Clustering

see more at http://ml.memect.com

Contents

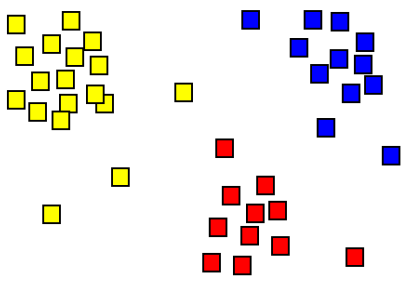
Chapter 1

Types

1.1 Cluster analysis

For the supervised learning approach, see Statistical classification.

Cluster analysis or **clustering** is the task of grouping a set of objects in such a way that objects in the same group (called a **cluster**) are more similar (in some sense or another) to each other than to those in other groups (clusters). It is a main task of exploratory data mining, and a common technique for statistical data analysis, used in many fields, including machine learning, pattern recognition, image analysis, information retrieval, and bioinformatics.


The result of a cluster analysis shown as the coloring of the squares into three clusters.

Cluster analysis itself is not one specific algorithm, but the general task to be solved. It can be achieved by various algorithms that differ significantly in their notion of what constitutes a cluster and how to efficiently find them. Popular notions of clusters include groups with small distances among the cluster members, dense areas of the data space, intervals or particular statistical distributions. Clustering can therefore be formulated as a multi-objective optimization problem. The appropriate clustering algorithm and parameter settings (including values such as the distance function to use, a density threshold or the number of expected clusters) depend on the individual data set and intended use of the results. Cluster analysis as such is not an automatic task, but an iterative process of knowledge discovery or interactive multi-objective optimization that involves trial and failure. It will often be necessary to modify data preprocessing and model parameters until the result achieves the desired properties.

Besides the term *clustering*, there are a number of terms with similar meanings, including *automatic classification*, *numerical taxonomy*, *botryology* (from Greek βότρυς "grape") and *typological analysis*. The subtle differences are often in the usage of the results: while in data mining, the resulting groups are the matter of interest, in automatic classification the resulting discriminative power is of interest. This often leads to misunderstandings between researchers coming from the fields of data mining and machine learning, since they use the same terms and often the same algorithms, but have different goals.

Cluster analysis was originated in anthropology by Driver and Kroeber in 1932 and introduced to psychology by Zubin in 1938 and Robert Tryon in 1939*[1]*[2] and famously used by Cattell beginning in 1943*[3] for trait theory classification in personality psychology.

1.1.1 Definition

According to Vladimir Estivill-Castro, the notion of a "cluster" cannot be precisely defined, which is one of the reasons why there are so many clustering algorithms.*[4] There is a common denominator: a group of data objects. However, different researchers employ different cluster models, and for each of these cluster models again different algorithms can be given. The notion of a cluster, as found by different algorithms, varies significantly in its properties. Understanding these "cluster models" is key to understanding the differences between the various algorithms. Typical cluster models include:

- Connectivity models: for example hierarchical clustering builds models based on distance connectivity.
- Centroid models: for example the k-means algorithm represents each cluster by a single mean vector.
- Distribution models: clusters are modeled using statistical distributions, such as multivariate normal

distributions used by the Expectation-maximization algorithm.

- Density models: for example DBSCAN and OPTICS defines clusters as connected dense regions in the data space.

- Subspace models: in Biclustering (also known as Co-clustering or two-mode-clustering), clusters are modeled with both cluster members and relevant attributes.

- Group models: some algorithms do not provide a refined model for their results and just provide the grouping information.

- Graph-based models: a clique, i.e., a subset of nodes in a graph such that every two nodes in the subset are connected by an edge can be considered as a prototypical form of cluster. Relaxations of the complete connectivity requirement (a fraction of the edges can be missing) are known as quasi-cliques.

A "clustering" is essentially a set of such clusters, usually containing all objects in the data set. Additionally, it may specify the relationship of the clusters to each other, for example a hierarchy of clusters embedded in each other. Clusterings can be roughly distinguished as:

- hard clustering: each object belongs to a cluster or not

- soft clustering (also: fuzzy clustering): each object belongs to each cluster to a certain degree (e.g. a likelihood of belonging to the cluster)

There are also finer distinctions possible, for example:

- strict partitioning clustering: here each object belongs to exactly one cluster

- strict partitioning clustering with outliers: objects can also belong to no cluster, and are considered outliers.

- overlapping clustering (also: alternative clustering, multi-view clustering): while usually a hard clustering, objects may belong to more than one cluster.

- hierarchical clustering: objects that belong to a child cluster also belong to the parent cluster

- subspace clustering: while an overlapping clustering, within a uniquely defined subspace, clusters are not expected to overlap.

1.1.2 Algorithms

Main category: Data clustering algorithms

Clustering algorithms can be categorized based on their cluster model, as listed above. The following overview will only list the most prominent examples of clustering algorithms, as there are possibly over 100 published clustering algorithms. Not all provide models for their clusters and can thus not easily be categorized. An overview of algorithms explained in Wikipedia can be found in the list of statistics algorithms.

There is no objectively "correct" clustering algorithm, but as it was noted, "clustering is in the eye of the beholder." *[4] The most appropriate clustering algorithm for a particular problem often needs to be chosen experimentally, unless there is a mathematical reason to prefer one cluster model over another. It should be noted that an algorithm that is designed for one kind of model has no chance on a data set that contains a radically different kind of model.*[4] For example, k-means cannot find non-convex clusters.*[4]

Connectivity based clustering (hierarchical clustering)

Main article: Hierarchical clustering

Connectivity based clustering, also known as *hierarchical clustering*, is based on the core idea of objects being more related to nearby objects than to objects farther away. These algorithms connect "objects" to form "clusters" based on their distance. A cluster can be described largely by the maximum distance needed to connect parts of the cluster. At different distances, different clusters will form, which can be represented using a dendrogram, which explains where the common name "hierarchical clustering" comes from: these algorithms do not provide a single partitioning of the data set, but instead provide an extensive hierarchy of clusters that merge with each other at certain distances. In a dendrogram, the y-axis marks the distance at which the clusters merge, while the objects are placed along the x-axis such that the clusters don't mix.

Connectivity based clustering is a whole family of methods that differ by the way distances are computed. Apart from the usual choice of distance functions, the user also needs to decide on the linkage criterion (since a cluster consists of multiple objects, there are multiple candidates to compute the distance to) to use. Popular choices are known as single-linkage clustering (the minimum of object distances), complete linkage clustering (the maximum of object distances) or UPGMA ("Unweighted Pair Group Method with Arithmetic Mean" , also known as average linkage clustering). Furthermore, hierarchical clustering can be agglomerative (starting with single elements and aggregating them into clusters) or divisive (starting with the complete data set and dividing it into partitions).

These methods will not produce a unique partitioning of the data set, but a hierarchy from which the user still needs to choose appropriate clusters. They are not very robust towards outliers, which will either show up as additional clusters or even cause other clusters to merge (known as "chaining phenomenon", in particular with single-linkage clustering). In the general case, the complexity is $\mathcal{O}(n^3)$, which makes them too slow for large data sets. For some special cases, optimal efficient methods (of complexity $\mathcal{O}(n^2)$) are known: SLINK*[5] for single-linkage and CLINK*[6] for complete-linkage clustering. In the data mining community these methods are recognized as a theoretical foundation of cluster analysis, but often considered obsolete. They did however provide inspiration for many later methods such as density based clustering.

- Linkage clustering examples
- Single-linkage on Gaussian data. At 35 clusters, the biggest cluster starts fragmenting into smaller parts, while before it was still connected to the second largest due to the single-link effect.
- Single-linkage on density-based clusters. 20 clusters extracted, most of which contain single elements, since linkage clustering does not have a notion of "noise".

Centroid-based clustering

Main article: k-means clustering

In centroid-based clustering, clusters are represented by a central vector, which may not necessarily be a member of the data set. When the number of clusters is fixed to k, *k*-means clustering gives a formal definition as an optimization problem: find the k cluster centers and assign the objects to the nearest cluster center, such that the squared distances from the cluster are minimized.

The optimization problem itself is known to be NP-hard, and thus the common approach is to search only for approximate solutions. A particularly well known approximative method is Lloyd's algorithm,*[7] often actually referred to as "*k-means algorithm*". It does however only find a local optimum, and is commonly run multiple times with different random initializations. Variations of k-means often include such optimizations as choosing the best of multiple runs, but also restricting the centroids to members of the data set (k-medoids), choosing medians (k-medians clustering), choosing the initial centers less randomly (K-means++) or allowing a fuzzy cluster assignment (Fuzzy c-means).

Most k-means-type algorithms require the number of clusters - k - to be specified in advance, which is considered to be one of the biggest drawbacks of these algorithms. Furthermore, the algorithms prefer clusters of approximately similar size, as they will always assign an object to the nearest centroid. This often leads to incorrectly cut borders in between of clusters (which is not surprising, as the algorithm optimized cluster centers, not cluster borders).

K-means has a number of interesting theoretical properties. On the one hand, it partitions the data space into a structure known as a Voronoi diagram. On the other hand, it is conceptually close to nearest neighbor classification, and as such is popular in machine learning. Third, it can be seen as a variation of model based classification, and Lloyd's algorithm as a variation of the Expectation-maximization algorithm for this model discussed below.

- k-Means clustering examples
- K-means separates data into Voronoi-cells, which assumes equal-sized clusters (not adequate here)
- K-means cannot represent density-based clusters

Distribution-based clustering

The clustering model most closely related to statistics is based on distribution models. Clusters can then easily be defined as objects belonging most likely to the same distribution. A convenient property of this approach is that this closely resembles the way artificial data sets are generated: by sampling random objects from a distribution.

While the theoretical foundation of these methods is excellent, they suffer from one key problem known as overfitting, unless constraints are put on the model complexity. A more complex model will usually be able to explain the data better, which makes choosing the appropriate model complexity inherently difficult.

One prominent method is known as Gaussian mixture models (using the expectation-maximization algorithm). Here, the data set is usually modelled with a fixed (to avoid overfitting) number of Gaussian distributions that are initialized randomly and whose parameters are iteratively optimized to fit better to the data set. This will converge to a local optimum, so multiple runs may produce different results. In order to obtain a hard clustering, objects are often then assigned to the Gaussian distribution they most likely belong to; for soft clusterings, this is not necessary.

Distribution-based clustering produces complex models for clusters that can capture correlation and dependence between attributes. However, these algorithms put an extra burden on the user: for many real data sets, there may be no concisely defined mathematical model (e.g. assuming Gaussian distributions is a rather strong assumption on the data).

- Expectation-Maximization (EM) clustering examples
- On Gaussian-distributed data, EM works well, since it uses Gaussians for modelling clusters
- Density-based clusters cannot be modeled using Gaussian distributions

Density-based clustering

In density-based clustering,*[8] clusters are defined as areas of higher density than the remainder of the data set. Objects in these sparse areas - that are required to separate clusters - are usually considered to be noise and border points.

The most popular*[9] density based clustering method is DBSCAN.*[10] In contrast to many newer methods, it features a well-defined cluster model called "density-reachability" . Similar to linkage based clustering, it is based on connecting points within certain distance thresholds. However, it only connects points that satisfy a density criterion, in the original variant defined as a minimum number of other objects within this radius. A cluster consists of all density-connected objects (which can form a cluster of an arbitrary shape, in contrast to many other methods) plus all objects that are within these objects' range. Another interesting property of DBSCAN is that its complexity is fairly low - it requires a linear number of range queries on the database - and that it will discover essentially the same results (it is deterministic for core and noise points, but not for border points) in each run, therefore there is no need to run it multiple times. OPTICS*[11] is a generalization of DBSCAN that removes the need to choose an appropriate value for the range parameter ε , and produces a hierarchical result related to that of linkage clustering. DeLi-Clu,*[12] Density-Link-Clustering combines ideas from single-linkage clustering and OPTICS, eliminating the ε parameter entirely and offering performance improvements over OPTICS by using an R-tree index.

The key drawback of DBSCAN and OPTICS is that they expect some kind of density drop to detect cluster borders. Moreover, they cannot detect intrinsic cluster structures which are prevalent in the majority of real life data. A variation of DBSCAN, EnDBSCAN,*[13] efficiently detects such kinds of structures. On data sets with, for example, overlapping Gaussian distributions - a common use case in artificial data - the cluster borders produced by these algorithms will often look arbitrary, because the cluster density decreases continuously. On a data set consisting of mixtures of Gaussians, these algorithms are nearly always outperformed by methods such as EM clustering that are able to precisely model this kind of data.

Mean-shift is a clustering approach where each object is moved to the densest area in its vicinity, based on kernel density estimation. Eventually, objects converge to local maxima of density. Similar to k-means clustering, these "density attractors" can serve as representatives for the data set, but mean-shift can detect arbitrary-shaped clusters similar to DBSCAN. Due to the expensive iterative procedure and density estimation, mean-shift is usually slower than DBSCAN or k-Means.

- Density-based clustering examples
- Density-based clustering with DBSCAN.
- DBSCAN assumes clusters of similar density, and may have problems separating nearby clusters
- OPTICS is a DBSCAN variant that handles different densities much better

Recent developments

In recent years considerable effort has been put into improving the performance of existing algorithms.*[14]*[15] Among them are *CLARANS* (Ng and Han, 1994),*[16] and *BIRCH* (Zhang et al., 1996).*[17] With the recent need to process larger and larger data sets (also known as big data), the willingness to trade semantic meaning of the generated clusters for performance has been increasing. This led to the development of pre-clustering methods such as canopy clustering, which can process huge data sets efficiently, but the resulting "clusters" are merely a rough pre-partitioning of the data set to then analyze the partitions with existing slower methods such as k-means clustering. Various other approaches to clustering have been tried such as seed based clustering.*[18]

For high-dimensional data, many of the existing methods fail due to the curse of dimensionality, which renders particular distance functions problematic in high-dimensional spaces. This led to new clustering algorithms for high-dimensional data that focus on subspace clustering (where only some attributes are used, and cluster models include the relevant attributes for the cluster) and correlation clustering that also looks for arbitrary rotated ("correlated") subspace clusters that can be modeled by giving a correlation of their attributes. Examples for such clustering algorithms are CLIQUE*[19] and SUBCLU.*[20]

Ideas from density-based clustering methods (in particular the DBSCAN/OPTICS family of algorithms) have been adopted to subspace clustering (HiSC,*[21] hierarchical subspace clustering and DiSH*[22]) and correlation clustering (HiCO,*[23] hierarchical correlation clustering, 4C*[24] using "correlation connectivity" and ERiC*[25] exploring hierarchical density-based correlation clusters).

Several different clustering systems based on mutual information have been proposed. One is Marina Meilă's *variation of information* metric;*[26] another provides hierarchical clustering.*[27] Using genetic algorithms, a wide range of different fit-functions can be optimized, including mutual information.*[28] Also message passing algorithms, a recent development in Computer Science and Statistical Physics, has led to the creation of new types of clustering algorithms.*[29]

Other methods

- Basic sequential algorithmic scheme (BSAS)

1.1.3 Evaluation and assessment

Evaluation of clustering results sometimes is referred to as cluster validation.

There have been several suggestions for a measure of similarity between two clusterings. Such a measure can be used to compare how well different data clustering algorithms perform on a set of data. These measures are usually tied to the type of criterion being considered in assessing the quality of a clustering method.

Internal evaluation

When a clustering result is evaluated based on the data that was clustered itself, this is called internal evaluation. These methods usually assign the best score to the algorithm that produces clusters with high similarity within a cluster and low similarity between clusters. One drawback of using internal criteria in cluster evaluation is that high scores on an internal measure do not necessarily result in effective information retrieval applications.*[30] Additionally, this evaluation is biased towards algorithms that use the same cluster model. For example k-Means clustering naturally optimizes object distances, and a distance-based internal criterion will likely overrate the resulting clustering.

Therefore, the internal evaluation measures are best suited to get some insight into situations where one algorithm performs better than another, but this shall not imply that one algorithm produces more valid results than another.*[4] Validity as measured by such an index depends on the claim that this kind of structure exists in the data set. An algorithm designed for some kind of models has no chance if the data set contains a radically different set of models, or if the evaluation measures a radically different criterion.*[4] For example, k-means clustering can only find convex clusters, and many evaluation indexes assume convex clusters. On a data set with non-convex clusters neither the use of k-means, nor of an evaluation criterion that assumes convexity, is sound.

The following methods can be used to assess the quality of clustering algorithms based on internal criterion:

- **Davies–Bouldin index**

  The Davies–Bouldin index can be calculated by the following formula:

  $$DB = \frac{1}{n}\sum_{i=1}^{n} \max_{j\neq i}\left(\frac{\sigma_i+\sigma_j}{d(c_i,c_j)}\right)$$

  where n is the number of clusters, c_x is the centroid of cluster x , σ_x is the average distance of all elements in cluster x to centroid c_x , and $d(c_i, c_j)$ is the distance between centroids c_i and c_j . Since algorithms that produce clusters with low intra-cluster distances (high intra-cluster similarity) and high inter-cluster distances (low inter-cluster similarity) will have a low Davies–Bouldin index, the clustering algorithm that produces a collection of clusters with the smallest Davies–Bouldin index is considered the best algorithm based on this criterion.

- **Dunn index**

  The Dunn index aims to identify dense and well-separated clusters. It is defined as the ratio between the minimal inter-cluster distance to maximal intra-cluster distance. For each cluster partition, the Dunn index can be calculated by the following formula:*[31]

  $$D = \frac{\min_{1\leq i<j\leq n} d(i,j)}{\max_{1\leq k\leq n} d'(k)} ,$$

  where *d*(*i*,*j*) represents the distance between clusters *i* and *j*, and *d* '(*k*) measures the intra-cluster distance of cluster *k*. The inter-cluster distance *d*(*i*,*j*) between two clusters may be any number of distance measures, such as the distance between the centroids of the clusters. Similarly, the intra-cluster distance *d* '(*k*) may be measured in a variety ways, such as the maximal distance between any pair of elements in cluster *k*. Since internal criterion seek clusters with high intra-cluster similarity and low inter-cluster similarity, algorithms that produce clusters with high Dunn index are more desirable.

- Silhouette coefficient

  The silhouette coefficient contrasts the average distance to elements in the same cluster with the average distance to elements in other clusters. Objects with a high silhouette value are considered well clustered, objects with a low value may be outliers. This index works well with k-means clustering, and is also used to determine the optimal number of clusters.

External evaluation

In external evaluation, clustering results are evaluated based on data that was not used for clustering, such as known class labels and external benchmarks. Such benchmarks consist of a set of pre-classified items, and these sets are often created by human (experts). Thus, the benchmark sets can be thought of as a gold standard for evaluation. These types of evaluation methods measure how close the clustering is to the predetermined benchmark classes. However, it has recently been discussed whether this is adequate for real data, or only on synthetic data sets with a factual ground truth, since classes can contain internal structure, the attributes present may not allow separation of clusters or the classes may contain anomalies.*[32] Additionally, from a knowledge discovery point of view, the reproduction of known knowledge may not necessarily be the intended result.*[32]

A number of measures are adapted from variants used to evaluate classification tasks. In place of counting the number of times a class was correctly assigned to a single data point (known as true positives), such *pair counting* metrics assess whether each pair of data points that is truly in the same cluster is predicted to be in the same cluster.

Some of the measures of quality of a cluster algorithm using external criterion include:

- **Rand measure** (William M. Rand 1971)*[33]

  The Rand index computes how similar the clusters (returned by the clustering algorithm) are to the benchmark classifications. One can also view the Rand index as a measure of the percentage of correct decisions made by the algorithm. It can be computed using the following formula:

  $RI = \frac{TP+TN}{TP+FP+FN+TN}$

  where TP is the number of true positives, TN is the number of true negatives, FP is the number of false positives, and FN is the number of false negatives. One issue with the Rand index is that false positives and false negatives are equally weighted. This may be an undesirable characteristic for some clustering applications. The F-measure addresses this concern, as does the chance-corrected adjusted Rand index.

- **F-measure**

  The F-measure can be used to balance the contribution of false negatives by weighting recall through a parameter $\beta \geq 0$. Let precision and recall be defined as follows:

  $P = \frac{TP}{TP+FP}$

  $R = \frac{TP}{TP+FN}$

  where P is the precision rate and R is the recall rate. We can calculate the F-measure by using the following formula:*[30]

  $F_\beta = \frac{(\beta^2+1)\cdot P\cdot R}{\beta^2\cdot P+R}$

  Notice that when $\beta = 0$, $F_0 = P$. In other words, recall has no impact on the F-measure when $\beta = 0$, and increasing β allocates an increasing amount of weight to recall in the final F-measure.

- **Jaccard index**

  The Jaccard index is used to quantify the similarity between two datasets. The Jaccard index takes on a value between 0 and 1. An index of 1 means that the two dataset are identical, and an index of 0 indicates that the datasets have no common elements. The Jaccard index is defined by the following formula:

  $J(A,B) = \frac{|A\cap B|}{|A\cup B|} = \frac{TP}{TP+FP+FN}$

  This is simply the number of unique elements common to both sets divided by the total number of unique elements in both sets.

- **Fowlkes–Mallows index** (E. B. Fowlkes & C. L. Mallows 1983)*[34]

  The Fowlkes-Mallows index computes the similarity between the clusters returned by the clustering algorithm and the benchmark classifications. The higher the value of the Fowlkes-Mallows index the more similar the clusters and the benchmark classifications are. It can be computed using the following formula:

  $FM = \sqrt{\frac{TP}{TP+FP}\cdot\frac{TP}{TP+FN}}$

  where TP is the number of true positives, FP is the number of false positives, and FN is the number of false negatives. The FM index is the geometric mean of the precision and recall P and R , while the F-measure is their harmonic mean.*[35] Moreover, precision and recall are also known as Wallace's indices B^I and B^{II} .*[36]

- The **Mutual Information** is an information theoretic measure of how much information is shared between a clustering and a ground-truth classification that can detect a non-linear similarity between two clusterings. Adjusted mutual information is the corrected-for-chance variant of this that has a reduced bias for varying cluster numbers.

- **Confusion matrix**

A confusion matrix can be used to quickly visualize the results of a classification (or clustering) algorithm. It shows how different a cluster is from the gold standard cluster.

1.1.4 Applications

1.1.5 See also

Specialized types of cluster analysis

Others
Social science
Computer science
World wide web
usiness and marketing
ine
mputational biology and bioinformatics

nimal ecologycluster analysis is used to be and to make spatial and temporal com- ons of communities (assemblages) of or- ns in heterogeneous environments; it is sed in plant systematics to generate artifi- nylogenies or clusters of organisms (indi- s) at the species, genus or higher level that a number of attributes

omicsclustering is used to build groups nes with related expression patterns (also n as coexpressed genes). Often such s contain functionally related proteins, s enzymes for a specific pathway, or genes re co-regulated. High throughput exper- s using expressed sequence tags (ESTs) NA microarrays can be a powerful tool enome annotation, a general aspect of nics.

Medical imaging

nalysisclustering is used to group homol- sequences into gene families. This is a mportant concept in bioinformatics, and ionary biology in general. See evolution ne duplication.

ghput genotyping platformsclustering al- ms are used to automatically assign geno-

Market research

etic clusteringThe similarity of genetic s used in clustering to infer population ures.

On PET scans, cluster analysis can be used to ifferentiate between different types of tissue nd blood in a three-dimensional image. In this pplication, actual position does not matter, ut the voxel intensity is considered as a vector, ith a dimension for each image that was taken ver time. This technique allows, for example, ccurate measurement of the rate a radioactive racer is delivered to the area of interest, ithout a separate sampling of arterial blood, n intrusive technique that is most common oday.

Social network analysis

sis of antimicrobial activityCluster analysis an be used to analyse patterns of antibiotic esistance, to classify antimicrobial compounds ccording to their mechanism of action, to clas- ify antibiotics according to their antibacterial ctivity.

Software evolution

segmentationClustering can be used to divide fluence map into distinct regions for conver- ion into deliverable fields in MLC-based Radi- tion Therapy.

Crime analysis

Cluster analysis is widely used in market re-

Clustering high-dimensional data

- Conceptual clustering
- Consensus clustering
- Constrained clustering
- Data stream clustering
- Sequence clustering
- Spectral clustering

Techniques used in cluster analysis

- Artificial neural network (ANN)
- Nearest neighbor search
- Neighbourhood components analysis
- Latent class analysis

Data projection and preprocessing

- Dimension reduction
- Principal component analysis
- Multidimensional scaling

Other

- Cluster-weighted modeling
- Curse of dimensionality
- Determining the number of clusters in a data set
- Parallel coordinates
- Structured data analysis

1.1.6 References

[1] Bailey, Ken (1994). "Numerical Taxonomy and Cluster Analysis". *Typologies and Taxonomies*. p. 34. ISBN 9780803952591.

[2] Tryon, Robert C. (1939). *Cluster Analysis: Correlation Profile and Orthometric (factor) Analysis for the Isolation of Unities in Mind and Personality*. Edwards Brothers.

[3] Cattell, R. B. (1943). "The description of personality: Basic traits resolved into clusters". *Journal of Abnormal and Social Psychology* **38**: 476–506. doi:10.1037/h0054116.

[4] Estivill-Castro, Vladimir (20 June 2002). "Why so many clustering algorithms —A Position Paper". *ACM SIGKDD Explorations Newsletter* **4** (1): 65–75. doi:10.1145/568574.568575.

[5] Sibson, R. (1973). "SLINK: an optimally efficient algorithm for the single-link cluster method" (PDF). *The Computer Journal* (British Computer Society) **16** (1): 30–34. doi:10.1093/comjnl/16.1.30.

[6] Defays, D. (1977). "An efficient algorithm for a complete link method" . *The Computer Journal* (British Computer Society) **20** (4): 364–366. doi:10.1093/comjnl/20.4.364.

[7] Lloyd, S. (1982). "Least squares quantization in PCM" . *IEEE Transactions on Information Theory* **28** (2): 129–137. doi:10.1109/TIT.1982.1056489.

[8] Kriegel, Hans-Peter; Kröger, Peer; Sander, Jörg; Zimek, Arthur (2011). "Density-based Clustering" . *WIREs Data Mining and Knowledge Discovery* **1** (3): 231–240. doi:10.1002/widm.30.

[9] Microsoft academic search: most cited data mining articles: DBSCAN is on rank 24, when accessed on: 4/18/2010

[10] Ester, Martin; Kriegel, Hans-Peter; Sander, Jörg; Xu, Xiaowei (1996). "A density-based algorithm for discovering clusters in large spatial databases with noise" . In Simoudis, Evangelos; Han, Jiawei; Fayyad, Usama M. *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96)*. AAAI Press. pp. 226–231. ISBN 1-57735-004-9. CiteSeerX: 10.1.1.71.1980.

[11] Ankerst, Mihael; Breunig, Markus M.; Kriegel, Hans-Peter; Sander, Jörg (1999). "OPTICS: Ordering Points To Identify the Clustering Structure" . *ACM SIGMOD international conference on Management of data*. ACM Press. pp. 49–60. CiteSeerX: 10.1.1.129.6542.

[12] Achtert, E.; Böhm, C.; Kröger, P. (2006). "DeLi-Clu: Boosting Robustness, Completeness, Usability, and Efficiency of Hierarchical Clustering by a Closest Pair Ranking" . *LNCS: Advances in Knowledge Discovery and Data Mining*. Lecture Notes in Computer Science **3918**: 119–128. doi:10.1007/11731139_16. ISBN 978-3-540-33206-0.

[13] Roy, S.; Bhattacharyya, D. K. (2005). "An Approach to find Embedded Clusters Using Density Based Techniques" . *LNCS Vol.3816*. Springer Verlag. pp. 523–535.

[14] Sculley, D. (2010). *Web-scale k-means clustering*. Proc. 19th WWW.

[15] Huang, Z. (1998). "Extensions to the *k*-means algorithm for clustering large data sets with categorical values" . *Data Mining and Knowledge Discovery* **2**: 283–304.

[16] R. Ng and J. Han. "Efficient and effective clustering method for spatial data mining" . In: Proceedings of the 20th VLDB Conference, pages 144-155, Santiago, Chile, 1994.

[17] Tian Zhang, Raghu Ramakrishnan, Miron Livny. "An Efficient Data Clustering Method for Very Large Databases."In: Proc. Int'l Conf. on Management of Data, ACM SIGMOD, pp. 103–114.

[18] Can, F.; Ozkarahan, E. A. (1990). "Concepts and effectiveness of the cover-coefficient-based clustering methodology for text databases" . *ACM Transactions on Database Systems* **15** (4): 483. doi:10.1145/99935.99938.

[19] Agrawal, R.; Gehrke, J.; Gunopulos, D.; Raghavan, P. (2005). "Automatic Subspace Clustering of High Dimensional Data" . *Data Mining and Knowledge Discovery* **11**: 5. doi:10.1007/s10618-005-1396-1.

[20] Karin Kailing, Hans-Peter Kriegel and Peer Kröger. *Density-Connected Subspace Clustering for High-Dimensional Data*. In: *Proc. SIAM Int. Conf. on Data Mining (SDM'04)*, pp. 246-257, 2004.

[21] Achtert, E.; Böhm, C.; Kriegel, H. P.; Kröger, P.; Müller-Gorman, I.; Zimek, A. (2006). "Finding Hierarchies of Subspace Clusters" . *LNCS: Knowledge Discovery in Databases: PKDD 2006*. Lecture Notes in Computer Science **4213**: 446–453. doi:10.1007/11871637_42. ISBN 978-3-540-45374-1.

[22] Achtert, E.; Böhm, C.; Kriegel, H. P.; Kröger, P.; Müller-Gorman, I.; Zimek, A. (2007). "Detection and Visualization of Subspace Cluster Hierarchies" . *LNCS: Advances in Databases: Concepts, Systems and Applications*. Lecture Notes in Computer Science **4443**: 152–163. doi:10.1007/978-3-540-71703-4_15. ISBN 978-3-540-71702-7.

[23] Achtert, E.; Böhm, C.; Kröger, P.; Zimek, A. (2006). "Mining Hierarchies of Correlation Clusters" . *Proc. 18th International Conference on Scientific and Statistical Database Management (SSDBM)*: 119–128. doi:10.1109/SSDBM.2006.35. ISBN 0-7695-2590-3.

[24] Böhm, C.; Kailing, K.; Kröger, P.; Zimek, A. (2004). "Computing Clusters of Correlation Connected objects" . *Proceedings of the 2004 ACM SIGMOD international conference on Management of data - SIGMOD '04*. p. 455. doi:10.1145/1007568.1007620. ISBN 1581138598.

[25] Achtert, E.; Bohm, C.; Kriegel, H. P.; Kröger, P.; Zimek, A. (2007). "On Exploring Complex Relationships of Correlation Clusters" . *19th International Conference on Scientific and Statistical Database Management (SSDBM 2007)*. p. 7. doi:10.1109/SSDBM.2007.21. ISBN 0-7695-2868-6.

[26] Meilă, Marina (2003). "Comparing Clusterings by the Variation of Information" . *Learning Theory and Kernel Machines*. Lecture Notes in Computer Science **2777**: 173–187. doi:10.1007/978-3-540-45167-9_14. ISBN 978-3-540-40720-1.

[27] Kraskov, Alexander; Stögbauer, Harald; Andrzejak, Ralph G.; Grassberger, Peter (1 December 2003) [28 November 2003]. "Hierarchical Clustering Based on Mutual Information" . arXiv:q-bio/0311039.

[28] Auffarth, B. (July 18–23, 2010). "Clustering by a Genetic Algorithm with Biased Mutation Operator" . *WCCI CEC* (IEEE). CiteSeerX: 10.1.1.170.869.

[29] Frey, B. J.; Dueck, D. (2007). "Clustering by Passing Messages Between Data Points" . *Science* **315** (5814): 972–976. doi:10.1126/science.1136800. PMID 17218491.

[30] Manning, Christopher D.; Raghavan, Prabhakar; Schütze, Hinrich. *Introduction to Information Retrieval*. Cambridge University Press. ISBN 978-0-521-86571-5.

[31] Dunn, J. (1974). "Well separated clusters and optimal fuzzy partitions". *Journal of Cybernetics* **4**: 95–104. doi:10.1080/01969727408546059.

[32] Färber, Ines; Günnemann, Stephan; Kriegel, Hans-Peter; Kröger, Peer; Müller, Emmanuel; Schubert, Erich; Seidl, Thomas; Zimek, Arthur (2010). "On Using Class-Labels in Evaluation of Clusterings" (PDF). In Fern, Xiaoli Z.; Davidson, Ian; Dy, Jennifer. *MultiClust: Discovering, Summarizing, and Using Multiple Clusterings*. ACM SIGKDD.

[33] Rand, W. M. (1971). "Objective criteria for the evaluation of clustering methods". *Journal of the American Statistical Association* (American Statistical Association) **66** (336): 846–850. doi:10.2307/2284239. JSTOR 2284239.

[34] E. B. Fowlkes & C. L. Mallows (1983), "A Method for Comparing Two Hierarchical Clusterings", Journal of the American Statistical Association 78, 553–569.

[35] L. Hubert et P. Arabie. Comparing partitions. J. of Classification, 2(1), 1985.

[36] D. L. Wallace. Comment. Journal of the American Statistical Association, 78 :569– 579, 1983.

[37] Bewley, A. et al. "Real-time volume estimation of a dragline payload". *IEEE International Conference on Robotics and Automation* **2011**: 1571–1576.

[38] Basak, S.C.; Magnuson, V.R.; Niemi, C.J.; Regal, R.R. "Determining Structural Similarity of Chemicals Using Graph Theoretic Indices". *Discr. Appl. Math., 19* **1988**: 17–44.

[39] Huth, R. et al. (2008). "Classifications of Atmospheric Circulation Patterns: Recent Advances and Applications". *Ann. N.Y. Acad. Sci.* **1146**: 105–152.

1.1.7 External links

- Data Mining at DMOZ

1.2 Hierarchical clustering

In data mining, **hierarchical clustering** (also called **hierarchical cluster analysis** or **HCA**) is a method of cluster analysis which seeks to build a hierarchy of clusters. Strategies for hierarchical clustering generally fall into two types: [1]

- **Agglomerative**: This is a "bottom up" approach: each observation starts in its own cluster, and pairs of clusters are merged as one moves up the hierarchy.
- **Divisive**: This is a "top down" approach: all observations start in one cluster, and splits are performed recursively as one moves down the hierarchy.

In general, the merges and splits are determined in a greedy manner. The results of hierarchical clustering are usually presented in a dendrogram.

In the general case, the complexity of agglomerative clustering is $O(n^3)$, which makes them too slow for large data sets. Divisive clustering with an exhaustive search is $O(2^n)$, which is even worse. However, for some special cases, optimal efficient agglomerative methods (of complexity $O(n^2)$) are known: SLINK[2] for single-linkage and CLINK[3] for complete-linkage clustering.

1.2.1 Cluster dissimilarity

In order to decide which clusters should be combined (for agglomerative), or where a cluster should be split (for divisive), a measure of dissimilarity between sets of observations is required. In most methods of hierarchical clustering, this is achieved by use of an appropriate metric (a measure of distance between pairs of observations), and a linkage criterion which specifies the dissimilarity of sets as a function of the pairwise distances of observations in the sets.

Metric

Further information: metric (mathematics)

The choice of an appropriate metric will influence the shape of the clusters, as some elements may be close to one another according to one distance and farther away according to another. For example, in a 2-dimensional space, the distance between the point (1,0) and the origin (0,0) is always 1 according to the usual norms, but the distance between the point (1,1) and the origin (0,0) can be 2 under Manhattan distance, $\sqrt{2}$ under Euclidean distance, or 1 under maximum distance.

Some commonly used metrics for hierarchical clustering are:[4]

For text or other non-numeric data, metrics such as the Hamming distance or Levenshtein distance are often used.

A review of cluster analysis in health psychology research found that the most common distance measure in published studies in that research area is the Euclidean distance or the squared Euclidean distance.

Linkage criteria

The linkage criterion determines the distance between sets of observations as a function of the pairwise distances between observations.

Some commonly used linkage criteria between two sets of observations *A* and *B* are:*[5]*[6]

where *d* is the chosen metric. Other linkage criteria include:

- The sum of all intra-cluster variance.
- The decrease in variance for the cluster being merged (Ward's criterion).*[7]
- The probability that candidate clusters spawn from the same distribution function (V-linkage).
- The product of in-degree and out-degree on a k-nearest-neighbor graph (graph degree linkage).*[8]
- The increment of some cluster descriptor (i.e., a quantity defined for measuring the quality of a cluster) after merging two clusters.*[9]*[10]*[11]

1.2.2 Discussion

Hierarchical clustering has the distinct advantage that any valid measure of distance can be used. In fact, the observations themselves are not required: all that is used is a matrix of distances.

1.2.3 Example for Agglomerative Clustering

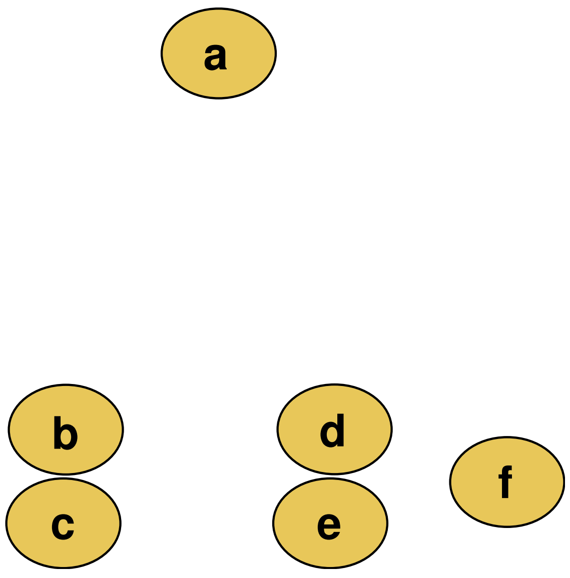
For example, suppose this data is to be clustered, and the Euclidean distance is the distance metric.

Cutting the tree at a given height will give a partitioning clustering at a selected precision. In this example, cutting after the second row of the dendrogram will yield clusters {a} {b c} {d e} {f}. Cutting after the third row will yield clusters {a} {b c} {d e f}, which is a coarser clustering, with a smaller number but larger clusters.

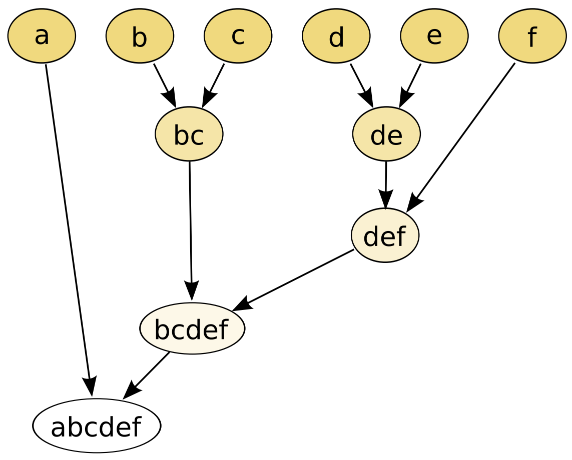
The hierarchical clustering dendrogram would be as such:

This method builds the hierarchy from the individual elements by progressively merging clusters. In our example, we have six elements {a} {b} {c} {d} {e} and {f}. The first step is to determine which elements to merge in a cluster. Usually, we want to take the two closest elements, according to the chosen distance.

Optionally, one can also construct a distance matrix at this stage, where the number in the *i*-th row *j*-th column is the distance between the *i*-th and *j*-th elements. Then, as clustering progresses, rows and columns are merged as the clusters are merged and the distances updated. This is a common way to implement this type of clustering, and has the benefit of caching distances between clusters. A simple agglomerative clustering algorithm is described in the single-linkage clustering page; it can easily be adapted to different types of linkage (see below).



Raw data



Traditional representation

Suppose we have merged the two closest elements *b* and *c*, we now have the following clusters $\{a\}$, $\{b, c\}$, $\{d\}$, $\{e\}$ and $\{f\}$, and want to merge them further. To do that, we need to take the distance between {a} and {b c}, and therefore define the distance between two clusters. Usually the distance between two clusters $\mathcal{A}$ and $\mathcal{B}$ is one of the following:

- The maximum distance between elements of each cluster (also called complete-linkage clustering):

$$\max\{\, d(x,y) : x \in \mathcal{A},\, y \in \mathcal{B}\,\}.$$

- The minimum distance between elements of each cluster (also called single-linkage clustering):

$$\min\{\, d(x,y) : x \in \mathcal{A},\, y \in \mathcal{B}\,\}.$$

- The mean distance between elements of each cluster (also called average linkage clustering, used e.g. in UPGMA):

$$\frac{1}{|\mathcal{A}|\cdot|\mathcal{B}|}\sum_{x\in\mathcal{A}}\sum_{y\in\mathcal{B}}d(x,y).$$

- The sum of all intra-cluster variance.
- The increase in variance for the cluster being merged (Ward's method<ref name="*[7])
- The probability that candidate clusters spawn from the same distribution function (V-linkage).

Each agglomeration occurs at a greater distance between clusters than the previous agglomeration, and one can decide to stop clustering either when the clusters are too far apart to be merged (distance criterion) or when there is a sufficiently small number of clusters (number criterion).

1.2.4 Software

Open Source Frameworks

- R has several functions for hierarchical clustering: see CRAN Task View: Cluster Analysis & Finite Mixture Models for more information.
- Cluster 3.0 provides a nice Graphical User Interface to access to different clustering routines and is available for Windows, Mac OS X, Linux, Unix.
- ELKI includes multiple hierarchical clustering algorithms, various linkage strategies and also includes the efficient SLINK*[2] algorithm, flexible cluster extraction from dendrograms and various other cluster analysis algorithms.
- Octave, the GNU analog to MATLAB implements hierarchical clustering in linkage function
- Orange, a free data mining software suite, module orngClustering for scripting in Python, or cluster analysis through visual programming.
- scikit-learn implements a hierarchical clustering.
- Weka includes hierarchical cluster analysis.
- fastCluster efficiently implements the seven most widely used clustering schemes.
- SCaViS computing environment in Java that implements this algorithm.

Standalone implementations

- CrimeStat implements two hierarchical clustering routines, a nearest neighbor (Nnh) and a risk-adjusted(Rnnh).
- figue is a JavaScript package that implements some agglomerative clustering functions (single-linkage, complete-linkage, average-linkage) and functions to visualize clustering output (e.g. dendrograms).
- hcluster is a Python implementation, based on NumPy, which supports hierarchical clustering and plotting.
- Hierarchical Agglomerative Clustering implemented as C# visual studio project that includes real text files processing, building of document-term matrix with stop words filtering and stemming.
- MultiDendrograms An open source Java application for variable-group agglomerative hierarchical clustering, with graphical user interface.
- Graph Agglomerative Clustering (GAC) toolbox implemented several graph-based agglomerative clustering algorithms.
- Hierarchical Clustering Explorer provides tools for interactive exploration of multidimensional data.

Commercial

- MATLAB includes hierarchical cluster analysis.
- SAS includes hierarchical cluster analysis.
- Mathematica includes a Hierarchical Clustering Package.
- NCSS (statistical software) includes hierarchical cluster analysis.
- SPSS includes hierarchical cluster analysis.
- Qlucore Omics Explorer includes hierarchical cluster analysis.
- Stata includes hierarchical cluster analysis.

1.2.5 See also

- Statistical distance
- Brown clustering
- Cluster analysis
- CURE data clustering algorithm
- Dendrogram
- Determining the number of clusters in a data set

- Hierarchical clustering of networks
- Nearest-neighbor chain algorithm
- Numerical taxonomy
- OPTICS algorithm
- Nearest neighbor search
- Locality-sensitive hashing

1.2.6 Notes

[1] Rokach, Lior, and Oded Maimon. "Clustering methods." Data mining and knowledge discovery handbook. Springer US, 2005. 321-352.

[2] R. Sibson (1973). "SLINK: an optimally efficient algorithm for the single-link cluster method" (PDF). *The Computer Journal* (British Computer Society) **16** (1): 30–34. doi:10.1093/comjnl/16.1.30.

[3] D. Defays (1977). "An efficient algorithm for a complete link method" . *The Computer Journal* (British Computer Society) **20** (4): 364–366. doi:10.1093/comjnl/20.4.364.

[4] "The DISTANCE Procedure: Proximity Measures" . *SAS/STAT 9.2 Users Guide*. SAS Institute. Retrieved 2009-04-26.

[5] "The CLUSTER Procedure: Clustering Methods" . *SAS/STAT 9.2 Users Guide*. SAS Institute. Retrieved 2009-04-26.

[6] Székely, G. J. and Rizzo, M. L. (2005) Hierarchical clustering via Joint Between-Within Distances: Extending Ward's Minimum Variance Method, Journal of Classification 22, 151-183.

[7] Ward, Joe H. (1963). "Hierarchical Grouping to Optimize an Objective Function" . *Journal of the American Statistical Association* **58** (301): 236–244. doi:10.2307/2282967. JSTOR 2282967. MR 0148188.

[8] Zhang, et al. "Graph degree linkage: Agglomerative clustering on a directed graph." 12th European Conference on Computer Vision, Florence, Italy, October 7–13, 2012. http://arxiv.org/abs/1208.5092

[9] Zhang, et al. "Agglomerative clustering via maximum incremental path integral." Pattern Recognition (2013).

[10] Zhao, and Tang. "Cyclizing clusters via zeta function of a graph."Advances in Neural Information Processing Systems. 2008.

[11] Ma, et al. "Segmentation of multivariate mixed data via lossy data coding and compression."IEEE Transactions on Pattern Analysis and Machine Intelligence, 29(9) (2007): 1546-1562.

1.2.7 References and further reading

- Kaufman, L.; Rousseeuw, P.J. (1990). *Finding Groups in Data: An Introduction to Cluster Analysis* (1 ed.). New York: John Wiley. ISBN 0-471-87876-6.
- Hastie, Trevor; Tibshirani, Robert; Friedman, Jerome (2009). "14.3.12 Hierarchical clustering" . *The Elements of Statistical Learning* (PDF) (2nd ed.). New York: Springer. pp. 520–528. ISBN 0-387-84857-6. Retrieved 2009-10-20.
- Press, WH; Teukolsky, SA; Vetterling, WT; Flannery, BP (2007). "Section 16.4. Hierarchical Clustering by Phylogenetic Trees" . *Numerical Recipes: The Art of Scientific Computing* (3rd ed.). New York: Cambridge University Press. ISBN 978-0-521-88068-8.
- Hierarchical Cluster Analysis
- Free statistical software. An overview of statistical software and methods used in published microbiological studies

1.3 Conceptual clustering

Conceptual clustering is a machine learning paradigm for unsupervised classification developed mainly during the 1980s. It is distinguished from ordinary data clustering by generating a **concept description** for each generated class. Most conceptual clustering methods are capable of generating hierarchical category structures; see Categorization for more information on hierarchy. Conceptual clustering is closely related to formal concept analysis, decision tree learning, and mixture model learning.

1.3.1 Conceptual clustering vs. data clustering

Conceptual clustering is obviously closely related to data clustering; however, in conceptual clustering it is not only the inherent structure of the data that drives cluster formation, but also the Description language (disambiguation) which is available to the learner. Thus, a statistically strong grouping in the data may fail to be extracted by the learner if the prevailing concept description language is incapable of describing that particular *regularity*. In most implementations, the description language has been limited to feature conjunction, although in COBWEB (see "COBWEB" below), the feature language is probabilistic.

1.3.2 List of published algorithms

A fair number of algorithms have been proposed for conceptual clustering. Some examples are given below:

- CLUSTER/2 (Michalski & Stepp 1983)
- COBWEB (Fisher 1987)
- CYRUS (Kolodner 1983)
- GALOIS (Carpineto & Romano 1993),
- GCF (Talavera & Béjar 2001)
- INC (Hadzikadic & Yun 1989)
- ITERATE (Biswas, Weinberg & Fisher 1998),
- LABYRINTH (Thompson & Langley 1989)
- SUBDUE (Jonyer, Cook & Holder 2001).
- UNIMEM (Lebowitz 1987)
- WITT (Hanson & Bauer 1989),

More general discussions and reviews of conceptual clustering can be found in the following publications:

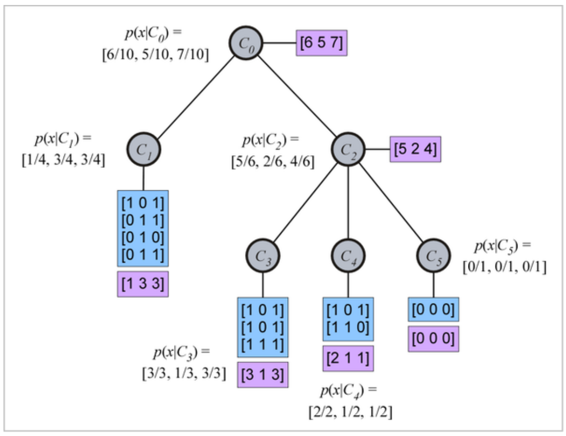
- Michalski (1980)
- Gennari, Langley, & Fisher (1989)
- Fisher & Pazzani (1991)
- Fisher & Langley (1986)
- Stepp & Michalski (1986)

1.3.3 Example: A basic conceptual clustering algorithm

This section discusses the rudiments of the conceptual clustering algorithm COBWEB. There are many other algorithms using different heuristics and "category goodness" or category evaluation criteria, but COBWEB is one of the best known. The reader is referred to the bibliography for other methods.

Knowledge representation

The COBWEB data structure is a hierarchy (tree) wherein each node represents a given *concept*. Each concept represents a set (actually, a multiset or bag) of objects, each object being represented as a binary-valued property list. The data associated with each tree node (i.e., concept) are the integer property counts for the objects in that concept. For example (see figure), let a concept C_1 contain the following four objects (repeated objects being permitted).



Sample COBWEB knowledge representation, probabilistic concept hierarchy. Blue boxes list actual objects, purple boxes list attribute counts. See text for details. ***Note:*** *The diagram is intended to be illustrative only of COBWEB's data structure; it does not necessarily represent a "good" concept tree, or one that COBWEB would actually construct from real data.*

1. [1 0 1]
2. [0 1 1]
3. [0 1 0]
4. [0 1 1]

The three properties might be, for example, [is_male, has_wings, is_nocturnal]. Then what is stored at this concept node is the property count [1 3 3], indicating that 1 of the objects in the concept is male, 3 of the objects have wings, and 3 of the objects are nocturnal. The concept *description* is the category-conditional probability (likelihood) of the properties at the node. Thus, given that an object is a member of category (concept) C_1 , the likelihood that it is male is $1/4 = 0.25$. Likewise, the likelihood that the object has wings and likelihood that the object is nocturnal or both is $3/4 = 0.75$. The concept description can therefore simply be given as [.25 .75 .75], which corresponds to the C_1 -conditional feature likelihood, i.e., $p(x|C_1) = (0.25, 0.75, 0.75)$.

The figure to the right shows a concept tree with five concepts. C_0 is the root concept, which contains all ten objects in the data set. Concepts C_1 and C_2 are the children of C_0 , the former containing four objects, and the later containing six objects. Concept C_2 is also the parent of concepts C_3 , C_4 , and C_5 , which contain three, two, and one object, respectively. Note that each parent node (relative superordinate concept) contains all the objects contained by its child nodes (relative subordinate concepts). In Fisher's (1987) description of COBWEB, he indicates that only the total attribute counts (not conditional probabilities, and not object lists) are stored at the nodes. Any probabilities are computed from the attribute counts as needed.

The COBWEB language The description language of COBWEB is a “language” only in a loose sense, because being fully probabilistic it is capable of describing any concept. However, if constraints are placed on the probability ranges which concepts may represent, then a stronger language is obtained. For example, we might permit only concepts wherein at least one probability differs from 0.5 by more than α . Under this constraint, with $\alpha = 0.3$, a concept such as [.6 .5 .7] could not be constructed by the learner; however a concept such as [.6 .5 .9] would be accessible because at least one probability differs from 0.5 by more than α . Thus, under constraints such as these, we obtain something like a traditional concept language. In the limiting case where $\alpha = 0.5$ for every feature, and thus every probability in a concept must be 0 or 1, the result is a feature language base on conjunction; that is, every concept that can be represented can then be described as a conjunction of features (and their negations), and concepts that cannot be described in this way cannot be represented.

Evaluation criterion

In Fisher's (1987) description of COBWEB, the measure he uses to evaluate the quality of the hierarchy is Gluck and Corter's (1985) category utility (CU) measure, which he re-derives in his paper. The motivation for the measure is highly similar to the "information gain" measure introduced by Quinlan for decision tree learning. It has previously been shown that the CU for feature-based classification is the same as the mutual information between the feature variables and the class variable (Gluck & Corter, 1985; Corter & Gluck, 1992), and since this measure is much better known, we proceed here with mutual information as the measure of category “goodness” .

What we wish to evaluate is the overall utility of grouping the objects into a particular hierarchical categorization structure. Given a set of possible classification structures, we need to determine whether one is better than another.

1.3.4 References

- Biswas, G.; Weinberg, J. B.; Fisher, Douglas H. (1998). “Iterate: A conceptual clustering algorithm for data mining” . *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews* **28**: 100–111.
- Carpineto, C.; Romano, G. (1993). “Galois: An order-theoretic approach to conceptual clustering” . *Proceedings of 10th International Conference on Machine Learning, Amherst*. pp. 33–40.
- Fisher, Douglas H. (1987). “Knowledge acquisition via incremental conceptual clustering” . *Machine Learning* **2** (2): 139–172. doi:10.1007/BF00114265.
- Fisher, Douglas H. (1996). “Iterative optimization and simplification of hierarchical clusterings” . *Journal of Artificial Intelligence Research* **4**: 147–178. doi:10.1613/jair.276.
- Fisher, Douglas H.; Langley, Patrick W. (1986). “Conceptual clustering and its relation to numerical taxonomy” . In Gale, W. A. (Ed.). *Artificial Intelligence and Statistics*. Reading, MA: Addison-Wesley. pp. 77–116.
- Fisher, Douglas H.; Pazzani, Michael J. (1991). “Computational models of concept learning” . In Fisher, D. H.; Pazzani, M. J.; Langley, P. (Eds.). *Concept Formation: Knowledge and Experience in Unsupervised Learning*. San Mateo, CA: Morgan Kaufmann. pp. 3–43.
- Gennari, John H.; Langley, Patrick W.; Fisher, Douglas H. (1989). “Models of incremental concept formation” . *Artificial Intelligence* **40**: 11–61. doi:10.1016/0004-3702(89)90046-5.
- Hanson, S. J.; Bauer, M. (1989). “Conceptual clustering, categorization, and polymorphy” . *Machine Learning* **3** (4): 343–372. doi:10.1007/BF00116838.
- Jonyer, I.; Cook, D. J.; Holder, L. B. (2001). “Graph-based hierarchical conceptual clustering” . *Journal of Machine Learning Research* **2**: 19–43. doi:10.1162/153244302760185234.
- Lebowitz, M. (1987). “Experiments with incremental concept formation” . *Machine Learning* **2** (2): 103–138. doi:10.1007/BF00114264.
- Michalski, R. S. (1980). “Knowledge acquisition through conceptual clustering: A theoretical framework and an algorithm for partitioning data into conjunctive concepts” . *International Journal of Policy Analysis and Information Systems* **4**: 219–244.
- Michalski, R. S.; Stepp, R. E. (1983). “Learning from observation: Conceptual clustering” . In Michalski, R. S.; Carbonell, J. G.; Mitchell, T. M. (Eds.). *Machine Learning: An Artificial Intelligence Approach*. Palo Alto, CA: Tioga. pp. 331–363.
- Stepp, R. E.; Michalski, R. S. (1986). “Conceptual clustering: Inventing goal-oriented classifications of structured objects” . In Michalski, R. S.; Carbonell, J. G.; Mitchell, T. M. (Eds.). *Machine Learning: An Artificial Intelligence Approach*. Los Altos, CA: Morgan Kaufmann. pp. 471–498.

- Talavera, L.; Béjar, J. (2001). "Generality-based conceptual clustering with probabilistic concepts" . *IEEE Transactions on Pattern Analysis and Machine Intelligence* **23** (2): 196–206. doi:10.1109/34.908969.

1.3.5 External links

- Bibliography of conceptual clustering
- Working python implementation of COBWEB

1.4 Consensus clustering

Clustering is the assignment of objects into groups (called *clusters*) so that objects from the same cluster are more similar to each other than objects from different clusters.*[1] Often similarity is assessed according to a distance measure. Clustering is a common technique for statistical data analysis, which is used in many fields, including machine learning, data mining, pattern recognition, image analysis*[2]*[3] and bioinformatics.

Consensus clustering has emerged as an important elaboration of the classical clustering problem. Consensus clustering, also called aggregation of clustering (or partitions), refers to the situation in which a number of different (input) clusterings have been obtained for a particular dataset and it is desired to find a single (consensus) clustering which is a better fit in some sense than the existing clusterings.*[4]*[5] Consensus clustering is thus the problem of reconciling clustering information about the same data set coming from different sources or from different runs of the same algorithm. When cast as an optimization problem, consensus clustering is known as median partition, and has been shown to be NP-complete.*[6] Consensus clustering for unsupervised learning is analogous to ensemble learning in supervised learning.*[5]

1.4.1 Issues with existing clustering techniques

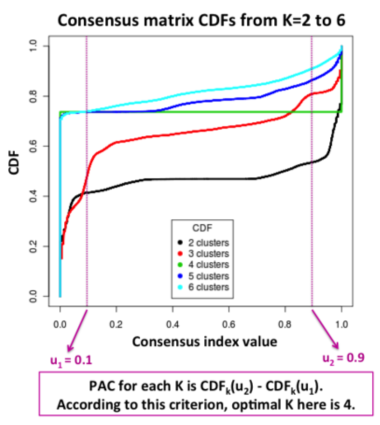
- Current clustering techniques do not address all the requirements adequately.
- Dealing with large number of dimensions and large number of data items can be problematic because of time complexity;
- Effectiveness of the method depends on the definition of "distance" (for distance based clustering)
- If an obvious distance measure doesn't exist we must "define" it, which is not always easy, especially in multidimensional spaces.
- The result of the clustering algorithm (that in many cases can be arbitrary itself) can be interpreted in different ways.

1.4.2 Justification for using consensus clustering

There are potential shortcomings for all existing clustering techniques. This may cause interpretation of results to become difficult, especially when there is no knowledge about the number of clusters. Clustering methods are also very sensitive to the initial clustering settings, which can cause non-significant data to be amplified in non-reiterative methods. An extremely important issue in cluster analysis is the validation of the clustering results, that is, how to gain confidence about the significance of the clusters provided by the clustering technique (cluster numbers and cluster assignments). Lacking an external objective criterion (the equivalent of a known class label in supervised analysis), this validation becomes somewhat elusive. Iterative descent clustering methods, such as the SOM and *K*-means clustering circumvent some of the shortcomings of Hierarchical clustering by providing for univocally defined clusters and cluster boundaries. Consensus clustering provides a method that represents the consensus across multiple runs of a clustering algorithm, to determine the number of clusters in the data, and to assess the stability of the discovered clusters. The method can also be used to represent the consensus over multiple runs of a clustering algorithm with random restart (such as K-means, model-based Bayesian clustering, SOM, etc.), so as to account for its sensitivity to the initial conditions. It can provide data for a visualization tool to inspect cluster number, membership, and boundaries. However, they lack the intuitive and visual appeal of Hierarchical clustering dendrograms, and the number of clusters must be chosen a priori.

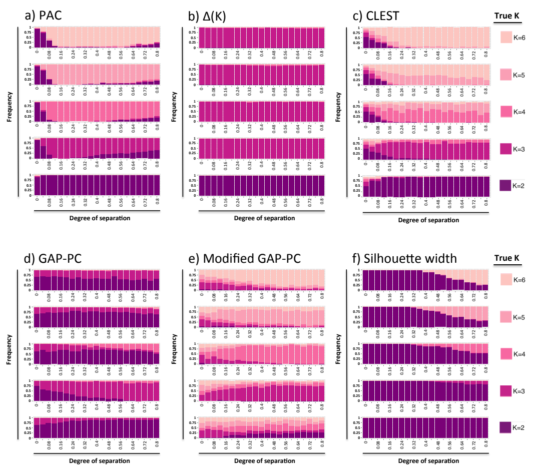
1.4.3 Over-interpretation potential of consensus clustering

Consensus clustering can be a powerful tool for identifying clusters, but it needs to be applied with caution. It has been shown that consensus clustering is able to claim apparent stability of chance partitioning of null datasets drawn from a unimodal distribution, and thus has the potential to lead to over-interpretation of cluster stability in a real study.*[7]*[8] If clusters are not well separated, consensus clustering could lead one to conclude apparent structure when there is none, or declare cluster stability when it is subtle. To reduce the false positive potential in clustering samples (observations), Şenbabaoğlu *et al* *[7] recommends (1) doing a formal test of cluster strength using simulated unimodal data with the same feature-space correlation structure as in the empirical data, (2) not relying solely on the consensus matrix heatmap to declare



PAC measure (proportion of ambiguous clustering) explained. Optimal K is the K with lowest PAC value.

the existence of clusters, or to estimate optimal K, (3) applying the proportion of ambiguous clustering (**PAC**) as a simple yet powerful method to infer optimal K.



Inferred optimal K values of different methods on simulated datasets with known number of clusters and known degree of separation between clusters. Consensus clustering followed by PAC outperforms other methods.

PAC: In the CDF curve of a consensus matrix, the lower left portion represents sample pairs rarely clustered together, the upper right portion represents those almost always clustered together, whereas the middle segment represent those with ambiguous assignments in different clustering runs. The "proportion of ambiguous clustering" (PAC) measure quantifies this middle segment; and is defined as the fraction of sample pairs with consensus indices falling in the interval $(u_1, u_2) \in [0, 1]$ where u_1 is a value close to 0 and u_2 is a value close to 1 (for instance u_1=0.1 and u_2=0.9). A low value of PAC indicates a flat middle segment, and a low rate of discordant assignments across permuted clustering runs. We can therefore infer the optimal number of clusters by the K value having the lowest PAC.

In simulated datasets with known number of clusters, consensus clustering+PAC has been shown to perform better than several other commonly used methods such as consensus clustering+Δ(K), CLEST, GAP, and silhouette width.[*][7][*][8]

1.4.4 Related work

1. **Clustering ensemble (Strehl and Ghosh)**: They considered various formulations for the problem, most of which reduce the problem to a hyper-graph partitioning problem. In one of their formulations they considered the same graph as in the correlation clustering problem. The solution they proposed is to compute the best *k*-partition of the graph, which does not take into account the penalty for merging two nodes that are far apart.

2. **Clustering aggregation (Fern and Brodley)**: They applied the clustering aggregation idea to a collection of soft clusterings they obtained by random projections. They used an agglomerative algorithm and did not penalize for merging dissimilar nodes.

3. **Fred and Jain**: They proposed to use a single linkage algorithm to combine multiple runs of the *k*-means algorithm.

4. **Dana Cristofor and Dan Simovici**: They observed the connection between clustering aggregation and clustering of categorical data. They proposed information theoretic distance measures, and they propose genetic algorithms for finding the best aggregation solution.

5. **Topchy et al.**: They defined clustering aggregation as a maximum likelihood estimation problem, and they proposed an EM algorithm for finding the consensus clustering.

6. **Abu-Jamous et al.**: They proposed their binarization of consensus partition matrices (Bi-CoPaM) method to enhance ensemble clustering in two major aspects. The first is to consider clustering the same set of objects by various clustering methods as well as by considering their features measured in multiple datasets; this seems perfectly relevant in the context of microarray gene expression clustering, which is the context they initially proposed the method in. The second aspect is the format of the final result; based on the consistency of inclusion of a data object in the same cluster by the multiple single clustering results, they allowed any single data object to have any of the three eventualities; to be exclusively assigned to one and only one cluster, to be unassigned from all clusters, or to be simultaneously assigned to multiple clusters at the same time. They made it possible to produce, in a perfectly tunable way, wide overlapping clusters, tight specific clusters, as well as complementary clusters. Therefore, they proposed their work as a new paradigm of clustering rather than merely a new ensemble clustering method.[*][5][*][9]

1.4.5 Hard ensemble clustering

This approach by *Strehl* and *Ghosh* introduces the problem of combining multiple partitionings of a set of objects into a single consolidated clustering without accessing the features or algorithms that determined these partitionings. They discuss three approaches towards solving this problem to obtain high quality consensus functions. Their techniques have low computational costs and this makes it feasible to evaluate each of the techniques discussed below and arrive at the best solution by comparing the results against the objective function.

Efficient consensus functions

1. Cluster-based similarity partitioning algorithm (CSPA)

In CSPA the similarity between two data-points is defined to be directly proportional to number of constituent clusterings of the ensemble in which they are clustered together. The intuition is that the more similar two data-points are the higher is the chance that constituent clusterings will place them in the same cluster. CSPA is the simplest heuristic, but its computational and storage complexity are both quadratic in n. The following two methods are computationally less expensive:

2. Hyper-graph partitioning algorithm (HGPA)

The HGPA algorithm takes a very different approach to finding the consensus clustering than the previous method. The cluster ensemble problem is formulated as partitioning the hypergraph by cutting a minimal number of hyperedges. They make use of hMETIS which is a hypergraph partitioning package system.

3. Meta-clustering algorithm (MCLA)

The meta-cLustering algorithm (MCLA) is based on clustering clusters. First, it tries to solve the cluster correspondence problem and then uses voting to place data-points into the final consensus clusters. The cluster correspondence problem is solved by grouping the clusters identified in the individual clusterings of the ensemble. The clustering is performed using METIS and Spectral clustering.

1.4.6 Soft clustering ensembles

Punera and *Ghosh* extended the idea of hard clustering ensembles to the soft clustering scenario. Each instance in a soft ensemble is represented by a concatenation of r posterior membership probability distributions obtained from the constituent clustering algorithms. We can define a distance measure between two instances using the Kullback–Leibler (KL) divergence, which calculates the “distance” between two probability distributions.

1. sCSPA

sCSPA extends CSPA by calculating a similarity matrix. Each object is visualized as a point in dimensional space, with each dimension corresponding to probability of its belonging to a cluster. This technique first transforms the objects into a label-space and then interprets the dot product between the vectors representing the objects as their similarity.

2. sMCLA

sMCLA extends MCLA by accepting soft clusterings as input. sMCLA’ s working can be divided into the following steps:

- Construct Soft Meta-Graph of Clusters
- Group the Clusters into Meta-Clusters
- Collapse Meta-Clusters using Weighting
- Compete for Objects

3. sHBGF

HBGF represents the ensemble as a bipartite graph with clusters and instances as nodes, and edges between the instances and the clusters they belong to.*[10] This approach can be trivially adapted to consider soft ensembles since the graph partitioning algorithm METIS accepts weights on the edges of the graph to be partitioned. In sHBGF, the graph has $n + t$ vertices, where t is the total number of underlying clusters.

1.4.7 Tunable-tightness partitions

In this different form of clustering, each data object is allowed to be exclusively assigned to one and only one cluster, to be unassigned from all clusters, or to be simultaneously assigned to multiple clusters, in a completely tunable way.*[5] In some applications like gene clustering, this matches the biological reality that many of the genes considered for clustering in a particular gene discovery study might be irrelevant to the case of study in hand and should be ideally not assigned to any of the output clusters, moreover, any single gene can be participating in multiple processes and would be useful to be included in multiple clusters simultaneously. This has been proposed in the recent method of the binarization of consensus partition matrices (Bi-CoPaM)*[5] and is being used currently in the field of bioinformatics.*[9]

1.4.8 Sources

[1] Gibbons, F. D.; Johnson, GF; Yalow, RS (30 September 2002). “Judging the Quality of Gene Expression-Based Clustering Methods Using Gene Annotation” . *Genome Research* **12** (10): 1574–1581. doi:10.1101/gr.397002.

[2] Ozay, Mete; Yarman Vural, F. T.; Kulkarni, Sanjeev R.; Vincent, H. Vincent (2013). *Fusion of image segmentation*

algorithms using consensus clustering. IEEE International Conference on Image Processing (ICIP). pp. 4049–4053. doi:10.1109/ICIP.2013.6738834.

[3] Ozay, Mete (2014). *Semi-supervised Segmentation Fusion of Multi-spectral and Aerial Images*. 22nd International Conference on Pattern Recognition (ICPR). pp. 3839–3844. doi:10.1109/ICPR.2014.659.

[4] VEGA-PONS, SANDRO; RUIZ-SHULCLOPER, JOSÉ (1 May 2011). "A SURVEY OF CLUSTERING ENSEMBLE ALGORITHMS" . *International Journal of Pattern Recognition and Artificial Intelligence* **25** (03): 337–372. doi:10.1142/S0218001411008683.

[5] Abu-Jamous, Basel; Fa, Rui; Roberts, David J.; Nandi, Asoke K.; Peddada, Shyamal D. (11 February 2013). "Paradigm of Tunable Clustering Using Binarization of Consensus Partition Matrices (Bi-CoPaM) for Gene Discovery" . *PLoS ONE* **8** (2): e56432. doi:10.1371/journal.pone.0056432. PMC 3569426. PMID 23409186.

[6] Filkov, Vladimir (2003). "Integrating microarray data by consensus clustering" . *In Proceedings of the 15th IEEE International Conference on Tools with Artificial Intelligence.*: 418–426. doi:10.1109/TAI.2003.1250220.

[7] Şenbabaoğlu, Y.; Michailidis, G.; Li, J. Z. (2014). "Critical limitations of consensus clustering in class discovery" . *Scientific Reports* **4**: 6207. doi:10.1038/srep06207.

[8] Şenbabaoğlu, Y.; Michailidis, G.; Li, J. Z. (Feb 2014). "A reassessment of consensus clustering for class discovery" . *bioRxiv*. doi:10.1101/002642.

[9] Abu-Jamous, B.; Fa, R.; Roberts, D. J.; Nandi, A. K. (24 January 2013). "Yeast gene CMR1/YDL156W is consistently co-expressed with genes participating in DNA-metabolic processes in a variety of stringent clustering experiments" . *Journal of The Royal Society Interface* **10** (81): 20120990–20120990. doi:10.1098/rsif.2012.0990. PMC 3627109. PMID 23349438.

[10] Solving cluster ensemble problems by bipartite graph partitioning, Xiaoli Zhang Fern and Carla E. Brodley,Proceedings of the twenty-first international conference on Machine learning

1.4.9 References

- Alexander Strehl and J. Ghosh, Cluster ensembles – a knowledge reuse framework for combining multiple partitions, Journal on Machine Learning Research (JMLR) 2002
- Kunal Punera, Joydeep Ghosh. Consensus Based Ensembles of Soft Clusterings.
- Aristides Gionis, Heikki Mannila, Panayiotis Tsaparas. Clustering Aggregation. 21st International Conference on Data Engineering (ICDE 2005)
- Hongjun Wang, Hanhuai Shan, Arindam Banerjee. Bayesian Cluster Ensembles, SIAM International Conference on Data Mining, SDM 09
- Nam Nguyen, Rich Caruana. Consensus Clusterings. Seventh IEEE International Conference on Data Mining.
- Alexander Topchy, Anil K. Jain, William Punch. Clustering Ensembles: Models of Consensus and Weak Partitions. IEEE International Conference on Data Mining, ICDM 03 & SIAM International Conference on Data Mining, SDM 04
- Basel Abu-Jamous, Rui Fa, David J. Roberts and Asoke K. Nandi. Paradigm of Tunable Clustering using Binarization of Consensus Partition Matrices (Bi-CoPaM) for Gene Discovery, PLOS ONE 8(2) (doi:10.1371/journal.pone.0056432) 2013

1.4.10 Further reading

- Andrey Goder and Vladimir Filkov. "Consensus Clustering Algorithms: Comparison and Refinement" (PDF). *2008 Proceedings of the Ninth Workshop on Algorithm Engineering and Experiments (ALENEX) —San Francisco, January 19, 2008*. Society for Industrial and Applied Mathematics.
- Tao Li and Chris Ding. "Weighted Consensus Clustering" (PDF). *Proceedings of the 2008 SIAM International Conference on Data Mining —Atlanta, April 24–26, 2008*. Society for Industrial and Applied Mathematics.
- Stefano Monti, Pablo Tamayo, Jill P. Mesirov and Todd Golub. "Consensus Clustering – A resampling-based method for class discovery and visualization of gene expression microarray data"

1.5 Sequence clustering

In bioinformatics, **sequence clustering** algorithms attempt to group biological sequences that are somehow related. The sequences can be either of genomic, "transcriptomic" (ESTs) or protein origin. For proteins, homologous sequences are typically grouped into families. For EST data, clustering is important to group sequences originating from the same gene before the ESTs are assembled to reconstruct the original mRNA.

Some clustering algorithms use single-linkage clustering, constructing a transitive closure of sequences with a similarity over a particular threshold. UCLUST*[1] and CD-HIT*[2] use a greedy algorithm that identifies a representative sequence for each cluster and assigns a new sequence to that cluster if it is sufficiently similar to the representative; if a sequence is not matched then it becomes the representative sequence for a new cluster. The similarity score is often based on sequence alignment. Sequence clustering is often used to make a non-redundant set of representative sequences.

Sequence clusters are often synonymous with (but not identical to) protein families. Determining a representative tertiary structure for each sequence cluster is the aim of many structural genomics initiatives.

1.5.1 Sequence clustering algorithms and packages

- UCLUST in USEARCH*[1]
- CD-HIT*[2]
- nrdb90.pl*[3]
- TribeMCL: a method for clustering proteins into related groups*[4]
- BAG: a graph theoretic sequence clustering algorithm*[5]
- JESAM: Open source parallel scalable DNA alignment engine with optional clustering software component
- UICluster: Parallel Clustering of EST (Gene) Sequences
- BLASTClust single-linkage clustering with BLAST
- (Multi)netclust: fast and memory-efficient detection of connected clusters in (multi-parametric) data networks*[6]
- Clusterer: extendable java application for sequence grouping and cluster analyses
- PATDB: a program for rapidly identifying perfect substrings
- nrdb: a program for merging trivially redundant (identical) sequences
- CluSTr: A single-linkage protein sequence clustering database from Smith-Waterman sequence similarities; covers over 7 mln sequences including UniProt and IPI
- ICAtools - original (ancient) DNA clustering package with many algorithms useful for artifact discovery or EST clustering
- Virus Orthologous Clusters: A viral protein sequence clustering database; contains all predicted genes from eleven virus families organized into ortholog groups by BLASTP similarity
- Skipredudant EMBOSS tool to remove redundant sequences from a set

1.5.2 Non-redundant sequence databases

- PISCES: A Protein Sequence Culling Server*[7]
- RDB90*[3]
- UniRef: A non-redundant UniProt sequence database*[8]

1.5.3 See also

- Cluster analysis

1.5.4 References

[1] USEARCH: An exceptionally fast sequence clustering program for nucleotide and protein sequences

[2] CD-HIT: a ultra-fast method for clustering protein and nucleotide sequences, with many new applications in next generation sequencing (NGS) data

[3] Holm L1, Sander C. (Jun 1998). "Removing near-neighbour redundancy from large protein sequence collections." . *Bioinformatics* **14** (5): 423–9. doi:10.1093/bioinformatics/14.5.423. PMID 9682055.

[4] Enright AJ, Van Dongen S, Ouzounis CA. (Apr 2002). "An efficient algorithm for large-scale detection of protein families." . *Nucleic Acids Res.* **30** (7): 1575–84. doi:10.1093/nar/30.7.1575. PMID 11917018.

[5] http://bio.informatics.indiana.edu/sunkim/BAG/

[6] Kuzniar, A., Dhir, S., Nijveen, H., Pongor, S. and Leunissen, J. A. M. (Oct 2010). "Multi-netclust: an efficient tool for finding connected clusters in multi-parametric networks" . *Bioinformatics* **26** (19): 2482–2483. doi:10.1093/bioinformatics/btq435. PMID 20679333.

[7] http://dunbrack.fccc.edu/pisces/

[8] UniRef: A non-redundant UniProt sequence database

1.6 Data stream clustering

In computer science, **data stream clustering** is defined as the clustering of data that arrive continuously such as telephone records, multimedia data, financial transactions etc. Data stream clustering is usually studied as a streaming algorithm and the objective is, given a sequence of points, to construct a good clustering of the stream, using a small amount of memory and time.

1.6.1 History

Data stream clustering has recently attracted attention for emerging applications that involve large amounts of streaming data. For clustering, k-means is a widely used heuristic but alternate algorithms have also been developed such as k-medoids, CURE and the popular BIRCH. For data streams, one of the first results appeared in 1980*[1] but the model was formalized in 1998.*[2]

1.6.2 Definition

The problem of data stream clustering is defined as:

Input: a sequence of *n* points in metric space and an integer *k*.
Output: *k* centers in the set of the *n* points so as to minimize the sum of distances from data points to their closest cluster centers.

This is the streaming version of the k-median problem.

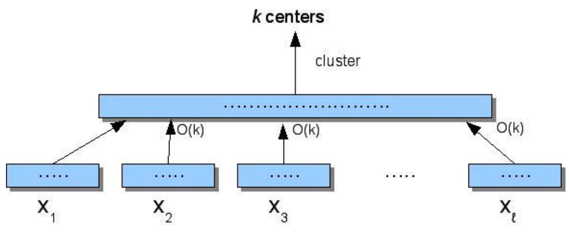
1.6.3 Algorithms

STREAM

STREAM is an algorithm for clustering data streams described by Guha, Mishra, Motwani and O'Callaghan*[3] which achieves a constant factor approximation for the k-Median problem in a single pass and using small space.

***Theorem**: STREAM can solve the* k-*Median problem on a data stream in a single pass, with time* O(n*1+e) *and space* θ(n*ε) *up to a factor* 2*O(1/e), *where* n *the number of points and* e<1/2.

To understand STREAM, the first step is to show that clustering can take place in small space (not caring about the number of passes). Small-Space is a divide-and-conquer algorithm that divides the data, *S*, into ℓ pieces, clusters each one of them (using *k*-means) and then clusters the centers obtained.



Small-Space Algorithm representation

Algorithm Small-Space(S)

1. Divide *S* into ℓ disjoint pieces $X_1,...,X_\ell$.
2. *For each* i, *find* O(k) *centers in* X_i. *Assign* each point in X_i to its closest center.
3. Let X' be the $O(\ell k)$ centers obtained in (2),

   where each center *c* is weighted by the number

   of points assigned to it.
4. Cluster X' to find *k* centers.

Where, if in Step 2 we run a bicriteria *(a,b)*-approximation algorithm which outputs at most *ak* medians with cost at most *b* times the optimum k-Median solution and in Step 4 we run a *c*-approximation algorithm then the approximation factor of Small-Space() algorithm is *2c(1+2b)+2b*. We can also generalize Small-Space so that it recursively calls itself *i* times on a successively smaller set of weighted centers and achieves a constant factor approximation to the *k*-median problem.

The problem with the Small-Space is that the number of subsets ℓ that we partition *S* into is limited, since it has to store in memory the intermediate medians in *X*. ***So, if* M *is the size of memory, we need to partition* S *into* ℓ *subsets such that each subset fits in memory, (n/* ℓ *) and so that the weighted* ℓ k *centers also fit in memory,* ℓ k<M". But such an <math>\ell</math> may not always exist.**

The STREAM algorithm solves the problem of storing intermediate medians and achieves better running time and space requirements. The algorithm works as follows:*[3]

1. Input the first *m* points; using the randomized algorithm presented in*[3] reduce these to *O(k)* (say *2k*) points.
2. Repeat the above till we have seen $m^2/(2k)$ of the original data points. We now have *m* intermediate medians.
3. Using a local search algorithm, cluster these *m* first-level medians into *2k* second-level medians and proceed.
4. In general, maintain at most *m* level-*i* medians, and, on seeing *m*, generate *2k* level-*i*+ 1 medians, with the weight of a new median as the sum of the weights of the intermediate medians assigned to it.
5. When we have seen all the original data points, we cluster all the intermediate medians into *k* final medians, using the primal dual algorithm.*[4]

Other Algorithms

Other well-known algorithms used for data stream clustering are:

- BIRCH:*[5] builds a hierarchical data structure to incrementally cluster the incoming points using the available memory and minimizing the amount of I/O required. The complexity of the algorithm is *O(N)* since one pass suffices to get a good clustering (though, results can be improved by allowing several passes).

- COBWEB:*[6]*[7] is an incremental clustering technique that keeps a hierarchical clustering model in the form of a classification tree. For each new point. COBWEB descends the tree, updates the nodes along the way and looks for the best node to put the point on (using a category utility function).
- C2ICM:*[8] builds a flat partitioning clustering structure by selecting some objects as cluster seeds/initiators and a non-seed is assigned to the seed that provides the highest coverage, addition of new objects can introduce new seeds and falsify some existing old seeds, during incremental clustering new objects and the members of the falsified clusters are assigned to one of the existing new/old seeds.

1.6.4 References

[1] Munro, J.; Paterson, M. (1980). "Selection and Sorting with Limited Storage". *Theoretical Computer Science*: 315–323.

[2] Henzinger, M.; Raghavan, P.; Rajagopalan, S. (August 1998). "Computing on Data Streams". *Digital Equipment Corporation*. TR-1998-011. CiteSeerX: 10.1.1.19.9554.

[3] Guha, S.; Mishra, N.; Motwani, R.; O'Callaghan, L. (2000). "Clustering Data Streams". *Proceedings of the Annual Symposium on Foundations of Computer Science*. CiteSeerX: 10.1.1.32.1927.

[4] Jain, K.; Vazirani, V. (1999). "Primal-dual approximation algorithms for metric facility location and k-median problems". *Proc. FOCS*.

[5] Zhang, T.; Ramakrishnan, R.; Linvy, M. (1996). "BIRCH: An Efficient Data Clustering Method for Very Large Databases". *Proceedings of the ACM SIGMOD Conference on Management of Data* **25**: 103–114. doi:10.1145/235968.233324.

[6] Fisher, D. H. (1987). "Knowledge Acquisition Via Incremental Conceptual Clustering". *Machine Learning* **2**: 139–172. doi:10.1023/A:1022852608280.

[7] Fisher, D. H. (1996). "Iterative Optimization and Simplification of Hierarchical Clusterings". *Journal of AI Research* **4**. CiteSeerX: 10.1.1.6.9914.

[8] Can, F. (1993). "Incremental Clustering for Dynamic Information Processing". *ACM Transactions on Information Systems* **11** (2): 143–164. doi:10.1145/130226.134466.

1.7 Constrained clustering

In computer science, **constrained clustering** is a class of semi-supervised learning algorithms. Typically, constrained clustering incorporates either a set of must-link constraints, cannot-link constraints, or both, with a Data clustering algorithm. Both a must-link and a cannot-link constraint define a relationship between two data instances. A must-link constraint is used to specify that the two instances in the must-link relation should be associated with the same cluster. A cannot-link constraint is used to specify that the two instances in the cannot-link relation should *not* be associated with the same cluster. These sets of constraints acts as a guide for which a constrained clustering algorithm will attempt to find clusters in a data set which satisfy the specified must-link and cannot-link constraints. Some constrained clustering algorithms will abort if no such clustering exists which satisfies the specified constraints. Others will try to minimize the amount of constraint violation should it be impossible to find a clustering which satisfies the constraints.

Examples of constrained clustering algorithms include:

- COP K-means *[1]
- PCKmeans
- CMWK-Means *[2]

1.7.1 References

[1] Wagstaff, K.; Cardie, C.; Rogers, S.; Schrödl, S. (2001). "Constrained K-means Clustering with Background Knowledge". *Proceedings of the Eighteenth International Conference on Machine Learning*. pp. 577–584.

[2] de Amorim, R. C. (2012). "Constrained Clustering with Minkowski Weighted K-Means". *Proceedings of the 13th IEEE International Symposium on Computational Intelligence and Informatics*. pp. 13–17. doi:10.1109/CINTI.2012.6496753.

1.7.2 Further reading

- Biswas, A.; Jacobs, D. (2012). "Active image clustering: Seeking constraints from humans to complement algorithms". *2012 IEEE Conference on Computer Vision and Pattern Recognition*. p. 2152. doi:10.1109/CVPR.2012.6247922. ISBN 978-1-4673-1228-8.
- Huang, H.; Cheng, Y.; Zhao, R. (2008). "A Semi-supervised Clustering Algorithm Based on Must-Link Set". *Advanced Data Mining and Applications*. Lecture Notes in Computer Science **5139**. p. 492. doi:10.1007/978-3-540-88192-6_48. ISBN 978-3-540-88191-9.
- Zhang, S.; Wong, H. S. (2008). "Partial closure-based constrained clustering with order ranking". *2008 19th International Conference on Pattern Recognition*. p. 1. doi:10.1109/ICPR.2008.4760984. ISBN 978-1-4244-2174-9.

- Grira, N.; Crucianu, M.; Boujemaa, N. (2005). "Semi-supervised image database categorization using pairwise constraints" . *IEEE International Conference on Image Processing 2005*. pp. III. doi:10.1109/ICIP.2005.1530620. ISBN 0-7803-9134-9.

- Shi Rui; Wanjun Jin; Tat-Seng Chua (2005). "A Novel Approach to Auto Image Annotation Based on Pairwise Constrained Clustering and Semi-Naïve Bayesian Model" . *11th International Multimedia Modelling Conference*. p. 322. doi:10.1109/MMMC.2005.14. ISBN 0-7695-2164-9.

- Ismail, M. M. B.; Frigui, H. (2009). "Image annotation based on constrained clustering and semi-naive bayesian model" . *2009 IEEE Symposium on Computers and Communications*. p. 431. doi:10.1109/ISCC.2009.5202230. ISBN 978-1-4244-4672-8.

1.8 Fuzzy clustering

Data clustering is the process of dividing data elements into classes or clusters so that items in the same class are as similar as possible, and items in different classes are as dissimilar as possible. Depending on the nature of the data and the purpose for which clustering is being used, different measures of similarity may be used to place items into classes, where the similarity measure controls how the clusters are formed. Some examples of measures that can be used as in clustering include distance, connectivity, and intensity.

In hard clustering, data is divided into distinct clusters, where each data element belongs to exactly one cluster. In **fuzzy clustering** (also referred to as **soft clustering**), data elements can belong to more than one cluster, and associated with each element is a set of membership levels. These indicate the strength of the association between that data element and a particular cluster. Fuzzy clustering is a process of assigning these membership levels, and then using them to assign data elements to one or more clusters.

One of the most widely used fuzzy clustering algorithms is the Fuzzy C-Means (FCM) Algorithm (Bezdek 1981). The FCM algorithm attempts to partition a finite collection of n elements $X = \{\mathbf{x}_1, ..., \mathbf{x}_n\}$ into a collection of c fuzzy clusters with respect to some given criterion. Given a finite set of data, the algorithm returns a list of c cluster centres $C = \{\mathbf{c}_1, ..., \mathbf{c}_c\}$ and a partition matrix $W = w_{i,j} \in [0,1],\ i = 1, ..., n,\ j = 1, ..., c$, where each element w_{ij} tells the degree to which element $\mathbf{x}_i$ belongs to cluster $\mathbf{c}_j$. Like the K-means clustering, the FCM aims to minimize an objective function:

$$\underset{C}{\arg\min} \sum_{i=1}^{n} \sum_{j=1}^{c} w_{ij}^{m} \left\|\mathbf{x}_i - \mathbf{c}_j\right\|^2 ,$$

where:

$$w_{ij} = \frac{1}{\sum_{k=1}^{c} \left(\frac{\|\mathbf{x}_i - \mathbf{c}_j\|}{\|\mathbf{x}_i - \mathbf{c}_k\|} \right)^{\frac{2}{m-1}}} .$$

This differs from the k-means objective function by the addition of the membership values w_{ij} and the fuzzifier $m \in R$, with $m \geq 1$. The fuzzifier m determines the level of cluster fuzziness. A large m results in smaller memberships w_{ij} and hence, fuzzier clusters. In the limit $m = 1$, the memberships w_{ij} converge to 0 or 1, which implies a crisp partitioning. In the absence of experimentation or domain knowledge, m is commonly set to 2.

1.8.1 Fuzzy c-means clustering

In fuzzy clustering, every point has a degree of belonging to clusters, as in fuzzy logic, rather than belonging completely to just one cluster. Thus, points on the edge of a cluster, may be *in the cluster* to a lesser degree than points in the center of cluster. An overview and comparison of different fuzzy clustering algorithms is available.*[1]

Any point x has a set of coefficients giving the degree of being in the kth cluster $w_k(x)$. With fuzzy c-means, the centroid of a cluster is the mean of all points, weighted by their degree of belonging to the cluster:

$$c_k = \frac{\sum_x w_k(x)^m x}{\sum_x w_k(x)^m} .$$

The degree of belonging, $w_k(x)$, is related inversely to the distance from x to the cluster center as calculated on the previous pass. It also depends on a parameter m that controls how much weight is given to the closest center. The fuzzy c-means algorithm is very similar to the k-means algorithm:*[2]

- Choose a number of clusters.
- Assign randomly to each point coefficients for being in the clusters.
- Repeat until the algorithm has converged (that is, the coefficients' change between two iterations is no more than ε , the given sensitivity threshold) :
  - Compute the centroid for each cluster, using the formula above.
  - For each point, compute its coefficients of being in the clusters, using the formula above.

The algorithm minimizes intra-cluster variance as well, but has the same problems as *k*-means; the minimum is a local minimum, and the results depend on the initial choice of weights.

Using a mixture of Gaussians along with the expectation-maximization algorithm is a more statistically formalized method which includes some of these ideas: partial membership in classes.

Another algorithm closely related to Fuzzy C-Means is Soft K-means.

Fuzzy c-means has been a very important tool for image processing in clustering objects in an image. In the 70's, mathematicians introduced the spatial term into the FCM algorithm to improve the accuracy of clustering under noise.*[3]

1.8.2 See also

- FLAME Clustering
- Cluster Analysis
- Expectation-maximization algorithm (a similar, but more statistically formalized method)

1.8.3 References

[1] Nock, R. and Nielsen, F. (2006) "On Weighting Clustering" , IEEE Trans. on Pattern Analysis and Machine Intelligence, 28 (8), 1–13

[2] Bezdek, James C. (1981). *Pattern Recognition with Fuzzy Objective Function Algorithms*. ISBN 0-306-40671-3.

[3] Ahmed, Mohamed N.; Yamany, Sameh M.; Mohamed, Nevin; Farag, Aly A.; Moriarty, Thomas (2002). "A Modified Fuzzy C-Means Algorithm for Bias Field Estimation and Segmentation of MRI Data" (PDF). *IEEE Transactions on Medical Imaging* **21** (3): 193–199. doi:10.1109/42.996338. PMID 11989844.

1.8.4 External links

- Fuzzy Clustering in Wolfram Research
- *Extended Fuzzy Clustering Algorithms* by Kaymak, U. and Setnes, M.
- *Fuzzy Clustering in C++ and Boost* by Antonio Gulli
- Concise description with examples

1.9 Spectral clustering

In multivariate statistics and the clustering of data, **spectral clustering** techniques make use of the spectrum (eigenvalues) of the similarity matrix of the data to perform dimensionality reduction before clustering in fewer dimensions. The similarity matrix is provided as an input and consists of a quantitative assessment of the relative similarity of each pair of points in the dataset.

In application to image segmentation, spectral clustering is known as segmentation-based object categorization.

1.9.1 Algorithms

Given an enumerated set of data points, the similarity matrix may be defined as a symmetric matrix A , where $A_{ij} \geq 0$ represents a measure of the similarity between data points with indexes i and j .

One spectral clustering technique is the **normalized cuts algorithm** or *Shi–Malik algorithm* introduced by Jianbo Shi and Jitendra Malik,*[1] commonly used for image segmentation. It partitions points into two sets (B_1, B_2) based on the eigenvector v corresponding to the second-smallest eigenvalue of the symmetric normalized Laplacian defined as

$$L^{norm} := I - D^{-1/2} A D^{-1/2}$$

where D is the diagonal matrix

$$D_{ii} = \sum_j A_{ij}.$$

A mathematically equivalent algorithm *[2] takes the eigenvector corresponding to the largest eigenvalue of the random walk normalized Laplacian matrix $P = D^{-1}A$.

Another possibility is to use the Laplacian matrix defined as

$$L := D - A$$

rather than the symmetric normalized Laplacian matrix.

Partitioning may be done in various ways, such as by computing the median m of the components of the second smallest eigenvector v , and placing all points whose component in v is greater than m in B_1 , and the rest in B_2 . The algorithm can be used for hierarchical clustering by repeatedly partitioning the subsets in this fashion.

Alternatively to computing just one eigenvector, *k* eigenvectors for some *k*, are computed, and then another algorithm (e.g. k-means clustering) is used to cluster points by their respective *k* components in these eigenvectors.

The efficiency of spectral clustering may be improved if the solution to the corresponding eigenvalue problem is performed in a matrix-free fashion, i.e., without explicitly manipulating or even computing the similarity matrix, as, e.g., in the Lanczos algorithm.

For large-sized graphs, the second eigenvalue of the (normalized) graph Laplacian matrix is often ill-conditioned, leading to slow convergence of iterative eigenvalue solvers. Preconditioning is a key technology accelerating the convergence, e.g., in the matrix-free LOBPCG method. Spectral clustering has been successfully applied on large graphs by first identifying their community structure, and then clustering communities.*[3]

Spectral clustering is closely related to Nonlinear dimensionality reduction, and dimension reduction techniques such as locally-linear embedding can be used to reduce errors from noise or outliers.*[4]

1.9.2 Relationship with *k*-means

The kernel k-means problem is an extension of the k-means problem where the input data points are mapped non-linearly into a higher-dimensional feature space via a kernel function $k(x_i, x_j) = \phi^T(x_i)\phi(x_j)$. The weighted kernel k-means problem further extends this problem by defining a weight w_r for each cluster as the reciprocal of the number of elements in the cluster,

$$\max_{\{C_s\}} \sum_{r=1}^{k} w_r \sum_{x_i, x_j \in C_r} k(x_i, x_j).$$

Suppose F is a matrix of the normalizing coefficients for each point for each cluster $F_{ij} = w_r$ if $i, j \in C_r$ and zero otherwise. Suppose K is the kernel matrix for all points. The weighted kernel k-means problem with n points and k clusters is given as,

$$\max_F \text{trace}\,(KF)$$

such that,

$$F = G_{n \times k} G_{n \times k}^T$$

$$G^T G = I$$

such that rank$(G) = k$. In addition, there are identity constrains on F given by,

$$F \cdot \mathbb{I} = \mathbb{I}$$

where $\mathbb{I}$ represents a vector of ones.

$$F^T \mathbb{I} = \mathbb{I}$$

This problem can be recast as,

$$\max_G \text{ trace } \left(G^T G\right).$$

This problem is equivalent to the spectral clustering problem when the identity constraints on F are relaxed. In particular, the weighted kernel k-means problem can be reformulated as a spectral clustering (graph partitioning) problem and vice versa. The output of the algorithms are eigenvectors which do not satisfy the identity requirements for indicator variables defined by F . Hence, post-processing of the eigenvectors is required for the equivalence between the problems.*[5] Transforming the spectral clustering problem into a weighted kernel k-means problem greatly reduces the computational burden.*[6]

1.9.3 See also

- Affinity propagation
- Kernel principal component analysis
- Cluster analysis
- Spectral graph theory

1.9.4 References

[1] Jianbo Shi and Jitendra Malik, “Normalized Cuts and Image Segmentation” , IEEE Transactions on PAMI, Vol. 22, No. 8, Aug 2000.

[2] Marina Meilă & Jianbo Shi, "Learning Segmentation by Random Walks", Neural Information Processing Systems 13 (NIPS 2000), 2001, pp. 873–879.

[3] Zare, Habil; P. Shooshtari, A. Gupta and R. Brinkman (2010). “Data reduction for spectral clustering to analyze high throughput flow cytometry data” . *BMC Bioinformatics.*

[4] Arias-Castro, E. and Chen, G. and Lerman, G. (2011), “Spectral clustering based on local linear approximations.” , *Electronic Journal of Statistics* **5**: 1537–1587, doi:10.1214/11-ejs651

[5] Dhillon, I.S. and Guan, Y. and Kulis, B. (2004). “Kernel *k*-means: spectral clustering and normalized cuts” . *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining.* pp. 551–556.

[6] Dhillon, Inderjit; Yuqiang Guan; Brian Kulis (November 2007). “Weighted Graph Cuts without Eigenvectors: A Multilevel Approach” . *IEEE Transactions on Pattern Analysis and Machine Intelligence* **29** (11): 1–14.

Chapter 2

General

2.1 Determining the number of clusters in a data set

Determining the number of clusters in a data set, a quantity often labeled k as in the k-means algorithm, is a frequent problem in data clustering, and is a distinct issue from the process of actually solving the clustering problem.

For a certain class of clustering algorithms (in particular k-means, k-medoids and Expectation-maximization algorithm), there is a parameter commonly referred to as k that specifies the number of clusters to detect. Other algorithms such as DBSCAN and OPTICS algorithm do not require the specification of this parameter; hierarchical clustering avoids the problem altogether.

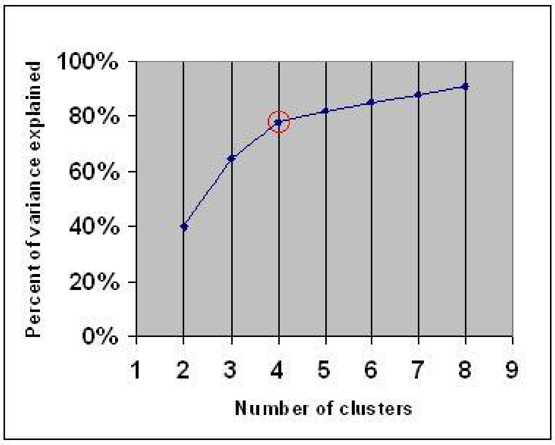
The correct choice of k is often ambiguous, with interpretations depending on the shape and scale of the distribution of points in a data set and the desired clustering resolution of the user. In addition, increasing k without penalty will always reduce the amount of error in the resulting clustering, to the extreme case of zero error if each data point is considered its own cluster (i.e., when k equals the number of data points, n). Intuitively then, *the optimal choice of* k *will strike a balance between maximum compression of the data using a single cluster, and maximum accuracy by assigning each data point to its own cluster*. If an appropriate value of k is not apparent from prior knowledge of the properties of the data set, it must be chosen somehow. There are several categories of methods for making this decision.

2.1.1 Rule of thumb

One simple rule of thumb sets the number to[*][1][*]:365

$$k \approx \sqrt{n/2}$$

with n as the number of objects (data points).



Explained Variance. The "elbow" is indicated by the red circle. The number of clusters chosen should therefore be 4.

2.1.2 The Elbow Method

Another method looks at the percentage of variance explained as a function of the number of clusters: One should choose a number of clusters so that adding another cluster doesn't give much better modeling of the data. More precisely, if one plots the percentage of variance explained by the clusters against the number of clusters, the first clusters will add much information (explain a lot of variance), but at some point the marginal gain will drop, giving an angle in the graph. The number of clusters is chosen at this point, hence the "elbow criterion". This "elbow" cannot always be unambiguously identified.[*][2] Percentage of variance explained is the ratio of the between-group variance to the total variance, also known as an F-test. A slight variation of this method plots the curvature of the within group variance.[*][3] The method can be traced to speculation by Robert L. Thorndike in 1953.[*][4]

2.1.3 Information Criterion Approach

Another set of methods for determining the number of clusters are information criteria, such as the Akaike information criterion (AIC), Bayesian information criterion (BIC), or the Deviance information criterion (DIC) —if it is possible to make a likelihood function for the clustering model. For example: The k-means model is "almost" a Gaussian mixture model and one can construct a likelihood for the Gaussian mixture model and thus also determine information criterion values.*[5]

2.1.4 An Information Theoretic Approach*[6]

Rate distortion theory has been applied to choosing k called the "jump" method, which determines the number of clusters that maximizes efficiency while minimizing error by information theoretic standards. The strategy of the algorithm is to generate a distortion curve for the input data by running a standard clustering algorithm such as k-means for all values of k between 1 and n, and computing the distortion (described below) of the resulting clustering. The distortion curve is then transformed by a negative power chosen based on the dimensionality of the data. Jumps in the resulting values then signify reasonable choices for k, with the largest jump representing the best choice.

The distortion of a clustering of some input data is formally defined as follows: Let the data set be modeled as a p-dimensional random variable, X, consisting of a mixture distribution of G components with common covariance, Γ . If we let $c_1 ... c_K$ be a set of K cluster centers, with c_X the closest center to a given sample of X, then the minimum average distortion per dimension when fitting the K centers to the data is:

$d_K = \frac{1}{p} \min_{c_1 \ldots c_K} E[(X - c_X)^T \Gamma^{-1} (X - c_X)]$

This is also the average Mahalanobis distance per dimension between X and the set of cluster centers C. Because the minimization over all possible sets of cluster centers is prohibitively complex, the distortion is computed in practice by generating a set of cluster centers using a standard clustering algorithm and computing the distortion using the result. The pseudo-code for the jump method with an input set of p-dimensional data points X is:

JumpMethod(X): Let Y = (p/2) Init a list D, of size n+1 Let D[0] = 0 For k = 1 ... n: Cluster X with k clusters (e.g., with k-means) Let d = Distortion of the resulting clustering D[k] = d^(-Y) Define J(i) = D[i] - D[i-1] Return the k between 1 and n that maximizes J(k)

The choice of the transform power $Y = (p/2)$ is motivated by asymptotic reasoning using results from rate distortion theory. Let the data X have a single, arbitrarily p-dimensional Gaussian distribution, and let fixed K = floor(α^p), for some α greater than zero. Then the distortion of a clustering of K clusters in the limit as p goes to infinity is α^{-2} . It can be seen that asymptotically, the distortion of a clustering to the power $(-p/2)$ is proportional to α^p , which by definition is approximately the number of clusters K. In other words, for a single Gaussian distribution, increasing K beyond the true number of clusters, which should be one, causes a linear growth in distortion. This behavior is important in the general case of a mixture of multiple distribution components.

Let X be a mixture of G p-dimensional Gaussian distributions with common covariance. Then for any fixed K less than G, the distortion of a clustering as p goes to infinity is infinite. Intuitively, this means that a clustering of less than the correct number of clusters is unable to describe asymptotically high-dimensional data, causing the distortion to increase without limit. If, as described above, K is made an increasing function of p, namely, K = floor(α^p), the same result as above is achieved, with the value of the distortion in the limit as p goes to infinity being equal to α^{-2} . Correspondingly, there is the same proportional relationship between the transformed distortion and the number of clusters, K.

Putting the results above together, it can be seen that for sufficiently high values of p, the transformed distortion $d_K^{-p/2}$ is approximately zero for $K < G$, then jumps suddenly and begins increasing linearly for $K >= G$. The jump algorithm for choosing K makes use of these behaviors to identify the most likely value for the true number of clusters.

Although the mathematical support for the method is given in terms of asymptotic results, the algorithm has been empirically verified to work well in a variety of data sets with reasonable dimensionality. In addition to the localized jump method described above, there exists a second algorithm for choosing K using the same transformed distortion values known as the broken line method. The broken line method identifies the jump point in the graph of the transformed distortion by doing a simple least squares error line fit of two line segments, which in theory will fall along the x-axis for $K < G$, and along the linearly increasing phase of the transformed distortion plot for $K >= G$. The broken line method is more robust than the jump method in that its decision is global rather than local, but it also relies on the assumption of Gaussian mixture components, whereas the jump method is fully non-parametric and has been shown to be viable for general mixture distributions.

2.1.5 Choosing *k* Using the Silhouette

The average silhouette of the data is another useful criterion for assessing the natural number of clusters. The silhouette of a datum is a measure of how closely it is matched to data within its cluster and how loosely it is matched to data of the neighbouring cluster, i.e. the cluster whose average distance from the datum is lowest.*[7] A silhouette close to 1 implies the datum is in an appropriate cluster, while a silhouette close to -1 implies the datum is in the wrong cluster. Optimization techniques such as genetic algorithms are useful in determining the number of clusters that gives rise to the largest silhouette.*[8]

2.1.6 Cross-validation

One can also use the process of cross-validation to analyze the number of clusters. In this process, the data is partitioned into v parts. Each of the parts is then set aside at turn as a test set, a clustering model computed on the other $v-1$ training sets, and the value of the goal function (for example, the sum of the squared distances to the centroids for k-means) calculated for the test set. These v values are calculated and averaged for each alternative number of clusters, and the cluster number selected that minimizes the test set errors.*[9]

2.1.7 Finding Number of Clusters in Text Databases

In text databases, a document collection defined by a document by term D matrix (of size m by n, m: number of documents, n: number of terms) number of clusters can roughly be estimated by the following formula $(m \times n)/t$ where t is the number of non-zero entries in D. Note that in D each row and each column must contain at least one non-zero element. *[10]

2.1.8 Analyzing the Kernel Matrix

Kernel matrix defines the proximity of the input information. For example, in Gaussian Radial basis function, determines the dot product of the inputs in a higher-dimensional space, called feature space. It is believed that the data become more linearly separable in the feature space, and hence, linear algorithms can be applied on the data with a higher success.

The kernel matrix can thus be analyzed in order to find the optimal number of clusters .*[11] The method proceeds by the eigenvalue decomposition of the kernel matrix. It will then analyze the eigenvalues and eigenvectors to obtain a measure of the compactness of the input distribution. Finally, a plot will be drawn, where the elbow of that plot indicates the optimal number of clusters in the data set. Unlike previous methods, this technique does not need to perform any clustering a-priori. It directly find the number of clusters from the data.

2.1.9 External links

- Clustergram - cluster diagnostic plot - for visual diagnostics of choosing the number of (k) clusters (R code)
- Six methods for determining an optimal k value for k-means analysis - Answer on stackoverflow containing R code for several methods of computing an optimal value of k for k-means cluster analysis

2.1.10 Bibliography

[1] Kanti Mardia et al. (1979). *Multivariate Analysis*. Academic Press.

[2] See, e.g., David J. Ketchen, Jr & Christopher L. Shook (1996). "The application of cluster analysis in Strategic Management Research: An analysis and critique" . *Strategic Management Journal* **17** (6): 441–458. doi:10.1002/(SICI)1097-0266(199606)17:6<441::AID-SMJ819>3.0.CO;2-G.

[3] See, e.g., Figure 6 in

- Cyril Goutte, Peter Toft, Egill Rostrup, Finn Årup Nielsen, Lars Kai Hansen (March 1999). "On Clustering fMRI Time Series" . *NeuroImage* **9** (3): 298–310. doi:10.1006/nimg.1998.0391. PMID 10075900.

[4] Robert L. Thorndike (December 1953). "Who Belongs in the Family?". *Psychometrika* **18** (4): 267–276. doi:10.1007/BF02289263.

[5] Cyril Goutte, Lars Kai Hansen, Matthew G. Liptrot & Egill Rostrup (2001). "Feature-Space Clustering for fMRI Meta-Analysis" . *Human Brain Mapping* **13** (3): 165–183. doi:10.1002/hbm.1031. PMID 11376501. see especially Figure 14 and appendix.

[6] Catherine A. Sugar and Gareth M. James (2003). "Finding the number of clusters in a data set: An information theoretic approach" . *Journal of the American Statistical Association* **98** (January): 750–763. doi:10.1198/016214503000000666.

[7] Peter J. Rousseuw (1987). "Silhouettes: a Graphical Aid to the Interpretation and Validation of Cluster Analysis" . *Computational and Applied Mathematics* **20**: 53–65. doi:10.1016/0377-0427(87)90125-7.

[8] R. Lleti, M.C. Ortiz, L.A. Sarabia, M.S. Sánchez (2004). "Selecting Variables for *k*-Means Cluster Analysis by Using a Genetic Algorithm that Optimises the Silhouettes" . *Analytica Chimica Acta* **515**: 87–100. doi:10.1016/j.aca.2003.12.020.

[9] See e.g. "Finding the Right Number of Clusters in k-Means and EM Clustering: v-Fold Cross-Validation" . *Electronic Statistics Textbook*. StatSoft. 2010. Retrieved 2010-05-03.

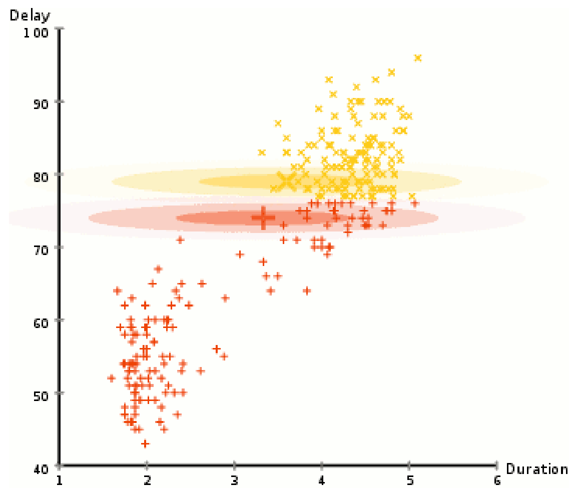
[10] Can, F.; Ozkarahan, E. A. (1990). "Concepts and effectiveness of the cover-coefficient-based clustering methodology for text databases" . *ACM Transactions on Database Systems* **15** (4): 483. doi:10.1145/99935.99938. especially see Section 2.7.

[11] Honarkhah, M and Caers, J (2010). "Stochastic Simulation of Patterns Using Distance-Based Pattern Modeling" . *Mathematical Geosciences* **42** (5): 487–517. doi:10.1007/s11004-010-9276-7.

- Ralf Wagner, Sören W. Scholz, Reinhold Decker (2005): The Number of Clusters in Market Segmentation, in: Daniel Baier, Reinhold Decker; Lars Schmidt-Thieme (Eds.): Data Analysis and Decision Support, Berlin, Springer, 157 - 176.

2.2 Expectation–maximization algorithm

In statistics, an **expectation–maximization (EM) algorithm** is an iterative method for finding maximum likelihood or maximum a posteriori (MAP) estimates of parameters in statistical models, where the model depends on unobserved latent variables. The EM iteration alternates between performing an expectation (E) step, which creates a function for the expectation of the log-likelihood evaluated using the current estimate for the parameters, and a maximization (M) step, which computes parameters maximizing the expected log-likelihood found on the *E* step. These parameter-estimates are then used to determine the distribution of the latent variables in the next E step.



EM clustering of Old Faithful eruption data. The random initial model (which, due to the different scales of the axes, appears to be two very flat and wide spheres) is fit to the observed data. In the first iterations, the model changes substantially, but then converges to the two modes of the geyser. Visualized using ELKI.

2.2.1 History

The EM algorithm was explained and given its name in a classic 1977 paper by Arthur Dempster, Nan Laird, and Donald Rubin.*[1] They pointed out that the method had been "proposed many times in special circumstances" by earlier authors. In particular, a very detailed treatment of the EM method for exponential families was published by Rolf Sundberg in his thesis and several papers*[2]*[3]*[4] following his collaboration with Per Martin-Löf and Anders Martin-Löf.*[5]*[6]*[7]*[8]*[9]*[10]*[11] The Dempster-Laird-Rubin paper in 1977 generalized the method and sketched a convergence analysis for a wider class of problems. Regardless of earlier inventions, the innovative Dempster-Laird-Rubin paper in the *Journal of the Royal Statistical Society* received an enthusiastic discussion at the Royal Statistical Society meeting with Sundberg calling the paper "brilliant". The Dempster-Laird-Rubin paper established the EM method as an important tool of statistical analysis.

The convergence analysis of the Dempster-Laird-Rubin paper was flawed and a correct convergence analysis was published by C.F. Jeff Wu in 1983.*[12] Wu's proof established the EM method's convergence outside of the exponential family, as claimed by Dempster-Laird-Rubin.*[13]

2.2.2 Introduction

The EM algorithm is used to find (locally) maximum likelihood parameters of a statistical model in cases where the equations cannot be solved directly. Typically these models involve latent variables in addition to unknown parameters and known data observations. That is, either there are missing values among the data, or the model can be formulated more simply by assuming the existence of additional unobserved data points. For example, a mixture model can be described more simply by assuming that each observed data point has a corresponding unobserved data point, or latent variable, specifying the mixture component that each data point belongs to.

Finding a maximum likelihood solution typically requires taking the derivatives of the likelihood function with respect to all the unknown values —viz. the parameters and the latent variables —and simultaneously solving the resulting equations. In statistical models with latent variables, this usually is not possible. Instead, the result is typically a set of interlocking equations in which the solution to the parameters requires the values of the latent variables and vice versa, but substituting one set of equations into the other produces an unsolvable equation.

The EM algorithm proceeds from the observation that the following is a way to solve these two sets of equations numerically. One can simply pick arbitrary values for one of the two sets of unknowns, use them to estimate the second set, then use these new values to find a better estimate of the first set, and then keep alternating between the two until the resulting values both converge to fixed points. It's not obvious that this will work at all, but in fact it can be proven that in this particular context it does, and that the derivative of the likelihood is (arbitrarily close to) zero at that point, which in turn means that the point is either a maximum or a saddle point. In general there may be multiple maxima, and there is no guarantee that the global maximum will be found. Some likelihoods also have singularities in them, i.e. nonsensical maxima. For example, one of the "solutions" that may be found by EM in a mixture model involves setting one of the components to have zero variance and the mean parameter for the same component to be equal to one of the data points.

2.2.3 Description

Given a statistical model which generates a set $\mathbf{X}$ of observed data, a set of unobserved latent data or missing values $\mathbf{Z}$, and a vector of unknown parameters $\boldsymbol{\theta}$, along with a likelihood function $L(\boldsymbol{\theta};\mathbf{X},\mathbf{Z}) = p(\mathbf{X},\mathbf{Z}|\boldsymbol{\theta})$, the maximum likelihood estimate (MLE) of the unknown parameters is determined by the marginal likelihood of the observed data

$$L(\boldsymbol{\theta};\mathbf{X}) = p(\mathbf{X}|\boldsymbol{\theta}) = \sum_{\mathbf{Z}} p(\mathbf{X},\mathbf{Z}|\boldsymbol{\theta})$$

However, this quantity is often intractable (e.g. if $\mathbf{Z}$ is a sequence of events, so that the number of values grows exponentially with the sequence length, making the exact calculation of the sum extremely difficult).

The EM algorithm seeks to find the MLE of the marginal likelihood by iteratively applying the following two steps:

Expectation step (E step): Calculate the expected value of the log likelihood function, with respect to the conditional distribution of $\mathbf{Z}$ given $\mathbf{X}$ under the current estimate of the parameters $\boldsymbol{\theta}^{(t)}$:

$$Q(\boldsymbol{\theta}|\boldsymbol{\theta}^{(t)}) = \operatorname{E}_{\mathbf{Z}|\mathbf{X},\boldsymbol{\theta}^{(t)}}\left[\log L(\boldsymbol{\theta};\mathbf{X},\mathbf{Z})\right]$$

Maximization step (M step): Find the parameter that maximizes this quantity:

$$\boldsymbol{\theta}^{(t+1)} = \underset{\boldsymbol{\theta}}{\arg\max}\ Q(\boldsymbol{\theta}|\boldsymbol{\theta}^{(t)})$$

Note that in typical models to which EM is applied:

1. The observed data points $\mathbf{X}$ may be discrete (taking values in a finite or countably infinite set) or continuous (taking values in an uncountably infinite set). There may in fact be a vector of observations associated with each data point.
2. The missing values (aka latent variables) $\mathbf{Z}$ are discrete, drawn from a fixed number of values, and there is one latent variable per observed data point.
3. The parameters are continuous, and are of two kinds: Parameters that are associated with all data points, and parameters associated with a particular value of a latent variable (i.e. associated with all data points whose corresponding latent variable has a particular value).

However, it is possible to apply EM to other sorts of models.

The motivation is as follows. If we know the value of the parameters $\boldsymbol{\theta}$, we can usually find the value of the latent variables $\mathbf{Z}$ by maximizing the log-likelihood over all possible values of $\mathbf{Z}$, either simply by iterating over $\mathbf{Z}$ or through an algorithm such as the Viterbi algorithm for hidden Markov models. Conversely, if we know the value of the latent variables $\mathbf{Z}$, we can find an estimate of the parameters $\boldsymbol{\theta}$ fairly easily, typically by simply grouping the observed data points according to the value of the associated latent variable and averaging the values, or some function of the values, of the points in each group. This suggests an iterative algorithm, in the case where both $\boldsymbol{\theta}$ and $\mathbf{Z}$ are unknown:

1. First, initialize the parameters $\boldsymbol{\theta}$ to some random values.
2. Compute the best value for $\mathbf{Z}$ given these parameter values.
3. Then, use the just-computed values of $\mathbf{Z}$ to compute a better estimate for the parameters $\boldsymbol{\theta}$. Parameters associated with a particular value of $\mathbf{Z}$ will use only those data points whose associated latent variable has that value.
4. Iterate steps 2 and 3 until convergence.

The algorithm as just described monotonically approaches a local minimum of the cost function, and is commonly called *hard EM*. The k-means algorithm is an example of this class of algorithms.

However, one can do somewhat better: Rather than making a hard choice for $\mathbf{Z}$ given the current parameter values and averaging only over the set of data points associated with a particular value of $\mathbf{Z}$, one can instead determine the probability of each possible value of $\mathbf{Z}$ for each data point, and then use the probabilities associated with a particular value of $\mathbf{Z}$ to compute a weighted average over the entire set of data points. The resulting algorithm is commonly called *soft EM*, and is the type of algorithm normally associated with EM. The counts used to compute these weighted averages are called *soft counts* (as opposed to the *hard counts* used in a hard-EM-type algorithm such as k-means). The probabilities computed for $\mathbf{Z}$ are posterior probabilities and are what is computed in the E step. The soft counts used to compute new parameter values are what is computed in the M step.

2.2.4 Properties

Speaking of an expectation (E) step is a bit of a misnomer. What is calculated in the first step are the fixed, data-dependent parameters of the function Q. Once the parameters of Q are known, it is fully determined and is maximized in the second (M) step of an EM algorithm.

Although an EM iteration does increase the observed data (i.e. marginal) likelihood function there is no guarantee that the sequence converges to a maximum likelihood estimator. For multimodal distributions, this means that an EM algorithm may converge to a local maximum of the observed data likelihood function, depending on starting values. There are a variety of heuristic or metaheuristic approaches for escaping a local maximum such as random restart (starting with several different random initial estimates $\theta^*(t)$), or applying simulated annealing methods.

EM is particularly useful when the likelihood is an exponential family: the E step becomes the sum of expectations of sufficient statistics, and the M step involves maximizing a linear function. In such a case, it is usually possible to derive closed form updates for each step, using the Sundberg formula (published by Rolf Sundberg using unpublished results of Per Martin-Löf and Anders Martin-Löf).*[3]*[4]*[7]*[8]*[9]*[10]*[11]

The EM method was modified to compute maximum a posteriori (MAP) estimates for Bayesian inference in the original paper by Dempster, Laird, and Rubin.

There are other methods for finding maximum likelihood estimates, such as gradient descent, conjugate gradient or variations of the Gauss–Newton method. Unlike EM, such methods typically require the evaluation of first and/or second derivatives of the likelihood function.

2.2.5 Proof of correctness

Expectation-maximization works to improve $Q(\boldsymbol\theta|\boldsymbol\theta^{(t)})$ rather than directly improving $\log p(\mathbf{X}|\boldsymbol\theta)$. Here we show that improvements to the former imply improvements to the latter.*[14]

For any $\mathbf{Z}$ with non-zero probability $p(\mathbf{Z}|\mathbf{X},\boldsymbol\theta)$, we can write

$$\log p(\mathbf{X}|\boldsymbol\theta) = \log p(\mathbf{X},\mathbf{Z}|\boldsymbol\theta) - \log p(\mathbf{Z}|\mathbf{X},\boldsymbol\theta)\,.$$

We take the expectation over values of $\mathbf{Z}$ by multiplying both sides by $p(\mathbf{Z}|\mathbf{X},\boldsymbol\theta^{(t)})$ and summing (or integrating) over $\mathbf{Z}$. The left-hand side is the expectation of a constant, so we get:

$$\begin{aligned}\log p(\mathbf{X}|\boldsymbol\theta) &= \sum_{\mathbf{Z}} p(\mathbf{Z}|\mathbf{X},\boldsymbol\theta^{(t)}) \log p(\mathbf{X},\mathbf{Z}|\boldsymbol\theta) - \sum_{\mathbf{Z}} p(\mathbf{Z}|\mathbf{X},\boldsymbol\theta^{(t)}) \log p(\mathbf{Z}|\mathbf{X},\boldsymbol\theta) \\ &= Q(\boldsymbol\theta|\boldsymbol\theta^{(t)}) + H(\boldsymbol\theta|\boldsymbol\theta^{(t)})\,,\end{aligned}$$

where $H(\boldsymbol\theta|\boldsymbol\theta^{(t)})$ is defined by the negated sum it is replacing. This last equation holds for any value of $\boldsymbol\theta$ including $\boldsymbol\theta = \boldsymbol\theta^{(t)}$,

$$\log p(\mathbf{X}|\boldsymbol\theta^{(t)}) = Q(\boldsymbol\theta^{(t)}|\boldsymbol\theta^{(t)}) + H(\boldsymbol\theta^{(t)}|\boldsymbol\theta^{(t)})\,,$$

and subtracting this last equation from the previous equation gives

$$\log p(\mathbf{X}|\boldsymbol\theta) - \log p(\mathbf{X}|\boldsymbol\theta^{(t)}) = Q(\boldsymbol\theta|\boldsymbol\theta^{(t)}) - Q(\boldsymbol\theta^{(t)}|\boldsymbol\theta^{(t)}) + H(\boldsymbol\theta|\boldsymbol\theta^{(t)}) - H(\boldsymbol\theta^{($$

However, Gibbs' inequality tells us that $H(\boldsymbol\theta|\boldsymbol\theta^{(t)}) \ge H(\boldsymbol\theta^{(t)}|\boldsymbol\theta^{(t)})$, so we can conclude that

$$\log p(\mathbf{X}|\boldsymbol\theta) - \log p(\mathbf{X}|\boldsymbol\theta^{(t)}) \ge Q(\boldsymbol\theta|\boldsymbol\theta^{(t)}) - Q(\boldsymbol\theta^{(t)}|\boldsymbol\theta^{(t)})\,.$$

In words, choosing $\boldsymbol\theta$ to improve $Q(\boldsymbol\theta|\boldsymbol\theta^{(t)})$ beyond $Q(\boldsymbol\theta^{(t)}|\boldsymbol\theta^{(t)})$ will improve $\log p(\mathbf{X}|\boldsymbol\theta)$ beyond $\log p(\mathbf{X}|\boldsymbol\theta^{(t)})$ at least as much.

2.2.6 Alternative description

Under some circumstances, it is convenient to view the EM algorithm as two alternating maximization steps.*[15]*[16] Consider the function:

$$F(q,\theta) = \operatorname{E}_q[\log L(\theta;x,Z)] + H(q) = -D_{\mathrm{KL}}\big(q \big\| p_{Z|X}(\cdot|x;\theta)\big) + \log L(\theta;$$

where q is an arbitrary probability distribution over the unobserved data z, $p_{Z|X}(\cdot|x;\theta)$ is the conditional distribution of the unobserved data given the observed data x, H is the entropy and D_{KL} is the Kullback–Leibler divergence.

Then the steps in the EM algorithm may be viewed as:

Expectation step: Choose q to maximize F:

$q^{(t)} = *\, arg\, max_q\ F(q,\theta^{(t)})$

Maximization step: Choose θ to maximize F:

$\theta^{(t+1)} = *\, arg\, max_\theta\ F(q^{(t)},\theta)$

2.2.7 Applications

EM is frequently used for data clustering in machine learning and computer vision. In natural language processing, two prominent instances of the algorithm are the Baum-Welch algorithm and the inside-outside algorithm for unsupervised induction of probabilistic context-free grammars.

In psychometrics, EM is almost indispensable for estimating item parameters and latent abilities of item response theory models.

With the ability to deal with missing data and observe unidentified variables, EM is becoming a useful tool to price and manage risk of a portfolio.[ref?]

The EM algorithm (and its faster variant Ordered subset expectation maximization) is also widely used in medical image reconstruction, especially in positron emission tomography and single photon emission computed tomography. See below for other faster variants of EM.

2.2.8 Filtering and smoothing EM algorithms

A Kalman filter is typically used for on-line state estimation and a minimum-variance smoother may be employed for off-line or batch state estimation. However, these minimum-variance solutions require estimates of the state-space model parameters. EM algorithms can be used for solving joint state and parameter estimation problems.

Filtering and smoothing EM algorithms arise by repeating the following two-step procedure:

E-step Operate a Kalman filter or a minimum-variance smoother designed with current parameter estimates to obtain updated state estimates.

M-step Use the filtered or smoothed state estimates within maximum-likelihood calculations to obtain updated parameter estimates.

Suppose that a Kalman filter or minimum-variance smoother operates on noisy measurements of a single-input-single-output system. An updated measurement noise variance estimate can be obtained from the maximum likelihood calculation

$$\hat{\sigma}_v^2 = \frac{1}{N}\sum_{k=1}^{N}(z_k - \hat{x}_k)^2$$

where $\hat{x}_k$ are scalar output estimates calculated by a filter or a smoother from N scalar measurements z_k . Similarly, for a first-order auto-regressive process, an updated process noise variance estimate can be calculated by

$$\hat{\sigma}_w^2 = \frac{1}{N}\sum_{k=1}^{N}(\hat{x}_{k+1} - \hat{F}\hat{x}_k)^2$$

where $\hat{x}_k$ and $\hat{x}_{k+1}$ are scalar state estimates calculated by a filter or a smoother. The updated model coefficient estimate is obtained via

$$\hat{F} = \frac{\sum_{k=1}^{N}(\hat{x}_{k+1} - \hat{F}\hat{x}_k)}{\sum_{k=1}^{N}\hat{x}_k^2}$$

The convergence of parameter estimates such as those above are well studied.*[17]*[18]*[19]

2.2.9 Variants

A number of methods have been proposed to accelerate the sometimes slow convergence of the EM algorithm, such as those using conjugate gradient and modified Newton–Raphson techniques.*[20] Additionally EM can be used with constrained estimation techniques.

Expectation conditional maximization (ECM) replaces each M step with a sequence of conditional maximization (CM) steps in which each parameter θ_i is maximized individually, conditionally on the other parameters remaining fixed.*[21]

This idea is further extended in **generalized expectation maximization (GEM)** algorithm, in which one only seeks an increase in the objective function F for both the E step and M step under the alternative description.*[15]

It is also possible to consider the EM algorithm as a subclass of the **MM** (Majorize/Minimize or Minorize/Maximize, depending on context) algorithm,*[22] and therefore use any machinery developed in the more general case.

α-EM algorithm

The Q-function used in the EM algorithm is based on the log likelihood. Therefore, it is regarded as the log-EM algorithm. The use of the log likelihood can be generalized to that of the α-log likelihood ratio. Then, the α-log likelihood ratio of the observed data can be exactly expressed as equality by using the Q-function of the α-log likelihood ratio and the α-divergence. Obtaining this Q-function is a generalized E step. Its maximization is a generalized M step. This pair is called the α-EM algorithm *[23] which contains the log-EM algorithm as its subclass. Thus, the α-EM algorithm by Yasuo Matsuyama is an exact generalization of the log-EM algorithm. No computation of gradient or Hessian matrix is needed. The α-EM shows faster convergence than the log-EM algorithm by choosing an appropriate α. The α-EM algorithm leads to a faster version of the Hidden Markov model estimation algorithm α-HMM. *[24]

2.2.10 Relation to variational Bayes methods

EM is a partially non-Bayesian, maximum likelihood method. Its final result gives a probability distribution over the latent variables (in the Bayesian style) together with a point estimate for θ (either a maximum likelihood estimate or a posterior mode). We may want a fully Bayesian version of this, giving a probability distribution over θ as well as the latent variables. In fact the Bayesian approach to inference is simply to treat θ as another latent variable. In this paradigm, the distinction between the E and M steps disappears. If we use the factorized

Q approximation as described above (variational Bayes), we may iterate over each latent variable (now including θ) and optimize them one at a time. There are now k steps per iteration, where k is the number of latent variables. For graphical models this is easy to do as each variable's new Q depends only on its Markov blanket, so local message passing can be used for efficient inference.

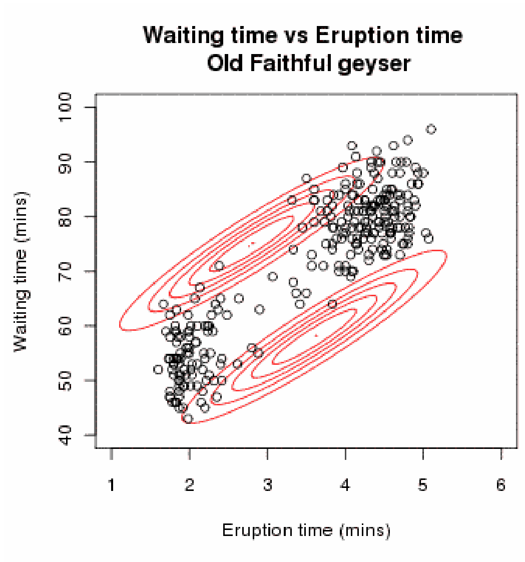
2.2.11 Geometric interpretation

For more details on this topic, see Information geometry.

In information geometry, the E step and the M step are interpreted as projections under dual affine connections, called the e-connection and the m-connection; the Kullback–Leibler divergence can also be understood in these terms.

2.2.12 Examples

Gaussian mixture



An animation demonstrating the EM algorithm fitting a two component Gaussian mixture model to the Old Faithful dataset. The algorithm steps through from a random initialization to convergence.

Let $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n)$ be a sample of n independent observations from a mixture of two multivariate normal distributions of dimension d, and let $\mathbf{z} = (z_1, z_2, \ldots, z_n)$ be the latent variables that determine the component from which the observation originates.*[16]

$$X_i|(Z_i = 1) \sim \mathcal{N}_d(\boldsymbol{\mu}_1, \Sigma_1) \text{ and } X_i|(Z_i = 2) \sim \mathcal{N}_d(\boldsymbol{\mu}_2, \Sigma_2)$$

where

$$\operatorname{P}(Z_i = 1) = \tau_1 \text{ and } \operatorname{P}(Z_i = 2) = \tau_2 = 1 - \tau_1$$

The aim is to estimate the unknown parameters representing the “mixing” value between the Gaussians and the means and covariances of each:

$$\theta = \big(\boldsymbol{\tau}, \boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \Sigma_1, \Sigma_2\big)$$

where the incomplete-data likelihood function is

$$L(\theta; \mathbf{x}) = \prod_{i=1}^{n} \sum_{j=1}^{2} \tau_j \, f(\mathbf{x}_i; \boldsymbol{\mu}_j, \Sigma_j)$$

and the complete-data likelihood function is

$$L(\theta; \mathbf{x}, \mathbf{z}) = P(\mathbf{x}, \mathbf{z}|\theta) = \prod_{i=1}^{n} \sum_{j=1}^{2} \mathbb{I}(z_i = j) \, f(\mathbf{x}_i; \boldsymbol{\mu}_j, \Sigma_j)\tau_j$$

or

$$L(\theta; \mathbf{x}, \mathbf{z}) = \exp\Bigg\{ \sum_{i=1}^{n} \sum_{j=1}^{2} \mathbb{I}(z_i = j) \big[-\tfrac{1}{2} \log |\Sigma_j| - \tfrac{1}{2} (\mathbf{x}_i - \boldsymbol{\mu}_j)^\top \Sigma_j^{-1} (\mathbf{x}$$

where $\mathbb{I}$ is an indicator function and f is the probability density function of a multivariate normal.

To see the last equality, note that for each i all indicators $\mathbb{I}(z_i = j)$ are equal to zero, except for one which is equal to one. The inner sum thus reduces to a single term.

E step Given our current estimate of the parameters $\theta^*(t)$, the conditional distribution of the Z_i is determined by Bayes theorem to be the proportional height of the normal density weighted by τ:

$$T_{j,i}^{(t)} := \operatorname{P}(Z_i = j | X_i = \mathbf{x}_i; \theta^{(t)}) = \frac{\tau_j^{(t)} \, f(\mathbf{x}_i; \boldsymbol{\mu}_j^{(t)}, \Sigma_j^{(t)})}{\tau_1^{(t)} \, f(\mathbf{x}_i; \boldsymbol{\mu}_1^{(t)}, \Sigma_1^{(t)}) + \tau_2^{(t)} \, f(\mathbf{x}_i; \boldsymbol{\mu}_2^{(t}}$$

These are called the “membership probabilities” which are normally considered the output of the E step (although this is not the Q function of below).

Note that this E step corresponds with the following function for Q:

$$
\begin{aligned}
Q(\theta|\theta^{(t)}) &= \mathrm{E}[\log L(\theta;\mathbf{x},\mathbf{Z})] \\
&= \mathrm{E}[\log \prod_{i=1}^{n} L(\theta;\mathbf{x}_i,\mathbf{z}_i)] \\
&= \mathrm{E}[\sum_{i=1}^{n} \log L(\theta;\mathbf{x}_i,\mathbf{z}_i)] \\
&= \sum_{i=1}^{n} \mathrm{E}[\log L(\theta;\mathbf{x}_i,\mathbf{z}_i)] \\
&= \sum_{i=1}^{n}\sum_{j=1}^{2} T_{j,i}^{(t)} \left[\log \tau_j - \tfrac{1}{2}\log|\Sigma_j| - \tfrac{1}{2}(\mathbf{x}_i-\boldsymbol{\mu}_j)^\top \Sigma_j^{-1}(\mathbf{x}_i-\boldsymbol{\mu}_j) - \tfrac{d}{2}\log(2\pi) \right]
\end{aligned}
$$

This does not need to be calculated, because in the M step we only require the terms depending on τ when we maximize for τ, or only the terms depending on $\boldsymbol{\mu}$ if we maximize for $\boldsymbol{\mu}$.

M step The fact that $Q(\theta|\theta^*(t))$ is quadratic in form means that determining the maximizing values of θ is relatively straightforward. Note that τ, ($\boldsymbol{\mu}_1$,Σ_1) and ($\boldsymbol{\mu}_2$,Σ_2) may all be maximized independently since they all appear in separate linear terms.

To begin, consider τ, which has the constraint $\tau_1 + \tau_2$=1:

$$
\begin{aligned}
\boldsymbol{\tau}^{(t+1)} &= \underset{\boldsymbol{\tau}}{\arg\max}\ Q(\theta|\theta^{(t)}) \\
&= \underset{\boldsymbol{\tau}}{\arg\max} \left\{ \left[\sum_{i=1}^{n} T_{1,i}^{(t)}\right] \log\tau_1 + \left[\sum_{i=1}^{n} T_{2,i}^{(t)}\right] \log\tau_2 \right\}
\end{aligned}
$$

This has the same form as the MLE for the binomial distribution, so

$$
\tau_j^{(t+1)} = \frac{\sum_{i=1}^{n} T_{j,i}^{(t)}}{\sum_{i=1}^{n}(T_{1,i}^{(t)} + T_{2,i}^{(t)})} = \frac{1}{n}\sum_{i=1}^{n} T_{j,i}^{(t)}
$$

For the next estimates of ($\boldsymbol{\mu}_1$,σ_1):

$$
\begin{aligned}
(\boldsymbol{\mu}_1^{(t+1)}, \Sigma_1^{(t+1)}) &= \underset{\boldsymbol{\mu}_1,\Sigma_1}{\arg\max}\ Q(\theta|\theta^{(t)}) \\
&= \underset{\boldsymbol{\mu}_1,\Sigma_1}{\arg\max} \sum_{i=1}^{n} T_{1,i}^{(t)} \left\{ -\tfrac{1}{2}\log|\Sigma_1| - \tfrac{1}{2}(\mathbf{x}_i-\boldsymbol{\mu}_1)^\top \Sigma_1^{-1}(\mathbf{x}_i-\boldsymbol{\mu}_1) \right\}
\end{aligned}
$$

This has the same form as a weighted MLE for a normal distribution, so

$$
\boldsymbol{\mu}_1^{(t+1)} = \frac{\sum_{i=1}^{n} T_{1,i}^{(t)}\mathbf{x}_i}{\sum_{i=1}^{n} T_{1,i}^{(t)}} \text{ and } \Sigma_1^{(t+1)} = \frac{\sum_{i=1}^{n} T_{1,i}^{(t)}(\mathbf{x}_i-\boldsymbol{\mu}_1^{(t+1)})(\mathbf{x}_i-\boldsymbol{\mu}_1^{(t+1)})^\top}{\sum_{i=1}^{n} T_{1,i}^{(t)}}
$$

and, by symmetry

$$
\boldsymbol{\mu}_2^{(t+1)} = \frac{\sum_{i=1}^{n} T_{2,i}^{(t)}\mathbf{x}_i}{\sum_{i=1}^{n} T_{2,i}^{(t)}} \text{ and } \Sigma_2^{(t+1)} = \frac{\sum_{i=1}^{n} T_{2,i}^{(t)}(\mathbf{x}_i-\boldsymbol{\mu}_2^{(t+1)})(\mathbf{x}_i-\boldsymbol{\mu}_2^{(t+1)})^\top}{\sum_{i=1}^{n} T_{2,i}^{(t)}} .
$$

Termination Conclude the iterative process if $\log L(\theta^t;\mathbf{x},\mathbf{Z}) \leq \log L(\theta^{(t-1)};\mathbf{x},\mathbf{Z}) + \epsilon$ for ϵ below some preset threshold.

Generalization The algorithm illustrated above can be generalized for mixtures of more than two multivariate normal distributions.

Truncated and censored regression

The EM algorithm has been implemented in the case where there is an underlying linear regression model explaining the variation of some quantity, but where the values actually observed are censored or truncated versions of those represented in the model.*[25] Special cases of this model include censored or truncated observations from a single normal distribution.*[25]

2.2.13 Alternatives to EM

EM typically converges to a local optimum--not necessarily the global optimum--and there is no bound on the convergence rate in general. It is possible that it can be arbitrarily poor in high dimensions and there can be an exponential number of local optima. Hence, there is a need for alternative techniques for guaranteed learning, especially in the high-dimensional setting. There are alternatives to EM with better guarantees in terms of consistency which are known as moment-based approaches or the so-called "spectral techniques" . Moment-based approaches*[26] to learning the parameters of a probabilistic model are of increasing interest recently since they enjoy guarantees such as global convergence under certain conditions unlike EM which is often plagued by the issue of getting stuck in local optima. Algorithms with guarantees for learning can be derived for a number of important models such as mixture models, Hidden Markov models *[27] and community models.*[28] For these spectral methods, there are no spurious local optima and the true parameters can be consistently estimated under some regularity conditions.

2.2.14 See also

- Density estimation
- Total absorption spectroscopy
- The EM algorithm can be viewed as a special case of the majorize-minimization (MM) algorithm.*[29]

2.2.15 Further reading

- Robert Hogg, Joseph McKean and Allen Craig. *Introduction to Mathematical Statistics*. pp. 359–364. Upper Saddle River, NJ: Pearson Prentice Hall, 2005.
- The on-line textbook: Information Theory, Inference, and Learning Algorithms, by David J.C. MacKay includes simple examples of the EM algorithm such as clustering using the soft *k*-means algorithm, and emphasizes the variational view of the EM algorithm, as described in Chapter 33.7 of version 7.2 (fourth edition).
- Dellaert, Frank. “The Expectation Maximization Algorithm”. CiteSeerX: 10.1.1.9.9735, gives an easier explanation of EM algorithm in terms of lowerbound maximization.
- Bishop, Christopher M. (2006). *Pattern Recognition and Machine Learning*. Springer. ISBN 0-387-31073-8.
- M. R. Gupta and Y. Chen (2010). *Theory and Use of the EM Algorithm*. doi:10.1561/2000000034. A well-written short book on EM, including detailed derivation of EM for GMMs, HMMs, and Dirichlet.
- Bilmes, Jeff. “A Gentle Tutorial of the EM Algorithm and its Application to Parameter Estimation for Gaussian Mixture and Hidden Markov Models”. CiteSeerX: 10.1.1.28.613, includes a simplified derivation of the EM equations for Gaussian Mixtures and Gaussian Mixture Hidden Markov Models.
- Variational Algorithms for Approximate Bayesian Inference, by M. J. Beal includes comparisons of EM to Variational Bayesian EM and derivations of several models including Variational Bayesian HMMs (chapters).
- The Expectation Maximization Algorithm: A short tutorial, A self-contained derivation of the EM Algorithm by Sean Borman.
- The EM Algorithm, by Xiaojin Zhu.
- EM algorithm and variants: an informal tutorial by Alexis Roche. A concise and very clear description of EM and many interesting variants.
- Einicke, G.A. (2012). *Smoothing, Filtering and Prediction: Estimating the Past, Present and Future*. Rijeka, Croatia: Intech. ISBN 978-953-307-752-9.

2.2.16 References

[1] Dempster, A.P.; Laird, N.M.; Rubin, D.B. (1977). “Maximum Likelihood from Incomplete Data via the EM Algorithm”. *Journal of the Royal Statistical Society, Series B* **39** (1): 1–38. JSTOR 2984875. MR 0501537.

[2] Sundberg, Rolf (1974). “Maximum likelihood theory for incomplete data from an exponential family”. *Scandinavian Journal of Statistics* **1** (2): 49–58. JSTOR 4615553. MR 381110.

[3] Rolf Sundberg. 1971. *Maximum likelihood theory and applications for distributions generated when observing a function of an exponential family variable*. Dissertation, Institute for Mathematical Statistics, Stockholm University.

[4] Sundberg, Rolf (1976). “An iterative method for solution of the likelihood equations for incomplete data from exponential families”. *Communications in Statistics – Simulation and Computation* **5** (1): 55–64. doi:10.1080/03610917608812007. MR 443190.

[5] See the acknowledgement by Dempster, Laird and Rubin on pages 3, 5 and 11.

[6] G. Kulldorff. 1961. *Contributions to the theory of estimation from grouped and partially grouped samples*. Almqvist & Wiksell.

[7] Anders Martin-Löf. 1963. “Utvärdering av livslängder i subnanosekundsområdet” (“Evaluation of subnanosecond lifetimes”). (“Sundberg formula”)

[8] Per Martin-Löf. 1966. *Statistics from the point of view of statistical mechanics*. Lecture notes, Mathematical Institute, Aarhus University. (“Sundberg formula” credited to Anders Martin-Löf).

[9] Per Martin-Löf. 1970. *Statistika Modeller (Statistical Models): Anteckningar från seminarier läsåret 1969–1970 (Notes from seminars in the academic year 1969-1970), with the assistance of Rolf Sundberg*. Stockholm University. (“Sundberg formula”)

[10] Martin-Löf, P. The notion of redundancy and its use as a quantitative measure of the deviation between a statistical hypothesis and a set of observational data. With a discussion by F. Abildgård, A. P. Dempster, D. Basu, D. R. Cox, A. W. F. Edwards, D. A. Sprott, G. A. Barnard, O. Barndorff-Nielsen, J. D. Kalbfleisch and G. Rasch and a reply by the author. *Proceedings of Conference on Foundational Questions in Statistical Inference* (Aarhus, 1973), pp. 1–42. Memoirs, No. 1, Dept. Theoret. Statist., Inst. Math., Univ. Aarhus, Aarhus, 1974.

[11] Martin-Löf, Per The notion of redundancy and its use as a quantitative measure of the discrepancy between a statistical hypothesis and a set of observational data. *Scand. J. Statist.* 1 (1974), no. 1, 3–18.

[12] Wu, C. F. Jeff (1983). “On the Convergence Properties of the EM Algorithm”. *The Annals of Statistics* (Institute of Mathematical Statistics) **11** (1): 95–103. Retrieved 11 December 2014.

[13] Wu, C. F. Jeff (Mar 1983). “On the Convergence Properties of the EM Algorithm”. *Annals of Statistics* **11** (1): 95–103. doi:10.1214/aos/1176346060. JSTOR 2240463. MR 684867.

[14] Little, Roderick J.A.; Rubin, Donald B. (1987). *Statistical Analysis with Missing Data*. Wiley Series in Probability and Mathematical Statistics. New York: John Wiley & Sons. pp. 134–136. ISBN 0-471-80254-9.

[15] Neal, Radford; Hinton, Geoffrey (1999). Michael I. Jordan, ed. "A view of the EM algorithm that justifies incremental, sparse, and other variants" (PDF). *Learning in Graphical Models* (Cambridge, MA: MIT Press): 355–368. ISBN 0-262-60032-3. Retrieved 2009-03-22.

[16] Hastie, Trevor; Tibshirani, Robert; Friedman, Jerome (2001). "8.5 The EM algorithm". *The Elements of Statistical Learning*. New York: Springer. pp. 236–243. ISBN 0-387-95284-5.

[17] Einicke, G.A.; Malos, J.T.; Reid, D.C.; Hainsworth, D.W. (January 2009). "Riccati Equation and EM Algorithm Convergence for Inertial Navigation Alignment" . *IEEE Trans. Signal Processing* **57** (1): 370–375. doi:10.1109/TSP.2008.2007090.

[18] Einicke, G.A.; Falco, G.; Malos, J.T. (May 2010). "EM Algorithm State Matrix Estimation for Navigation" . *IEEE Signal Processing Letters* **17** (5): 437–440. Bibcode:2010ISPL...17..437E. doi:10.1109/LSP.2010.2043151.

[19] Einicke, G.A.; Falco, G.; Dunn, M.T.; Reid, D.C. (May 2012). "Iterative Smoother-Based Variance Estimation" . *IEEE Signal Processing Letters* **19** (5): 275–278. Bibcode:2012ISPL...19..275E. doi:10.1109/LSP.2012.2190278.

[20] Jamshidian, Mortaza; Jennrich, Robert I. (1997). "Acceleration of the EM Algorithm by using Quasi-Newton Methods" . *Journal of the Royal Statistical Society, Series B* **59** (2): 569–587. doi:10.1111/1467-9868.00083. MR 1452026.

[21] Meng, Xiao-Li; Rubin, Donald B. (1993). "Maximum likelihood estimation via the ECM algorithm: A general framework" . *Biometrika* **80** (2): 267–278. doi:10.1093/biomet/80.2.267. MR 1243503.

[22] Hunter DR and Lange K (2004), A Tutorial on MM Algorithms, The American Statistician, 58: 30-37

[23] Matsuyama, Yasuo (2003). "The α-EM algorithm: Surrogate likelihood maximization using α-logarithmic information measures" . *IEEE Transactions on Information Theory* **49** (3): 692–706. doi:10.1109/TIT.2002.808105.

[24] Matsuyama, Yasuo (2011). "Hidden Markov model estimation based on alpha-EM algorithm: Discrete and continuous alpha-HMMs" . *International Joint Conference on Neural Networks*: 808–816.

[25] Wolynetz, M.S. (1979). "Maximum likelihood estimation in a linear model from confined and censored normal data" . *Journal of the Royal Statistical Society, Series C* **28** (2): 195–206.

[26] Anandkumar, Animashree; Ge, Rong; Hsu, Daniel; Kakade, Sham M; Telgarsky, Matus (2014). "Tensor decompositions for learning latent variable models" . *The Journal of Machine Learning Research* **15** (1): 2773–2832.

[27] Anandkumar, Animashree; Hsu, Daniel; Kakade, Sham M (2012). "A method of moments for mixture models and hidden Markov models" . *arXiv preprint arXiv:1203.0683*.

[28] Anandkumar, Animashree; Ge, Rong; Hsu, Daniel; Kakade, Sham M (2014). "A tensor approach to learning mixed membership community models" . *The Journal of Machine Learning Research* **15** (1): 2239–2312.

[29] Lange, Kenneth. "The MM Algorithm" (PDF).

2.2.17 External links

- Various 1D, 2D and 3D demonstrations of EM together with Mixture Modeling are provided as part of the paired SOCR activities and applets. These applets and activities show empirically the properties of the EM algorithm for parameter estimation in diverse settings.
- k-MLE: A fast algorithm for learning statistical mixture models
- Class hierarchy in C++ (GPL) including Gaussian Mixtures
- Fast and clean C implementation of the Expectation Maximization (EM) algorithm for estimating Gaussian Mixture Models (GMMs).

2.3 Clustering high-dimensional data

Clustering high-dimensional data is the cluster analysis of data with anywhere from a few dozen to many thousands of dimensions. Such high-dimensional data spaces are often encountered in areas such as medicine, where DNA microarray technology can produce a large number of measurements at once, and the clustering of text documents, where, if a word-frequency vector is used, the number of dimensions equals the size of the vocabulary.

2.3.1 Problems

Four problems need to be overcome for clustering in high-dimensional data:*[1]

- Multiple dimensions are hard to think in, impossible to visualize, and, due to the exponential growth of the number of possible values with each dimension, complete enumeration of all subspaces becomes intractable with increasing dimensionality. This problem is known as the curse of dimensionality.
- The concept of distance becomes less precise as the number of dimensions grows, since the distance between any two points in a given dataset converges.

The discrimination of the nearest and farthest point in particular becomes meaningless:

$$\lim_{d\to\infty} \frac{dist_{\max} - dist_{\min}}{dist_{\min}} = 0$$

- A cluster is intended to group objects that are related, based on observations of their attribute's values. However, given a large number of attributes some of the attributes will usually not be meaningful for a given cluster. For example, in newborn screening a cluster of samples might identify newborns that share similar blood values, which might lead to insights about the relevance of certain blood values for a disease. But for different diseases, different blood values might form a cluster, and other values might be uncorrelated. This is known as the *local feature relevance* problem: different clusters might be found in different subspaces, so a global filtering of attributes is not sufficient.

- Given a large number of attributes, it is likely that some attributes are correlated. Hence, clusters might exist in arbitrarily oriented affine subspaces.

Recent research indicates that the discrimination problems only occur when there is a high number of irrelevant dimensions, and that shared-nearest-neighbor approaches can improve results.*[2]

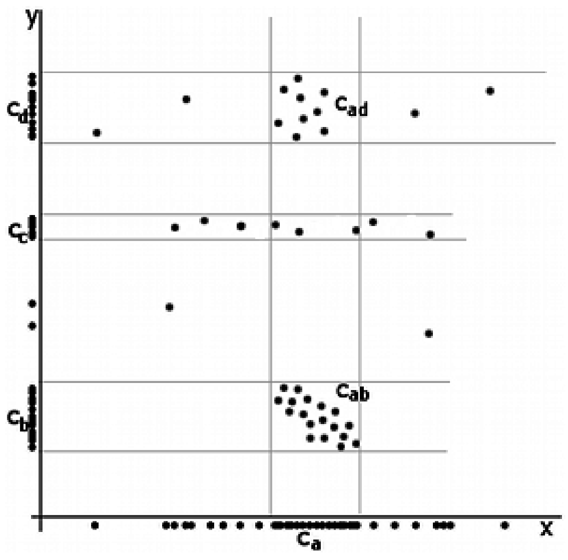
2.3.2 Approaches

Approaches towards clustering in axis-parallel or arbitrarily oriented affine subspaces differ in how they interpret the overall goal, which is finding clusters in data with high dimensionality.*[1] An overall different approach is to find clusters based on pattern in the data matrix, often referred to as biclustering, which is a technique frequently utilized in bioinformatics.

Subspace clustering

Subspace clustering is the task of detecting *all* clusters in *all subspaces*. This means that a point might be a member of multiple clusters, each existing in a different subspace. Subspaces can either be axis-parallel or affine. The term is often used synonymously with general clustering in high-dimensional data.

The image on the right shows a mere two-dimensional space where a number of clusters can be identified. In the one-dimensional subspaces, the clusters c_a (in subspace $\{x\}$) and c_b , c_c , c_d (in subspace $\{y\}$) can be found. c_c cannot be considered a cluster in a two-dimensional (sub-)space, since it is too sparsely distributed in the x axis.



Example 2D space with subspace clusters

In two dimensions, the two clusters c_{ab} and c_{ad} can be identified.

The problem of subspace clustering is given by the fact that there are 2^d different subspaces of a space with d dimensions. If the subspaces are not axis-parallel, an infinite number of subspaces is possible. Hence, subspace clustering algorithm utilize some kind of heuristic to remain computationally feasible, at the risk of producing inferior results. For example, the *downward-closure property* (cf. association rules) can be used to build higher-dimensional subspaces only by combining lower-dimensional ones, as any subspace T containing a cluster, will result in a full space S also to contain that cluster (i.e. S ⊆ T), an approach taken by most of the traditional algorithms such as CLIQUE,*[3] SUBCLU.*[4] It is also possible to define a subspace using different degrees of relevance for each dimension, an approach taken by iMWK-Means.*[5]

Projected clustering

Projected clustering seeks to assign each point to a unique cluster, but clusters may exist in different subspaces. The general approach is to use a special distance function together with a regular clustering algorithm.

For example, the PreDeCon algorithm checks which attributes seem to support a clustering for each point, and adjusts the distance function such that dimensions with low variance are amplified in the distance function.*[6] In the figure above, the cluster c_c might be found using DBSCAN with a distance function that places less emphasis on the x -axis and thus exaggerates the low difference in the y -axis sufficiently enough to group the points into a cluster.

PROCLUS uses a similar approach with a k-medoid clustering.*[7] Initial medoids are guessed, and for each medoid the subspace spanned by attributes with low variance is determined. Points are assigned to the medoid closest, considering only the subspace of that medoid in determining the distance. The algorithm then proceeds as the regular PAM algorithm.

If the distance function weights attributes differently, but never with 0 (and hence never drops irrelevant attributes), the algorithm is called a *"soft"-projected clustering algorithm*.

Hybrid approaches

Not all algorithms try to either find a unique cluster assignment for each point or all clusters in all subspaces; many settle for a result in between, where a number of possibly overlapping, but not necessarily exhaustive set of clusters are found. An example is FIRES, which is from its basic approach a subspace clustering algorithm, but uses a heuristic too aggressive to credibly produce all subspace clusters.*[8]

Correlation clustering

Another type of subspaces is considered in Correlation clustering (Data Mining).

2.3.3 Software

- ELKI includes various subspace and correlation clustering algorithms

2.3.4 References

[1] Kriegel, H. P.; Kröger, P.; Zimek, A. (2009). "Clustering high-dimensional data" . *ACM Transactions on Knowledge Discovery from Data* **3**: 1. doi:10.1145/1497577.1497578.

[2] Houle, M. E.; Kriegel, H. P.; Kröger, P.; Schubert, E.; Zimek, A. (2010). *Can Shared-Neighbor Distances Defeat the Curse of Dimensionality?* (PDF). Scientific and Statistical Database Management. Lecture Notes in Computer Science **6187**. p. 482. doi:10.1007/978-3-642-13818-8_34. ISBN 978-3-642-13817-1.

[3] Agrawal, R.; Gehrke, J.; Gunopulos, D.; Raghavan, P. (2005). "Automatic Subspace Clustering of High Dimensional Data" . *Data Mining and Knowledge Discovery* **11**: 5. doi:10.1007/s10618-005-1396-1.

[4] Kailing, K.; Kriegel, H. P.; Kröger, P. (2004). *Density-Connected Subspace Clustering for High-Dimensional Data*. Proceedings of the 2004 SIAM International Conference on Data Mining. p. 246. doi:10.1137/1.9781611972740.23. ISBN 978-0-89871-568-2.

[5] Cordeiro De Amorim, R.; Mirkin, B. (2012). "Minkowski metric, feature weighting and anomalous cluster initializing in K-Means clustering" . *Pattern Recognition* **45** (3): 1061. doi:10.1016/j.patcog.2011.08.012.

[6] Böhm, C.; Kailing, K.; Kriegel, H. -P.; Kröger, P. (2004). *Density Connected Clustering with Local Subspace Preferences*. Fourth IEEE International Conference on Data Mining (ICDM'04). p. 27. doi:10.1109/ICDM.2004.10087. ISBN 0-7695-2142-8.

[7] Aggarwal, C. C.; Wolf, J. L.; Yu, P. S.; Procopiuc, C.; Park, J. S. (1999). "Fast algorithms for projected clustering" . *ACM SIGMOD Record* **28** (2): 61. doi:10.1145/304181.304188.

[8] Kriegel, H.; Kröger, P.; Renz, M.; Wurst, S. (2005). *A Generic Framework for Efficient Subspace Clustering of High-Dimensional Data*. Fifth IEEE International Conference on Data Mining (ICDM'05). p. 250. doi:10.1109/ICDM.2005.5. ISBN 0-7695-2278-5.

2.4 Canopy clustering algorithm

The **canopy clustering algorithm** is an unsupervised pre-clustering algorithm introduced by Andrew McCallum, Kamal Nigam and Lyle Ungar in 2000.*[1] It is often used as preprocessing step for the K-means algorithm or the Hierarchical clustering algorithm. It is intended to speed up clustering operations on large data sets, where using another algorithm directly may be impractical due to the size of the data set.

The algorithm proceeds as follows, using two thresholds T_1 (the loose distance) and T_2 (the tight distance), where $T_1 > T_2$.*[1]*[2]

1. Begin with the set of data points to be clustered.
2. Remove a point from the set, beginning a new 'canopy'.
3. For each point left in the set, assign it to the new canopy if the distance less than the loose distance T_1 .
4. If the distance of the point is additionally less than the tight distance T_2 , remove it from the original set.
5. Repeat from step 2 until there are no more data points in the set to cluster.
6. These relatively cheaply clustered canopies can be sub-clustered using a more expensive but accurate algorithm.

An important note is that individual data points may be part of several canopies. As an additional speed-up, an approximate and fast distance metric can be used for 3, where a more accurate and slow distance metric can be used for step 4.

Since the algorithm uses distance functions and requires the specification of distance thresholds, its applicability for high-dimensional data is limited by the curse of dimensionality. Only when a cheap and approximative – low-dimensional – distance function is available, the produced canopies will preserve the clusters produced by K-means.

2.4.1 Benefits

- The number of instances of training data that must be compared at each step is reduced
- There is some evidence that the resulting clusters are improved*[3]

2.4.2 References

[1] McCallum, A.; Nigam, K.; and Ungar L.H. (2000) "Efficient Clustering of High Dimensional Data Sets with Application to Reference Matching" , Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining, 169-178 doi:10.1145/347090.347123

[2] http://courses.cs.washington.edu/courses/cse590q/04au/slides/DannyMcCallumKDD00.ppt Retrieved 2014-09-06.

[3] Mahout description of Canopy-Clustering Retrieved 2011-04-02.

Chapter 3

Connectivity models

3.1 Single-linkage clustering

Single-linkage clustering is one of several methods of agglomerative hierarchical clustering. In the beginning of the process, each element is in a cluster of its own. The clusters are then sequentially combined into larger clusters, until all elements end up being in the same cluster. At each step, the two clusters separated by the shortest distance are combined. The definition of 'shortest distance' is what differentiates between the different agglomerative clustering methods. In single-linkage clustering, the link between two clusters is made by a single element pair, namely those two elements (one in each cluster) that are closest to each other. The shortest of these links that remains at any step causes the fusion of the two clusters whose elements are involved. The method is also known as **nearest neighbour clustering**. The result of the clustering can be visualized as a dendrogram, which shows the sequence of cluster fusion and the distance at which each fusion took place.*[1]

Mathematically, the linkage function – the distance $D(X,Y)$ between clusters X and Y – is described by the expression

$$D(X,Y) = \min_{x\in X, y\in Y} d(x,y),$$

where X and Y are any two sets of elements considered as clusters, and $d(x,y)$ denotes the distance between the two elements x and y.

A drawback of this method is the so-called *chaining phenomenon*, which refers to the gradual growth of a cluster as one element at a time gets added to it. This may lead to impractically heterogeneous clusters and difficulties in defining classes that could usefully subdivide the data.

3.1.1 Naive Algorithm

The following algorithm is an agglomerative scheme that erases rows and columns in a proximity matrix as old clusters are merged into new ones. The $N \times N$ proximity matrix D contains all distances $d(i,j)$. The clusterings are assigned sequence numbers 0,1,......, $(n-1)$ and $L(k)$ is the level of the kth clustering. A cluster with sequence number m is denoted (m) and the proximity between clusters (r) and (s) is denoted $d[(r),(s)]$.

The algorithm is composed of the following steps:

1. Begin with the disjoint clustering having level $L(0) = 0$ and sequence number m = 0.
2. Find the most similar pair of clusters in the current clustering, say pair (r), (s), according to $d[(r),(s)] = \min d[(i),(j)]$ where the minimum is over all pairs of clusters in the current clustering.
3. Increment the sequence number: $m = m + 1$. Merge clusters (r) and (s) into a single cluster to form the next clustering m. Set the level of this clustering to $L(m) = d[(r),(s)]$
4. Update the proximity matrix, D, by deleting the rows and columns corresponding to clusters (r) and (s) and adding a row and column corresponding to the newly formed cluster. The proximity between the new cluster, denoted (r,s) and old cluster (k) is defined as $d[(k), (r,s)] =$ **min** $\boldsymbol{d[(k),(r)], d[(k),(s)]}$.
5. If all objects are in one cluster, stop. Else, go to step 2.

3.1.2 Optimally efficient algorithm

The algorithm explained above is easy to understand but of complexity $\mathcal{O}(n^3)$. In 1973, R. Sibson proposed an optimally efficient algorithm of only complexity $\mathcal{O}(n^2)$ known as SLINK.*[2]

3.1.3 Other linkages

This is essentially the same as Kruskal's algorithm for minimum spanning trees. However, in single linkage clustering, the order in which clusters are formed is important, while for minimum spanning trees what matters is the set of pairs of points that form distances chosen by the algorithm.

Alternative linkage schemes include complete linkage and Average linkage clustering - implementing a different linkage in the naive algorithm is simply a matter of using a different formula to calculate inter-cluster distances in the initial computation of the proximity matrix and in step 4 of the above algorithm. An optimally efficient algorithm is however not available for arbitrary linkages. The formula that should be adjusted has been highlighted using bold text.

3.1.4 References

[1] Legendre, P. & Legendre, L. 1998. Numerical Ecology. Second English Edition. 853 pages.

[2] R. Sibson (1973). "SLINK: an optimally efficient algorithm for the single-link cluster method" (PDF). *The Computer Journal* (British Computer Society) **16** (1): 30–34. doi:10.1093/comjnl/16.1.30.

3.1.5 External links

- Single linkage clustering algorithm implementation in Ruby (AI4R)
- Linkages used in Matlab

3.2 Complete-linkage clustering

Complete-linkage clustering is one of several methods of agglomerative hierarchical clustering. At the beginning of the process, each element is in a cluster of its own. The clusters are then sequentially combined into larger clusters until all elements end up being in the same cluster. At each step, the two clusters separated by the shortest distance are combined. The definition of 'shortest distance' is what differentiates between the different agglomerative clustering methods. In complete-linkage clustering, the link between two clusters contains all element pairs, and the distance between clusters equals the distance between those two elements (one in each cluster) that are farthest away from each other. The shortest of these links that remains at any step causes the fusion of the two clusters whose elements are involved. The method is also known as **farthest neighbour clustering**. The result of the clustering can be visualized as a dendrogram, which shows the sequence of cluster fusion and the distance at which each fusion took place.*[1]*[2]*[3]

Mathematically, the complete linkage function —the distance $D(X,Y)$ between clusters X and Y —is described by the following expression : $D(X,Y) = \max_{x\in X, y\in Y} d(x,y)$

where

- $d(x,y)$ is the distance between elements $x \in X$ and $y \in Y$;
- X and Y are two sets of elements (clusters)

Complete linkage clustering avoids a drawback of the alternative single linkage method - the so-called *chaining phenomenon*, where clusters formed via single linkage clustering may be forced together due to single elements being close to each other, even though many of the elements in each cluster may be very distant to each other. Complete linkage tends to find compact clusters of approximately equal diameters.*[4]

3.2.1 Naive Algorithm

The following algorithm is an agglomerative scheme that erases rows and columns in a proximity matrix as old clusters are merged into new ones. The $N \times N$ proximity matrix *D* contains all distances *d*(*i*,*j*). The clusterings are assigned sequence numbers 0,1,......, (*n* − 1) and *L*(*k*) is the level of the kth clustering. A cluster with sequence number *m* is denoted (*m*) and the proximity between clusters (*r*) and (*s*) is denoted *d*[(*r*),(*s*)].

The algorithm is composed of the following steps:

1. Begin with the disjoint clustering having level *L*(0) = 0 and sequence number m = 0.
2. Find the most similar pair of clusters in the current clustering, say pair (r), (s), according to *d*[(*r*),(*s*)] = max *d*[(*i*),(*j*)] where the maximum is over all pairs of clusters in the current clustering.
3. Increment the sequence number: *m* = *m* + 1. Merge clusters (*r*) and (*s*) into a single cluster to form the next clustering *m*. Set the level of this clustering to *L*(*m*) = *d*[(*r*),(*s*)]
4. Update the proximity matrix, *D*, by deleting the rows and columns corresponding to clusters (*r*) and (*s*) and adding a row and column corresponding to the newly formed cluster. The proximity between the new cluster, denoted (*r*,*s*) and old cluster (*k*) is defined as *d*[(*k*), (*r*,*s*)] = **max *d*[(*k*),(*r*)], *d*[(*k*),(*s*)]**.
5. If all objects are in one cluster, stop. Else, go to step 2.

3.2.2 Optimally efficient algorithm

The algorithm explained above is easy to understand but of complexity $\mathcal{O}(n^3)$. In May 1976, D. Defays proposed an optimally efficient algorithm of only complexity $\mathcal{O}(n^2)$ known as CLINK (published 1977)*[5] inspired by the similar algorithm SLINK for single-linkage clustering.

3.2.3 Other linkages

Alternative linkage schemes include single linkage and average linkage clustering - implementing a different linkage in the naive algorithm is simply a matter of using a different formula to calculate inter-cluster distances in the initial computation of the proximity matrix and in step 4 of the above algorithm. An optimally efficient algorithm is however not available for arbitrary linkages. The formula that should be adjusted has been highlighted using bold text.

3.2.4 References

[1] T. Sorensen (1948). "A method of establishing groups of equal amplitude in plant sociology based on similarity of species and its application to analyses of the vegetation on Danish commons." . *Biologiske Skrifter* **5**: 1–34.

[2] Legendre, P. & Legendre, L. 1998. Numerical Ecology. Second English Edition. 853 pages.

[3] Brian S. Everitt; Sabine Landau; Morven Leese (2001). *Cluster Analysis* (Fourth ed.). London: Arnold. ISBN 0-340-76119-9.

[4] Everitt, Landau and Leese (2001), pp. 62-64.

[5] D. Defays (1977). "An efficient algorithm for a complete link method" (PDF). *The Computer Journal* (British Computer Society) **20** (4): 364–366. doi:10.1093/comjnl/20.4.364.

3.2.5 Other literature

- H. Späth (1980). *Cluster Analysis Algorithms*. Chichester: Ellis Horwood.

3.3 Nearest-neighbor chain algorithm

In the theory of cluster analysis, the **nearest-neighbor chain algorithm** is a method that can be used to perform several types of agglomerative hierarchical clustering, using an amount of memory that is linear in the number of points to be clustered and an amount of time linear in the number of distinct distances between pairs of points.*[1] The main idea of the algorithm is to find pairs of clusters to merge by following paths in the nearest neighbor graph of the clusters until the paths terminate in pairs of mutual nearest neighbors. The algorithm was developed and implemented in 1982 by J. P. Benzécri*[2] and J. Juan,*[3] based on earlier methods that constructed hierarchical clusterings using mutual nearest neighbor pairs without taking advantage of nearest neighbor chains.*[4]*[5]

3.3.1 Background

The input to a clustering problem consists of a set of points. A *cluster* is any proper subset of the points, and a hierarchical clustering is a maximal family of clusters with the property that any two clusters in the family are either nested or disjoint. Alternatively, a hierarchical clustering may be represented as a binary tree with the points at its leaves; the clusters of the clustering are the sets of points in subtrees descending from each node of the tree.

In agglomerative clustering methods, the input also includes a distance function defined on the points, or a numerical measure of their dissimilarity that is symmetric (insensitive to the ordering within each pair of points) but (unlike a distance) may not satisfy the triangle inequality. Depending on the method, this dissimilarity function can be extended in several different ways to pairs of clusters; for instance, in the single-linkage clustering method, the distance between two clusters is defined to be the minimum distance between any two points from each cluster. Given this distance between clusters, a hierarchical clustering may be defined by a greedy algorithm that initially places each point in its own single-point cluster and then repeatedly merges the closest pair of clusters.*[6]

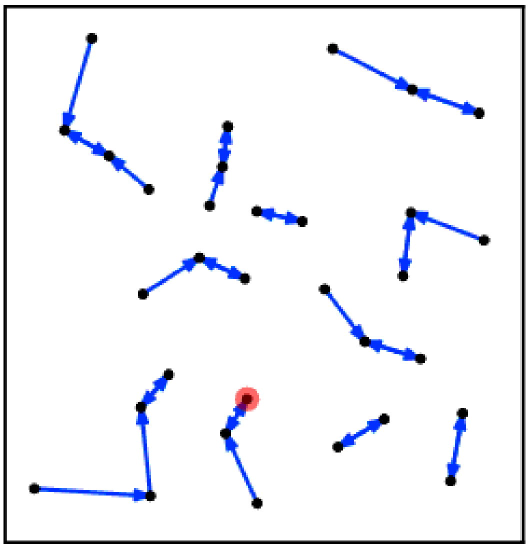
However, known methods for repeatedly finding the closest pair of clusters in a dynamic set of clusters either require superlinear space to maintain a data structure that can find closest pairs quickly, or they take greater than linear time to find each closest pair.*[7]*[8] The nearest-neighbor chain algorithm uses a smaller amount of time and space than the greedy algorithm by merging pairs of clusters in a different order. However, for many types of clustering problem, it can be guaranteed to come up with the same hierarchical clustering as the greedy algorithm despite the different merge order.

3.3.2 The algorithm

Intuitively, the nearest neighbor chain algorithm repeatedly follows a chain of clusters $A \to B \to C \to \ldots$ where each cluster is the nearest neighbor of the previous one, until reaching a pair of clusters that are mutual nearest neighbors.*[6]

More formally, the algorithm performs the following steps:*[1]*[6]

- Initialize the set of active clusters to consist of n one-point clusters, one for each input point.
- Let S be a stack data structure, initially empty, the elements of which will be active clusters.
- While there is more than one cluster in the set of clusters:
  - If S is empty, choose an active cluster arbitrarily and push it onto S.



Animation of the algorithm using Ward's distance. Black dots are points, grey regions are larger clusters, blue arrows point to nearest neighbors, and the red bar indicates the current chain. For visual simplicity, when a merge leaves the chain empty, it continues with the recently merged cluster.

- Let C be the active cluster on the top of S. Compute the distances from C to all other clusters, and let D be the nearest other cluster.
- If D is already in S, it must be the immediate predecessor of C. Pop both clusters from S, merge them, and push the merged cluster onto S.
- Otherwise, if D is not already in S, push it onto S.

If there may be multiple equal nearest neighbors to a cluster, the algorithm requires a consistent tie-breaking rule: for instance, in this case, the nearest neighbor may be chosen, among the clusters at equal minimum distance from C, by numbering the clusters arbitrarily and choosing the one with the smallest index.

3.3.3 Time and space analysis

Each iteration of the loop performs a single search for the nearest neighbor of a cluster, and either adds one cluster to the stack or removes two clusters from it. Every cluster is only ever added once to the stack, because when it is removed again it is immediately made inactive and merged. There are a total of $2n - 2$ clusters that ever get added to the stack: n single-point clusters in the initial set, and $n - 2$ internal nodes other than the root in the binary tree representing the clustering. Therefore, the algorithm performs $2n - 2$ pushing iterations and $n - 1$ popping iterations, each time scanning as many as $n - 1$ inter-cluster distances to find the nearest neighbor. The total number of distance calculations it makes is therefore less than $3n^2$, and the total time it uses outside of the distance calculations is $O(n^2)$.

Since the only data structure is the set of active clusters and the stack containing a subset of the active clusters, the space required is linear in the number of input points.

3.3.4 Correctness

The correctness of this algorithm relies on a property of its distance function called *reducibility*, identified by Bruynooghe (1977) in connection with an earlier clustering method that used mutual nearest neighbor pairs but not chains of nearest neighbors.*[4] A distance function d on clusters is defined to be reducible if, for every three clusters A, B and C in the greedy hierarchical clustering such that A and B are mutual nearest neighbors, the following inequality holds:

$$d(A \cup B, C) \geq \min(\mathrm{d}(A,C), \mathrm{d}(B,C)).$$

If a distance function has the reducibility property, then merging two clusters C and D can only cause the nearest neighbor of E to change if that nearest neighbor was one of C and D. This has two important consequences for the nearest neighbor chain algorithm: first, it can be shown using this property that, at each step of the algorithm, the clusters on the stack S form a valid chain of nearest neighbors, because whenever a nearest neighbor becomes invalidated it is immediately removed from the stack.

Second, and even more importantly, it follows from this property that, if two clusters C and D both belong to the greedy hierarchical clustering, and are mutual nearest neighbors at any point in time, then they will be merged by the greedy clustering, for they must remain mutual nearest neighbors until they are merged. It follows that each mutual nearest neighbor pair found by the nearest neighbor chain algorithm is also a pair of clusters found by the greedy algorithm, and therefore that the nearest neighbor chain algorithm computes exactly the same clustering (although in a different order) as the greedy algorithm.

3.3.5 Application to specific clustering distances

Ward's method

Ward's method is an agglomerative clustering method in which the dissimilarity between two clusters A and B is measured by the amount by which merging the two clusters into a single larger cluster would increase the average squared distance of a point to its cluster centroid.*[9] That is,

$$d(A,B)=\sum_{x\in A,y\in B}\frac{d^2(x,y)}{|A|+|B|}-\sum_{x,y\in A}\frac{d^2(x,y)}{|A|}-\sum_{x,y\in B}\frac{d^2(x,y)}{|B|}$$

Expressed in terms of the centroid c_A and cardinality n_A of the two clusters, it has the simpler formula

$$d(A,B)=\frac{d^2(c_a,c_b)}{1/n_A+1/n_B},$$

allowing it to be computed in constant time per distance calculation. Although highly sensitive to outliers, Ward's method is the most popular variation of agglomerative clustering both because of the round shape of the clusters it typically forms and because of its principled definition as the clustering that at each step has the smallest variance within its clusters.*[10] Alternatively, this distance can be seen as the difference in k-means cost between the new cluster and the two old clusters.

Ward's distance is also reducible, as can be seen more easily from a different formula of Lance–Williams type for calculating the distance of a merged cluster from the distances of the clusters it was merged from:*[9]*[11]

$$d(A\cup B,C)=\frac{n_A+n_C}{n_A+n_B+n_C}d(A,C)+\frac{n_B+n_C}{n_A+n_B+n_C}d(B,C)-\frac{n_C}{n_A+n_B+n_C}d(A,B).$$

If $d(A,B)$ is the smallest of the three distances on the right hand side (as would necessarily be true if A and B are mutual nearest-neighbors) then the negative contribution from its term is cancelled by the n_C coefficient of one of the two other terms, leaving a positive value added to the weighted average of the other two distances. Therefore, the combined distance is always at least as large as the minimum of $d(A,C)$ and $d(B,C)$, meeting the definition of reducibility.

Therefore, the nearest-neighbor chain algorithm using Ward's distance calculates exactly the same clustering as the standard greedy algorithm. For n points in a Euclidean space of constant dimension, it takes time $O(n^2)$ and space $O(n)$.*[1]

Complete linkage and average distance

Complete-linkage or furthest-neighbor clustering is a form of agglomerative clustering that uses the maximum distance between any two points from the two clusters as the dissimilarity, and similarly average-distance clustering uses the average pairwise distance. Like Ward's distance, these forms of clustering obey a formula of Lance-Williams type: in complete linkage, the distance $d(A\cup B,C)$ is the average of the distances $d(A,C)$ and $d(B,C)$ plus a positive correction term, while for average distance it is just a weighted average of the distances $d(A,C)$ and $d(B,C)$.*[9]*[11] Thus, in both of these cases, the distance is reducible.

Unlike Ward's method, these two forms of clustering do not have a constant-time method for computing distances between pairs of clusters. Instead it is possible to maintain an array of distances between all pairs of clusters, using the Lance–Williams formula to update the array as pairs of clusters are merged, in time and space $O(n^2)$. The nearest-neighbor chain algorithm may be used in conjunction with this array of distances to find the same clustering as the greedy algorithm for these cases in total time and space $O(n^2)$. The same $O(n^2)$ time and space bounds can also be achieved by a different and more general technique that overlays a quadtree-based priority queue data structure on top of the distance matrix and uses it to perform the standard greedy clustering algorithm, avoiding the need for reducibility,*[7] but the nearest-neighbor chain algorithm matches its time and space bounds while using simpler data structures.*[12]

Single linkage

In single-linkage or nearest-neighbor clustering, the oldest form of agglomerative hierarchical clustering,*[11] the dissimilarity between clusters is measured as the minimum distance between any two points from the two clusters. With this dissimilarity,

$$d(A\cup B,C)=\min(d(A,C),d(B,C)),$$

meeting as an equality rather than an inequality the requirement of reducibility. (Single-linkage also obeys a Lance–Williams formula,*[9]*[11] but with a negative coefficient from which it is more difficult to prove reducibility.)

As with complete linkage and average distance, the difficulty of calculating cluster distances causes the nearest-neighbor chain algorithm to take time and space $O(n^2)$ to compute the single-linkage clustering. However, the single-linkage clustering can be found more efficiently by an alternative algorithm that computes the minimum spanning tree of the input distances using Prim's algorithm (with an unsorted list of vertices and their priorities in place of the usual priority queue), and then sorts the minimum spanning tree edges and uses this sorted list to guide the merger of pairs of clusters. This alternative method would take time $O(n^2)$ and space $O(n)$, matching the best bounds that could be achieved with the nearest-neighbor chain algorithm for distances with constant-time calculations.*[13]

Centroid distance

Another distance measure commonly used in agglomerative clustering is the distance between the centroids of pairs of clusters, also known as the weighted group method.*[9]*[11] It can be calculated easily in constant time per distance calculation. However, it is not reducible: for instance, if the input forms the set of three points of an equilateral triangle, merging two of these points into a larger cluster causes the inter-cluster distance to decrease, a violation of reducibility. Therefore, the nearest-neighbor chain algorithm will not necessarily find the same clustering as the greedy algorithm. A different algorithm by Day and Edelsbrunner can be used to find the clustering in $O(n^2)$ time for this distance measure.*[8]

Distances sensitive to merge order

The above presentation explicitly disallowed distances sensitive to merge order; indeed, allowing such distances can cause problems. In particular, there exist order-sensitive cluster distances which satisfy reducibility, but the above algorithm will return a hierarchy with suboptimal costs.*[14] Following the earlier discussion of the value of defining cluster distances recursively (so that memoization can be used to greatly speed up distance computations), care must be taken with recursively defined distances so that they are not using the hierarchy in a way which is sensitive to merge order.

3.3.6 References

[1] Murtagh, Fionn (2002), "Clustering in massive data sets" , in Abello, James M.; Pardalos, Panos M.; Resende, Mauricio G. C., *Handbook of massive data sets*, Massive Computing **4**, Springer, pp. 513–516, ISBN 978-1-4020-0489-6.

[2] Benzécri, J.-P. (1982), "Construction d'une classification ascendante hiérarchique par la recherche en chaîne des voisins réciproques" , *Les Cahiers de l'Analyse des Données* **7** (2): 209–218.

[3] Juan, J. (1982), "Programme de classification hiérarchique par l'algorithme de la recherche en chaîne des voisins réciproques" , *Les Cahiers de l'Analyse des Données* **7** (2): 219–225.

[4] Bruynooghe, Michel (1977), "Méthodes nouvelles en classification automatique de données taxinomiqes nombreuses" , *Statistique et Analyse des Données* **3**: 24–42.

[5] de Rham, C. (1980), "La classification hiérarchique ascendante selon la méthode des voisins réciproques" , *Les Cahiers de l'Analyse des Données* **5** (2): 135–144.

[6] Murtagh, Fionn (1983), "A survey of recent advances in hierarchical clustering algorithms" (PDF), *The Computer Journal* **26** (4): 354–359, doi:10.1093/comjnl/26.4.354.

[7] Eppstein, David (2000), "Fast hierarchical clustering and other applications of dynamic closest pairs" , *J. Experimental Algorithmics* (ACM) **5** (1): 1–23, arXiv:cs.DS/9912014.

[8] Day, William H. E.; Edelsbrunner, Herbert (1984), "Efficient algorithms for agglomerative hierarchical clustering methods" (PDF), *Journal of Classification* **1** (1): 7–24, doi:10.1007/BF01890115.

[9] Mirkin, Boris (1996), *Mathematical classification and clustering*, Nonconvex Optimization and its Applications **11**, Dordrecht: Kluwer Academic Publishers, pp. 140–144, ISBN 0-7923-4159-7, MR 1480413.

[10] Tuffery, Stéphane (2011), "9.10 Agglomerative hierarchical clustering" , *Data Mining and Statistics for Decision Making*, Wiley Series in Computational Statistics, pp. 253–261, ISBN 978-0-470-68829-8.

[11] Lance, G. N.; Williams, W. T. (1967), "A general theory of classificatory sorting strategies. I. Hierarchical systems" , *The Computer Journal* **9** (4): 373–380, doi:10.1093/comjnl/9.4.373.

[12] Gronau, Ilan; Moran, Shlomo (2007), "Optimal implementations of UPGMA and other common clustering algorithms" , *Information Processing Letters* **104** (6): 205–210, doi:10.1016/j.ipl.2007.07.002, MR 2353367.

[13] Gower, J. C.; Ross, G. J. S. (1969), "Minimum spanning trees and single linkage cluster analysis" , *Journal of the Royal Statistical Society, Series C* **18** (1): 54–64, JSTOR 2346439, MR 0242315.

[14] Müllner, Daniel (2011), *Modern hierarchical, agglomerative clustering algorithms*, arXiv:1109.2378v1.

3.4 UPGMA

UPGMA (**U**nweighted **P**air **G**roup **M**ethod with **A**rithmetic Mean) is a simple agglomerative (bottom-up) hierarchical clustering method. It is one of the most popular methods in ecology for the classification of sampling units (such as vegetation plots) on the basis of their pairwise similarities in relevant descriptor variables (such as species composition).*[1] In bioinformatics, UPGMA is used for the creation of phenetic trees (phenograms). In a phylogenetic context, UPGMA assumes a constant rate of evolution (molecular clock hypothesis), and is not a well-regarded method for inferring relationships unless this assumption has been tested and justified for the data set being used. UPGMA was initially designed for use in protein electrophoresis studies, but is currently most often used to produce guide trees for more sophisticated phylogenetic reconstruction algorithms.

The UPGMA algorithm constructs a rooted tree (dendrogram) that reflects the structure present in a pairwise similarity matrix (or a dissimilarity matrix).

At each step, the nearest two clusters are combined into a higher-level cluster. The distance between any two clusters A and B is taken to be the average of all distances between pairs of objects "x" in A and "y" in B, that is, the mean distance between elements of each cluster:

$$\frac{1}{|\mathcal{A}| \cdot |\mathcal{B}|} \sum_{x \in \mathcal{A}} \sum_{y \in \mathcal{B}} d(x,y)$$

The method is generally attributed to Sokal and Michener.*[2] Fionn Murtagh found a time optimal $O(n^2)$ time algorithm to construct the UPGMA tree.*[3]

3.4.1 See also

- Neighbor-joining
- Cluster analysis
- Single-linkage clustering
- Complete-linkage clustering
- Hierarchical clustering
- Models of DNA evolution
- Molecular clock

3.4.2 References

[1] Legendre, P. and Legendre, L. 1998. Numerical Ecology. Second English Edition. Developments in Environmental Modelling 20. Elsevier, Amsterdam.

[2] Sokal R and Michener C (1958). “A statistical method for evaluating systematic relationships”. *University of Kansas Science Bulletin* **38**: 1409–1438.

[3] Murtagh F (1984). “Complexities of Hierarchic Clustering Algorithms: the state of the art” . *Computational Statistics Quarterly* **1**: 101–113.

3.4.3 External links

- UPGMA clustering algorithm implementation in Ruby (AI4R)
- Example calculation of UPGMA using a similarity matrix
- Example calculation of UPGMA using a distance matrix

3.5 BIRCH

This article is about the clustering algorithm. For the tree, see Birch. For other uses, see Birch (disambiguation).

BIRCH (balanced iterative reducing and clustering using hierarchies) is an unsupervised data mining algorithm used to perform hierarchical clustering over particularly large data-sets.*[1] An advantage of BIRCH is its ability to incrementally and dynamically cluster incoming, multi-dimensional metric data points in an attempt to produce the best quality clustering for a given set of resources (memory and time constraints). In most cases, BIRCH only requires a single scan of the database.

Its inventors claim BIRCH to be the “first clustering algorithm proposed in the database area to handle 'noise' (data points that are not part of the underlying pattern) effectively” ,*[1] beating DBSCAN by two months. The algorithm received the SIGMOD 10 year test of time award in 2006.*[2]

3.5.1 Problem with previous methods

Previous clustering algorithms performed less effectively over very large databases and did not adequately consider the case wherein a data-set was too large to fit in main memory. As a result, there was a lot of overhead maintaining high clustering quality while minimizing the cost of addition IO (input/output) operations. Furthermore, most of BIRCH's predecessors inspect all data points (or all currently existing clusters) equally for each 'clustering decision' and do not perform heuristic weighting based on the distance between these data points.

3.5.2 Advantages with BIRCH

It is local in that each clustering decision is made without scanning all data points and currently existing clusters. It exploits the observation that data space is not usually uniformly occupied and not every data point is equally important. It makes full use of available memory to derive the finest possible sub-clusters while minimizing I/O costs. It is also an incremental method that does not require the whole data set in advance.

3.5.3 Algorithm

The BIRCH algorithm takes as input a set of N data points, represented as real-valued vectors, and a desired number of clusters K. It operates in four phases, the second of which is optional.

The first phase builds a *CF tree* out of the data points, a height-balanced tree data structure, defined as follows:

- Given a set of N d-dimensional data points, the *clustering feature* CF of the set is defined as the triple $CF = (N, LS, SS)$, where LS is the linear sum and SS is the square sum of data points.
- Clustering features are organized in a *CF tree*, a height-balanced tree with two parameters: branching factor B and threshold T . Each non-leaf node contains at most B entries of the form

$[CF_i, child_i]$, where $child_i$ is a pointer to its i th child node and CF_i the clustering feature representing the associated subcluster. A leaf node contains at most L entries each of the form $[CF_i]$. It also has two pointers prev and next which are used to chain all leaf nodes together. The tree size depends on the parameter T. A node is required to fit in a page of size P. B and L are determined by P. So P can be varied for performance tuning. It is a very compact representation of the dataset because each entry in a leaf node is not a single data point but a subcluster.

In the second step, the algorithm scans all the leaf entries in the initial CF tree to rebuild a smaller CF tree, while removing outliers and grouping crowded subclusters into larger ones. This step is marked optional in the original presentation of BIRCH.

In step three an existing clustering algorithm is used to cluster all leaf entries. Here an agglomerative hierarchical clustering algorithm is applied directly to the subclusters represented by their CF vectors. It also provides the flexibility of allowing the user to specify either the desired number of clusters or the desired diameter threshold for clusters. After this step a set of clusters is obtained that captures major distribution pattern in the data. However there might exist minor and localized inaccuracies which can be handled by an optional step 4. In step 4 the centroids of the clusters produced in step 3 are used as seeds and redistribute the data points to its closest seeds to obtain a new set of clusters. Step 4 also provides us with an option of discarding outliers. That is a point which is too far from its closest seed can be treated as an outlier.

3.5.4 Notes

[1] Zhang, T.; Ramakrishnan, R.; Livny, M. (1996). "BIRCH: an efficient data clustering method for very large databases" . *Proceedings of the 1996 ACM SIGMOD international conference on Management of data - SIGMOD '96*. pp. 103—114. doi:10.1145/233269.233324.

[2] "2006 SIGMOD Test of Time Award" .

Chapter 4

Centroid models

4.1 k-nearest neighbors algorithm

In pattern recognition, the ***k*-Nearest Neighbors algorithm** (or ***k*-NN** for short) is a non-parametric method used for classification and regression.*[1] In both cases, the input consists of the k closest training examples in the feature space. The output depends on whether k-NN is used for classification or regression:

- In *k-NN classification*, the output is a class membership. An object is classified by a majority vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors (k is a positive integer, typically small). If $k = 1$, then the object is simply assigned to the class of that single nearest neighbor.

- In *k-NN regression*, the output is the property value for the object. This value is the average of the values of its k nearest neighbors.

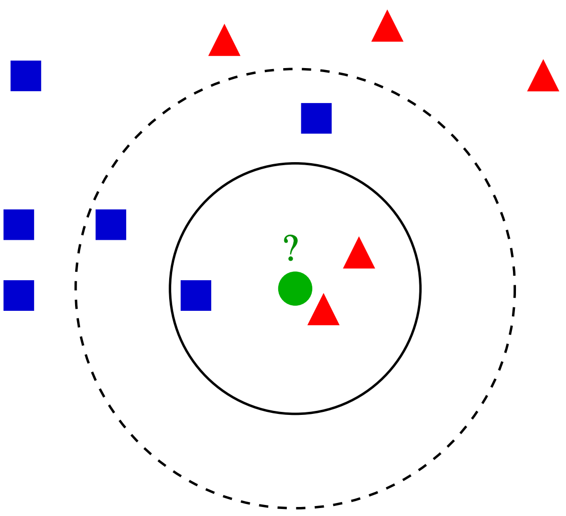
k-NN is a type of instance-based learning, or lazy learning, where the function is only approximated locally and all computation is deferred until classification. The k-NN algorithm is among the simplest of all machine learning algorithms.

Both for classification and regression, it can be useful to assign weight to the contributions of the neighbors, so that the nearer neighbors contribute more to the average than the more distant ones. For example, a common weighting scheme consists in giving each neighbor a weight of $1/d$, where d is the distance to the neighbor.*[2]

The neighbors are taken from a set of objects for which the class (for k-NN classification) or the object property value (for k-NN regression) is known. This can be thought of as the training set for the algorithm, though no explicit training step is required.

A shortcoming of the k-NN algorithm is that it is sensitive to the local structure of the data. The algorithm has nothing to do with and is not to be confused with k-means, another popular machine learning technique.

4.1.1 Algorithm



Example of k-*NN classification. The test sample (green circle) should be classified either to the first class of blue squares or to the second class of red triangles. If* k = 3 *(solid line circle) it is assigned to the second class because there are 2 triangles and only 1 square inside the inner circle. If* k = 5 *(dashed line circle) it is assigned to the first class (3 squares vs. 2 triangles inside the outer circle).*

The training examples are vectors in a multidimensional feature space, each with a class label. The training phase of the algorithm consists only of storing the feature vectors and class labels of the training samples.

In the classification phase, k is a user-defined constant, and an unlabeled vector (a query or test point) is classified by assigning the label which is most frequent among the k training samples nearest to that query point.

A commonly used distance metric for continuous variables is Euclidean distance. For discrete variables, such as for text classification, another metric can be used, such as the **overlap metric** (or Hamming distance). In the context of gene expression microarray data, for example, k-NN has also been employed with correlation coefficients such as Pearson and Spearman.*[3] Often, the classification accuracy of k-NN can be improved significantly if the distance metric is learned with specialized algorithms such as Large Margin Nearest Neighbor or

Neighbourhood components analysis.

A drawback of the basic "majority voting" classification occurs when the class distribution is skewed. That is, examples of a more frequent class tend to dominate the prediction of the new example, because they tend to be common among the *k* nearest neighbors due to their large number.*[4] One way to overcome this problem is to weight the classification, taking into account the distance from the test point to each of its *k* nearest neighbors. The class (or value, in regression problems) of each of the *k* nearest points is multiplied by a weight proportional to the inverse of the distance from that point to the test point. Another way to overcome skew is by abstraction in data representation. For example in a self-organizing map (SOM), each node is a representative (a center) of a cluster of similar points, regardless of their density in the original training data. *K*-NN can then be applied to the SOM.

4.1.2 Parameter selection

The best choice of *k* depends upon the data; generally, larger values of *k* reduce the effect of noise on the classification,*[5] but make boundaries between classes less distinct. A good *k* can be selected by various heuristic techniques (see hyperparameter optimization). The special case where the class is predicted to be the class of the closest training sample (i.e. when $k = 1$) is called the nearest neighbor algorithm.

The accuracy of the *k*-NN algorithm can be severely degraded by the presence of noisy or irrelevant features, or if the feature scales are not consistent with their importance. Much research effort has been put into selecting or scaling features to improve classification. A particularly popular approach is the use of evolutionary algorithms to optimize feature scaling.*[6] Another popular approach is to scale features by the mutual information of the training data with the training classes.

In binary (two class) classification problems, it is helpful to choose *k* to be an odd number as this avoids tied votes. One popular way of choosing the empirically optimal *k* in this setting is via bootstrap method.*[7]

4.1.3 Properties

k-NN is a special case of a variable-bandwidth, kernel density "balloon" estimator with a uniform kernel.*[8] *[9]

The naive version of the algorithm is easy to implement by computing the distances from the test example to all stored examples, but it is computationally intensive for large training sets. Using an appropriate nearest neighbor search algorithm makes *k*-NN computationally tractable even for large data sets. Many nearest neighbor search algorithms have been proposed over the years; these generally seek to reduce the number of distance evaluations actually performed.

k-NN has some strong consistency results. As the amount of data approaches infinity, the algorithm is guaranteed to yield an error rate no worse than twice the Bayes error rate (the minimum achievable error rate given the distribution of the data).*[10] *k*-NN is guaranteed to approach the Bayes error rate for some value of *k* (where *k* increases as a function of the number of data points). Various improvements to *k*-NN are possible by using proximity graphs.*[11]

4.1.4 Metric Learning

The K-nearest neighbor classification performance can often be significantly improved through (supervised) metric learning. Popular algorithms are Neighbourhood components analysis and Large margin nearest neighbor. Supervised metric learning algorithms use the label information to learn a new metric or pseudo-metric.

4.1.5 Feature extraction

When the input data to an algorithm is too large to be processed and it is suspected to be notoriously redundant (e.g. the same measurement in both feet and meters) then the input data will be transformed into a reduced representation set of features (also named features vector). Transforming the input data into the set of features is called feature extraction. If the features extracted are carefully chosen it is expected that the features set will extract the relevant information from the input data in order to perform the desired task using this reduced representation instead of the full size input. Feature extraction is performed on raw data prior to applying *k*-NN algorithm on the transformed data in feature space.

An example of a typical computer vision computation pipeline for face recognition using *k*-NN including feature extraction and dimension reduction pre-processing steps (usually implemented with OpenCV):

1. Haar face detection
2. Mean-shift tracking analysis
3. PCA or Fisher LDA projection into feature space, followed by *k*-NN classification

4.1.6 Dimension reduction

For high-dimensional data (e.g., with number of dimensions more than 10) dimension reduction is usually performed prior to applying the *k*-NN algorithm in order to avoid the effects of the curse of dimensionality. *[12]

The curse of dimensionality in the k-NN context basically means that Euclidean distance is unhelpful in high dimensions because all vectors are almost equidistant to the search query vector (imagine multiple points lying more or less on a circle with the query point at the center; the distance from the query to all data points in the search space is almost the same).

Feature extraction and dimension reduction can be combined in one step using principal component analysis (PCA), linear discriminant analysis (LDA), or canonical correlation analysis (CCA) techniques as a pre-processing step, followed by clustering by k-NN on feature vectors in reduced-dimension space. In machine learning this process is also called low-dimensional embedding.*[13]

For very-high-dimensional datasets (e.g. when performing a similarity search on live video streams, DNA data or high-dimensional time series) running a fast **approximate** k-NN search using locality sensitive hashing, "random projections",*[14] "sketches" *[15] or other high-dimensional similarity search techniques from VLDB toolbox might be the only feasible option.

4.1.7 Decision boundary

Nearest neighbor rules in effect implicitly compute the decision boundary. It is also possible to compute the decision boundary explicitly, and to do so efficiently, so that the computational complexity is a function of the boundary complexity.*[16]

4.1.8 Data reduction

Data reduction is one of the most important problems for work with huge data sets. Usually, only some of the data points are needed for accurate classification. Those data are called the *prototypes* and can be found as follows:

1. Select the *class-outliers*, that is, training data that are classified incorrectly by k-NN (for a given k)
2. Separate the rest of the data into two sets: (i) the prototypes that are used for the classification decisions and (ii) the *absorbed points* that can be correctly classified by k-NN using prototypes. The absorbed points can then be removed from the training set.

Selection of class-outliers

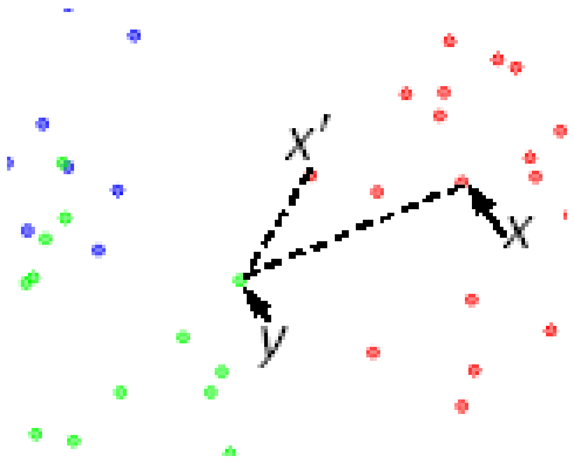
A training example surrounded by examples of other classes is called a class outlier. Causes of class outliers include:

- random error
- insufficient training examples of this class (an isolated example appears instead of a cluster)
- missing important features (the classes are separated in other dimensions which we do not know)
- too many training examples of other classes (unbalanced classes) that create a "hostile" background for the given small class

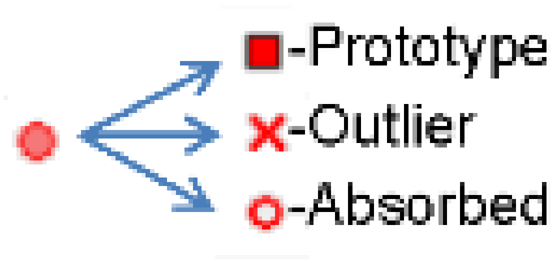
Class outliers with k-NN produce noise. They can be detected and separated for future analysis. Given two natural numbers, $k>r>0$, a training example is called a (k,r)NN class-outlier if its k nearest neighbors include more than r examples of other classes.

CNN for data reduction

Condensed nearest neighbor (CNN, the *Hart algorithm*) is an algorithm designed to reduce the data set for k-NN classification.*[17] It selects the set of prototypes U from the training data, such that 1NN with U can classify the examples almost as accurately as 1NN does with the whole data set.



Calculation of the border ratio.



Three types of points: prototypes, class-outliers, and absorbed points.

Given a training set X, CNN works iteratively:

1. Scan all elements of X, looking for an element x whose nearest prototype from U has a different label than x.

2. Remove x from X and add it to U
3. Repeat the scan until no more prototypes are added to U.

Use U instead of X for classification. The examples that are not prototypes are called "absorbed" points.

It is efficient to scan the training examples in order of decreasing border ratio.*[18] The border ratio of a training example x is defined as

$$a(x) = \|x'-y\| / \|x-y\|$$

where $\|x-y\|$ is the distance to the closest example y having a different color than x, and $\|x'-y\|$ is the distance from y to its closest example x' with the same label as x.

The border ratio is in the interval [0,1] because $\|x'-y\|$ never exceeds $\|x-y\|$. This ordering gives preference to the borders of the classes for inclusion in the set of prototypes$U$. A point of a different label than x is called external to x. The calculation of the border ratio is illustrated by the figure on the right. The data points are labeled by colors: the initial point is x and its label is red. External points are blue and green. The closest to x external point is y. The closest to y red point is x'. The border ratio $a(x)=\|x'-y\|/\|x-y\|$ is the attribute of the initial point x.

Below is an illustration of CNN in a series of figures. There are three classes (red, green and blue). Fig. 1: initially there are 60 points in each class. Fig. 2 shows the 1NN classification map: each pixel is classified by 1NN using all the data. Fig. 3 shows the 5NN classification map. White areas correspond to the unclassified regions, where 5NN voting is tied (for example, if there are two green, two red and one blue points among 5 nearest neighbors). Fig. 4 shows the reduced data set. The crosses are the class-outliers selected by the (3,2)NN rule (all the three nearest neighbors of these instances belong to other classes); the squares are the prototypes, and the empty circles are the absorbed points. The left bottom corner shows the numbers of the class-outliers, prototypes and absorbed points for all three classes. The number of prototypes varies from 15% to 20% for different classes in this example. Fig. 5 shows that the 1NN classification map with the prototypes is very similar to that with the initial data set. The figures were produced using the Mirkes applet.*[18]

- CNN model reduction for k-NN classifiers
- Fig. 1. The dataset.
- Fig. 2. The 1NN classification map.
- Fig. 3. The 5NN classification map.
- Fig. 4. The CNN reduced dataset.
- Fig. 5. The 1NN classification map based on the CNN extracted prototypes.

4.1.9 *k*-NN regression

In k-NN regression, the k-NN algorithm is used for estimating continuous variables. One such algorithm uses a weighted average of the k nearest neighbors, weighted by the inverse of their distance. This algorithm works as follows:

1. Compute the Euclidean or Mahalanobis distance from the query example to the labeled examples.
2. Order the labeled examples by increasing distance.
3. Find a heuristically optimal number k of nearest neighbors, based on RMSE. This is done using cross validation.
4. Calculate an inverse distance weighted average with the k-nearest multivariate neighbors.

4.1.10 Validation of results

A confusion matrix or "matching matrix" is often used as a tool to validate the accuracy of k-NN classification. More robust statistical methods such as likelihood-ratio test can also be applied.

4.1.11 See also

- Instance-based learning
- Nearest neighbor search
- Statistical classification
- Cluster analysis
- Data mining
- Nearest centroid classifier
- Pattern recognition
- Curse of dimensionality
- Dimension reduction
- Principal Component Analysis
- Locality Sensitive Hashing
- MinHash
- Cluster hypothesis
- Closest pair of points problem

4.1.12 References

[1] Altman, N. S. (1992). “An introduction to kernel and nearest-neighbor nonparametric regression”. *The American Statistician* **46** (3): 175–185. doi:10.1080/00031305.1992.10475879.

[2] This scheme is a generalization of linear interpolation.

[3] Jaskowiak, P. A.; Campello, R. J. G. B. “Comparing Correlation Coefficients as Dissimilarity Measures for Cancer Classification in Gene Expression Data”. *http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.208.993". Brazilian Symposium on Bioinformatics (BSB 2011). pp. 1–8. Retrieved 16 October 2014.*

[4] D. Coomans; D.L. Massart (1982). “Alternative k-nearest neighbour rules in supervised pattern recognition : Part 1. k-Nearest neighbour classification by using alternative voting rules”. *Analytica Chimica Acta* **136**: 15–27. doi:10.1016/S0003-2670(01)95359-0.

[5] Everitt, B. S., Landau, S., Leese, M. and Stahl, D. (2011) Miscellaneous Clustering Methods, in Cluster Analysis, 5th Edition, John Wiley & Sons, Ltd, Chichester, UK.

[6] Nigsch F, Bender A, van Buuren B, Tissen J, Nigsch E, Mitchell JB (2006). “Melting point prediction employing k-nearest neighbor algorithms and genetic parameter optimization”. *Journal of Chemical Information and Modeling* **46** (6): 2412–2422. doi:10.1021/ci060149f. PMID 17125183.

[7] Hall P, Park BU, Samworth RJ (2008). “Choice of neighbor order in nearest-neighbor classification”. *Annals of Statistics* **36** (5): 2135–2152. doi:10.1214/07-AOS537.

[8] D. G. Terrell; D. W. Scott (1992). “Variable kernel density estimation”. *Annals of Statistics* **20** (3): 1236–1265. doi:10.1214/aos/1176348768.

[9] Mills, Peter. “Efficient statistical classification of satellite measurements”. *International Journal of Remote Sensing*.

[10] Cover TM, Hart PE (1967). “Nearest neighbor pattern classification”. *IEEE Transactions on Information Theory* **13** (1): 21–27. doi:10.1109/TIT.1967.1053964.

[11] Toussaint GT (April 2005). “Geometric proximity graphs for improving nearest neighbor methods in instance-based learning and data mining”. *International Journal of Computational Geometry and Applications* **15** (2): 101–150. doi:10.1142/S0218195905001622.

[12] Beyer, Kevin, et al.. 'When is “nearest neighbor” meaningful?. Database Theory—ICDT’99, 217-235|year 1999

[13] Shaw, Blake, and Tony Jebara. 'Structure preserving embedding. Proceedings of the 26th Annual International Conference on Machine Learning. ACM,2009

[14] Bingham, Ella, and Heikki Mannila. Random projection in dimensionality reduction: applications to image and text data. Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining. ACM | year 2001

[15] Shasha, D High Performance Discovery in Time Series.Berlin: Springer, 2004, ISBN 0-387-00857-8

[16] Bremner D, Demaine E, Erickson J, Iacono J, Langerman S, Morin P, Toussaint G (2005). “Output-sensitive algorithms for computing nearest-neighbor decision boundaries”. *Discrete and Computational Geometry* **33** (4): 593–604. doi:10.1007/s00454-004-1152-0.

[17] P. E. Hart, The Condensed Nearest Neighbor Rule. IEEE Transactions on Information Theory 18 (1968) 515–516. doi: 10.1109/TIT.1968.1054155

[18] E. M. Mirkes, KNN and Potential Energy: applet. University of Leicester, 2011.

4.1.13 Further reading

- When Is “Nearest Neighbor” Meaningful?
- Belur V. Dasarathy, ed. (1991). *Nearest Neighbor (NN) Norms: NN Pattern Classification Techniques*. ISBN 0-8186-8930-7.
- Shakhnarovish, Darrell, and Indyk, ed. (2005). *Nearest-Neighbor Methods in Learning and Vision*. MIT Press. ISBN 0-262-19547-X.
- Mäkelä H Pekkarinen A (2004-07-26). “Estimation of forest stand volumes by Landsat TM imagery and stand-level field-inventory data”. *Forest Ecology and Management* **196** (2–3): 245–255. doi:10.1016/j.foreco.2004.02.049.
- Fast k nearest neighbor search using GPU. In Proceedings of the CVPR Workshop on Computer Vision on GPU, Anchorage, Alaska, USA, June 2008. V. Garcia and E. Debreuve and M. Barlaud.
- Scholarpedia article on *k*-NN
- google-all-pairs-similarity-search

4.2 k-means clustering

***k*-means clustering** is a method of vector quantization, originally from signal processing, that is popular for cluster analysis in data mining. *k*-means clustering aims to partition *n* observations into *k* clusters in which each observation belongs to the cluster with the nearest mean, serving as a prototype of the cluster. This results in a partitioning of the data space into Voronoi cells.

The problem is computationally difficult (NP-hard); however, there are efficient heuristic algorithms that are commonly employed and converge quickly to a local optimum. These are usually similar to the expectation-maximization algorithm for mixtures of Gaussian distributions via an iterative refinement approach employed by both algorithms. Additionally, they both use cluster centers to model the data; however, *k*-means clustering tends to find clusters of comparable spatial extent, while the expectation-maximization mechanism allows clusters to have different shapes.

The algorithm has nothing to do with and should not be confused with k-nearest neighbor, another popular machine learning technique.

4.2.1 Description

Given a set of observations ($\mathbf{x}_1$, $\mathbf{x}_2$, $\cdots$, $\mathbf{x}_n$), where each observation is a d-dimensional real vector, k-means clustering aims to partition the n observations into k ($\leq n$) sets $\mathbf{S} = \{S_1, S_2, \cdots, S_k\}$ so as to minimize the within-cluster sum of squares (WCSS). In other words, its objective is to find:

$$\underset{\mathbf{S}}{\arg\min} \sum_{i=1}^{k} \sum_{\mathbf{x}\in S_i} \left\|\mathbf{x} - \boldsymbol{\mu}_i\right\|^2$$

where $\boldsymbol{\mu}_i$ is the mean of points in S_i.

4.2.2 History

The term "k-means" was first used by James MacQueen in 1967,[1] though the idea goes back to Hugo Steinhaus in 1957.[2] The standard algorithm was first proposed by Stuart Lloyd in 1957 as a technique for pulse-code modulation, though it wasn't published outside of Bell Labs until 1982.[3] In 1965, E.W.Forgy published essentially the same method, which is why it is sometimes referred to as Lloyd-Forgy.[4] A more efficient version was proposed and published in Fortran by Hartigan and Wong in 1975/1979.[5][6]

4.2.3 Algorithms

Standard algorithm

The most common algorithm uses an iterative refinement technique. Due to its ubiquity it is often called the ***k*-means algorithm**; it is also referred to as **Lloyd's algorithm**, particularly in the computer science community.

Given an initial set of k means $m_1{}^{*}(1)$,$\cdots$,$m_k{}^{*}(1)$ (see below), the algorithm proceeds by alternating between two steps:[7]

Assignment step: Assign each observation to the cluster whose mean yields the least within-cluster sum of squares (WCSS). Since the sum of squares is the squared Euclidean distance, this is intuitively the "nearest" mean.[8] (Mathematically, this means partitioning the observations according to the Voronoi diagram generated by the means).

$S_i^{(t)} = \big\{x_p : \big\|x_p - m_i^{(t)}\big\|^2 \leq \big\|x_p - m_j^{(t)}\big\|^2 \ \forall j, 1 \leq j \leq k\big\}$,

where each x_p is assigned to exactly one $S^{(t)}$, even if it could be assigned to two or more of them.

Update step: Calculate the new means to be the centroids of the observations in the new clusters.

$m_i^{(t+1)} = \frac{1}{|S_i^{(t)}|} \sum_{x_j \in S_i^{(t)}} x_j$

Since the arithmetic mean is a least-squares estimator, this also minimizes the within-cluster sum of squares (WCSS) objective.

The algorithm has converged when the assignments no longer change. Since both steps optimize the WCSS objective, and there only exists a finite number of such partitionings, the algorithm must converge to a (local) optimum. There is no guarantee that the global optimum is found using this algorithm.

The algorithm is often presented as assigning objects to the nearest cluster by distance. The standard algorithm aims at minimizing the WCSS objective, and thus assigns by "least sum of squares", which is exactly equivalent to assigning by the smallest Euclidean distance. Using a different distance function other than (squared) Euclidean distance may stop the algorithm from converging. Various modifications of k-means such as spherical k-means and k-medoids have been proposed to allow using other distance measures.

Initialization methods Commonly used initialization methods are Forgy and Random Partition.[9] The Forgy method randomly chooses k observations from the data set and uses these as the initial means. The Random Partition method first randomly assigns a cluster to each observation and then proceeds to the update step, thus computing the initial mean to be the centroid of the cluster's randomly assigned points. The Forgy method tends to spread the initial means out, while Random Partition places all of them close to the center of the data set. According to Hamerly et al.,[9] the Random Partition method is generally preferable for algorithms such as the k-harmonic means and fuzzy k-means. For expectation maximization and standard k-means algorithms, the Forgy method of initialization is preferable.

- Demonstration of the standard algorithm
- 1. k initial "means" (in this case k=3) are randomly generated within the data domain (shown in color).
- 2. k clusters are created by associating every observation with the nearest mean. The partitions here represent the Voronoi diagram generated by the means.

- 3. The centroid of each of the *k* clusters becomes the new mean.
- 4. Steps 2 and 3 are repeated until convergence has been reached.

As it is a heuristic algorithm, there is no guarantee that it will converge to the global optimum, and the result may depend on the initial clusters. As the algorithm is usually very fast, it is common to run it multiple times with different starting conditions. However, in the worst case, *k*-means can be very slow to converge: in particular it has been shown that there exist certain point sets, even in 2 dimensions, on which *k*-means takes exponential time, that is 2*Ω(n), to converge.*[10] These point sets do not seem to arise in practice: this is corroborated by the fact that the smoothed running time of *k*-means is polynomial.*[11]

The "assignment" step is also referred to as **expectation step**, the "update step" as **maximization step**, making this algorithm a variant of the *generalized* expectation-maximization algorithm.

Complexity

Regarding computational complexity, finding the optimal solution to the *k*-means clustering problem for observations in *d* dimensions is:

- NP-hard in general Euclidean space *d* even for 2 clusters *[12]*[13]
- NP-hard for a general number of clusters *k* even in the plane *[14]
- If *k* and *d* (the dimension) are fixed, the problem can be exactly solved in time $O(n^{dk+1}\log n)$, where *n* is the number of entities to be clustered *[15]

Thus, a variety of heuristic algorithms such as Lloyds algorithm given above are generally used.

The running time of Lloyds algorithm is often given as $O(nkdi)$, where *n* is the number of *d*-dimensional vectors, *k* the number of clusters and *i* the number of iterations needed until convergence. On data that does have a clustering structure, the number of iterations until convergence is often small, and results only improve slightly after the first dozen iterations. Lloyds algorithm is therefore often considered to be of "linear" complexity in practice.

Following are some recent insights into this algorithm complexity behaviour.

- Lloyd's *k*-means algorithm has polynomial smoothed running time. It is shown that *[11] for arbitrary set of *n* points in $[0,1]^d$, if each point is independently perturbed by a normal distribution with mean 0 and variance σ^2 , then the expected running time of k -means algorithm is bounded by $O(n^{34}k^{34}d^8log^4(n)/\sigma^6)$, which is a polynomial in n , k , d and $1/\sigma$.
- Better bounds are proved for simple cases. For example,*[16] showed that the running time of *k*-means algorithm is bounded by $O(dn^4M^2)$ for n points in an integer lattice $\{1,\dots,M\}^d$.

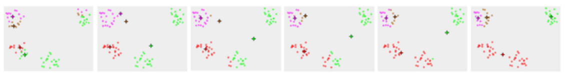
Variations

- Jenks natural breaks optimization: *k*-means applied to univariate data
- k-medians clustering uses the median in each dimension instead of the mean, and this way minimizes L_1 norm (Taxicab geometry).
- k-medoids (also: Partitioning Around Medoids, PAM) uses the medoid instead of the mean, and this way minimizes the sum of distances for *arbitrary* distance functions.
- Fuzzy C-Means Clustering is a soft version of K-means, where each data point has a fuzzy degree of belonging to each cluster.
- Gaussian mixture models trained with expectation-maximization algorithm (EM algorithm) maintains probabilistic assignments to clusters, instead of deterministic assignments, and multivariate Gaussian distributions instead of means.
- k-means++ chooses initial centers in a way that gives a provable upper bound on the WCCS objective.
- The filtering algorithm uses kd-trees to speed up each k-means step.*[17]
- Some methods attempt to speed up each k-means step using coresets*[18] or the triangle inequality.*[19]
- Escape local optima by swapping points between clusters.*[6]
- The Spherical k-means clustering algorithm is suitable for directional data.*[20]
- The Minkowski metric weighted k-means deals with irrelevant features by assigning cluster specific weights to each feature*[21]

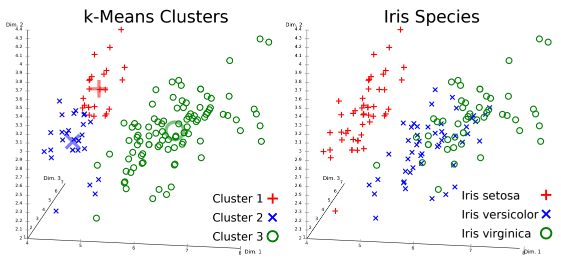
4.2.4 Discussion

The two key features of *k*-means which make it efficient are often regarded as its biggest drawbacks:

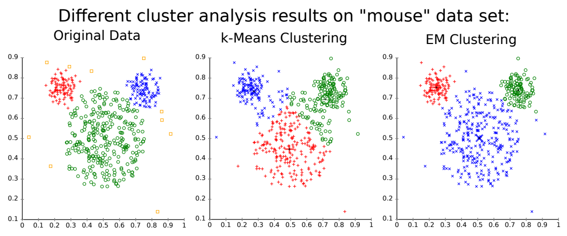
- Euclidean distance is used as a metric and variance is used as a measure of cluster scatter.



A typical example of the k-means convergence to a local minimum. In this example, the result of k-means clustering (the right figure) contradicts the obvious cluster structure of the data set. The small circles are the data points, the four ray stars are the centroids (means). The initial configuration is on the left figure. The algorithm converges after five iterations presented on the figures, from the left to the right. The illustration was prepared with the Mirkes Java applet.[*][22]



k-*means clustering result for the* Iris flower data set *and actual species visualized using* ELKI*. Cluster means are marked using larger, semi-transparent symbols.*



k-*means clustering and* EM clustering *on an artificial dataset ("mouse"). The tendency of* k-*means to produce equi-sized clusters leads to bad results, while EM benefits from the Gaussian distribution present in the data set*

- The number of clusters k is an input parameter: an inappropriate choice of k may yield poor results. That is why, when performing k-means, it is important to run diagnostic checks for determining the number of clusters in the data set.
- Convergence to a local minimum may produce counterintuitive ("wrong") results (see example in Fig.).

A key limitation of k-means is its cluster model. The concept is based on spherical clusters that are separable in a way so that the mean value converges towards the cluster center. The clusters are expected to be of similar size, so that the assignment to the nearest cluster center is the correct assignment. When for example applying k-means with a value of $k = 3$ onto the well-known Iris flower data set, the result often fails to separate the three Iris species contained in the data set. With $k = 2$, the two visible clusters (one containing two species) will be discovered, whereas with $k = 3$ one of the two clusters will be split into two even parts. In fact, $k = 2$ is more appropriate for this data set, despite the data set containing 3 *classes*. As with any other clustering algorithm, the k-means result relies on the data set to satisfy the assumptions made by the clustering algorithms. It works well on some data sets, while failing on others.

The result of k-means can also be seen as the Voronoi cells of the cluster means. Since data is split halfway between cluster means, this can lead to suboptimal splits as can be seen in the "mouse" example. The Gaussian models used by the Expectation-maximization algorithm (which can be seen as a generalization of k-means) are more flexible here by having both variances and covariances. The EM result is thus able to accommodate clusters of variable size much better than k-means as well as correlated clusters (not in this example).

4.2.5 Applications

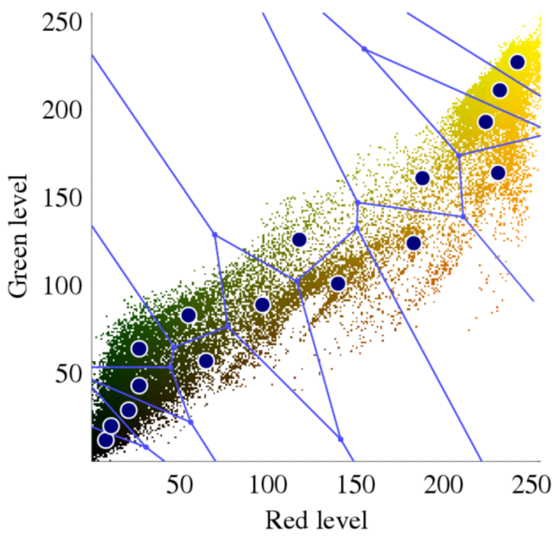
k-means clustering in particular when using heuristics such as Lloyd's algorithm is rather easy to implement and apply even on large data sets. As such, it has been successfully used in various topics, including market segmentation, computer vision, geostatistics,[*][23] astronomy and agriculture. It often is used as a preprocessing step for other algorithms, for example to find a starting configuration.

Vector quantization

Main article: Vector quantization
k-means originates from signal processing, and still finds

Two-channel (for illustration purposes -- red and green only) color image.

use in this domain. For example in computer graphics, color quantization is the task of reducing the color palette



Vector quantization of colors present in the image above into Voronoi cells using k*-means.*

of an image to a fixed number of colors k. The k-means algorithm can easily be used for this task and produces competitive results. Other uses of vector quantization include non-random sampling, as k-means can easily be used to choose k different but prototypical objects from a large data set for further analysis.

Cluster analysis

Main article: Cluster analysis

In cluster analysis, the k-means algorithm can be used to partition the input data set into k partitions (clusters).

However, the pure k-means algorithm is not very flexible, and as such of limited use (except for when vector quantization as above is actually the desired use case!). In particular, the parameter k is known to be hard to choose (as discussed above) when not given by external constraints. Another limitation of the algorithm is that it cannot be used with arbitrary distance functions or on non-numerical data. For these use cases, many other algorithms have been developed since.

Feature learning

k-means clustering has been used as a feature learning (or dictionary learning) step, in either (semi-)supervised learning or unsupervised learning.*[24] The basic approach is first to train a k-means clustering representation, using the input training data (which need not be labelled). Then, to project any input datum into the new feature space, we have a choice of "encoding" functions, but we can use for example the thresholded matrix-product of the datum with the centroid locations, the distance from the datum to each centroid, or simply an indicator function for the nearest centroid,*[24]*[25] or some smooth transformation of the distance.*[26] Alternatively, by transforming the sample-cluster distance through a Gaussian RBF, one effectively obtains the hidden layer of a radial basis function network.*[27]

This use of k-means has been successfully combined with simple, linear classifiers for semi-supervised learning in NLP (specifically for named entity recognition)*[28] and in computer vision. On an object recognition task, it was found to exhibit comparable performance with more sophisticated feature learning approaches such as autoencoders and restricted Boltzmann machines.*[26] However, it generally requires more data than the sophisticated methods, for equivalent performance, because each data point only contributes to one "feature" rather than multiple.*[24]

4.2.6 Relation to other statistical machine learning algorithms

k-means clustering, and its associated expectation-maximization algorithm, is a special case of a Gaussian mixture model, specifically, the limit of taking all covariances as diagonal, equal, and small. It is often easy to generalize a k-means problem into a Gaussian mixture model.*[29] Another generalization of the k-means algorithm is the K-SVD algorithm, which estimates data points as a sparse linear combination of "codebook vectors". K-means corresponds to the special case of using a single codebook vector, with a weight of 1.*[30]

Mean shift clustering

Basic mean shift clustering algorithms maintain a set of data points the same size as the input data set. Initially, this set is copied from the input set. Then this set is iteratively replaced by the mean of those points in the set that are within a given distance of that point. By contrast, k-means restricts this updated set to k points usually much less than the number of points in the input data set, and replaces each point in this set by the mean of all points in the *input set* that are closer to that point than any other (e.g. within the Voronoi partition of each updating point). A mean shift algorithm that is similar then to k-means, called *likelihood mean shift*, replaces the set of points undergoing replacement by the mean of all points in the input set that are within a given distance of the changing set.*[31] One of the advantages of mean shift over k-means is that there is no need to choose the number of clusters, because mean shift is likely to find only a few clusters if indeed only a small number exist. However, mean shift can be much slower than k-means, and still requires selection of a bandwidth parameter. Mean shift has soft variants much as k-means does.

Principal component analysis (PCA)

It was asserted in [32][33] that the relaxed solution of k-means clustering, specified by the cluster indicators, is given by the PCA (principal component analysis) principal components, and the PCA subspace spanned by the principal directions is identical to the cluster centroid subspace. However, that PCA is a useful relaxation of k-means clustering was not a new result (see, for example,[34]), and it is straightforward to uncover counterexamples to the statement that the cluster centroid subspace is spanned by the principal directions.[35]

Independent component analysis (ICA)

It has been shown in [36] that under sparsity assumptions and when input data is pre-processed with the whitening transformation *k*-means produces the solution to the linear Independent component analysis task. This aids in explaining the successful application of *k*-means to feature learning.

Bilateral filtering

k-means implicitly assumes that the ordering of the input data set does not matter. The bilateral filter is similar to K-means and mean shift in that it maintains a set of data points that are iteratively replaced by means. However, the bilateral filter restricts the calculation of the (kernel weighted) mean to include only points that are close in the ordering of the input data.[31] This makes it applicable to problems such as image denoising, where the spatial arrangement of pixels in an image is of critical importance.

4.2.7 Similar problems

The set of squared error minimizing cluster functions also includes the k-medoids algorithm, an approach which forces the center point of each cluster to be one of the actual points, i.e., it uses medoids in place of centroids.

4.2.8 Software Implementations

Free

- Apache Mahout
- Apache Spark MLlib implements a *k*-means algorithm.
- CrimeStat implements two spatial *k*-means algorithms, one of which allows the user to define the starting locations.
- ELKI contains *k*-means (with Lloyd and MacQueen iteration, along with different initializations such as *k*-means++ initialization) and various more advanced clustering algorithms
- Julia contains a *k*-means implementation in the Clustering package[37]
- MLPACK contains a C++ implementation of *k*-means
- R [1][3][6]
- SciPy
- Torch contains an *unsup* package that provides *k*-means clustering.
- Weka contains *k*-means and a few variants of it, including *k*-means++ and *x*-means.
- OpenCV contains a *k*-means implementation under BSD licence.

Commercial

- IDL Cluster, Clust_Wts
- MATLAB
- SAS
- Stata
- Grapheme

4.2.9 See also

- Canopy clustering algorithm
- Centroidal Voronoi tessellation
- k q-flats
- Linde–Buzo–Gray algorithm
- Nearest centroid classifier
- Self-organizing map
- Silhouette clustering
- Head/tail Breaks

4.2.10 References

[1] MacQueen, J. B. (1967). *Some Methods for classification and Analysis of Multivariate Observations*. Proceedings of 5th Berkeley Symposium on Mathematical Statistics and Probability **1**. University of California Press. pp. 281–297. MR 0214227. Zbl 0214.46201. Retrieved 2009-04-07.

[2] Steinhaus, H. (1957). "Sur la division des corps matériels en parties". *Bull. Acad. Polon. Sci.* (in French) **4** (12): 801–804. MR 0090073. Zbl 0079.16403.

[3] Lloyd, S. P. (1957). "Least square quantization in PCM" . *Bell Telephone Laboratories Paper*. Published in journal much later: Lloyd., S. P. (1982). "Least squares quantization in PCM" (PDF). *IEEE Transactions on Information Theory* **28** (2): 129–137. doi:10.1109/TIT.1982.1056489. Retrieved 2009-04-15.

[4] E.W. Forgy (1965). "Cluster analysis of multivariate data: efficiency versus interpretability of classifications" . *Biometrics* **21**: 768–769.

[5] J.A. Hartigan (1975). *Clustering algorithms*. John Wiley & Sons, Inc.

[6] Hartigan, J. A.; Wong, M. A. (1979). "Algorithm AS 136: A K-Means Clustering Algorithm" . *Journal of the Royal Statistical Society, Series C* **28** (1): 100–108. JSTOR 2346830.

[7] MacKay, David (2003). "Chapter 20. An Example Inference Task: Clustering" (PDF). *Information Theory, Inference and Learning Algorithms*. Cambridge University Press. pp. 284–292. ISBN 0-521-64298-1. MR 2012999.

[8] Since the square root is a monotone function, this also is the minimum Euclidean distance assignment.

[9] Hamerly, G. and Elkan, C. (2002). "Alternatives to the k-means algorithm that find better clusterings" (PDF). *Proceedings of the eleventh international conference on Information and knowledge management (CIKM)*.

[10] Vattani., A. (2011). "k-means requires exponentially many iterations even in the plane" (PDF). *Discrete and Computational Geometry* **45** (4): 596–616. doi:10.1007/s00454-011-9340-1.

[11] Arthur, D.; Manthey, B.; Roeglin, H. (2009). "k-means has polynomial smoothed complexity" . *Proceedings of the 50th Symposium on Foundations of Computer Science (FOCS)*.

[12] Aloise, D.; Deshpande, A.; Hansen, P.; Popat, P. (2009). "NP-hardness of Euclidean sum-of-squares clustering" . *Machine Learning* **75**: 245–249. doi:10.1007/s10994-009-5103-0.

[13] Dasgupta, S. and Freund, Y. (July 2009). "Random Projection Trees for Vector Quantization" . *Information Theory, IEEE Transactions on* **55**: 3229–3242. arXiv:0805.1390. doi:10.1109/TIT.2009.2021326.

[14] Mahajan, M.; Nimbhorkar, P.; Varadarajan, K. (2009). "The Planar k-Means Problem is NP-Hard" . *Lecture Notes in Computer Science* **5431**: 274–285. doi:10.1007/978-3-642-00202-1_24.

[15] Inaba, M.; Katoh, N.; Imai, H. (1994). *Applications of weighted Voronoi diagrams and randomization to variance-based* k*-clustering*. Proceedings of 10th ACM Symposium on Computational Geometry. pp. 332–339. doi:10.1145/177424.178042.

[16] Arthur; Abhishek Bhowmick (2009). *A theoretical analysis of Lloyd's algorithm for k-means clustering* (Thesis).

[17] Kanungo, T.; Mount, D. M.; Netanyahu, N. S.; Piatko, C. D.; Silverman, R.; Wu, A. Y. (2002). "An efficient k-means clustering algorithm: Analysis and implementation" (PDF). *IEEE Trans. Pattern Analysis and Machine Intelligence* **24**: 881–892. doi:10.1109/TPAMI.2002.1017616. Retrieved 2009-04-24.

[18] Frahling, G.; Sohler, C. (2006). *A fast k-means implementation using coresets* (PDF). Proceedings of the twenty-second annual symposium on Computational geometry (SoCG).

[19] Elkan, C. (2003). "Using the triangle inequality to accelerate k-means" (PDF). *Proceedings of the Twentieth International Conference on Machine Learning (ICML)*.

[20] Dhillon, I. S.; Modha, D. M. (2001). "Concept decompositions for large sparse text data using clustering" . *Machine Learning* **42** (1): 143–175.

[21] Amorim, R. C.; Mirkin, B (2012). "Minkowski metric, feature weighting and anomalous cluster initializing in K-Means clustering" . *Pattern Recognition* **45** (3): 1061–1075. doi:10.1016/j.patcog.2011.08.012.

[22] Mirkes, E.M. "K-means and K-medoids applet." . *http://www.math.le.ac.uk". Retrieved 1 May 2015.*

[23] Honarkhah, M and Caers, J, 2010, *Stochastic Simulation of Patterns Using Distance-Based Pattern Modeling*, Mathematical Geosciences, 42: 487 - 517

[24] Coates, Adam; Ng, Andrew Y. (2012). "Learning feature representations with k-means" (PDF). In G. Montavon, G. B. Orr, K.-R. Müller. *Neural Networks: Tricks of the Trade*. Springer.

[25] Csurka, Gabriella; Dance, Christopher C.; Fan, Lixin; Willamowski, Jutta; Bray, Cédric (2004). *Visual categorization with bags of keypoints* (PDF). ECCV Workshop on Statistical Learning in Computer Vision.

[26] Coates, Adam; Lee, Honglak; Ng, Andrew Y. (2011). *An analysis of single-layer networks in unsupervised feature learning* (PDF). International Conference on Artificial Intelligence and Statistics (AISTATS).

[27] Schwenker, Friedhelm; Kestler, Hans A.; Palm, Günther (2001). "Three learning phases for radial-basis-function networks" . *Neural Networks* **14**: 439–458. doi:10.1016/s0893-6080(01)00027-2. CiteSeerX: 10.1.1.109.312.

[28] Lin, Dekang; Wu, Xiaoyun (2009). *Phrase clustering for discriminative learning* (PDF). Annual Meeting of the ACL and IJCNLP. pp. 1030–1038.

[29] Press, WH; Teukolsky, SA; Vetterling, WT; Flannery, BP (2007). "Section 16.1. Gaussian Mixture Models and k-Means Clustering" . *Numerical Recipes: The Art of Scientific Computing* (3rd ed.). New York: Cambridge University Press. ISBN 978-0-521-88068-8.

[30] Aharon, Michal; Elad, Michael; Bruckstein, Alfred (2006). "K-SVD: An Algorithm for Designing Overcomplete Dictionaries for Sparse Representation" (PDF).

[31] Little, M.A.; Jones, N.S. (2011). "Generalized Methods and Solvers for Piecewise Constant Signals: Part I" (PDF). *Proceedings of the Royal Society A*.

[32] H. Zha, C. Ding, M. Gu, X. He and H.D. Simon (Dec 2001). "Spectral Relaxation for K-means Clustering" (PDF). *Neural Information Processing Systems vol.14 (NIPS 2001)* (Vancouver, Canada): 1057–1064.

[33] Chris Ding and Xiaofeng He (July 2004). "K-means Clustering via Principal Component Analysis" (PDF). *Proc. of Int'l Conf. Machine Learning (ICML 2004)*: 225–232.

[34] Drineas, P.; A. Frieze; R. Kannan; S. Vempala; V. Vinay (2004). "Clustering large graphs via the singular value decomposition" (PDF). *Machine learning* **56**: 9–33. doi:10.1023/b:mach.0000033113.59016.96. Retrieved 2012-08-02.

[35] Cohen, M.; S. Elder; C. Musco; C. Musco; M. Persu (2014). "Dimensionality reduction for k-means clustering and low rank approximation (Appendix B)". *ArXiv*. Retrieved 2014-11-29.

[36] Alon Vinnikov and Shai Shalev-Shwartz (2014). "K-means Recovers ICA Filters when Independent Components are Sparse" (PDF). *Proc. of Int'l Conf. Machine Learning (ICML 2014)*.

[37] Clustering.jl www.github.com

4.2.11 External links

- Clustergram: visualization and diagnostics for cluster analysis (R code)
- Hyper-threaded Java, Use the Java concurrency API to speed up time-consuming tasks.
- K-means and K-medoids applet,E.M. Mirkes, University of Leicester, 2011
- Clustering colors in an image
- K-means Clustering Demo
- Visualizing K-Means Clustering
- A Tutorial on Clustering Algorithms: K-means Demo
- Yet Another K-means Visualization
- k-Means and Voronoi Tesselation Demo

4.3 k-medians clustering

In statistics and data mining, ***k*-medians clustering**[1][2] is a cluster analysis algorithm. It is a variation of *k*-means clustering where instead of calculating the mean for each cluster to determine its centroid, one instead calculates the median. This has the effect of minimizing error over all clusters with respect to the 1-norm distance metric, as opposed to the square of the 2-norm distance metric (which *k*-means does.)

This relates directly to the ***k*-median problem** which is the problem of finding k centers such that the clusters formed by them are the most compact. Formally, given a set of data points x, the k centers c_i are to be chosen so as to minimize the sum of the distances from each x to the nearest c_i.

The criterion function formulated in this way is sometimes a better criterion than that used in the *k*-means clustering algorithm, in which the sum of the squared distances is used. The sum of distances is widely used in applications such as facility location.

The proposed algorithm uses Lloyd-style iteration which alternates between an expectation (E) and maximization (M) step, making this an Expectation–maximization algorithm. In the E step, all objects are assigned to their nearest median. In the M step, the medians are recomputed by using the median in each single dimension.

4.3.1 Medians and medoids

The median is computed in each single dimension in the Manhattan-distance formulation of the *k*-medians problem, so the individual attributes will come from the dataset. This makes the algorithm more reliable for discrete or even binary data sets. In contrast, the use of means or Euclidean-distance medians will not necessarily yield individual attributes from the dataset. Even with the Manhattan-distance formulation, the individual attributes may come from different instances in the dataset; thus, the resulting median may not be a member of the input dataset.

This algorithm is often confused with the *k*-medoids algorithm. However, a medoid has to be an actual instance from the dataset, while for the multivariate Manhattan-distance median this only holds for single attribute values. The actual median can thus be a combination of multiple instances. For example, given the vectors (0,1), (1,0) and (2,2), the Manhattan-distance median is (1,1), which does not exist in the original data, and thus cannot be a medoid.

4.3.2 Software

- ELKI includes various k-means variants, including k-medians.
- GNU R includes k-medians in the "flexclust" package.
- Stata kmedians

4.3.3 See also

- cluster analysis

- k-means
- medoid
- silhouette

4.3.4 References

[1] A. K. Jain and R. C. Dubes, *Algorithms for Clustering Data*. Prentice-Hall, 1988.

[2] P. S. Bradley, O. L. Mangasarian, and W. N. Street, "Clustering via Concave Minimization," in Advances in Neural Information Processing Systems, vol. 9, M. C. Mozer, M. I. Jordan, and T. Petsche, Eds. Cambridge, MA: MIT Press, 1997, pp. 368–374.

4.4 K-means++

In data mining, ***k*-means++**[*][1][*][2] is an algorithm for choosing the initial values (or "seeds") for the *k*-means clustering algorithm. It was proposed in 2007 by David Arthur and Sergei Vassilvitskii, as an approximation algorithm for the NP-hard *k*-means problem—a way of avoiding the sometimes poor clusterings found by the standard *k*-means algorithm. It is similar to the first of three seeding methods proposed, in independent work, in 2006[*][3] by Rafail Ostrovsky, Yuval Rabani, Leonard Schulman and Chaitanya Swamy. (The distribution of the first seed is different.)

4.4.1 Background

The *k*-means problem is to find cluster centers that minimize the intra-class variance, i.e. the sum of squared distances from each data point being clustered to its cluster center (the center that is closest to it). Although finding an exact solution to the *k*-means problem for arbitrary input is NP-hard,[*][4] the standard approach to finding an approximate solution (often called Lloyd's algorithm or the *k*-means algorithm) is used widely and frequently finds reasonable solutions quickly.

However, the *k*-means algorithm has at least two major theoretic shortcomings:

- First, it has been shown that the worst case running time of the algorithm is super-polynomial in the input size.[*][5]
- Second, the approximation found can be arbitrarily bad with respect to the objective function compared to the optimal clustering.

The *k*-means++ algorithm addresses the second of these obstacles by specifying a procedure to initialize the cluster centers before proceeding with the standard *k*-means optimization iterations. With the *k*-means++ initialization, the algorithm is guaranteed to find a solution that is O(log k) competitive to the optimal *k*-means solution.

4.4.2 Initialization algorithm

The intuition behind this approach is that spreading out the k initial cluster centers is a good thing: the first cluster center is chosen uniformly at random from the data points that are being clustered, after which each subsequent cluster center is chosen from the remaining data points with probability proportional to its squared distance from the point's closest existing cluster center.

The exact algorithm is as follows:

1. Choose one center uniformly at random from among the data points.
2. For each data point x, compute D(x), the distance between x and the nearest center that has already been chosen.
3. Choose one new data point at random as a new center, using a weighted probability distribution where a point x is chosen with probability proportional to D(x)^2.
4. Repeat Steps 2 and 3 until k centers have been chosen.
5. Now that the initial centers have been chosen, proceed using standard *k*-means clustering.

This seeding method yields considerable improvement in the final error of k-means. Although the initial selection in the algorithm takes extra time, the k-means part itself converges very quickly after this seeding and thus the algorithm actually lowers the computation time. The authors tested their method with real and synthetic datasets and obtained typically 2-fold improvements in speed, and for certain datasets, close to 1000-fold improvements in error. In these simulations the new method almost always performed at least as well as vanilla k-means in both speed and error.

Additionally, the authors calculate an approximation ratio for their algorithm. The k-means++ algorithm guarantees an approximation ratio O(log k) in expectation (over the randomness of the algorithm), where k is the number of clusters used. This is in contrast to vanilla k-means, which can generate clusterings arbitrarily worse than the optimum.[*][6]

4.4.3 Example bad case

To illustrate the potential of the *k*-means algorithm to perform arbitrarily poorly with respect to the objective function of minimizing the sum of squared distances of cluster points to the centroid of their assigned clusters, consider the example of four points in $\mathbf{R}^2$ that form an axis-aligned rectangle whose width is greater than its height.

If $k = 2$ and the two initial cluster centers lie at the midpoints of the top and bottom line segments of the rectangle formed by the four data points, the k-means algorithm converges immediately, without moving these cluster centers. Consequently, the two bottom data points are clustered together and the two data points forming the top of the rectangle are clustered together—a suboptimal clustering because the width of the rectangle is greater than its height.

Now, consider stretching the rectangle horizontally to an arbitrary width. The standard k-means algorithm will continue to cluster the points suboptimally, and by increasing the horizontal distance between the two data points in each cluster, we can make the algorithm perform arbitrarily poorly with respect to the k-means objective function.

4.4.4 Applications

The k-means++ approach has been applied since its initial proposal. In a review by Shindler,*[7] which includes many types of clustering algorithms, the method is said to successfully overcome some of the problems associated with other ways of defining initial cluster-centres for k-means clustering. Lee et al.*[8] report an application of k-means++ to create geographical cluster of photographs based on the latitude and longitude information attached to the photos. An application to financial diversification is reported by Howard and Johansen.*[9] Other support for the method and ongoing discussion is also available online.*[10] Since the k-means++ initialization needs k passes over the data, it does not scale very well to large data sets. Bahman Bahmani et al. have proposed a scalable variant of k-means++ called k-means|| which provides the same theoretical guarantees and yet is highly scalable.*[11]

4.4.5 Software

- Scikit-learn has a K-Means implementation that uses k-means++ by default.
- ELKI data-mining framework contains multiple k-means variations, including k-means++ for seeding.
- GNU R includes k-means, and the “flexclust” package can do k-means++
- OpenCV implementation
- Weka contains k-means (with optional k-means++) and x-means clustering.
- David Arthur's implementation
- Apache Commons Math Java implementation
- CMU's GraphLab GraphLab Efficient, open source clustering on multicore.

4.4.6 References

[1] Arthur, D. and Vassilvitskii, S. (2007). "k-means++: the advantages of careful seeding" (PDF). *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics Philadelphia, PA, USA. pp. 1027–1035.

[2] http://theory.stanford.edu/~{}sergei/slides/BATS-Means.pdf Slides for presentation of method by Arthur, D. and Vassilvitskii, S.

[3] Ostrovsky, R., Rabani, Y., Schulman, L. J. and Swamy, C. (2006). “The Effectiveness of Lloyd-Type Methods for the k-Means Problem” . *Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06)*. IEEE. pp. 165–174.

[4] Drineas, P. and Frieze, A. and Kannan, R. and Vempala, S. and Vinay, V. (2004). “Clustering Large Graphs via the Singular Value Decomposition” . *Machine Learning* (Kluwer Academic Publishers Hingham, MA, USA) **56** (1–3): 9–33. doi:10.1023/B:MACH.0000033113.59016.96.

[5] Arthur, D. and Vassilvitskii, S. (2006), “How slow is the k-means method?", *ACM New York, NY, USA*: 144–153

[6] T. Kanungo, D. Mount, N. Netanyahux, C. Piatko, R. Silverman, A. Wu “A Local Search Approximation Algorithm for k-Means Clustering” 2004 Computational Geometry: Theory and Applications.

[7] http://web.archive.org/web/20110927100642/http://www.cs.ucla.edu/~{}shindler/shindler-kMedian-survey.pdf Approximation Algorithms for the Metric k-Median Problem

[8] http://sir-lab.usc.edu/publications/2008-ICWSM2LEES.pdf Discovering Relationships among Tags and Geotags, 2007

[9] http://www.cse.ohio-state.edu/~{}johansek/clustering.pdf Clustering Techniques for Financial Diversification, March 2009

[10] http://lingpipe-blog.com/2009/03/23/arthur-vassilvitskii-2007-kmeans-the-advantages-of-careful-seeding/ Lingpipe Blog

[11] B. Bahmani, B. Moseley, A. Vattani, R. Kumar, S. Vassilvitskii “Scalable K-means++" 2012 Proceedings of the VLDB Endowment.

4.5 k-medoids

The **k-medoids algorithm** is a clustering algorithm related to the k-means algorithm and the medoidshift algorithm. Both the k-means and k-medoids algorithms are partitional (breaking the dataset up into groups) and both attempt to minimize the distance between points labeled to be in a cluster and a point designated as the center of that cluster. In contrast to the k-means algorithm, k-medoids chooses datapoints as centers (medoids or exemplars) and works with an arbitrary matrix of distances between datapoints instead of l_2 . This method was proposed in 1987*[1] for the work with l_1 norm and other distances.

k-medoid is a classical partitioning technique of clustering that clusters the data set of n objects into k clusters known *a priori*. A useful tool for determining k is the silhouette.

It is more robust to noise and outliers as compared to k-means because it minimizes a sum of pairwise dissimilarities instead of a sum of squared Euclidean distances.

A medoid can be defined as the object of a cluster whose average dissimilarity to all the objects in the cluster is minimal. i.e. it is a most centrally located point in the cluster.

The most common realisation of k-medoid clustering is the **Partitioning Around Medoids (PAM)** algorithm and is as follows:*[2]

1. Initialize: randomly select (without replacement) k of the n data points as the medoids
2. Associate each data point to the closest medoid. ("closest" here is defined using any valid distance metric, most commonly Euclidean distance, Manhattan distance or Minkowski distance)
3. For each medoid m
   (a) For each non-medoid data point o
      i. Swap m and o and compute the total cost of the configuration
4. Select the configuration with the lowest cost.
5. Repeat steps 2 to 4 until there is no change in the medoid.

4.5.1 Demonstration of PAM

Cluster the following data set of ten objects into two clusters i.e. k = 2.

Consider a data set of ten objects as follows:

Step 1

Initialize k centers.

Let us assume x_2 and x_8 are selected as medoids, so the centers are c_1 = (3,4) and c_2 = (7,4)

Calculate distances to each center so as to associate each data object to its nearest medoid. Cost is calculated using

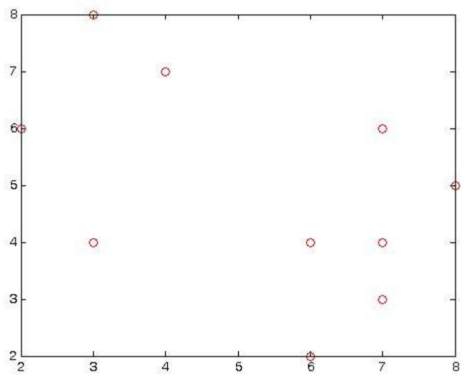


Figure 1.1 – distribution of the data

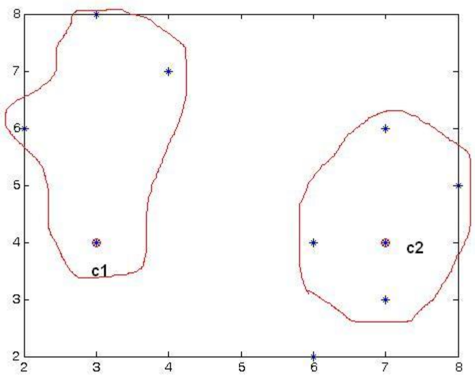


Figure 1.2 – clusters after step 1

Manhattan distance (Minkowski distance metric with r = 1). Costs to the nearest medoid are shown bold in the table.

Then the clusters become:

Cluster_1 = {(3,4)(2,6)(3,8)(4,7)}

Cluster_2 = {(7,4)(6,2)(6,4)(7,3)(8,5)(7,6)}

Since the points (2,6) (3,8) and (4,7) are closer to c_1 hence they form one cluster whilst remaining points form another cluster.

So the total cost involved is 20.

Where cost between any two points is found using formula

$\text{cost}(x, c) = \sum_{i=1}^{d} |x_i - c_i|$

where x is any data object, c is the medoid, and d is the dimension of the object which in this case is 2.

Total cost is the summation of the cost of data object from its medoid in its cluster so here:

$$\begin{aligned}\text{total cost} &= \{\text{cost}((3,4),(2,6)) + \text{cost}((3,4),(3,8)) + \text{cost}((3,4),(4,7))\}\\ &+ \{\text{cost}((7,4),(6,2)) + \text{cost}((7,4),(6,4)) + \text{cost}((7,4),(7,3))\\ &+ \text{cost}((7,4),(8,5)) + \text{cost}((7,4),(7,6))\}\\ &= (3+4+4) + (3+1+1+2+2)\\ &= 20\end{aligned}$$

Step 2

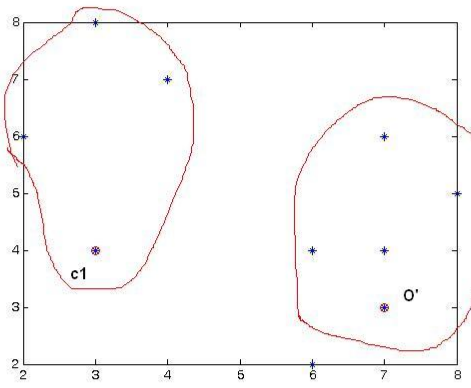


Figure 1.3 – clusters after step 2

Select one of the nonmedoids O′

Let us assume O′ = (7,3), i.e. x_7.

So now the medoids are c_1(3,4) and O′(7,3)

If c1 and O′ are new medoids, calculate the total cost involved

By using the formula in the step 1

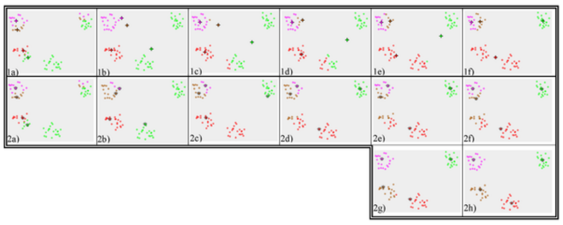


Figure 2. K-medoids versus k-means. Figs 2.1a-2.1f present a typical example of the k-means convergence to a local minimum. This result of k-means clustering contradicts the obvious cluster structure of data set. In this example, k-medoids algorithm (Figs 2.2a-2.2h) with the same initial position of medoids (Fig. 2.2a) converges to the obvious cluster structure. The small circles are data points, the four ray stars are centroids (means), the nine ray stars are medoids.[3]

$$\begin{aligned}\text{total cost} &= 3+4+4+2+2+1+3+3\\ &= 22\end{aligned}$$

So cost of swapping medoid from c_2 to O′ is

$$\begin{aligned}S &= \text{current total cost} - \text{past total cost}\\ &= 22 - 20\\ &= 2 > 0.\end{aligned}$$

=== program:

So moving to O′ would be a bad idea, so the previous choice was good. So we try other nonmedoids and found that our first choice was the best. So the configuration does not change and algorithm terminates here (i.e. there is no change in the medoids).

It may happen some data points may shift from one cluster to another cluster depending upon their closeness to medoid.

In some standard situations, k-medoids demonstrate better performance than k-means. An example is presented in Fig. 2. The most time-consuming part of the k-medoids algorithm is the calculation of the distances between objects. If a quadratic preprocessing and storage is applicable, the distances matrix can be precomputed to achieve consequent speed-up. See for example,*[4] where the authors also introduce a heuristic to choose the initial k medoids.

4.5.2 Software

- ELKI includes several k-means variants, including an EM-based k-medoids and the original PAM algorithm.
- Julia contains a k-medoid implementation in the JuliaStats clustering package.*[5]
- R includes variants of k-means in the "flexclust" package and PAM is implemented in the "cluster" package.
- RapidMiner has an operator named KMedoids, but it does *not* implement the KMedoids algorithm correctly. Instead, it is a k-means variant, that substitutes the mean with the closes data point (which is not the medoid).
- Java-ML. Includes a k-medoid implementation that is incorrect in the same way as the RapidMiner version.

4.5.3 External links

E.M. Mirkes, K-means and K-medoids (Applet), University of Leicester, 2011.

4.5.4 References

[1] Kaufman, L. and Rousseeuw, P.J. (1987), Clustering by means of Medoids, in Statistical Data Analysis Based on the L_1 –Norm and Related Methods, edited by Y. Dodge, North-Holland, 405–416.

[2] Sergios Theodoridis & Konstantinos Koutroumbas (2006). *Pattern Recognition 3rd ed.* p. 635.

[3] The illustration was prepared with the Java applet, E.M. Mirkes, K-means and K-medoids: applet. University of Leicester, 2011.

[4] H.S. Park , C.H. Jun, A simple and fast algorithm for K-medoids clustering, Expert Systems with Applications, 36, (2) (2009), 3336–3341.

[5] Clustering.jl

4.6 k q-flats clustering

In data mining and machine learning, k q -flats algorithm *[1] *[2] is an iterative method which aims to partition m observations into k clusters where each cluster is close to a q -flat, where q is a given integer.

It is a generalization of the k -means algorithm. In k -means algorithm, clusters are formed in the way that each cluster is close to one point, which is a 0 -flat. k q -flats algorithm gives better clustering result than k -means algorithm for some data set.

4.6.1 Description

Problem formulation

Given a set A of m observations $(a_1, a_2, \dots, a_m)$ where each observation a_i is an n-dimensional real vector, k q -flats algorithm aims to partition m observation points by generating k q -flats that minimize the sum of the squares of distances of each observation to a nearest q-flat.

A q -flat is a subset of R^n that is congruent to R^q . For example, a 0 -flat is a point; a 1 -flat is a plane; a $n-1$ -flat is a hyperplane. q -flat can be characterized by the solution set of a linear system of equations: $F = \{x|x \in R^n, W'x = \gamma\}$, where $W \in R^{n\times(n-q)}$, $\gamma \in R^{1\times(n-q)}$.

Denote a partition of $\{1, 2, \dots, n\}$ as $S = (S_1, S_2, \dots, S_k)$. The problem can be formulated as

$(P1)$ $\min_{F_l, l=1,\dots,k \text{q-flats are}} \min_S \sum_{l=1}^k \sum_{a_j \in S_i} \|a_j - P_{F_i}(a_j)\|^2$,

where $P_{F_i}(a_j)$ is the projection of a_j onto F_i . Note that $\|a_j - P_{F_i}(a_j)\| = dist(a_j, F_l)$ is the distance from a_j to F_l .

Algorithm

The algorithm is similar to the k-means algorithm (i.e. Lloyd's algorithm) in that it alternates between cluster assignment and cluster update. In specific, the algorithm starts with an initial set of q -flats $F_l^{(0)} = \{x \in R^n | (W_l^{(0)})'x = \gamma_l^{(0)}\}, l = 1, \dots, k$, and proceeds by alternating between the following two steps:

Cluster Assignment (given q -flats, assign each point to closest q -flat): the i'th cluster is updated as $S_i^{(t)} = \{a_j | \|(W_i^{(t)})'a_j - \gamma_i^{(t)}\|_F = \min_{l=1,\dots,k} \|(W_l^{(t)})'a_j - \gamma_l^{(t)}\|_F\}$.

Cluster Update (given cluster assignment, update the q -flats): For $l = 1, \dots, k$, let $A(l) \in R^{m(l)\times n}$ with rows corresponding to all a_i assigned to cluster l . Set $W_l^{(t+1)}$ to be the matrix whose columns are the orthonormal eigenvectors corresponding to the $(n-q)$ least eigenvalues of $A(l)'(I - \frac{ee'}{m})A(l)$ and $\gamma_l^{(t+1)} = \frac{e'A(l)W_l^{(t+1)}}{m}$.

Stop whenever the assignments no longer change.

The cluster assignment step uses the following fact: given a q-flat $F_l = \{x|W'x = \gamma\}$ and a vector a , where $W'W = I$, the distance from a to the q-flat F_l is $dist(a, F_l) = min_{x:W'x=\gamma} \|x - a\|_F^2 = \|W(W'W)^{-1}(W'x - \gamma)\|_F^2 = \|W'x - \gamma\|_F^2$.

The key part of this algorithm is how to update the cluster, i.e. given m points, how to find a q -flat that minimizes the sum of squares of distances of each point to the q -flat. Mathematically, this problem is: given $A \in R^{m\times n}$, solve the quadratic optimization problem

$(P2)$ $\min_{W\in R^{n\times(n-q)}, \gamma\in R^{1\times(n-q)}} \|AW - e\gamma\|_F^2$, subject to $W'W = I$,

where $A \in R^{m\times n}$ is given, and $e = (1, \dots, 1)' \in R^{m\times 1}$.

The problem can be solved using Lagrangian multiplier method and the solution is as given in the cluster update step.

It can be shown that the algorithm will terminate in a finite number of iterations (no more than the total number of possible assignments, which is bounded by k^m). In addition, the algorithm will terminate at a point that the overall objective cannot be decreased either by a different assignment or by defining new cluster planes for these clusters (such point is called “locally optimal” in the references).

This convergence result is a consequence of the fact that problem (P2) can be solved exactly. The same convergence result holds for k -means algorithm because the cluster update problem can be solved exactly.

4.6.2 Relation to other machine learning methods

k -means algorithm

k q -flats algorithm is a generalization of k -means algorithm. In fact, k -means algorithm is k 0-flats algorithm since a point is a 0-flat. Despite their connection, they should be used in different scenarios. k q -flats algorithm for the case that data lie in a few low-dimensional spaces.

k -means algorithm is desirable for the case the clusters are of the ambient dimension, . For example, if all observations lie in two lines, k q -flats algorithm with $q = 1$ may be used; if the observations are two Gaussian clouds, k -means algorithm may be used.

Sparse Dictionary Learning

Natural signals lie in a high-dimensional space. For example, the dimension of a 1024 by 1024 image is about *10^*6*, which is far too high for most signal processing algorithms. One way to get rid of the high dimensionality is to find a set of basis functions, such that the high-dimensional signal can be represented by only a few basis functions. In other words, the coefficients of the signal representation lies in a low-dimensional space, which is easier to apply signal processing algorithms. In the literature, wavelet transform is usually used in image processing, and fourier transform is usually used in audio processing. The set of basis functions is usually called a *dictionary*.

However, it is not clear what is the best dictionary to use once given a signal data set. One popular approach is to find a dictionary when given a data set using the idea of Sparse Dictionary Learning. It aims to find a dictionary, such that the signal can be sparsely represented by the dictionary. The optimization problem can be written as follows.

$\min_{B,R} \|X - BR\|_F^2$ subject to $\|R_i\|_0 \leq q$

where

- X is a *d* by *N* matrix. Each columns of X represent a signal, and there are total *N* signals.
- B is a *d* by *l* matrix. Each columns of B represent a basis function, and there are total *l* basis functions in the dictionary.
- R is a *l* by *N* matrix. R_i (*i^*th* columns of R) represent the coefficients when we use the dictionary B to represent the *i^*th* columns of X.
- $\|v\|_0$ denotes the zero-norm of the vector *v*.
- $\|V\|_F$ denotes the Frobenious norm of matrix *V*.

The idea of k $q-$ flats algorithm is similar to sparse dictionary learning in nature. If we restrict the q-flat to q-dimensional subspace, then the k $q-$ flats algorithm is simply finding the closed q-dimensional subspace to a given signal. Sparse dictionary learning is also doing the same thing, except for an additional constraints on the sparsity of the representation. Mathematically, it is possible to show that k $q-$ flats algorithm is of the form of sparse dictionary learning with an additional block structure on *R*.

Let B_k be a $d \times q$ matrix, where columns of B_k are basis of the k^{th} flat. Then the projection of the signal *x* to the k^{th} flat is $B_k r_k$, where r_k is a q-dimensional coefficient. Let $B = [B_1, \cdots, B_K]$ denote concatenation of basis of K flats, it is easy to show that the k -q-flat algorithm is the same as the following.

$\min_{B,R} \|X - BR\|_F^2$ subject to $\|R_i\|_0 \leq q$ and *R* has a block structure.

The block structure of *R* refers the fact that each signal is labeled to only one flat. Comparing the two formulations, k q-flat is the same as sparse dictionary modeling when $l = K \times q$ and with an additional block structure on *R*. Users may refer to Szlam's paper *[3] for more discussion about the relationship between the two concept.

4.6.3 Applications and Variations

Classification

Classification is a procedure that classifies an input signal into different classes. One example is to classify an email into *spam* or *non-spam* classes. Classification algorithms usually require a supervised learning stage. In the supervised learning stage, training data for each class is used for the algorithm to learn the characteristics of the class. In the classification stage, a new observation is classified into a class by using the characteristics that were already trained.

k q-flat algorithm can be used for classification. Suppose there are total of m classes. For each class, k flats are trained a priori via training data set. When a new data comes, find the flat that is closest to the new data. Then the new data is associate to class of the closest flat.

However, the classification performance can be further improved if we impose some structure on the flats. One possible choice is to require different flats from different class be sufficiently far apart. Some researchers *[4] use this idea and develop a discriminative k q-flat algorithm.

K-metrics *[3]

In k q -flats algorithm, $\|x - P_F(x)\|^2$ is used to measure the representation error. $P_F(x)$ denotes the projection of *x* to the flat *F*. If data lies in a q-dimension flat, then a single q-flat can represent the data very well. On the contrary, if data lies in a very high dimension space but near a common center, then k-means algorithm is a better way than k q-flat algorithm to represent the data. It is because k -means algorithm use $\|x - x_c\|^2$ to measure the error, where x_c denotes the center. K-metrics is a generalization that use both the idea of flat and mean. In k-metrics, error is measured by the following Mahalanobis metric.

$\|x - y\|_A^2 = (x - y)^T A(x - y)$

where *A* is a positive semi-definite matrix.

If *A* is the identity matrix, then the Mahalanobis metric is exactly the same as the error measure used in k-means.

If A is not the identity matrix, then $\|x-y\|_A^2$ will favor certain directions as the k q-flat error measure.

4.6.4 References

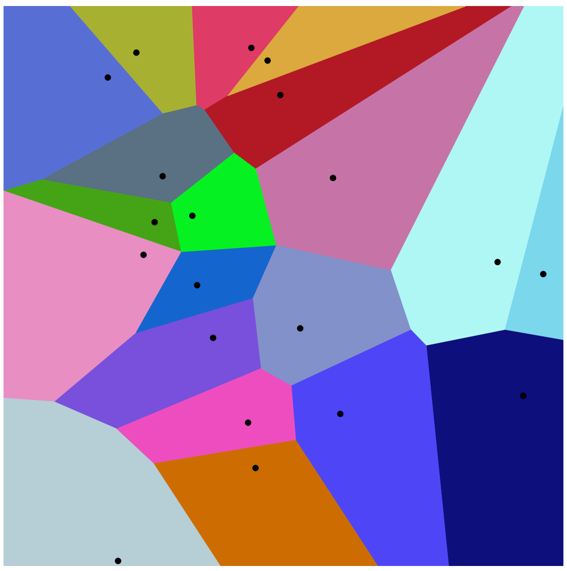
[1] Bradley, P S, and O L Mangasarian. 2000. k-Plane Clustering. Journal of Global Optimization 16, no. 1: 23-32. http://www.springerlink.com/index/H51L607707062TN7.pdf.

[2] Tseng, P. 2000. Nearest q-flat to m points. Journal of Optimization Theory and Applications 105, no. 1: 249–252.

[3] Szlam, A, and G Sapiro. 2009. "Discriminative k-metrics." Ed. Léon Bottou and Michael Littman. Processing (1) 744615-744615-10

[4] Szlam, A, and G Sapiro. "Supervised Learning via Discriminative k q-Flats"

4.7 Voronoi diagram



20 points and their Voronoi cells (larger version below).

In mathematics, a **Voronoi diagram** is a partitioning of a plane into regions based on distance to points in a specific subset of the plane. That set of points (called seeds, sites, or generators) is specified beforehand, and for each seed there is a corresponding region consisting of all points closer to that seed than to any other. These regions are called Voronoi cells. The Voronoi diagram of a set of points is dual to its Delaunay triangulation. Put simply, it's a diagram created by taking pairs of points that are close together and drawing a line that is equidistant between them and perpendicular to the line connecting them. That is, all points on the lines in the diagram are equidistant to the nearest two (or more) source points.

It is named after Georgy Voronoy, and is also called a **Voronoi tessellation**, a **Voronoi decomposition**, a **Voronoi partition**, or a **Dirichlet tessellation** (after Peter Gustav Lejeune Dirichlet). Voronoi diagrams have practical and theoretical applications to a large number of fields, mainly in science and technology but even including visual art.*[1]*[2]

4.7.1 The simplest case

In the simplest and most familiar case (shown in the first picture), we are given a finite set of points $\{p_1, \cdots, p_n\}$ in the Euclidean plane. In this case each site p_k is simply a point, and its corresponding Voronoi cell (also called Voronoi region or Dirichlet cell) R_k consists of every point whose distance to p_k is less than or equal to its distance to any other p_k. Each such cell is obtained from the intersection of half-spaces, and hence it is a convex polygon. The segments of the Voronoi diagram are all the points in the plane that are equidistant to the two nearest sites. The Voronoi vertices (nodes) are the points equidistant to three (or more) sites.

4.7.2 Formal definition

Let X be a space (a nonempty set) endowed with a distance function d. Let K be a set of indices and let $(P_k)_{k\in K}$ be a tuple (ordered collection) of nonempty subsets (the sites) in the space X. The Voronoi cell, or Voronoi region, R_k, associated with the site P_k is the set of all points in X whose distance to P_k is not greater than their distance to the other sites P_j, where j is any index different from k. In other words, if $d(x,A) = \inf\{d(x,a) | a \in A\}$ denotes the distance between the point x and the subset A, then

$$R_k = \{x \in X \mid d(x,P_k) \le d(x,P_j) \text{ all for } j \ne k\}$$

The Voronoi diagram is simply the tuple of cells $(R_k)_{k\in K}$. In principle some of the sites can intersect and even coincide (an application is described below for sites representing shops), but usually they are assumed to be disjoint. In addition, infinitely many sites are allowed in the definition (this setting has applications in geometry of numbers and crystallography), but again, in many cases only finitely many sites are considered.

In the particular case where the space is a finite-dimensional Euclidean space, each site is a point, there are finitely many points and all of them are different, then the Voronoi cells are convex polytopes and they can be represented in a combinatorial way using their vertices, sides, 2-dimensional faces, etc. Sometimes the induced combinatorial structure is referred to as the Voronoi diagram. However, in general the Voronoi cells may not be convex or even connected.

In the usual Euclidean space, we can rewrite the formal definition in usual terms. Each Voronoi polygon R_k is associated with a generator point P_k . Let X the set of all points in the Euclidean space. Let P_1 be a point that generates its Voronoi region R_1 , P_2 that generates R_2 , and P_3 that generates R_3 , and so on. Then, as expressed by Tran *et al*[*][3] "all locations in the Voronoi polygon are closer to the generator point of that polygon than any other generator point in the Voronoi diagram in Euclidian plane" .

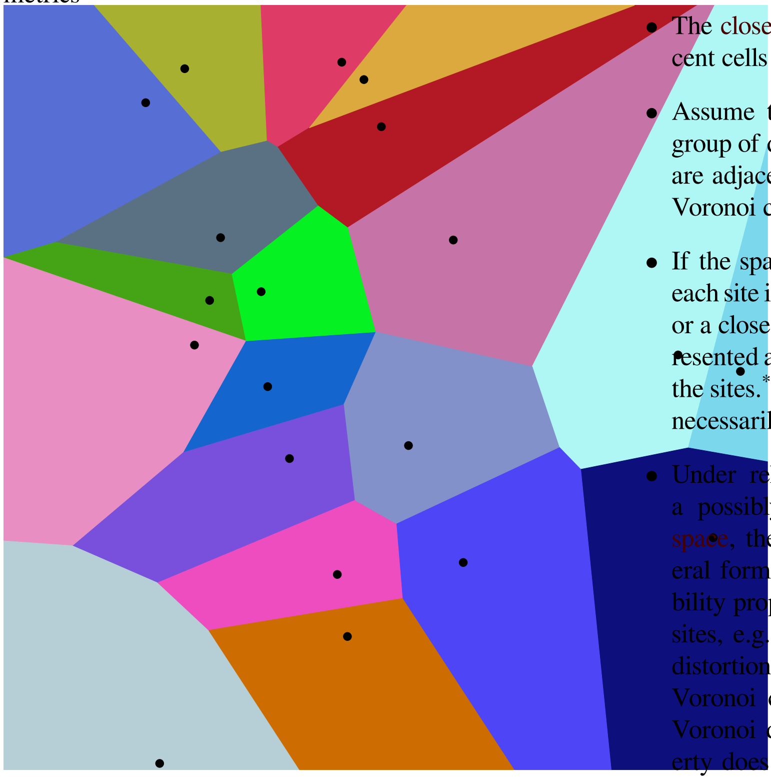
4.7.3 Illustration

As a simple illustration, consider a group of shops in a flat city. Suppose we want to estimate the number of customers of a given shop. With all else being equal (price, products, quality of service, etc.), it is reasonable to assume that customers choose their preferred shop simply by distance considerations: they will go to the shop located nearest to them. In this case the Voronoi cell R_k of a given shop P_k can be used for giving a rough estimate on the number of potential customers going to this shop (which is modeled by a point in our flat city).

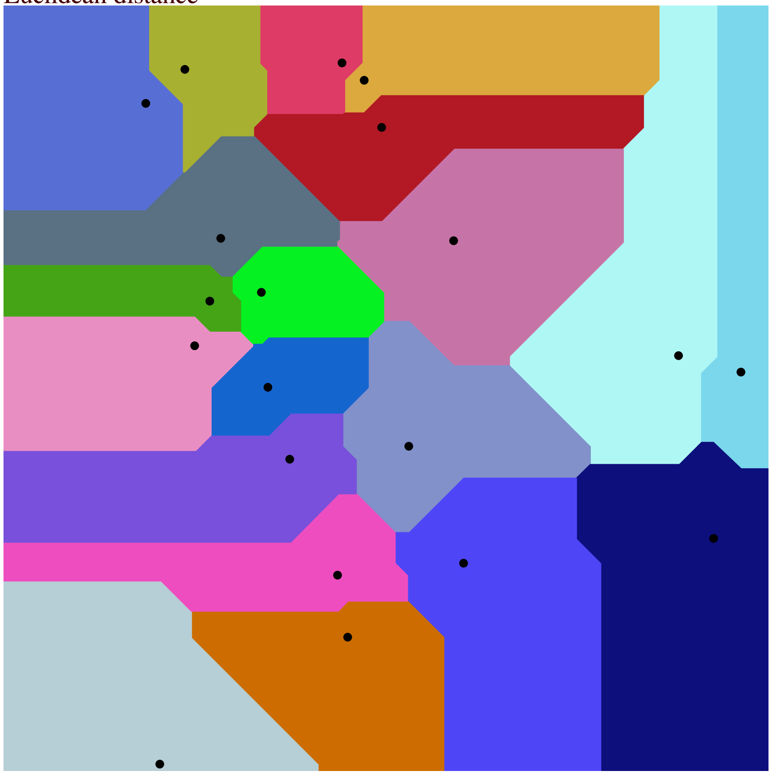
So far it was assumed that the distance between points in the city is measured using the standard distance, the familiar Euclidean distance: $\ell_2 = d\left[(a_1,a_2),(b_1,b_2)\right] = \sqrt{(a_1-b_1)^2+(a_2-b_2)^2}$

However, if we consider the case where customers only go to the shops by a vehicle and the traffic paths are parallel to the x and y axes, as in Manhattan, then a more realistic distance function will be the ℓ_1 distance, namely $d\left[(a_1,a_2),(b_1,b_2)\right] = |a_1-b_1|+|a_2-b_2|$.

Voronoi diagrams of 20 points under two different metrics



Euclidean distance



Manhattan distance

4.7.4 Properties

- The dual graph for a Voronoi diagram (in the case of a Euclidean space with point sites) corresponds to the Delaunay triangulation for the same set of points.
- The closest pair of points corresponds to two adjacent cells in the Voronoi diagram.
- Assume the setting is the Euclidean plane and a group of different points are given. Then two points are adjacent on the convex hull if and only if their Voronoi cells share an infinitely long side.
- If the space is a normed space and the distance to each site is attained (e.g., when a site is a compact set or a closed ball), then each Voronoi cell can be represented as a union of line segments emanating from the sites.[*][4] As shown there, this property does not necessarily hold when the distance is not attained.
- Under relatively general conditions (the space is a possibly infinite-dimensional uniformly convex space, there can be infinitely many sites of a general form, etc.) Voronoi cells enjoy a certain stability property: a small change in the shapes of the sites, e.g., a change caused by some translation or distortion, yields a small change in the shape of the Voronoi cells. This is the geometric stability of Voronoi diagrams.[*][5] As shown there, this property does not hold in general, even if the space is

two-dimensional (but non-uniformly convex, and, in particular, non-Euclidean) and the sites are points.

4.7.5 History and research

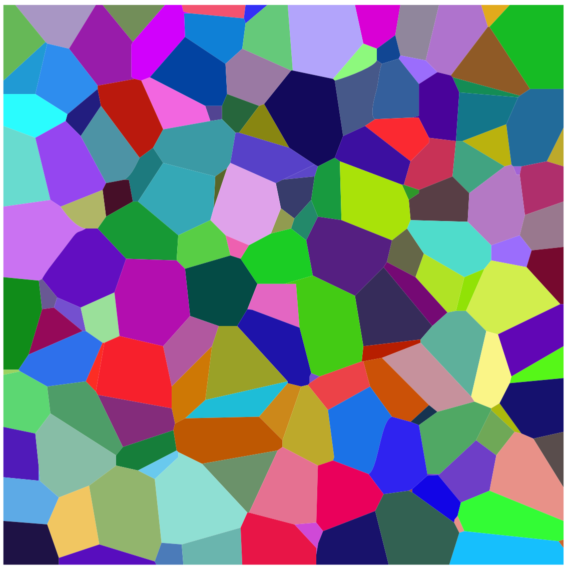
Informal use of Voronoi diagrams can be traced back to Descartes in 1644. Peter Gustav Lejeune Dirichlet used 2-dimensional and 3-dimensional Voronoi diagrams in his study of quadratic forms in 1850. British physician John Snow used a Voronoi diagram in 1854 to illustrate how the majority of people who died in the Soho cholera epidemic lived closer to the infected Broad Street pump than to any other water pump.

Voronoi diagrams are named after Ukrainian mathematician Georgy Fedosievych Voronyi (or *Voronoy*) who defined and studied the general n-dimensional case in 1908. Voronoi diagrams that are used in geophysics and meteorology to analyse spatially distributed data (such as rainfall measurements) are called Thiessen polygons after American meteorologist Alfred H. Thiessen. In condensed matter physics, such tessellations are also known as Wigner–Seitz unit cells. Voronoi tessellations of the reciprocal lattice of momenta are called Brillouin zones. For general lattices in Lie groups, the cells are simply called fundamental domains. In the case of general metric spaces, the cells are often called metric fundamental polygons. Other equivalent names for this concept (or particular important cases of it): Voronoi polyhedra, Voronoi polygons, domain(s) of influence, Voronoi decomposition, Voronoi tessellation(s), Dirichlet tessellation(s).

4.7.6 Examples

Voronoi tessellations of regular lattices of points in two or three dimensions give rise to many familiar tessellations.

- A 2D lattice gives an irregular honeycomb tessellation, with equal hexagons with point symmetry; in the case of a regular triangular lattice it is regular; in the case of a rectangular lattice the hexagons reduce to rectangles in rows and columns; a square lattice gives the regular tessellation of squares; note that the rectangles and the squares can also be generated by other lattices (for example the lattice defined by the vectors (1,0) and (1/2,1/2) gives squares). See here for a dynamic visual example.
- A simple cubic lattice gives the cubic honeycomb.
- A hexagonal close-packed lattice gives a tessellation of space with trapezo-rhombic dodecahedra.
- A face-centred cubic lattice gives a tessellation of space with rhombic dodecahedra.
- A body-centred cubic lattice gives a tessellation of space with truncated octahedra.



This is a slice of the Voronoi diagram of a random set of points in a 3D box. In general a cross section of a 3D Voronoi tessellation is not a 2D Voronoi tessellation itself. (The cells are all convex polyhedra.)

- Parallel planes with regular triangular lattices aligned with each other's centers give the hexagonal prismatic honeycomb.
- Certain body centered tetragonal lattices give a tessellation of space with rhombo-hexagonal dodecahedra.

For the set of points (x, y) with x in a discrete set X and y in a discrete set Y, we get rectangular tiles with the points not necessarily at their centers.

4.7.7 Higher-order Voronoi diagrams

Although a normal Voronoi cell is defined as the set of points closest to a single point in S, an nth-order Voronoi cell is defined as the set of points having a particular set of n points in S as its n nearest neighbors. Higher-order Voronoi diagrams also subdivide space.

Higher-order Voronoi diagrams can be generated recursively. To generate the n^*th-order Voronoi diagram from set S, start with the $(n-1)^*$th-order diagram and replace each cell generated by $X = \{x_1, x_2, ..., x_{n-1}\}$ with a Voronoi diagram generated on the set $S - X$.

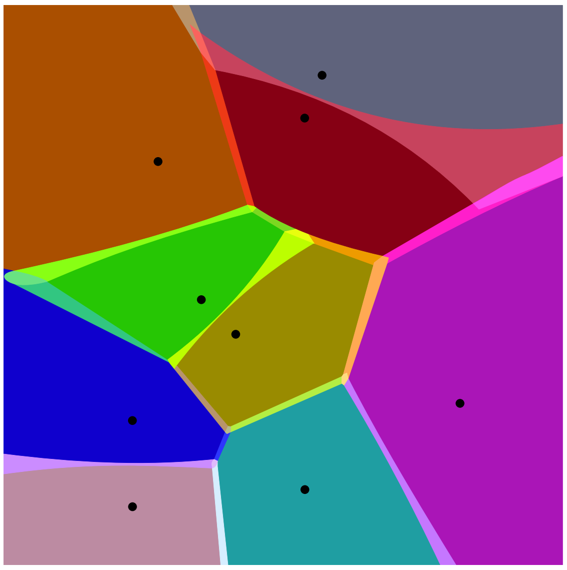
Farthest-point Voronoi diagram

For a set of n points the $(n-1)^*$th-order Voronoi diagram is called a farthest-point Voronoi diagram.

For a given set of points $S = \{p_1, p_2, ..., p_n\}$ the farthest-point Voronoi diagram divides the plane into cells in which the same point of P is the farthest point. Note that a point of P has a cell in the farthest-point Voronoi diagram if and only if it is a vertex of the convex hull of P. Thus, let $H = \{h_1, h_2, ..., h_k\}$ be the convex hull of P we define the farthest-point Voronoi diagram as the subdivision of the plane into k cells, one for each point in H, with the property that a point q lies in the cell corresponding to a site h_i if and only if $\text{dist}(q, h_i) > \text{dist}(q, p_j)$ for each $p_j \in S$ with $h_i \neq p_j$. Where $\text{dist}(p, q)$ is the Euclidean distance between two points p and q.*[6]*[7]

4.7.8 Generalizations and variations

As implied by the definition, Voronoi cells can be defined for metrics other than Euclidean (such as the Mahalanobis or Manhattan) distances. However in these cases the boundaries of the Voronoi cells may be more complicated than in the Euclidean case, since the equidistant locus for two points may fail to be subspace of codimension 1, even in the 2-dimensional case.



Approximate Voronoi diagram of a set of points. Notice the blended colors in the fuzzy boundary of the Voronoi cells.

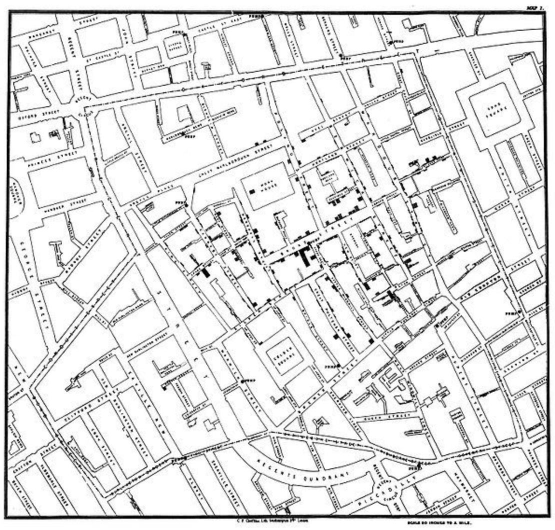
A weighted Voronoi diagram is the one in which the function of a pair of points to define a Voronoi cell is a distance function modified by multiplicative or additive weights assigned to generator points. In contrast to the case of Voronoi cells defined using a distance which is a metric, in this case some of the Voronoi cells may be empty. A power diagram is a type of Voronoi diagram defined from a set of circles using the power distance; it can also be thought of as a weighted Voronoi diagram in which a weight defined from the radius of each circle is added to the squared distance from the circle's center.*[8]

The Voronoi diagram of n points in d-dimensional space requires $O\left(n^{\left\lceil \frac{1}{2}d \right\rceil}\right)$ storage space. Therefore, Voronoi diagrams are often not feasible for $d > 2$. An alternative is to use approximate Voronoi diagrams, where the Voronoi cells have a fuzzy boundary, which can be approximated.*[9] Another alternative is when any site is a fuzzy circle and as a result the cells become fuzzy too.*[10]

Voronoi diagrams are also related to other geometric structures such as the medial axis (which has found applications in image segmentation, optical character recognition, and other computational applications), straight skeleton, and zone diagrams. Besides points, such diagrams use lines and polygons as seeds. By augmenting the diagram with line segments that connect to nearest points on the seeds, a planar subdivision of the environment is obtained.*[11] This structure can be used as a navigation mesh for path-finding through large spaces. The navigation mesh has been generalized to support 3D multi-layered environments, such as an airport or a multi-storey building.*[12]

4.7.9 Applications

- In astrophysics, Voronoi diagrams are used to generate adaptative smoothing zones on images, adding signal fluxes on each one. The main objective for these procedures is to maintain a relatively constant signal-to-noise ratio on all the image.



John Snow's original diagram

- In epidemiology, Voronoi diagrams can be used to correlate sources of infections in epidemics. One of the early applications of Voronoi diagrams was implemented by John Snow to study the 1854 Broad Street cholera outbreak in Soho, England. He showed the correlation between areas on the map of London using a particular water pump, and the areas with most deaths due to the outbreak.
- A point location data structure can be built on top of the Voronoi diagram in order to answer nearest

neighbor queries, where one wants to find the object that is closest to a given query point. Nearest neighbor queries have numerous applications. For example, one might want to find the nearest hospital, or the most similar object in a database. A large application is vector quantization, commonly used in data compression.

- In geometry, Voronoi diagrams can be used to find the largest empty circle amid a set of points, and in an enclosing polygon; e.g. to build a new supermarket as far as possible from all the existing ones, lying in a certain city.
- Voronoi diagrams together with farthest-point Voronoi diagrams are used for efficient algorithms to compute the roundness of a set of points.*[6]
- The Voronoi approach is also put to good use in the evaluation of circularity/roundness while assessing the dataset from a coordinate-measuring machine.
- In aviation, Voronoi diagrams are superimposed on oceanic plotting charts to identify the nearest airfield for in-flight diversion, as an aircraft progresses through its flight plan.
- In networking, Voronoi diagrams can be used in derivations of the capacity of a wireless network.
- In hydrology, Voronoi diagrams are used to calculate the rainfall of an area, based on a series of point measurements. In this usage, they are generally referred to as Thiessen polygons.
- In ecology, Voronoi diagrams are used to study the growth patterns of forests and forest canopies, and may also be helpful in developing predictive models for forest fires.
- In architecture, Voronoi patterns were the basis for the winning entry for redevelopment of The Arts Centre Gold Coast.*[13]
- In computational chemistry, Voronoi cells defined by the positions of the nuclei in a molecule are used to compute atomic charges. This is done using the Voronoi deformation density method.
- In polymer physics, Voronoi diagrams can be used to represent free volumes of polymers.
- In materials science, polycrystalline microstructures in metallic alloys are commonly represented using Voronoi tessellations. In solid state physics, the Wigner-Seitz cell is the Voronoi tessellation of a solid, and the Brillouin zone is the Voronoi tessellation of reciprocal (wave number) space of crystals which have the symmetry of a space group.
- In mining, Voronoi polygons are used to estimate the reserves of valuable materials, minerals, or other resources. Exploratory drillholes are used as the set of points in the Voronoi polygons.
- In computer graphics, Voronoi diagrams are used to calculate 3D shattering / fracturing geometry patterns. It is also used to procedurally generate organic or lava-looking textures.
- In autonomous robot navigation, Voronoi diagrams are used to find clear routes. If the points are obstacles, then the edges of the graph will be the routes furthest from obstacles (and theoretically any collisions).
- In machine learning, Voronoi diagrams are used to do 1-NN classifications.*[14]
- In biology, Voronoi diagrams are used to model a number of different biological structures, including cells*[15] and bone microarchitecture.*[16]
- In user interface development, Voronoi patterns can be used to compute the best hover state for a given point.*[17]
- In computational fluid dynamics, the Voronoi tessellation of a set of points can be used to define the computational domains used in finite volume methods, e.g. as in the moving-mesh cosmology code AREPO.*[18]

4.7.10 See also

Algorithms

Direct algorithms:

- Fortune's algorithm, an O(n log(n)) algorithm for generating a Voronoi diagram from a set of points in a plane.
- Lloyd's algorithm, aka k-means clustering, produces a Voronoi tessellation in a space of arbitrary dimensions

Starting with a Delaunay triangulation (obtain the dual):

- Bowyer–Watson algorithm, an O(n log(n)) to O(n^2) algorithm for generating a Delaunay triangulation in any number of dimensions, from which the Voronoi diagram can be obtained.

Related subjects

- Centroidal Voronoi tessellation
- Computational geometry
- Delaunay triangulation
- Mathematical diagram
- Natural neighbor interpolation

- Nearest neighbor search
- Nearest-neighbor interpolation
- Voronoi pole
- Power diagram

4.7.11 Notes

[1] Franz Aurenhammer (1991). *Voronoi Diagrams – A Survey of a Fundamental Geometric Data Structure*. ACM Computing Surveys, 23(3):345–405, 1991

[2] Atsuyuki Okabe, Barry Boots, Kokichi Sugihara & Sung Nok Chiu (2000). *Spatial Tessellations – Concepts and Applications of Voronoi Diagrams*. 2nd edition. John Wiley, 2000, 671 pages ISBN 0-471-98635-6

[3] Q.T.Tran, D.Tainar and M.Safar (2009) "Transactions on Large-Scale Data- and Knowledge-Centered Systems" , pag357. ISBN 9783642037214.

[4] Daniel Reem, *An algorithm for computing Voronoi diagrams of general generators in general normed spaces, In Proceedings of the sixth International Symposium on Voronoi Diagrams in science and engineering (ISVD 2009), 2009, pp. 144–152*

[5] Daniel Reem, *The geometric stability of Voronoi diagrams with respect to small changes of the sites*, Full version: arXiv 1103.4125 (2011), Extended abstract in Proceedings of the 27th Annual ACM Symposium on Computational Geometry (SoCG 2011), pp. 254–263

[6] Mark de Berg; Marc van Kreveld; Mark Overmars; Otfried Schwarzkopf (2008). *Computational Geometry* (Third ed.). Springer-Verlag. 7.4 Farthest-Point Voronoi Diagrams. Includes a description of the algorithm.

[7] Skyum, Sven (18 February 1991). "A simple algorithm for computing the smallest enclosing circle" . *Information Processing Letters* **37** (3): 121–125. doi:10.1016/0020-0190(91)90030-L., contains a simple algorithm to compute the farthest-point Voronoi diagram.

[8] Edelsbrunner, Herbert (1987), "13.6 Power Diagrams" , *Algorithms in Combinatorial Geometry*, EATCS Monographs on Theoretical Computer Science **10**, Springer-Verlag, pp. 327–328.

[9] S. Arya, T. Malamatos, and D. M. Mount, Space-Efficient Approximate Voronoi Diagrams, Proc. 34th ACM Symp. on Theory of Computing (STOC 2002), pp. 721–730.

[10] Jooyandeh, Mohammadreza; Mohades, Ali; Mirzakhah, Maryam (2009). "Uncertain Voronoi Diagram" (PDF). *Information Processing Letters* (Elsevier) **109** (13): 709–712. doi:10.1016/j.ipl.2009.03.007.

[11] Geraerts, Roland (2010), *Planning Short Paths with Clearance using Explicit Corridors* (PDF), International Conference on Robotics and Automation, IEEE, pp. 1997–2004.

[12] van Toll, Wouter G.; Cook IV, Atlas F.; Geraerts, Roland (2011), *Navigation Meshes for Realistic Multi-Layered Environments* (PDF), International Conference on Intelligent Robots and Systems, IEEE/RSJ, pp. 3526–3532.

[13] "GOLD COAST CULTURAL PRECINCT" . ARM Architecture.

[14] Tom M. Mitchell (1997). *Machine Learning* (International Edition 1997 ed.). McGraw-Hill. p. 233. ISBN 0-07-042807-7.

[15] Martin Bock (2009). "Generalized Voronoi Tessellation as a Model of Two-dimensional Cell Tissue Dynamics" .

[16] Hui Li (2012). "Spatial Modeling of Bone Microarchitecture" .

[17] "User Interface Algorithms" .

[18] Springel, Volker (2010). "E pur si muove: Galilean-invariant cosmological hydrodynamical simulations on a moving mesh" . *MNRAS* **401** (2): 791–851. doi:10.1111/j.1365-2966.2009.15715.x.

4.7.12 References

- G. Lejeune Dirichlet (1850). "Über die Reduktion der positiven quadratischen Formen mit drei unbestimmten ganzen Zahlen" . *Journal für die Reine und Angewandte Mathematik* **40**: 209–227.
- Voronoi, Georgy (1908). "Nouvelles applications des paramètres continus à la théorie des formes quadratiques" . *Journal für die Reine und Angewandte Mathematik* **133** (133): 97–178. doi:10.1515/crll.1908.133.97.
- Atsuyuki Okabe, Barry Boots, Kokichi Sugihara & Sung Nok Chiu (2000). *Spatial Tessellations – Concepts and Applications of Voronoi Diagrams*. 2nd edition. John Wiley, 2000, 671 pages, ISBN 0-471-98635-6
- Franz Aurenhammer, Rolf Klein & Der-Tsai Lee (2013) "Voronoi Diagrams and Delaunay Triangulations" . World Scientific, 2013, 337 pages, ISBN 978-9814447638
- Bowyer, Adrian (1981). "Computing Dirichlet tessellations" . *Comput. J.* **24** (2): 162–166. doi:10.1093/comjnl/24.2.162.
- Reem, Daniel (2009). "An algorithm for computing Voronoi diagrams of general generators in general normed spaces" . *Proceedings of the sixth International Symposium on Voronoi Diagrams in science and engineering (ISVD 2009)*. pp. 144–152. doi:10.1109/ISVD.2009.23.
- Daniel Reem (2011). *The geometric stability of Voronoi diagrams with respect to small changes of the sites*. Full version: arXiv 1103.4125 (2011),

Extended abstract: in Proceedings of the 27th Annual ACM Symposium on Computational Geometry (SoCG 2011), pp. 254–263.

- Watson, David F. (1981). "Computing the *n*-dimensional Delaunay tessellation with application to Voronoi polytopes" . *Comput. J.* **24** (2): 167–172. doi:10.1093/comjnl/24.2.167.
- Mark de Berg; Marc van Kreveld; Mark Overmars; and Otfried Schwarzkopf (2000). *Computational Geometry* (2nd revised ed.). Springer-Verlag. ISBN 3-540-65620-0. Chapter 7: Voronoi Diagrams: pp. 147–163. Includes a description of Fortune's algorithm.
- Rolf Klein (1989). *Abstract voronoi diagrams and their applications*. Lecture Notes in Computer Science **333**. Springer-Verlag. pp. 148–157. doi:10.1007/3-540-50335-8_31. ISBN 3-540-52055-4.

4.7.13 External links

- Real time interactive Voronoi / Delaunay diagrams with draggable points, Java 1.0.2, 1996–1997
- Real time interactive Voronoi and Delaunay diagrams with source code
- Interactive Voronoi diagrams with Natural Neighbor Interpolation visualization (in WebGL)
- Demo for various metrics
- Mathworld on Voronoi diagrams
- Qhull for computing the Voronoi diagram in 2-d, 3-d, etc.
- Voronoi Diagrams: Applications from Archaeology to Zoology
- Voronoi Diagrams in CGAL, the Computational Geometry Algorithms Library
- Voronoi Web Site : using Voronoi diagrams for spatial analysis
- More discussions and picture gallery on centroidal Voronoi tessellations
- Voronoi Diagrams by Ed Pegg, Jr., Jeff Bryant, and Theodore Gray, Wolfram Demonstrations Project.
- Nice explanation of voronoi diagram and visual implementation of fortune's algorithm
- A Voronoi diagram on a sphere
- Plot a Voronoi diagram with Mathematica
- Voronoi software for shattering 3D geometry
- Hand-drawing Voronoi diagrams
- Overlaid Voronoi diagram of the United States based on state capitals
- Overlaid Voronoi diagram of the world based on national capitals

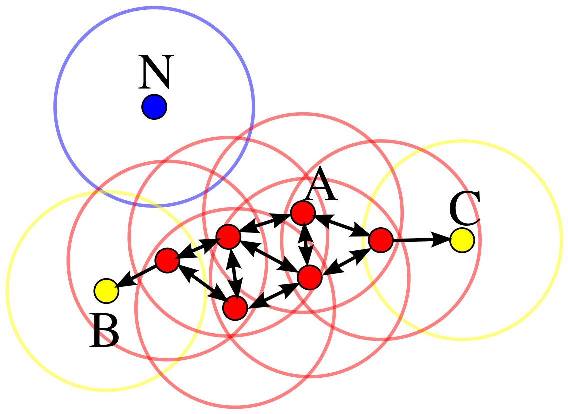
Chapter 5

Density Models

5.1 DBSCAN

Density-based spatial clustering of applications with noise (**DBSCAN**) is a data clustering algorithm proposed by Martin Ester, Hans-Peter Kriegel, Jörg Sander and Xiaowei Xu in 1996.*[1] It is a density-based clustering algorithm: given a set of points in some space, it groups together points that are closely packed together (points with many nearby neighbors), marking as outliers points that lie alone in low-density regions (whose nearest neighbors are too far away). DBSCAN is one of the most common clustering algorithms and also most cited in scientific literature.*[2]

In 2014, the algorithm was awarded the test of time award (an award given to algorithms which have received substantial attention in theory and practice) at the leading data mining conference, KDD.*[3]



In this diagram, minPts = 3. Point A and the other red points are core points, because at least three points surround it in an ε radius. Because they are all reachable from one another, they form a single cluster. Points B and C are not core points, but are reachable from A (via other core points) and thus belong to the cluster as well. Point N is a noise point that is neither a core point nor density-reachable.

5.1.1 Preliminaries

Consider a set of points in some space to be clustered. For the purpose of DBSCAN clustering, the points are classified as *core points*, (*density-*)*reachable points* and *outliers*, as follows:

- A point is a core point if at least minPts points are within distance ε of it, and those points are said to be *directly reachable* from p. No points are reachable from a non-core point.
- A point q is reachable from p if there is a path p_1, ..., p_n with $p_1 = p$ and $p_n = q$, where each p_{i+1} is directly reachable from p_i (so all the points on the path must be core points, with the possible exception of q).
- All points not reachable from any other point are outliers.

Now if p is a core point, then it forms a *cluster* together with all points (core or non-core) that are reachable from it. Each cluster contains at least one core point; non-core points can be part of a cluster, but they form its "edge" , since they cannot be used to reach more points.

Reachability is not a symmetric relation since, by definition, no point may be reachable from a non-core point, regardless of distance (so a non-core point may be reachable, but nothing can be reached from it). Therefore a further notion of *connectedness* is needed to formally define the extent of the clusters found by DBSCAN. Two points p and q are density-connected if there is a point o such that both p and q are density-reachable from o. Density-connectedness *is* symmetric.

A cluster then satisfies two properties:

1. All points within the cluster are mutually density-connected.
2. If a point is density-reachable from any point of the cluster, it is part of the cluster as well.

5.1.2 Algorithm

DBSCAN requires two parameters: ε (eps) and the minimum number of points required to form a dense region*[lower-alpha 1] (minPts). It starts with an arbitrary starting point that has not been visited. This point's ε-neighborhood is retrieved, and if it contains sufficiently many points, a cluster is started. Otherwise, the point is labeled as noise. Note that this point might later be found in a sufficiently sized ε-environment of a different point and hence be made part of a cluster.

If a point is found to be a dense part of a cluster, its ε-neighborhood is also part of that cluster. Hence, all points that are found within the ε-neighborhood are added, as is their own ε-neighborhood when they are also dense. This process continues until the density-connected cluster is completely found. Then, a new unvisited point is retrieved and processed, leading to the discovery of a further cluster or noise.

In pseudocode, the algorithm can be expressed as follows:

DBSCAN(D, eps, MinPts) C = 0 for each unvisited point P in dataset D mark P as visited NeighborPts = regionQuery(P, eps) if sizeof(NeighborPts) < MinPts mark P as NOISE else C = next cluster expandCluster(P, NeighborPts, C, eps, MinPts) expandCluster(P, NeighborPts, C, eps, MinPts) add P to cluster C for each point P' in NeighborPts if P' is not visited mark P' as visited NeighborPts' = regionQuery(P', eps) if sizeof(NeighborPts') >= MinPts NeighborPts = NeighborPts joined with NeighborPts' if P' is not yet member of any cluster add P' to cluster C regionQuery(P, eps) return all points within P's eps-neighborhood (including P)

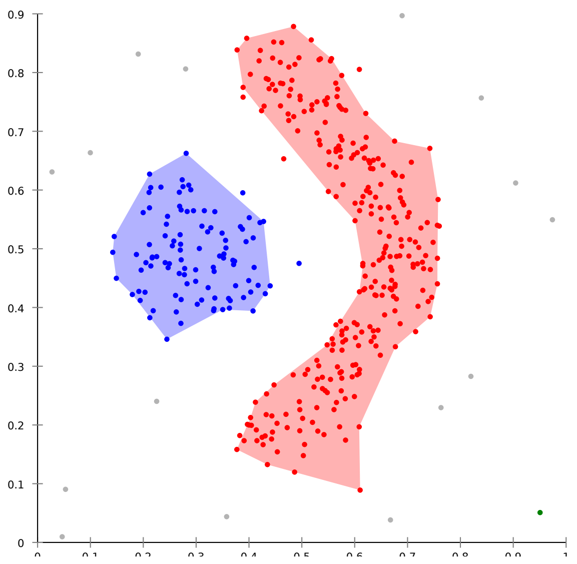
5.1.3 Complexity

DBSCAN visits each point of the database, possibly multiple times (e.g., as candidates to different clusters). For practical considerations, however, the time complexity is mostly governed by the number of regionQuery invocations. DBSCAN executes exactly one such query for each point, and if an indexing structure is used that executes such a neighborhood query in $O(\log n)$, an overall runtime complexity of $O(n \log n)$ is obtained. Without the use of an accelerating index structure, the run time complexity is $O(n^2)$. Often the distance matrix of size $(n^2-n)/2$ is materialized to avoid distance recomputations. This however also needs $O(n^2)$ memory, whereas a non-matrix based implementation only needs $O(n)$ memory.

5.1.4 Advantages

1. DBSCAN does not require one to specify the number of clusters in the data a priori, as opposed to k-means.

2. DBSCAN can find arbitrarily shaped clusters. It can even find a cluster completely surrounded by (but not connected to) a different cluster. Due to the MinPts parameter, the so-called single-link effect (different clusters being connected by a thin line of points) is reduced.



DBSCAN can find non-linearly separable clusters. This dataset cannot be adequately clustered with k-means or Gaussian Mixture EM clustering.

3. DBSCAN has a notion of noise, and is robust to outliers.

4. DBSCAN requires just two parameters and is mostly insensitive to the ordering of the points in the database. (However, points sitting on the edge of two different clusters might swap cluster membership if the ordering of the points is changed, and the cluster assignment is unique only up to isomorphism.)

5. DBSCAN is designed for use with databases that can accelerate region queries, e.g. using an R* tree.

5.1.5 Disadvantages

1. DBSCAN is not entirely deterministic: border points that are reachable from more than one cluster can be part of either cluster. Fortunately, this situation does not arise often, and has little impact on the clustering result: both on core points and noise points, DBSCAN is deterministic. DBSCAN**[4] is a variation that treats border points as noise, and this way achieves a fully deterministic result as well as a more consistent statistical interpretation of density-connected components.

2. The quality of DBSCAN depends on the distance measure used in the function regionQuery(P,ε). The most common distance metric used is Euclidean distance. Especially for high-dimensional data, this metric can be rendered almost useless due to the so-called "Curse of dimensionality", making it difficult to find an appropriate value for ε. This effect, however, is also present in any other algorithm based on Euclidean distance.

3. DBSCAN cannot cluster data sets well with large differences in densities, since the minPts-ε combination cannot then be chosen appropriately for all clusters.

See the section below on extensions for algorithmic modifications to handle these issues.

5.1.6 Parameter estimation

Every data mining task has the problem of parameters. Every parameter influences the algorithm in specific ways. For DBSCAN, the parameters ε and *minPts* are needed. The parameters must be specified by the user. Ideally, the value of ε is given by the problem to solve (e.g. a physical distance), and *minPts* is then the desired minimum cluster size.*[lower-alpha 1]

- *MinPts*: As a rule of thumb, a minimum *minPts* can be derived from the number of dimensions *D* in the data set, as *minPts*≥*D+1*. The low value of *minPts=1* does not make sense, as then every point on its own will already be a cluster. With *minPts=2*, the result will be the same as of hierarchical clustering with the single link metric, with the dendrogram cut at height ε. However, larger values are usually better for data sets with noise and will yield more significant clusters. The larger the data set, the larger the value of *minPts* should be chosen.
- ε: The value for ε can then be chosen by using a k-distance graph, plotting the distance to the *k=minPts* nearest neighbor. Good values of ε are where this plot shows a strong bend: if ε is chosen too small, a large part of the data will not be clustered; whereas for a too high value of ε, clusters will merge and the majority of objects will be in the same cluster.

OPTICS can be seen as a generalization of DBSCAN that replaces the ε parameter with a maximum value that mostly affects performance. *MinPts* then essentially becomes the minimum cluster size to find. While the algorithm is much easier to parameterize than DBSCAN, the results are a bit more difficult to use, as it will usually produce a hierarchical clustering instead of the simple data partitioning that DBSCAN produces.

Recently, one of the original authors of DBSCAN has revisited DBSCAN and OPTICS, and published a refined version of hierarchical DBSCAN (HDBSCAN*),*[4]*[5] which no longer has the notion of border points.

5.1.7 Extensions

Generalized DBSCAN (GDBSCAN)*[6]*[7] is a generalization by the same authors to arbitrary "neighborhood" and "dense" predicates. The ε and minpts parameters are removed from the original algorithm and moved to the predicates. For example on polygon data, the "neighborhood" could be any intersecting polygon, whereas the density predicate uses the polygon areas instead of just the object count.

Various extensions to the DBSCAN algorithm have been proposed, including methods for parallelization, parameter estimation and support for uncertain data. The basic idea has been extended to hierarchical clustering by the OPTICS algorithm. DBSCAN is also used as part of subspace clustering algorithms like PreDeCon and SUBCLU. HDBSCAN*[4] is a hierarchical version of DBSCAN which is also faster than OPTICS, from which a flat partition consisting of most prominent clusters can be extracted from the hierarchy.*[5]

5.1.8 Availability

- ELKI offers an implementation of DBSCAN as well as GDBSCAN and other variants. This implementation can use various index structures for sub-quadratic runtime and supports arbitrary distance functions and arbitrary data types, but it may be outperformed by low-level optimized (and specialized) implementations on small data sets.
- scikit-learn includes a Python implementation of DBSCAN for arbitrary Minkowski metrics, which can be accelerated using kd-trees and ball trees but which uses worst-case quadratic memory.
- GNU R contains DBSCAN in the "fpc" package with support for arbitrary distance functions via distance matrices. However it does not have index support (and thus has quadratic runtime and memory complexity) and is rather slow due to the R interpreter, whereas other clustering algorithms are optimized C and Fortran implementations.
- SPMF offers a minimalistic GPL-V3 Java implementation of the DBSCAN algorithm for Euclidean distance only with KD-Tree support.
- Weka contains (as an optional package in latest versions) a basic implementation of DBSCAN that run in quadratic time and linear memory.

5.1.9 See also

- OPTICS algorithm: a generalization of DBSCAN to multiple ranges, effectively replacing the ε parameter with a maximum search radius.

5.1.10 Notes

[1] While minPts intuitively is the minimum cluster size, in some cases DBSCAN *can* produce smaller clusters. A DBSCAN cluster consists of at least *one core point*. As other points may be border points to more than one cluster, there is no guarantee that at least minPts points are included in every cluster.

5.1.11 References

[1] Ester, Martin; Kriegel, Hans-Peter; Sander, Jörg; Xu, Xiaowei (1996). Simoudis, Evangelos; Han, Jiawei; Fayyad, Usama M., eds. *A density-based algorithm for discovering clusters in large spatial databases with noise*. Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96). AAAI Press. pp. 226–231. ISBN 1-57735-004-9. CiteSeerX: 10.1.1.71.1980.

[2] Most cited data mining articles according to Microsoft academic search; DBSCAN is on rank 24, when accessed on: 4/18/2010

[3] "2014 SIGKDD Test of Time Award" . ACM SIGKDD. 2014-08-18. Retrieved 2014-08-22.

[4] Campello, R. J. G. B.; Moulavi, D.; Sander, J. (2013). *Density-Based Clustering Based on Hierarchical Density Estimates*. Proceedings of the 17th Pacific-Asia Conference on Knowledge Discovery in Databases, PAKDD 2013. Lecture Notes in Computer Science **7819**. p. 160. doi:10.1007/978-3-642-37456-2_14. ISBN 978-3-642-37455-5.

[5] Campello, R. J. G. B.; Moulavi, D.; Zimek, A.; Sander, J. (2013). "A framework for semi-supervised and unsupervised optimal extraction of clusters from hierarchies" . *Data Mining and Knowledge Discovery* **27** (3): 344. doi:10.1007/s10618-013-0311-4.

[6] Sander, Jörg; Ester, Martin; Kriegel, Hans-Peter; Xu, Xiaowei (1998). "Density-Based Clustering in Spatial Databases: The Algorithm GDBSCAN and Its Applications" . *Data Mining and Knowledge Discovery* (Berlin: Springer-Verlag) **2** (2): 169–194. doi:10.1023/A:1009745219419.

[7] Sander, Jörg (1998). *Generalized Density-Based Clustering for Spatial Data Mining*. München: Herbert Utz Verlag. ISBN 3-89675-469-6.

Further reading

- Arlia, Domenica; Coppola, Massimo. "Experiments in Parallel Clustering with DBSCAN" . *Euro-Par 2001: Parallel Processing: 7th International Euro-Par Conference Manchester, UK August 28–31, 2001, Proceedings*. Springer Berlin.
- Kriegel, Hans-Peter; Kröger, Peer; Sander, Jörg; Zimek, Arthur (2011). "Density-based Clustering" . *WIREs Data Mining and Knowledge Discovery* **1** (3): 231–240. doi:10.1002/widm.30.

5.2 OPTICS algorithm

Ordering points to identify the clustering structure (**OPTICS**) is an algorithm for finding density-based clusters in spatial data. It was presented by Mihael Ankerst, Markus M. Breunig, Hans-Peter Kriegel and Jörg Sander.*[1] Its basic idea is similar to DBSCAN,*[2] but it addresses one of DBSCAN's major weaknesses: the problem of detecting meaningful clusters in data of varying density. In order to do so, the points of the database are (linearly) ordered such that points which are spatially closest become neighbors in the ordering. Additionally, a special distance is stored for each point that represents the density that needs to be accepted for a cluster in order to have both points belong to the same cluster. This is represented as a dendrogram.

5.2.1 Basic idea

Like DBSCAN, OPTICS requires two parameters: ε , which describes the maximum distance (radius) to consider, and $MinPts$, describing the number of points required to form a cluster. A point p is a *core point* if at least $MinPts$ points are found within its ε -neighborhood $N_\varepsilon(p)$. Contrary to DBSCAN, OPTICS also considers points that are part of a more densely packed cluster, so each point is assigned a *core distance* that describes the distance to the $MinPts$ th closest point:

$$\text{core-dist}_{\varepsilon,MinPts}(p) = \begin{cases} \text{UNDEFINED} & \text{if} |N_\varepsilon(\\ MinPts \text{ to distance smallest -th} N_\varepsilon(p) & \text{otherw} \end{cases}$$

The *reachability-distance* of another point o from a point p is the distance between o and p , or the core distance of p :

$$\text{reachability-dist}_{\varepsilon,MinPts}(o,p) = \begin{cases} \text{UNDEFINED} \\ \max(\text{core-dist}_{\varepsilon,MinPts}(p), \text{dist}(p,o)) \end{cases}$$

If p and o are nearest neighbors, this is the $\varepsilon' < \varepsilon$ we need to assume in order to have p and o belong to the same cluster.

Both the core-distance and the reachability-distance are undefined if no sufficiently dense cluster (w.r.t. ε) is available. Given a sufficiently large ε , this will never happen, but then every ε -neighborhood query will return the entire database, resulting in $O(n^2)$ runtime. Hence, the ε parameter is required to cut off the density of clusters that is no longer considered to be interesting and to speed up the algorithm this way.

The parameter ε is, strictly speaking, not necessary. It can simply be set to the maximum possible value. When a spatial index is available, however, it does play a practical role with regards to complexity. It is often claimed that OPTICS abstracts from DBSCAN by removing this parameter, at least to the extent of only having to give the maximum value.

5.2.2 Pseudocode

The basic approach of OPTICS is similar to DBSCAN, but instead of maintaining a set of known, but so far unprocessed cluster members, a priority queue (e.g. using an indexed heap) is used.

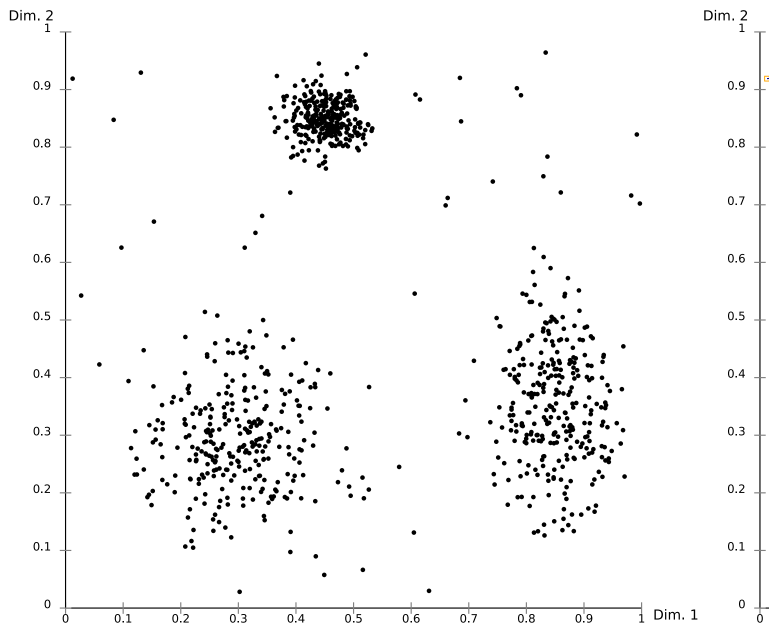
OPTICS(DB, eps, MinPts) for each point p of DB p.reachability-distance = UNDEFINED for each unprocessed point p of DB N = getNeighbors(p, eps) mark p as processed output p to the ordered list if (core-distance(p, eps, Minpts) != UNDEFINED) Seeds = empty priority queue update(N, p, Seeds, eps, Minpts) for each next q in Seeds N' = getNeighbors(q, eps) mark q as processed output q to the ordered list if (core-distance(q, eps, Minpts) != UNDEFINED) update(N', q, Seeds, eps, Minpts)

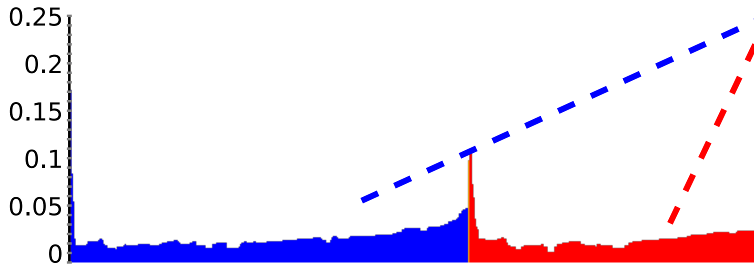
In update(), the priority queue Seeds is updated with the ε -neighborhood of p and q , respectively:

update(N, p, Seeds, eps, Minpts) coredist = core-distance(p, eps, MinPts) for each o in N if (o is not processed) new-reach-dist = max(coredist, dist(p,o)) if (o.reachability-distance == UNDEFINED) // o is not in Seeds o.reachability-distance = new-reach-dist Seeds.insert(o, new-reach-dist) else // o in Seeds, check for improvement if (new-reach-dist < o.reachability-distance) o.reachability-distance = new-reach-dist Seeds.move-up(o, new-reach-dist)

OPTICS hence outputs the points in a particular ordering, annotated with their smallest reachability distance (in the original algorithm, the core distance is also exported, but this is not required for further processing).

5.2.3 Extracting the clusters





Using a *reachability-plot* (a special kind of dendrogram), the hierarchical structure of the clusters can be obtained easily. It is a 2D plot, with the ordering of the points as processed by OPTICS on the x-axis and the reachability distance on the y-axis. Since points belonging to a cluster have a low reachability distance to their nearest neighbor, the clusters show up as valleys in the reachability plot. The deeper the valley, the denser the cluster.

The image above illustrates this concept. In its upper left area, a synthetic example data set is shown. The upper right part visualizes the spanning tree produced by OPTICS, and the lower part shows the reachability plot as computed by OPTICS. Colors in this plot are labels, and not computed by the algorithm; but it is well visible how the valleys in the plot correspond to the clusters in above data set. The yellow points in this image are considered noise, and no valley is found in their reachability plot. They will usually not be assigned to clusters except the omnipresent "all data" cluster in a hierarchical result.

Extracting clusters from this plot can be done manually by selecting a range on the x-axis after visual inspection, by selecting a threshold on the y-axis (the result will then be similar to a DBSCAN clustering result with the same ε and minPts parameters; here a value of 0.1 may yield good results), or by different algorithms that try to detect the valleys by steepness, knee detection, or local maxima. Clusterings obtained this way usually are hierarchical, and cannot be achieved by a single DBSCAN run.

5.2.4 Complexity

Like DBSCAN, OPTICS processes each point once, and performs one ε -neighborhood query during this processing. Given a spatial index that grants a neighborhood query in $O(\log n)$ runtime, an overall runtime of $O(n \cdot \log n)$ is obtained. The authors of the original OPTICS paper report an actual constant slowdown factor of 1.6 compared to DBSCAN. Note that the value of ε might heavily influence the cost of the algorithm, since a value too large might raise the cost of a neighborhood query to linear complexity.

In particular, choosing $\varepsilon > \max_{x,y} d(x,y)$ (larger than the maximum distance in the data set) is possible, but will obviously lead to quadratic complexity, since every neighborhood query will return the full data set. Even when no spatial index is available, this comes at additional cost in managing the heap. Therefore, ε should be chosen appropriately for the data set.

5.2.5 Extensions

OPTICS-OF*[3] is an outlier detection algorithm based on OPTICS. The main use is the extraction of outliers from an existing run of OPTICS at low cost compared to using a different outlier detection method.

DeLi-Clu,*[4] Density-Link-Clustering combines ideas from single-linkage clustering and OPTICS, eliminating the ε parameter and offering performance improvements over OPTICS.

HiSC*[5] is a hierarchical subspace clustering (axis-parallel) method based on OPTICS.

HiCO*[6] is a hierarchical correlation clustering algorithm based on OPTICS.

DiSH*[7] is an improvement over HiSC that can find more complex hierarchies.

FOPTICS*[8] is a faster implementation using random projections.

5.2.6 Availability

Implementations of OPTICS, OPTICS-OF, DeLi-Clu, HiSC, HiCO and DiSH are available in the ELKI data mining framework (with index acceleration). An incomplete and slow implementation can be found in the Weka extensions. The MRC National Institute for Medical Research provides a C reimplementation of OPTICS without index support.

5.2.7 References

[1] Mihael Ankerst, Markus M. Breunig, Hans-Peter Kriegel, Jörg Sander (1999). *OPTICS: Ordering Points To Identify the Clustering Structure*. ACM SIGMOD international conference on Management of data. ACM Press. pp. 49–60.

[2] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu (1996). Evangelos Simoudis, Jiawei Han, Usama M. Fayyad, ed. *A density-based algorithm for discovering clusters in large spatial databases with noise*. Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96). AAAI Press. pp. 226–231. ISBN 1-57735-004-9.

[3] Markus M. Breunig, Hans-Peter Kriegel, Raymond T. Ng and Jörg Sander (1999). "OPTICS-OF: Identifying Local Outliers" . *Principles of Data Mining and Knowledge Discovery*. Springer-Verlag. pp. 262–270. doi:10.1007/b72280. ISBN 978-3-540-66490-1.

[4] Achtert, E.; Böhm, C.; Kröger, P. (2006). "DeLi-Clu: Boosting Robustness, Completeness, Usability, and Efficiency of Hierarchical Clustering by a Closest Pair Ranking" . *LNCS: Advances in Knowledge Discovery and Data Mining*. Lecture Notes in Computer Science **3918**: 119–128. doi:10.1007/11731139_16. ISBN 978-3-540-33206-0.

[5] Achtert, E.; Böhm, C.; Kriegel, H. P.; Kröger, P.; Müller-Gorman, I.; Zimek, A. (2006). "Finding Hierarchies of Subspace Clusters" . *LNCS: Knowledge Discovery in Databases: PKDD 2006*. Lecture Notes in Computer Science **4213**: 446–453. doi:10.1007/11871637_42. ISBN 978-3-540-45374-1.

[6] Achtert, E.; Böhm, C.; Kröger, P.; Zimek, A. (2006). "Mining Hierarchies of Correlation Clusters" . *Proc. 18th International Conference on Scientific and Statistical Database Management (SSDBM)*: 119–128. doi:10.1109/SSDBM.2006.35. ISBN 0-7695-2590-3.

[7] Achtert, E.; Böhm, C.; Kriegel, H. P.; Kröger, P.; Müller-Gorman, I.; Zimek, A. (2007). "Detection and Visualization of Subspace Cluster Hierarchies" . *LNCS: Advances in Databases: Concepts, Systems and Applications*. Lecture Notes in Computer Science **4443**: 152–163. doi:10.1007/978-3-540-71703-4_15. ISBN 978-3-540-71702-7.

[8] Schneider, Johannes; Vlachos, Michail (2013). "Fast parameterless density-based clustering via random projections" . *22nd ACM International Conference on Information and Knowledge Management(CIKM)* (ACM).

5.3 Mean shift

Mean shift is a non-parametric feature-space analysis technique for locating the maxima of a density function, a so-called mode-seeking algorithm.*[1] Application domains include cluster analysis in computer vision and image processing.*[2]

5.3.1 History

The mean shift procedure was originally presented in 1975 by Fukunaga and Hostetler.*[3]

5.3.2 Overview

Mean shift is a procedure for locating the maxima of a density function given discrete data sampled from that function.*[1] It is useful for detecting the modes of this density.*[1] This is an iterative method, and we start with an initial estimate x . Let a kernel function $K(x_i - x)$ be given. This function determines the weight of nearby points for re-estimation of the mean. Typically a Gaussian kernel on the distance to the current estimate is used, $K(x_i - x) = e^{-c||x_i - x||^2}$. The weighted mean of the density in the window determined by K

$$m(x) = \frac{\sum_{x_i \in N(x)} K(x_i - x) x_i}{\sum_{x_i \in N(x)} K(x_i - x)}$$

where $N(x)$ is the neighborhood of x , a set of points for which $K(x) \neq 0$.

The mean-shift algorithm now sets $x \leftarrow m(x)$, and repeats the estimation until $m(x)$ converges.

5.3.3 Details

Let data be a finite set S embedded in the n-dimensional Euclidean space, X. Let K be a flat kernel that is the characteristic function of the λ -ball in X,

$$K(x) = \begin{cases} 1 & \text{if } \|x\| \leq \lambda \\ 0 & \text{if } \|x\| > \lambda \end{cases}$$

The difference $m(x) - x$ is called *mean shift* in Fukunaga and Hostetler.*[3] The repeated movement of data points to the sample means is called the *mean shift algorithm*. In each iteration of the algorithm, $s \leftarrow m(s)$ is performed for all $s \in S$ simultaneously. The first question, then, is how to estimate the density function given a sparse set of samples. One of the simplest approaches is to just smooth the data, e.g., by convolving it with a fixed kernel of width h ,

$$f(x) = \sum_i K(x - x_i) = \sum_i k\left(\frac{\|x - x_i\|^2}{h^2}\right)$$

where x_i are the input samples and $k(r)$ is the kernel function (or *Parzen window*). h is the only parameter in the algorithm and is called the bandwidth. This approach is known as *kernel density estimation* or the Parzen window technique. Once we have computed $f(x)$ from equation above, we can find its local maxima using gradient ascent or some other optimization technique. The problem with this "brute force" approach is that, for higher dimensions, it becomes computationally prohibitive to evaluate $f(x)$ over the complete search space. Instead, mean shift uses a variant of what is known in the optimization literature as *multiple restart gradient descent*. Starting at some guess for a local maximum, y_k , which can be a random input data point x_1 , mean shift computes the gradient of the density estimate $f(x)$ at y_k and takes an uphill step in that direction.

5.3.4 Types of kernels

Kernel definition: Let X be the n-dimensional Euclidean space, R^n . Denote the ith component of x by x_i . The norm of x is a non-negative number. $\|x\|^2 = x^T x$ A function K: $X \leftarrow R$ is said to be a kernel if there exists a profile, $k : [0, \infty] \rightarrow R$, such that

$K(x) = k(\|x\|^2)$ and

- k is non-negative.
- k is nonincreasing: $k(a) \geq k(b)$ if $a < b$.
- k is piecewise continuous and $\int_0^\infty k(r)\, dr < \infty$

The two frequently used kernels for mean shift are:

Flat kernel

$$F(x) = \begin{cases} 1 & \text{if } \|x\| \leq \lambda \\ 0 & \text{if } \|x\| > \lambda \end{cases}$$

Gaussian kernel

$$G(x) = c_{k,d} k(\|x\|^2)$$

where $c_{k,d}$, the normalization constant, makes G(x) integrate to one and $k(x)$ is called the *profile* of the kernel. It simplifies calculation in the case of multivariate data. The profile of the Gaussian kernel is: $e^{-1/2\|x\|^2}$ and therefore, the multivariate Gaussian kernel with the standard deviation σ , will be: $G(x) = \frac{1}{\sqrt{2\pi}\sigma^d} e^{-1/2 \frac{\|x\|^2}{\sigma^2}}$ where d is the number of dimensions. It's also worth mentioning that the standard deviation for the Gaussian kernel works as the bandwidth parameter, h

5.3.5 Applications

Clustering

Consider a set of points in two-dimensional space. Assume a circular window centered at C and having radius r as the kernel. Mean shift is a hill climbing algorithm which involves shifting this kernel iteratively to a higher density region until convergence. Every shift is defined by a mean shift vector. The mean shift vector always points toward the direction of the maximum increase in the density. At every iteration the kernel is shifted to the centroid or the mean of the points within it. The method of calculating this mean depends on the choice of the kernel. In this case if a Gaussian kernel is chosen instead of a flat kernel, then every point will first be assigned a weight which will decay exponentially as the distance from the kernel's center increases. At convergence, there will be no direction at which a shift can accommodate more points inside the kernel.

Tracking

The mean shift algorithm can be used for visual tracking. The simplest such algorithm would create a confidence map in the new image based on the color histogram of the object in the previous image, and use mean shift to find the peak of a confidence map near the object's old position. The confidence map is a probability density function on the new image, assigning each pixel of the new image a probability, which is the probability of the pixel color occurring in the object in the previous image. A few algorithms, such as ensemble tracking,*[4] CAMshift,*[5] expand on this idea.

Smoothing

Let x_i and $z_i, i = 1, ..., n,$ be the d-dimensional input and filtered image pixels in the joint spatial-range domain. For each pixel,

- Initialize $j = 1$ and $y_{i,1} = x_i$
- Compute $y_{i,j+1}$ according to $m(\cdot)$ until convergence, $y = y_{i,c}$.
- Assign $z_i = (x_i^s, y_{i,c}^r)$. The superscripts s and r denote the spatial and range components of a vector, respectively. The assignment specifies that the filtered data at the spatial location axis will have the range component of the point of convergence $y_{i,c}^r$.

5.3.6 Strengths

1. Mean shift is an application-independent tool suitable for real data analysis.
2. Does not assume any predefined shape on data clusters.
3. It is capable of handling arbitrary feature spaces.
4. The procedure relies on choice of a single parameter: bandwidth.
5. The bandwidth/window size 'h' has a physical meaning, unlike *k*-means.

5.3.7 Weaknesses

1. The selection of a window size is not trivial.
2. Inappropriate window size can cause modes to be merged, or generate additional "shallow" modes.
3. Often requires using adaptive window size.

5.3.8 Mean shift and *k*-means clustering

The mean shift clustering algorithm has two main drawbacks. First, the algorithm is calculation intensive; it requires in general $O(kN^2)$ operations, where N is the number of data points and k is the number of average iteration steps for each data point. Second, the mean shift algorithm relies on sufficient high data density with clear gradient to locate the cluster centers. In particular, the mean shift algorithm often fails to find appropriate clusters for so called data outliers, or those data points located between natural clusters.

The *k*-means algorithm does not have the above two problems. The *k*-means algorithm normally requires only $O(kN)$ operations, so that the *k*-means algorithm can be applied to relatively large dataset. However, *k*-means has two significant limitations. First, the *k*-means algorithm requires that the number of clusters to be pre-determined. In practise, it is often difficult to specify a priori an appropriate cluster number, resulting in some natural clusters being represented by multiple clusters found by the *k*-means algorithm. Second, the *k*-means algorithm is, in general, incapable of identifying non-convex clusters. The second limitation makes the *k*-means algorithm inadequate for complex non-linear data. These problems can be overcome, by simply combining the two algorithms mean shift and *k*-means together.*[6]

5.3.9 See also

- Kernel density estimation (KDE)
- Kernel (statistics)

5.3.10 References

[1] Cheng, Yizong (August 1995). "Mean Shift, Mode Seeking, and Clustering" . *IEEE Transactions on Pattern Analysis and Machine Intelligence* (IEEE) **17** (8): 790–799. doi:10.1109/34.400568.

[2] Comaniciu, Dorin; Peter Meer (May 2002). "Mean Shift: A Robust Approach Toward Feature Space Analysis" . *IEEE Transactions on Pattern Analysis and Machine Intelligence* (IEEE) **24** (5): 603–619. doi:10.1109/34.1000236.

[3] Fukunaga, Keinosuke; Larry D. Hostetler (January 1975). "The Estimation of the Gradient of a Density Function, with Applications in Pattern Recognition" . *IEEE Transactions on Information Theory* (IEEE) **21** (1): 32–40. doi:10.1109/TIT.1975.1055330. Retrieved 2008-02-29.

[4] Avidan, Shai (2005). "Ensemble Tracking". *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)* (San Diego, California: IEEE) **2**. ISBN 0-7695-2372-2.

[5] Emami, Ebrahim (2013). "Online failure detection and correction for CAMShift tracking algorithm". *2013 Iranian Conference on Machine Vision and Image Processing (MVIP)* (IEEE) **2**: 180–183.

[6] Li, James (2012). "Visualizing High Dimensional Data".

5.3.11 External links

Code implementations

- Scikit-learn library Numpy/Python implementation uses ball tree for efficient neighboring points lookup
- EDISON library. C++ implementation of mean-shift-based image segmentation. There is also a Matlab interface for EDISON.
- OpenCV contains mean-shift implementation via cvMeanShift Method
- Aiphial. Java-based mean-shift implementation for numeric data clustering and image segmentation
- Apache Mahout. An map-reduce based implementation of MeanShift clustering written on Apache Hadoop.
- CAMSHIFT project. A MATLAB implementation of CAMSHIFT algorithm.
- OTB MeanShift. A C++ implementation using the Orfeo Toolbox.
- ImageJ Plug-in. Image filtering using the mean shift filter.
- Mean-shift google code. A simple implementation of mean-shift as image filtering tool.

Short lessons

- A lesson from Prof. M. Shah on this topic

Chapter 6

Other

6.1 Biclustering

Biclustering, **block clustering** ,*[1] **co-clustering**, or **two-mode clustering** *[2] *[3] is a data mining technique which allows simultaneous clustering of the rows and columns of a matrix. The term was first introduced by Mirkin,*[4] although the technique was originally introduced much earlier*[4] (i.e., by J.A. Hartigan*[5]).

Given a set of m rows in n columns (i.e., an $m \times n$ matrix), the biclustering algorithm generates biclusters - a subset of rows which exhibit similar behavior across a subset of columns, or vice versa.

6.1.1 Development

Biclustering was originally introduced by J.A.Hartigan in 1972.*[6] The term biclustering was later used by Mirkin. This algorithm was not generalized until 2000 when Y.Cheng and G.M.Church proposed a biclustering algorithm based on variance and applied it to biological gene expression data.*[7] Their paper is still the most important literature in the gene expression biclustering field.

In 2001 and 2003, I.S.Dhillon put forward two algorithms applying biclustering to files and words. One version was based on bipartite spectral graph partitioning.*[8] The other was based on information theory. Dhillon assumed the loss of mutual information during biclustering was equal to the KL(Kullback-Leibler)-distance between P and Q. P means the distribution of files and feature words before biclustering. Q is the distribution after biclustering. KL-distance is for measuring the difference between two random distributions. KL=0 when the two distributions are the same and KL increases as the difference increases.*[9] Thus, the aim of the algorithm was to find the minimum KL-distance between P and Q. In 2004, A.Banerjee used a weighted-Bregman distance instead of KL-distance to design a biclustering algorithm which was suitable for any kind of matrix, unlike the KL-distance algoritm.*[10]

To cluster more than two types of objects, in 2005, Bekkerman expanded the mutual information in Dhillon' s theorem from a single pair into multiple pairs.

6.1.2 Complexity

The complexity of the biclustering problem depends on the exact problem formulation, and particularly on the merit function used to evaluate the quality of a given bicluster. However most interesting variants of this problem are NP-complete. NP-complete have two conditions. In the simple case that there is only element a_(i,j) either 0 or 1 in the binary matrix A, a bicluster is equal to a biclique in the corresponding bipartite graph. The maximum size bicluster is equivalent to maximum edge biclique in bipartite graph. In the complex case, the element in matrix A is used to compute the quality of a given bicluster and solve the more restricted version of the problem.*[11] It requires either large computational effort or the use of lossy heuristics to short-circuit the calculation.*[12]

6.1.3 Type of Bicluster

Different biclustering algorithms have different definitions of bicluster.*[12]

They are:

1. Bicluster with constant values (a),
2. Bicluster with constant values on rows (b) or columns (c),
3. Bicluster with coherent values (d, e).

1.Bicluster with constant values

When a biclustering algorithm tries to find a constant bicluster, the normal way for it is to reorder the rows and columns of the matrix so it can group together similar rows/columns and find biclusters with similar values. This method is OK when the data is tidy. But as the data can be noisy most of the times, so it can' t satisfy us. More sophisticated methods should be used. A perfect constant bicluster is a matrix(I,J) where all values a(i,j) are equal to μ. In real data, a(i,j) can be seen as n(i,j) +μ where n(i,j) is the noise. According to Hartigan' s algorithm, by spliting the original data matrix into a set of biclusters. Variance is used to compute constant biclusters. So a perfect bicluster is a matrix with variance zero. Also, in order to prevent the partitioning of the data matrix into biclusters with only one row and one column. Hartigan assumes that there are K biclusters within the data matrix. When the data matrix is partitioned into K biclusters, the algorithm ends.

2.Biclusters with constant values on rows or columns

This kind of biclusters can't be evaluated just by variance of its values. To finish the identification, the columns and the rows should be normalized at first. There are other algorithms, without normalization step, can find biclusters have rows and columns with different approaches.

3.Biclusters with coherent values

For biclusters with coherent values on rows and columns, an overall improvement over the algorithms for biclusters with constant values on rows or on columns should be considered. That means a sophisticated algorithm is needed. This algorithm may contain analysis of variance between groups, using co-variance between both rows and columns.In Cheng and Churchs' theorem, a bicluster is defined as a subset of rows and columns with almost the same score.the similarity score is used to measure the coherence of rows and columns.

The relationship between these cluster models and other types of clustering such as correlation clustering is discussed in.*[13]

6.1.4 Algorithms

There are many biclustering algorithms developed for bioinformatics, including: block clustering, CTWC (Coupled Two-Way Clustering), ITWC (Interrelated Two-Way Clustering), δ-bicluster, δ-pCluster, δ-pattern, FLOC, OPC, Plaid Model, OPSMs (Order-preserving submatrixes), Gibbs, SAMBA (Statistical-Algorithmic Method for Bicluster Analysis),*[14] Robust Biclustering Algorithm (RoBA), Crossing Minimization,*[15] cMonkey,*[16] PRMs, DCC, LEB (Localize and Extract Biclusters), QUBIC (QUalitative BIClustering), BCCA (Bi-Correlation Clustering Algorithm) BIMAX, ISA, SAMBA and FABIA (Factor Analysis for Bicluster Acquisition).*[17] Biclustering algorithms have also been proposed and used in other application fields under the names coclustering, bidimensional clustering, and subspace clustering.*[12]

Given the known importance of discovering local patterns in time-series data, recent proposals have addressed the biclustering problem in the specific case of time series gene expression data. In this case, the interesting biclusters can be restricted to those with contiguous columns. This restriction leads to a tractable problem and enables the development of efficient exhaustive enumeration algorithms such as CCC-Biclustering *[18] and *e*-CCC-Biclustering.*[19] The approximate patterns in CCC-Biclustering algorithms allow a given number of errors, per gene, relatively to an expression profile representing the expression pattern in the bicluster. The e-CCC-Biclustering algorithm uses approximate expressions to find and report all maximal CCC-Biclusters by a discretized matrix A and efficient string processing techniques.

These algorithms find and report all maximal biclusters with coherent and contiguous columns with perfect/approximate expression patterns, in time linear/polynomial which is obtained by manipulating a discretized version of original expression matrix in the size of the time series gene expression matrix using efficient string processing techniques based on suffix trees. These algorithms are also applied to solve problems and sketch the analysis of computational complexity.

Some recent algorithms have attempted to include additional support for biclustering rectangular matrices in the form of other datatypes, including cMonkey.

There is an ongoing debate about how to judge the results of these methods, as biclustering allows overlap between clusters and some algorithms allow the exclusion of hard-to-reconcile columns/conditions. Not all of the available algorithms are deterministic and the analyst must pay attention to the degree to which results represent stable minima. Because this is an unsupervised classification problem, the lack of a gold standard makes it difficult to spot errors in the results. One approach is to utilize multiple biclustering algorithms, with majority or super-majority voting amongst them deciding the best result. Another way is to analyse the quality of shifting and scaling patterns in biclusters.*[20] Biclustering has been used in the domain of text mining (or classification) where it is popularly known as co-clustering .*[21] Text corpora are represented in a vectorial form as a matrix D whose rows denote the documents and whose columns denote the words in the dictionary. Matrix elements D_{ij} denote occurrence of word j in document i. Co-clustering algorithms are then applied to discover blocks in D that correspond to a group of documents (rows) characterized by a group of words(columns).

Test clustering can solve the high-dimensional sparse problem, which means clustering text and words at the same time. When clustering text, we need to think about not only the words information, but also the information of words clusters that was composed by words. Then according to similarity of feature words in the text, will eventually cluster the feature words. This is called co-clustering. There are two advantages of co-clustering: one is clustering the test based on words clusters can extremely decrease the dimension of clustering, it can also appropriate to measure the distance between the tests. Second is mining more useful information and can get the corresponding information in test clusters and words clusters. This corresponding information can be used to describe the type of texts and words, at the same time, the result of words clustering can be also used to text mining and information retrival.

Several approaches have been proposed based on the information contents of the resulting blocks: matrix-based approaches such as SVD and BVD, and graph-based approaches. Information-theoretic algorithms iteratively assign each row to a cluster of documents and each column to a cluster of words such that the mutual information is maximized. Matrix-based methods focus on the decomposition of matrices into blocks such that the error between the original matrix and the regenerated matrices from the decomposition is minimized. Graph-based methods tend to minimize the cuts between the clusters. Given two groups of documents d_1 and d_2, the number of cuts can be measured as the number of words that occur in documents of groups d_1 and d_2.

More recently (Bisson and Hussain)*[21] have proposed a new approach of using the similarity between words and the similarity between documents to co-cluster the matrix. Their method (known as **χ-Sim**, for cross similarity) is based on finding document-document similarity and word-word similarity, and then using classical clustering methods such as hierarchical clustering. Instead of explicitly clustering rows and columns alternately, they consider higher-order occurrences of words, inherently taking into account the documents in which they occur. Thus, the similarity between two words is calculated based on the documents in which they occur and also the documents in which "similar" words occur. The idea here is that two documents about the same topic do not necessarily use the same set of words to describe it but a subset of the words and other similar words that are characteristic of that topic. This approach of taking higher-order similarities takes the latent semantic structure of the whole corpus into consideration with the result of generating a better clustering of the documents and words.

In text databases, for a document collection defined by a document by term D matrix (of size m by n, m: number of documents, n: number of terms) the cover-coefficient based clustering methodology*[22] yields the same number of clusters both for documents and terms (words) using a double-stage probability experiment. According to the cover coefficient concept number of clusters can also be roughly estimated by the following formula $(m \times n)/t$ where t is the number of non-zero entries in D. Note that in D each row and each column must contain at least one non-zero element.

In contrast to other approaches, FABIA is a multiplicative model that assumes realistic non-Gaussian signal distributions with heavy tails. FABIA utilizes well understood model selection techniques like variational approaches and applies the Bayesian framework. The generative framework allows FABIA to determine the information content of each bicluster to separate spurious biclusters from true biclusters.

6.1.5 See also

- Formal concept analysis
- Biclique
- Galois connection

6.1.6 References

[1] G. Govaert, M. Nadif (2008). "Block clustering with bernoulli mixture models: Comparison of different approaches," . *Computational Statistics and Data Analysis* (Elsevier) **52** (6): 3233–3245.

[2] G. Govaert, M. Nadif (2013). *Co-clustering: models, algorithms and applications*. ISTE, Wiley. ISBN 978-1-84821-473-6.

[3] Van Mechelen I, Bock HH, De Boeck P (2004). "Two-mode clustering methods:a structured overview" . *Statistical Methods in Medical Research* **13** (5): 363–94. doi:10.1191/0962280204sm373ra. PMID 15516031.

[4] Mirkin, Boris (1996). *Mathematical Classification and Clustering*. Kluwer Academic Publishers. ISBN 0-7923-4159-7.

[5] Hartigan JA (1972). "Direct clustering of a data matrix" . *Journal of the American Statistical Association* (American Statistical Association) **67** (337): 123–9. doi:10.2307/2284710. JSTOR 2284710.

[6] Hartigan J A. Direct clustering of a data matrix[J]. Journal of the american statistical association, 1972, 67(337): 123-129.

[7] https://www.cs.princeton.edu/courses/archive/fall03/cs597F/Articles/biclustering_of_expression_data.pdf Cheng Y, Church G M. Biclustering of expression data[C]//Ismb. 2000, 8: 93-103.

[8] Dhillon I S. Co-clustering documents and words using bipartite spectral graph partitioning[C]//Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2001: 269-274.

[9] Dhillon I S, Mallela S, Modha D S. Information-theoretic co-clustering[C]//Proceedings of the ninth ACM SIGKDD international conference on KKluwer Academic Publishersnowledge discovery and data mining. ACM, 2003: 89-98.

[10] Banerjee A, Dhillon I, Ghosh J, et al. A generalized maximum entropy approach to bregman co-clustering and matrix approximation[C]//Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2004: 509-514.

[11] Peeters R. The maximum edge biclique problem is NP-complete[J]. Discrete Applied Mathematics, 2003, 131(3): 651-654.

[12] Madeira SC, Oliveira AL (2004). "Biclustering Algorithms for Biological Data Analysis: A Survey". *IEEE Transactions on Computational Biology and Bioinformatics* **1** (1): 24–45. doi:10.1109/TCBB.2004.2. PMID 17048406.

[13] Kriegel, H.-P.; Kröger, P.; Zimek, A. (March 2009). "Clustering High Dimensional Data: A Survey on Subspace Clustering, Pattern-based Clustering, and Correlation Clustering". *ACM Transactions on Knowledge Discovery from Data (TKDD)* **3** (1): 1–58. doi:10.1145/1497577.1497578.

[14] Tanay A, Sharan R, Kupiec M and Shamir R (2004). "Revealing modularity and organization in the yeast molecular network by integrated analysis of highly heterogeneous genomewide data". *Proc Natl Acad Sci USA* **101** (9): 2981–2986. doi:10.1073/pnas.0308661100. PMC 365731. PMID 14973197.

[15] Abdullah, Ahsan; Hussain, Amir (2006). "A new biclustering technique based on crossing minimization". *Neurocomputing, vol. 69 issue 16-18* **69** (16–18): 1882–1896. doi:10.1016/j.neucom.2006.02.018.

[16] Reiss DJ, Baliga NS, Bonneau R (2006). "Integrated biclustering of heterogeneous genome-wide datasets for the inference of global regulatory networks". *BMC Bioinformatics* **2**: 280–302. doi:10.1186/1471-2105-7-280. PMC 1502140. PMID 16749936.

[17] Hochreiter S, Bodenhofer U, Heusel M, Mayr A, Mitterecker A, Kasim A, Khamiakova T, Van Sanden S, Lin D, Talloen W, Bijnens L, Gohlmann HWH, Shkedy Z, Clevert DA (2010). "FABIA: factor analysis for bicluster acquisition". *Bioinformatics* **26** (12): 1520–1527. doi:10.1093/bioinformatics/btq227. PMC 2881408. PMID 20418340.

[18] Madeira SC, Teixeira MC, Sá-Correia I, Oliveira AL (2010). "Identification of Regulatory Modules in Time Series Gene Expression Data using a Linear Time Biclustering Algorithm". *IEEE Transactions on Computational Biology and Bioinformatics* **1** (7): 153–165. doi:10.1109/TCBB.2008.34.

[19] Madeira SC, Oliveira AL (2009). "A polynomial time biclustering algorithm for finding approximate expression patterns in gene expression time series". *Algorithms for Molecular Biology* **4** (8).

[20] Aguilar-Ruiz JS (2005). "Shifting and scaling patterns from gene expression data". *Bioinformatics* **21** (10): 3840–3845. doi:10.1093/bioinformatics/bti641. PMID 16144809.

[21] Bisson G. and Hussain F. (2008). "Chi-Sim: A new similarity measure for the co-clustering task". *ICMLA*: 211–217. doi:10.1109/ICMLA.2008.103.

[22] Can, F., Ozkarahan, E. A. (1990). "Concepts and effectiveness of the cover coefficient based clustering methodology for text databases". *ACM Transactions on Database Systems* **15** (4): 483–517. doi:10.1145/99935.99938.

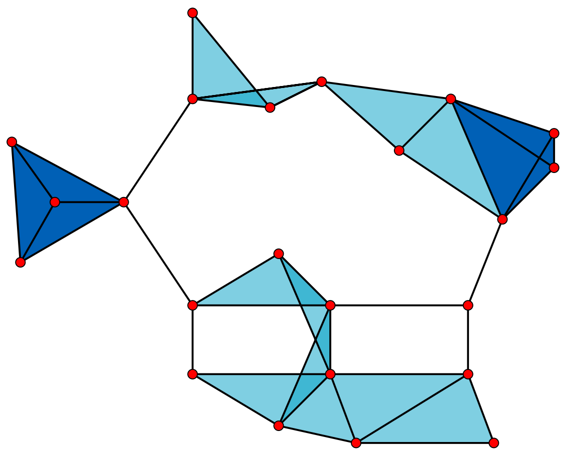
Others

- A. Tanay. R. Sharan, and R. Shamir, "Biclustering Algorithms: A Survey", In *Handbook of Computational Molecular Biology*, Edited by Srinivas Aluru, Chapman (2004)
- Kluger Y, Basri R, Chang JT, Gerstein MB (2003). "Spectral Biclustering of Microarray Data: Coclustering Genes and Conditions". *Genome Research* **13** (4): 703–716. doi:10.1101/gr.648603. PMC 430175. PMID 12671006.

6.1.7 External links

- FABIA: Factor Analysis for Bicluster Acquisition, an R package —software

6.2 Clique (graph theory)



A graph with

- *23 × 1-vertex cliques (the vertices),*
- *42 × 2-vertex cliques (the edges),*
- *19 × 3-vertex cliques (the light and dark blue triangles), and*
- *2 × 4-vertex cliques (just the dark blue areas).*

The 11 light blue triangles form maximal cliques. The two dark blue 4-cliques are both maximum and maximal, and the clique number of the graph is 4.

In the mathematical area of graph theory, a **clique** (/ˈkliːk/ or /ˈklɪk/) is subset of vertices of an undirected graph, such that its induced subgraph is complete; that is, every two distinct vertices in the clique are adjacent. Cliques are one of the basic concepts of graph theory and are used in many other mathematical problems and constructions on graphs. Cliques have also been studied in computer science: the task of finding whether there is a clique of a given size in a graph (the clique problem) is

NP-complete, but despite this hardness result, many algorithms for finding cliques have been studied.

Although the study of complete subgraphs goes back at least to the graph-theoretic reformulation of Ramsey theory by Erdős & Szekeres (1935),*[1] the term *clique* comes from Luce & Perry (1949), who used complete subgraphs in social networks to model cliques of people; that is, groups of people all of whom know each other. Cliques have many other applications in the sciences and particularly in bioinformatics.

6.2.1 Definitions

A **clique**, C, in an undirected graph $G = (V, E)$ is a subset of the vertices, $C \subseteq V$, such that every two distinct vertices are adjacent. This is equivalent to the condition that the subgraph of G induced by C is complete. In some cases, the term clique may also refer to the subgraph directly.

A **maximal clique** is a clique that cannot be extended by including one more adjacent vertex, that is, a clique which does not exist exclusively within the vertex set of a larger clique.

A **maximum clique** of a graph, G, is a clique, such that there is no clique with more vertices.

The **clique number** $\omega(G)$ of a graph G is the number of vertices in a maximum clique in G.

The **intersection number** of G is the smallest number of cliques that together cover all edges of G.

The opposite of a clique is an **independent set**, in the sense that every clique corresponds to an independent set in the complement graph. The clique cover problem concerns finding as few cliques as possible that include every vertex in the graph.

A related concept is a **biclique**, a complete bipartite subgraph. The bipartite dimension of a graph is the minimum number of bicliques needed to cover all the edges of the graph.

6.2.2 Mathematics

Mathematical results concerning cliques include the following.

- Turán's theorem (Turán 1941) gives a lower bound on the size of a clique in dense graphs. If a graph has sufficiently many edges, it must contain a large clique. For instance, every graph with n vertices and more than $\lfloor \frac{n}{2} \rfloor \cdot \lceil \frac{n}{2} \rceil$ edges must contain a three-vertex clique.
- Ramsey's theorem (Graham, Rothschild & Spencer 1990) states that every graph or its complement graph contains a clique with at least a logarithmic number of vertices.
- According to a result of Moon & Moser (1965), a graph with $3n$ vertices can have at most $3^{*}n$ maximal cliques. The graphs meeting this bound are the Moon–Moser graphs $K_{3,3,...}$, a special case of the Turán graphs arising as the extremal cases in Turán's theorem.
- Hadwiger's conjecture, still unproven, relates the size of the largest clique minor in a graph (its Hadwiger number) to its chromatic number.
- The Erdős–Faber–Lovász conjecture is another unproven statement relating graph coloring to cliques.

Several important classes of graphs may be defined by their cliques:

- A chordal graph is a graph whose vertices can be ordered into a perfect elimination ordering, an ordering such that the neighbors of each vertex v that come later than v in the ordering form a clique.
- A cograph is a graph all of whose induced subgraphs have the property that any maximal clique intersects any maximal independent set in a single vertex.
- An interval graph is a graph whose maximal cliques can be ordered in such a way that, for each vertex v, the cliques containing v are consecutive in the ordering.
- A line graph is a graph whose edges can be covered by edge-disjoint cliques in such a way that each vertex belongs to exactly two of the cliques in the cover.
- A perfect graph is a graph in which the clique number equals the chromatic number in every induced subgraph.
- A split graph is a graph in which some clique contains at least one endpoint of every edge.
- A triangle-free graph is a graph that has no cliques other than its vertices and edges.

Additionally, many other mathematical constructions involve cliques in graphs. Among them,

- The clique complex of a graph G is an abstract simplicial complex $X(G)$ with a simplex for every clique in G
- A simplex graph is an undirected graph $\kappa(G)$ with a vertex for every clique in a graph G and an edge connecting two cliques that differ by a single vertex. It is an example of median graph, and is associated with a median algebra on the cliques of a graph: the median $m(A,B,C)$ of three cliques A, B, and C is the clique whose vertices belong to at least two of the cliques A, B, and C.*[2]

- The clique-sum is a method for combining two graphs by merging them along a shared clique.
- Clique-width is a notion of the complexity of a graph in terms of the minimum number of distinct vertex labels needed to build up the graph from disjoint unions, relabeling operations, and operations that connect all pairs of vertices with given labels. The graphs with clique-width one are exactly the disjoint unions of cliques.
- The intersection number of a graph is the minimum number of cliques needed to cover all the graph's edges.
- The clique graph of a graph is the intersection graph of its maximal cliques.

Closely related concepts to complete subgraphs are subdivisions of complete graphs and complete graph minors. In particular, Kuratowski's theorem and Wagner's theorem characterize planar graphs by forbidden complete and complete bipartite subdivisions and minors, respectively.

6.2.3 Computer science

Main article: Clique problem

In computer science, the clique problem is the computational problem of finding a maximum clique, or all cliques, in a given graph. It is NP-complete, one of Karp's 21 NP-complete problems (Karp 1972). It is also fixed-parameter intractable, and hard to approximate. Nevertheless, many algorithms for computing cliques have been developed, either running in exponential time (such as the Bron–Kerbosch algorithm) or specialized to graph families such as planar graphs or perfect graphs for which the problem can be solved in polynomial time.

6.2.4 Free software for searching maximum clique

6.2.5 Applications

The word “clique”, in its graph-theoretic usage, arose from the work of Luce & Perry (1949), who used complete subgraphs to model cliques (groups of people who all know each other) in social networks. For continued efforts to model social cliques graph-theoretically, see e.g. Alba (1973), Peay (1974), and Doreian & Woodard (1994).

Many different problems from bioinformatics have been modeled using cliques. For instance, Ben-Dor, Shamir & Yakhini (1999) model the problem of clustering gene expression data as one of finding the minimum number of changes needed to transform a graph describing the data into a graph formed as the disjoint union of cliques; Tanay, Sharan & Shamir (2002) discuss a similar biclustering problem for expression data in which the clusters are required to be cliques. Sugihara (1984) uses cliques to model ecological niches in food webs. Day & Sankoff (1986) describe the problem of inferring evolutionary trees as one of finding maximum cliques in a graph that has as its vertices characteristics of the species, where two vertices share an edge if there exists a perfect phylogeny combining those two characters. Samudrala & Moult (1998) model protein structure prediction as a problem of finding cliques in a graph whose vertices represent positions of subunits of the protein. And by searching for cliques in a protein-protein interaction network, Spirin & Mirny (2003) found clusters of proteins that interact closely with each other and have few interactions with proteins outside the cluster. Power graph analysis is a method for simplifying complex biological networks by finding cliques and related structures in these networks.

In electrical engineering, Prihar (1956) uses cliques to analyze communications networks, and Paull & Unger (1959) use them to design efficient circuits for computing partially specified Boolean functions. Cliques have also been used in automatic test pattern generation: a large clique in an incompatibility graph of possible faults provides a lower bound on the size of a test set.*[3] Cong & Smith (1993) describe an application of cliques in finding a hierarchical partition of an electronic circuit into smaller subunits.

In chemistry, Rhodes et al. (2003) use cliques to describe chemicals in a chemical database that have a high degree of similarity with a target structure. Kuhl, Crippen & Friesen (1983) use cliques to model the positions in which two chemicals will bind to each other.

6.2.6 Notes

[1] The earlier work by Kuratowski (1930) characterizing planar graphs by forbidden complete and complete bipartite subgraphs was originally phrased in topological rather than graph-theoretic terms.

[2] Barthélemy, Leclerc & Monjardet (1986), page 200.

[3] Hamzaoglu & Patel (1998).

6.2.7 References

- Alba, Richard D. (1973), “A graph-theoretic definition of a sociometric clique” (PDF), *Journal of Mathematical Sociology* **3** (1): 113–126, doi:10.1080/0022250X.1973.9989826.
- Barthélemy, J.-P.; Leclerc, B.; Monjardet, B. (1986), “On the use of ordered sets in problems of comparison and consensus of classifications”, *Journal of Classification* **3** (2): 187–224, doi:10.1007/BF01894188.

- Ben-Dor, Amir; Shamir, Ron; Yakhini, Zohar (1999), "Clustering gene expression patterns." , *Journal of Computational Biology* **6** (3–4): 281–297, doi:10.1089/106652799318274, PMID 10582567.
- Cong, J.; Smith, M. (1993), "A parallel bottom-up clustering algorithm with applications to circuit partitioning in VLSI design" , *Proc. 30th International Design Automation Conference*, pp. 755–760, doi:10.1145/157485.165119.
- Day, William H. E.; Sankoff, David (1986), "Computational complexity of inferring phylogenies by compatibility" , *Systematic Zoology* **35** (2): 224–229, doi:10.2307/2413432, JSTOR 2413432.
- Doreian, Patrick; Woodard, Katherine L. (1994), "Defining and locating cores and boundaries of social networks" , *Social Networks* **16** (4): 267–293, doi:10.1016/0378-8733(94)90013-2.
- Erdős, Paul; Szekeres, George (1935), "A combinatorial problem in geometry" (PDF), *Compositio Mathematica* **2**: 463–470.
- Graham, R.; Rothschild, B.; Spencer, J. H. (1990), *Ramsey Theory*, New York: John Wiley and Sons, ISBN 0-471-50046-1.
- Hamzaoglu, I.; Patel, J. H. (1998), "Test set compaction algorithms for combinational circuits" , *Proc. 1998 IEEE/ACM International Conference on Computer-Aided Design*, pp. 283–289, doi:10.1145/288548.288615.
- Karp, Richard M. (1972), "Reducibility among combinatorial problems", in Miller, R. E.; Thatcher, J. W., *Complexity of Computer Computations* (PDF), New York: Plenum, pp. 85–103.
- Kuhl, F. S.; Crippen, G. M.; Friesen, D. K. (1983), "A combinatorial algorithm for calculating ligand binding", *Journal of Computational Chemistry* **5** (1): 24–34, doi:10.1002/jcc.540050105.
- Kuratowski, Kazimierz (1930), "Sur le probléme des courbes gauches en Topologie" (PDF), *Fundamenta Mathematicae* (in French) **15**: 271–283.
- Luce, R. Duncan; Perry, Albert D. (1949), "A method of matrix analysis of group structure" , *Psychometrika* **14** (2): 95–116, doi:10.1007/BF02289146, PMID 18152948.
- Moon, J. W.; Moser, L. (1965), "On cliques in graphs" , *Israel J. Math.* **3**: 23–28, doi:10.1007/BF02760024, MR 0182577.
- Paull, M. C.; Unger, S. H. (1959), "Minimizing the number of states in incompletely specified sequential switching functions" , *IRE Trans. on Electronic Computers* **EC–8** (3): 356–367, doi:10.1109/TEC.1959.5222697.
- Peay, Edmund R. (1974), "Hierarchical clique structures" , *Sociometry* **37** (1): 54–65, doi:10.2307/2786466, JSTOR 2786466.
- Prihar, Z. (1956), "Topological properties of telecommunications networks" , *Proceedings of the IRE* **44** (7): 927–933, doi:10.1109/JRPROC.1956.275149.
- Rhodes, Nicholas; Willett, Peter; Calvet, Alain; Dunbar, James B.; Humblet, Christine (2003), "CLIP: similarity searching of 3D databases using clique detection" , *Journal of Chemical Information and Computer Sciences* **43** (2): 443–448, doi:10.1021/ci025605o, PMID 12653507.
- Samudrala, Ram; Moult, John (1998), "A graph-theoretic algorithm for comparative modeling of protein structure", *Journal of Molecular Biology* **279** (1): 287–302, doi:10.1006/jmbi.1998.1689, PMID 9636717.
- Spirin, Victor; Mirny, Leonid A. (2003), "Protein complexes and functional modules in molecular networks" , *Proceedings of the National Academy of Sciences* **100** (21): 12123–12128, doi:10.1073/pnas.2032324100, PMC 218723, PMID 14517352.
- Sugihara, George (1984), "Graph theory, homology and food webs" , in Levin, Simon A., *Population Biology*, Proc. Symp. Appl. Math. **30**, pp. 83–101.
- Tanay, Amos; Sharan, Roded; Shamir, Ron (2002), "Discovering statistically significant biclusters in gene expression data" , *Bioinformatics* **18** (Suppl. 1): S136–S144, doi:10.1093/bioinformatics/18.suppl_1.S136, PMID 12169541.
- Turán, Paul (1941), "On an extremal problem in graph theory" , *Matematikai és Fizikai Lapok* (in Hungarian) **48**: 436–452

6.2.8 External links

- Weisstein, Eric W., "Clique" , *MathWorld*.
- Weisstein, Eric W., "Clique Number", *MathWorld*.

6.3 Affinity propagation

In statistics and data mining, **affinity propagation** (AP) is a clustering algorithm based on the concept of "message passing" between data points.*[1] Unlike clustering algorithms such as k-means or k-medoids, AP does not require the number of clusters to be determined or estimated before running the algorithm. Like k-medoids, AP finds "exemplars" , members of the input set that are representative of clusters.*[1]

6.3.1 Algorithm

Let x_1 through x_n be a set of data points, with no assumptions made about their internal structure, and let s be a function that quantifies the similarity between any two points, such that $s(x_i, x_j) > s(x_i, x_k)$ iff x_i is more similar to x_j than to x_k.

The algorithm proceeds by alternating two message passing steps, to update two matrices:*[1]

- The “responsibility” matrix **R** has values $r(i, k)$ that quantify how well-suited x_k is to serve as the exemplar for x_i, relative to other candidate exemplars for x_i.
- The “availability” matrix **A** contains values $a(i, k)$ represents how “appropriate” it would be for x_i to pick x_k as its exemplar, taking into account other points' preference for x_k as an exemplar.

Both matrices are initialized to all zeroes, and can be viewed as log-probability tables. The algorithm then performs the following updates iteratively:

- First, responsibility updates are sent around: $r(i,k) \leftarrow s(i,k) - \max_{k' \neq k} \{a(i,k') + s(i,k')\}$
- Then, availability is updated per

$$a(i,k) \leftarrow \min\left(0, r(k,k) + \sum_{i' \notin \{i,k\}} \max(0, r(i',k))\right)$$
for $i \neq k$ and
$$a(k,k) \leftarrow \sum_{i' \neq k} \max(0, r(i',k))$$
.

6.3.2 Applications

The inventors of affinity propagation showed it is better for certain computer vision and computational biology tasks, e.g. clustering of pictures of human faces and identifying regulated transcripts, than k-means,*[1] even when k-means was allowed many random restarts and initialized using PCA.*[2] A study comparing AP and Markov clustering on protein interaction graph partitioning found Markov clustering to work better for that problem.*[3] A semi-supervised variant has been proposed for text mining applications.*[4]

6.3.3 Software

- A Java implementation is included in the ELKI data mining framework.
- Java Apro library implements parallelized Affinity Propagation and Hierarchical AP.
- A Julia implementation of affinity propagation is contained in Julia Statistics's Clustering.jl package.*[5]
- A Python version is part of the scikit-learn library.*[6]
- An R implemenation is available in the “apcluster” package.*[7]

6.3.4 References

[1] Brendan J. Frey; Delbert Dueck (2007). “Clustering by passing messages between data points” . *Science* **315** (5814): 972–976. doi:10.1126/science.1136800. PMID 17218491.

[2] Delbert Dueck; Brendan J. Frey (2007). *Non-metric affinity propagation for unsupervised image categorization*. Int'l Conf. on Computer Vision.

[3] James Vlasblom; Shoshana Wodak (2009). “Markov clustering versus affinity propagation for the partitioning of protein interaction graphs” . *BMC Bioinformatics* **10** (1): 99. doi:10.1186/1471-2105-10-99.

[4] Renchu Guan; Xiaohu Shi; Maurizio Marchese; Chen Yang; Yanchun Liang (2011). “Text Clustering with Seeds Affinity Propagation” . *IEEE Transactions on Knowledge & Data Engineering* **23** (4): 627–637.

[5] Clustering.jl www.github.com

[6] “Clustering —scikit-learn 0.14.1 documentation” . Retrieved 15 July 2014.

[7] apcluster cran.r-project.org>

6.4 Basic sequential algorithmic scheme

The **basic sequential algorithmic scheme (BSAS)** is a very basic clustering algorithm that is easy to understand. In the basic form vectors are presented only once and the number of clusters is not known a priori. What is needed is the dissimilarity measured as the distance $d(x, C)$ between a vector point x and a cluster C, threshold of dissimilarity Θ and the number of maximum clusters allowed q. The idea is to assign every newly presented vector to an existing cluster or create a new cluster for this sample, depending on the distance to the already defined clusters. As pseudocode, the algorithm looks like the following:

1. m = 1; Cm = {x1}; // Init first cluster = first sample 2. for every sample x from 2 to N a. find cluster Ck such that min d(x, Ck) b. if d(x, Ck) > Θ AND (m < q) i. m = m + 1; Cm = {x} // Create a new cluster c. else
i. Ck = Ck + {x} // Add sample to the nearest cluster ii. Update representative if needed 3. end algorithm

As can be seen the algorithm is simple but still quite efficient. Different choices for the distance function lead to different results and unfortunately the order in which the samples are presented can also have a great effect to the final result. What' s also very important is a correct value for *Θ*. This value has a direct effect on the number of formed clusters. If Θ is too small unnecessary clusters are created and if too large a value is chosen less than required number of clusters are formed.

One detail is that if q is not defined the algorithm 'decides' the number of clusters on its own. This might be wanted under some circumstances but when dealing with limited resources a limited q is usually chosen. Also, BSAS can be used with a similarity function simply by replacing the min function with max.

There exists a modification to BSAS called modified BSAS (MBSAS), which runs twice through the samples. It overcomes the drawback that a final cluster for a single sample is decided before all the clusters have been created. The first phase of the algorithm creates the clusters (just like 2b in BSAS) and assigns only a single sample to each cluster. Then the second phase runs through the remaining samples and classifies them to the created clusters (step 2c in BSAS).

6.4.1 External links

- Clustering Algorithms: Basics and Visualization Jukka Kainulainen
- Pattern Recognition Lecture Sequential Clustering

6.5 Binarization of consensus partition matrices

Mainly in the context of gene clustering, the **binarization of consensus partition matrices (Bi-CoPaM)** was proposed by Abu-Jamous et al.[1] as a method for consensus clustering. In contrast to other conventional clustering and ensemble clustering methods, Bi-CoPaM has the ability to combine the results of clustering the same set of genes from various microarray datasets and by using many clustering methods to produce one consensus result. Moreover, Bi-CoPaM relaxes conventional clustering constraints by allowing each gene to have any of the three possible eventualities – to be exclusively assigned to one and only one cluster (as any conventional clustering method does), to be simultaneously assigned to multiple clusters, or to be unassigned from all of the clusters. At the clusters level, clusters can be complementary (as in the case of conventional clustering), can be wide and overlapping, and can be tight and distinct while leaving many genes unassigned from all of them. The Bi-CoPaM method has not been designed to only allow for these three forms of clusters; it has also been provided with tuning parameters which can be used to tune the level of tightness and wideness of the clusters based on research requirements.

Complete description of the method is given in the publication in which it was proposed (Abu-Jamous et al 2013).[1]

6.5.1 Applications

As the Bi-CoPaM specially meets many requirements of gene discovery studies, its current main applications are within this field of bioinformatics;[2] though, it was defined in a completely independent manner such that it is applicable for any other clustering problem. For example, a recent experiment in which the Bi-CoPaM was applied over multiple yeast cell-cycle datasets revealed important information about a poorly characterised gene, CMR1/YDL156W, and about its relation with many other genes.[3]

6.5.2 References

[1] Abu-Jamous, Basel; Fa, Rui; Roberts, David J.; Nandi, Asoke K.; Peddada, Shyamal D. (11 February 2013). "Paradigm of Tunable Clustering Using Binarization of Consensus Partition Matrices (Bi-CoPaM) for Gene Discovery" . *PLoS ONE* **8** (2): e56432. doi:10.1371/journal.pone.0056432. PMC 3569426. PMID 23409186.

[2] Garcia-Lapresta, Jose Luis; Perez-Roman, D. (June 2013). "Consensus-based hierarchical agglomerative clustering in the context of weak orders" . *2013 Joint IFSA World Congress and NAFIPS Annual Meeting (IFSA/NAFIPS)*: 1010–1015. doi:10.1109/IFSA-NAFIPS.2013.6608538.

[3] Abu-Jamous, B.; Fa, R.; Roberts, D. J.; Nandi, A. K. (24 January 2013). "Yeast gene CMR1/YDL156W is consistently co-expressed with genes participating in DNA-metabolic processes in a variety of stringent clustering experiments" . *Journal of The Royal Society Interface* **10** (81): 20120990–20120990. doi:10.1098/rsif.2012.0990. PMC 3627109. PMID 23349438.

6.6 Cluster-weighted modeling

In data mining, **cluster-weighted modeling (CWM)** is an algorithm-based approach to non-linear prediction of outputs (dependent variables) from inputs (independent variables) based on density estimation using a set of models (clusters) that are each notionally appropriate in a subregion of the input space. The overall approach works in jointly input-output space and an initial version was proposed by Neil Gershenfeld.[1][2]

6.6.1 Basic form of model

The procedure for cluster-weighted modeling of an input-output problem can be outlined as follows.*[2] In order to construct predicted values for an output variable y from an input variable x, the modeling and calibration procedure arrives at a joint probability density function, $p(y,x)$. Here the "variables" might be uni-variate, multivariate or time-series. For convenience, any model parameters are not indicated in the notation here and several different treatments of these are possible, including setting them to fixed values as a step in the calibration or treating them using a Bayesian analysis. The required predicted values are obtained by constructing the conditional probability density $p(y|x)$ from which the prediction using the conditional expected value can be obtained, with the conditional variance providing an indication of uncertainty.

The important step of the modeling is that $p(y|x)$ is assumed to take the following form, as a mixture model:

$$p(y,x) = \sum_{1}^{n} w_j p_j(y,x),$$

where n is the number of clusters and $\{w_j\}$ are weights that sum to one. The functions $p_j(y,x)$ are joint probability density functions that relate to each of the n clusters. These functions are modeled using a decomposition into a conditional and a marginal density:

$$p_j(y,x) = p_j(y|x)p_j(x),$$

where:

- $p_j(y|x)$ is a model for predicting y given x, and given that the input-output pair should be associated with cluster j on the basis of the value of x. This model might be a regression model in the simplest cases.

- $p_j(x)$ is formally a density for values of x, given that the input-output pair should be associated with cluster j. The relative sizes of these functions between the clusters determines whether a particular value of x is associated with any given cluster-center. This density might be a Gaussian function centered at a parameter representing the cluster-center.

In the same way as for regression analysis, it will be important to consider preliminary data transformations as part of the overall modeling strategy if the core components of the model are to be simple regression models for the cluster-wise condition densities, and normal distributions for the cluster-weighting densities $p_j(x)$.

6.6.2 General versions

The basic CWM algorithm gives a single output cluster for each input cluster. However, CWM can be extended to multiple clusters which are still associated with the same input cluster.*[3] Each cluster in CWM is localized to a Gaussian input region, and this contains its own trainable local model.*[4] It is recognized as a versatile inference algorithm which provides simplicity, generality, and flexibility; even when a feedforward layered network might be preferred, it is sometimes used as a "second opinion" on the nature of the training problem.*[5]

The original form proposed by Gershenfeld describes two innovations:

- Enabling CWM to work with continuous streams of data
- Addressing the problem of local minima encountered by the CWM parameter adjustment process*[5]

CWM can be used to classify media in printer applications, using at least two parameters to generate an output that has a joint dependency on the input parameters.*[6]

6.6.3 References

[1] Gershenfeld, N (1997). "Nonlinear Inference and Cluster-Weighted Modeling". *Annals of the New York Academy of Sciences* **808**: 18–24. doi:10.1111/j.1749-6632.1997.tb51651.x.

[2] Gershenfeld, N.; Schoner; Metois, E. (1999). "Cluster-weighted modelling for time-series analysis" (PDF). *Nature* **397** (6717): 329–332. doi:10.1038/16873.

[3] Feldkamp, L.A.; Prokhorov, D.V.; Feldkamp, T.M. (2001). "Cluster-weighted modeling with multiclusters" (PDF). *International Joint Conference on Neural Networks* **3** (1): 1710–1714.

[4] Boyden, Edward S. "Tree-based Cluster Weighted Modeling: Towards A Massively Parallel Real-Time Digital Stradivarius" (PDF). Cambridge, MA: MIT Media Lab.

[5] Prokhorov, A New Approach to Cluster-Weighted Modeling Danil V.; Lee A. Feldkamp; Timothy M. Feldkamp. "A New Approach to Cluster-Weighted Modeling" (PDF). Dearborn, MI: Ford Research Laboratory.

[6] Gao, Jun; Ross R. Allen (2003-07-24). "CLUSTER-WEIGHTED MODELING FOR MEDIA CLASSIFICATION". Palo Alto, CA: World Intellectual Property Organization.

6.7 Cobweb (clustering)

COBWEB is an incremental system for hierarchical conceptual clustering. COBWEB was invented by Professor Douglas H. Fisher, currently at Vanderbilt University.*[1]*[2]

COBWEB incrementally organizes observations into a classification tree. Each node in a classification tree represents a class (concept) and is labeled by a probabilistic concept that summarizes the attribute-value distributions of objects classified under the node. This classification tree can be used to predict missing attributes or the class of a new object.*[3]

There are four basic operations COBWEB employs in building the classification tree. Which operation is selected depends on the category utility of the classification achieved by applying it. The operations are:

- Merging Two Nodes
  Merging two nodes means replacing them by a node whose children is the union of the original nodes' sets of children and which summarizes the attribute-value distributions of all objects classified under them.
- Splitting a node
  A node is split by replacing it with its children.
- Inserting a new node
  A node is created corresponding to the object being inserted into the tree.
- Passing an object down the hierarchy
  Effectively calling the COBWEB algorithm on the object and the subtree rooted in the node.

6.7.1 The COBWEB Algorithm

COBWEB(*root*, *record*): Input: A COBWEB node *root*, an instance to insert *record if root* has no children *then children* := {*copy*(*root*)} *newcategory*(*record*) \\ adds child with record’ s feature values. *insert*(*record*, *root*) \\ update root’ s statistics *else insert*(*record*, *root*) *for child in root*’ s children *do* calculate Category Utility for *insert*(*record*, *child*), set *best1*, *best2* children w. best CU. *end for if newcategory*(*record*) yields best CU then *newcategory*(*record*) *else if merge*(*best1*, *best2*) yields best CU then *merge*(*best1*, *best2*) *COBWEB*(*root*, *record*) *else if split*(*best1*) yields best CU then *split*(*best1*) *COBWEB*(*root*, *record*) *else COBWEB*(*best1*, *record*) *end if end*

6.7.2 External links

- Working python implementation of COBWEB

6.7.3 References

[1] Fisher, Douglas (1987). “Knowledge acquisition via incremental conceptual clustering” (PDF). *Machine Learning* **2** (2): 139–172. doi:10.1007/BF00114265.

[2] Fisher, Douglas H. (July 1987). “Improving inference through conceptual clustering” . *Proceedings of the 1987 AAAI Conferences*. AAAI Conference. Seattle Washington. pp. 461–465.

[3] William Iba and Pat Langley. “Cobweb models of categorization and probabilistic concept formation” . In Emmanuel M. Pothos and Andy J. Wills,. *Formal approaches in categorization*. Cambridge: Cambridge University Press. pp. 253–273. ISBN 9780521190480.

6.8 CURE data clustering algorithm

CURE (Clustering Using REpresentatives) is an efficient data clustering algorithm for large databases that is more robust to outliers and identifies clusters having non-spherical shapes and wide variances in size.

6.8.1 Drawbacks of traditional algorithms

With the partitional clustering algorithms, which for example use the sum of squared errors criterion

$$E = \sum_{i=1}^{k} \sum_{p \in C_i} (p - m_i)^2,$$

when there are large differences in sizes or geometries of different clusters, the square error method could split the large clusters to minimize the square error which is not always correct. Also, with hierarchic clustering algorithms these problems exist as none of the distance measures between clusters (d_{min}, d_{mean}) tend to work with different shapes of clusters. Also the running time is high when n is very large. The problem with the BIRCH algorithm is that once the clusters are generated after step 3, it uses centroids of the clusters and assign each data point to the cluster with closest centroid. Using only the centroid to redistribute the data has problems when clusters do not have uniform sizes and shapes.

6.8.2 CURE clustering algorithm

To avoid the problems with non-uniform sized or shaped clusters, CURE employs a novel hierarchical clustering algorithm that adopts a middle ground between the centroid based and all point extremes. In CURE, a constant number c of well scattered points of a cluster are chosen and they are shrunk towards the centroid of the cluster by a fraction α. The scattered points after shrinking are used as representatives of the cluster. The clusters with the closest pair of representatives are the clusters that are merged at each step of CURE's hierarchical clustering algorithm. This enables CURE to correctly identify the clusters and makes it less sensitive to outliers.

The algorithm is given below.

The running time of the algorithm is O(n^2 log n) and space complexity is O(n).

The algorithm cannot be directly applied to large databases. So for this purpose we do the following enhancements

- Random sampling : To handle large data sets, we do random sampling and draw a sample data set. Generally the random sample fits in main memory. Also because of the random sampling there is a trade off between accuracy and efficiency.

- Partitioning for speed up : The basic idea is to partition the sample space into p partitions. Each partition contains n/p elements. Then in the first pass partially cluster each partition until the final number of clusters reduces to n/pq for some constant q ≥ 1. Then run a second clustering pass on n/q partial clusters for all the partitions. For the second pass we only store the representative points since the merge procedure only requires representative points of previous clusters before computing the new representative points for the merged cluster. The advantage of partitioning the input is that we can reduce the execution times.

- Labeling data on disk : Since we only have representative points for k clusters, the remaining data points should also be assigned to the clusters. For this a fraction of randomly selected representative points for each of the k clusters is chosen and data point is assigned to the cluster containing the representative point closest to it.

6.8.3 Pseudocode

CURE(no. of points,*k*)

Input : A set of points S

Output : k clusters

1. For every cluster u (each input point), in u.mean and u.rep store the mean of the points in the cluster and a set of c representative points of the cluster (initially $c = 1$ since each cluster has one data point). Also u.closest stores the cluster closest to u.
2. All the input points are inserted into a k-d tree T
3. Treat each input point as separate cluster, compute u.closest for each u and then insert each cluster into the heap Q. (clusters are arranged in increasing order of distances between u and u.closest).
4. While size(Q) > k
5. Remove the top element of Q(say u) and merge it with its closest cluster u.closest(say v) and compute the new representative points for the merged cluster w.
6. Also remove u and v from T and Q.
7. Also for all the clusters x in Q, update x.closest and relocate x
8. insert w into Q
9. repeat

6.8.4 References

- Guha, Sudipto; Rastogi, Rajeev; Shim, Kyuseok (2001). "CURE: An Efficient Clustering Algorithm for Large Databases" (PDF). *Information Systems* **26** (1): 35–58. doi:10.1016/S0306-4379(01)00008-4.
- Kogan, Jacob; Nicholas, Charles K.; Teboulle, M. (2006). *Grouping multidimensional data: recent advances in clustering*. Springer. ISBN 978-3-540-28348-5.
- Theodoridis, Sergios; Koutroumbas, Konstantinos (2006). *Pattern recognition*. Academic Press. pp. 572–574. ISBN 978-0-12-369531-4.

6.9 FLAME clustering

Fuzzy clustering by Local Approximation of MEmberships (FLAME) is a data clustering algorithm that defines clusters in the dense parts of a dataset and performs cluster assignment solely based on the neighborhood relationships among objects. The key feature of this algorithm is that the neighborhood relationships among neighboring objects in the feature space are used to constrain the memberships of neighboring objects in the fuzzy membership space.

6.9.1 Description of the FLAME algorithm

The FLAME algorithm is mainly divided into three steps:

1. Extraction of the structure information from the dataset:
   (a) Construct a neighborhood graph to connect each object to its K-Nearest Neighbors (KNN);
   (b) Estimate a density for each object based on its proximities to its KNN;
   (c) Objects are classified into 3 types:

   i. Cluster Supporting Object (CSO): object with density higher than all its neighbors;
   ii. Cluster Outliers: object with density lower than all its neighbors, and lower than a predefined threshold;
   iii. the rest.

2. Local/Neighborhood approximation of fuzzy memberships:
   (a) Initialization of fuzzy membership:
      i. Each CSO is assigned with fixed and full membership to itself to represent one cluster;
      ii. All outliers are assigned with fixed and full membership to the outlier group;
      iii. The rest are assigned with equal memberships to all clusters and the outlier group;
   (b) Then the fuzzy memberships of all type 3 objects are updated by a converging iterative procedure called *Local/Neighborhood Approximation of Fuzzy Memberships*, in which the fuzzy membership of each object is updated by a linear combination of the fuzzy memberships of its nearest neighbors.
3. Cluster construction from fuzzy memberships in two possible ways:
   (a) One-to-one object-cluster assignment, to assign each object to the cluster in which it has the highest membership;
   (b) One-to-multiple object-clusters assignment, to assign each object to the cluster in which it has a membership higher than a threshold.

6.9.2 The optimization problem in FLAME

The Local/Neighborhood Approximation of Fuzzy Memberships is a procedure to minimize the Local/Neighborhood Approximation Error (LAE/NAE) defined as the following:

$$E(\{\boldsymbol{p}\}) = \sum_{\boldsymbol{x}\in\boldsymbol{X}} \left\| \boldsymbol{p}(\boldsymbol{x}) - \sum_{\boldsymbol{y}\in\boldsymbol{\mathcal{N}}(\boldsymbol{x})} w_{\boldsymbol{xy}}\boldsymbol{p}(\boldsymbol{y}) \right\|^2$$

where $\boldsymbol{X}$ is the set of all type 3 objects, $\boldsymbol{p}(\boldsymbol{x})$ is the fuzzy membership vector of object $\boldsymbol{x}$, $\mathcal{N}(x)$ is the set of nearest neighbors of $\boldsymbol{x}$, and $w_{\boldsymbol{xy}}$ with $\sum_{\boldsymbol{y}\in\boldsymbol{\mathcal{N}}(\boldsymbol{x})} w_{\boldsymbol{xy}} = 1$ are the coefficients reflecting the relative proximities of the nearest neighbors.

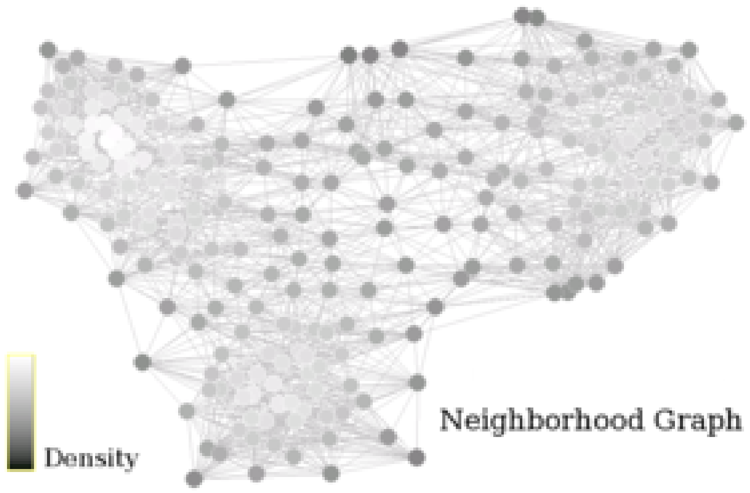
The NAE can be minimized by solving the following linear equations with unique solution which is the unique global minimum of NAE with value zero:

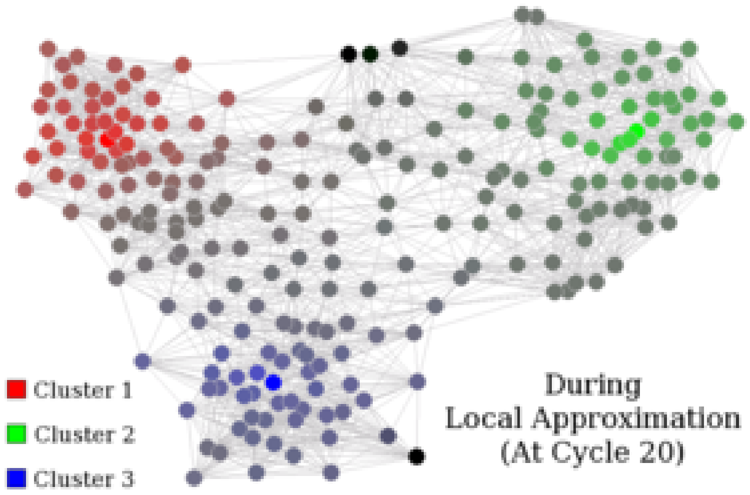
$$p_k(\boldsymbol{x}) - \sum_{\boldsymbol{y}\in\boldsymbol{\mathcal{N}}(\boldsymbol{x})} w_{\boldsymbol{xy}} p_k(\boldsymbol{y}) = 0, \quad \forall \boldsymbol{x}\in\boldsymbol{X}, \quad k = 1, ..., M$$

where M is the number of CSOs plus one (for the outlier group). The following iterative procedure can be used to solve these linear equations:

$$\boldsymbol{p}^{t+1}(\boldsymbol{x}) = \sum_{\boldsymbol{y}\in\boldsymbol{\mathcal{N}}(\boldsymbol{x})} w_{\boldsymbol{xy}}\boldsymbol{p}^t(\boldsymbol{y})$$

6.9.3 A simple illustration on a 2-Dimension testing dataset





6.9.4 See also

- Data clustering
- Fuzzy clustering

6.9.5 External links

- BMC Bioinformatics (2007): FLAME, a novel fuzzy clustering method for the analysis of DNA microarray data
- FLAME source codes in C released under FreeBSD-like license on GoogleCode

6.10 Information bottleneck method

The **information bottleneck method** is a technique introduced by Naftali Tishby et al. [1] for finding the best tradeoff between accuracy and complexity (compression) when summarizing (e.g. clustering) a random variable **X**, given a joint probability distribution between **X** and an observed relevant variable **Y**. Other applications include distributional clustering, and dimension reduction. In a well defined sense it generalized the classical notion of minimal sufficient statistics from parametric statistics to arbitrary distributions, not necessarily of exponential form. It does so by relaxing the sufficiency condition to capture some fraction of the mutual information with the relevant variable **Y**.

The compressed variable is T and the algorithm minimises the following quantity

$$\min_{p(t|x)} \; I(X;T) - \beta I(T;Y)$$

where $I(X;T)$ $I(T;Y)$ are the mutual information between $X;T$ and $T;Y$ respectively, and β is a Lagrange multiplier.

6.10.1 Gaussian information bottleneck

A relatively simple application of the information bottleneck is to Gaussian variates and this has some semblance to a least squares reduced rank or canonical correlation [2]. Assume X, Y are jointly multivariate zero mean normal vectors with covariances Σ_{XX}, Σ_{YY} and T is a compressed version of X which must maintain a given value of mutual information with Y . It can be shown that the optimum T is a normal vector consisting of linear combinations of the elements of X, $T = AX$ where matrix A has orthogonal rows.

The projection matrix A in fact contains M rows selected from the weighted left eigenvectors of the singular value decomposition of the following matrix (generally asymmetric)

$$\Omega = \Sigma_{X|Y}\Sigma_{XX}^{-1} = I - \Sigma_{XY}\Sigma_{YY}^{-1}\Sigma_{XY}^{T}\Sigma_{XX}^{-1}.$$

Define the singular value decomposition

$$\Omega = U\Lambda V^T \text{ with } \Lambda = \text{Diag}\left(\lambda_1 \leq \lambda_2 \cdots \lambda_N\right)$$

and the critical values

$$\beta_i^C \underset{\lambda_i<1}{=} (1-\lambda_i)^{-1}.$$

then the number M of active eigenvectors in the projection, or order of approximation, is given by

$$\beta_{M-1}^C < \beta \leq \beta_M^C$$

And we finally get

$$A = [w_1 U_1, \dots, w_M U_M]^T$$

In which the weights are given by

$$w_i = \sqrt{(\beta(1-\lambda_i)/\lambda_i r_i}$$

where $r_i = U_i^T \Sigma_{XX} U_i$.

Applying the Gaussian information bottleneck on time series, one gets optimal predictive coding. This procedure is formally equivalent to linear Slow Feature Analysis [3]. Optimal temporal structures in linear dynamic systems can be revealed in the so-called past-future information bottleneck [4].

Data clustering using the information bottleneck

This application of the bottleneck method to non-Gaussian sampled data is described in [4] by Tishby et. el. The concept, as treated there, is not without complication as there are two independent phases in the exercise: firstly estimation of the unknown parent probability densities from which the data samples are drawn and secondly the use of these densities within the information theoretic framework of the bottleneck.

Density estimation

Main article: Density estimation

Since the bottleneck method is framed in probabilistic rather than statistical terms, we first need to estimate the underlying probability density at the sample points $X = x_i$. This is a well known problem with a number of solutions described by Silverman in [5]. In the present method, joint probabilities of the samples are found by use of a Markov transition matrix method and this has some mathematical synergy with the bottleneck method itself.

Define an arbitrarily increasing distance metric f between all sample pairs and distance matrix $d_{i,j} = f\Big(\Big|x_i - x_j\Big|\Big)$. Then compute transition probabilities between sample pairs $P_{i,j} = \exp(-\lambda d_{i,j})$ for some $\lambda > 0$. Treating samples as states, and a normalised version of P as a Markov state transition probability matrix, the vector of probabilities of the 'states' after t steps, conditioned on the initial state $p(0)$, is $p(t) = P^t p(0)$. We are here interested only in the equilibrium probability vector $p(\infty)$ given, in the usual way, by the dominant eigenvector of matrix P which is independent of the initialising vector $p(0)$. This Markov transition method establishes a probability at the sample points which is claimed to be proportional to the probabilities densities there.

Other interpretations of the use of the eigenvalues of distance matrix d are discussed in [6].

Clusters

In the following soft clustering example, the reference vector Y contains sample categories and the joint probability $p(X, Y)$ is assumed known. A soft cluster c_k is defined by its probability distribution over the data samples x_i : $p(c_k|x_i)$. In [1] Tishby et al. present the following iterative set of equations to determine the clusters which are ultimately a generalization of the Blahut-Arimoto algorithm, developed in rate distortion theory. The application of this type of algorithm in neural networks appears to originate in entropy arguments arising in application of Gibbs Distributions in deterministic annealing [7].

$$\begin{cases} p(c|x) = Kp(c)\exp\Big(-\beta\, D^{KL}\Big[p(y|x)\,||\,p(y|c)\Big]\Big) \\ p(y|c) = \sum_x p(y|x)p(c|x)p(x)/p(c) \\ p(c) = \sum_x p(c|x)p(x) \end{cases}$$

The function of each line of the iteration is expanded as follows.

Line 1: This is a matrix valued set of conditional probabilities

$$A_{i,j} = p(c_i|x_j) = Kp(c_i)\exp\Big(-\beta\, D^{KL}\Big[p(y|x_j)\,||\,p(y|c_i)\Big]\Big)$$

The Kullback–Leibler distance D^{KL} between the Y vectors generated by the sample data x and those generated by its reduced information proxy c is applied to assess the fidelity of the compressed vector with respect to the reference (or categorical) data Y in accordance with the fundamental bottleneck equation. $D^{KL}(a||b)$ is the Kullback Leibler distance between distributions a, b

$$D^{KL}(a||b) = \sum_i p(a_i)\log\Big(\frac{p(a_i)}{p(b_i)}\Big)$$

and K is a scalar normalization. The weighting by the negative exponent of the distance means that prior cluster probabilities are downweighted in line 1 when the Kullback Liebler distance is large, thus successful clusters grow in probability while unsuccessful ones decay.

Line 2: This is a second matrix-valued set of conditional probabilities. The steps in deriving it are as follows. We have, by definition

$$\begin{aligned} p(y_i|c_k) &= \sum_j p(y_i|x_j)p(x_j|c_k) \\ &= \sum_j p(y_i|x_j)p(x_j, c_k)/p(c_k) \\ &= \sum_j p(y_i|x_j)p(c_k|x_j)p(x_j)/p(c_k) \end{aligned}$$

where the Bayes identities $p(a, b) = p(a|b)p(b) = p(b|a)p(a)$ are used.

Line 3: this line finds the marginal distribution of the clusters c

$$p(c_i) = \sum_j p(c_i, x_j) \quad = \sum_j p(c_i|x_j)p(x_j)$$

This is also a standard result.

Further inputs to the algorithm are the marginal sample distribution $p(x)$ which has already been determined by the dominant eigenvector of P and the matrix valued Kullback Leibler distance function

$$D^{KL}_{i,j} = D^{KL}\Big[p(y|x_j)\,||\,p(y|c_i)\Big]\Big)$$

derived from the sample spacings and transition probabilities.

The matrix $p(y_i|c_j)$ can be initialised randomly or as a reasonable guess, while matrix $p(c_i|x_j)$ needs no prior values. Although the algorithm is converging, multiple minima may exist which need some action to resolve. Further details, including hard clustering methods, are found in [5].

6.10.2 Defining decision contours

To categorize a new sample x' external to the training set X , apply the previous distance metric to find the transition probabilities between x' and all samples in X : , $\tilde{p}(x_i) = p(x_i|x') = \mathrm{K}\exp\Big(-\lambda f\Big(\Big|x_i - x'\Big|\Big)\Big)$ with K a normalisation. Secondly apply the last two lines of the 3-line algorithm to get cluster, and conditional category probabilities.

$$\tilde{p}(c_i) = p(c_i|x') = \sum_j p(c_i|x_j)p(x_j|x') = \sum_j p(c_i|x_j)\tilde{p}($$

$$p(y_i|c_j) = \sum_k p(y_i|x_k)p(c_j|x_k)p(x_k|x')/p(c_j|x') = \sum_k$$

Finally we have

$$p(y_i|x') = \sum_j p(y_i|c_j)p(c_j|x')) = \sum_j p(y_i|c_j)\tilde{p}(c_j)$$

Parameter β must be kept under close supervision since, as it is increased from zero, increasing numbers of features, in the category probability space, snap into focus at certain critical thresholds.

An example

The following case examines clustering in a four quadrant multiplier with random inputs u, v and two categories of output, ± 1 , generated by $y = \text{sign}(uv)$. This function has the property that there are two spatially separated clusters for each category and so it demonstrates that the method can handle such distributions.

20 samples are taken, uniformly distributed on the square $[-1, 1]^2$. The number of clusters used beyond the number of categories, two in this case, has little effect on performance and the results are shown for two clusters using parameters $\lambda = 3,\ \beta = 2.5$.

The distance function is $d_{i,j} = \left|x_i - x_j\right|^2$ where $x_i = (u_i, v_i)^T$ while the conditional distribution $p(y|x)$ is a 2 × 20 matrix

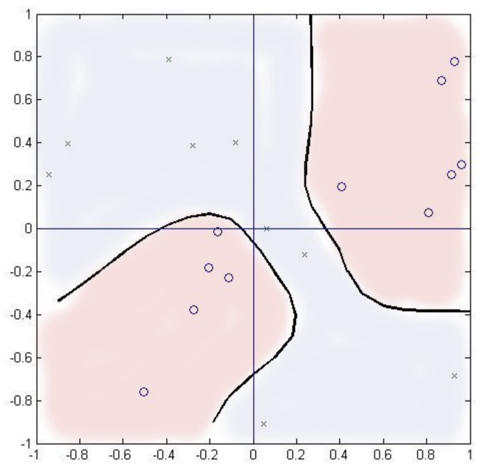
$$Pr(y_i = 1) = 1 \text{ if } \text{sign}(u_i v_i) = 1$$
$$Pr(y_i = -1) = 1 \text{ if } \text{sign}(u_i v_i) = -1$$

and zero elsewhere.

The summation in line 2 is only incorporates two values representing the training values of +1 or −1 but nevertheless seems to work quite well. Five iterations of the equations were used. The figure shows the locations of the twenty samples with '0' representing $Y = 1$ and 'x' representing $Y = -1$. The contour at the unity likelihood ratio level is shown,

$$L = \frac{\text{Pr}(1)}{\text{Pr}(-1)} = 1$$

as a new sample x' is scanned over the square. Theoretically the contour should align with the $u = 0$ and $v = 0$ coordinates but for such small sample numbers they have instead followed the spurious clusterings of the sample points.



Decision contours

Neural network/fuzzy logic analogies

There is some analogy between this algorithm and a neural network with a single hidden layer. The internal nodes are represented by the clusters c_j and the first and second layers of network weights are the conditional probabilities $p(c_j|x_i)$ and $p(y_k|c_j)$ respectively. However, unlike a standard neural network, the present algorithm relies entirely on probabilities as inputs rather than the sample values themselves while internal and output values are all conditional probability density distributions. Nonlinear functions are encapsulated in distance metric $f(.)$ (or *influence functions/radial basis functions*) and transition probabilities instead of sigmoid functions. The Blahut-Arimoto three-line algorithm is seen to converge rapidly, often in tens of iterations, and by varying β , λ and f and the cardinality of the clusters, various levels of focus on data features can be achieved.
The statistical soft clustering definition $p(c_i|x_j)$ has some overlap with the verbal fuzzy membership concept of fuzzy logic.

6.10.3 Bibliography

[1] N. Tishby, F.C. Pereira, and W. Bialek: "The Information Bottleneck method" . The 37th annual Allerton Conference on Communication, Control, and Computing, Sep 1999: pp. 368–377

[2] G. Chechik, A Globerson, N. Tishby and Y. Weiss: "Information Bottleneck for Gaussian Variables" . Journal of Machine Learning Research 6, Jan 2005, pp. 165–188

[3] F. Creutzig, H. Sprekeler: Predictive Coding and the Slowness Principle: an Information-Theoretic Approach, 2008, Neural Computation 20(4): 1026–1041

[4] F. Creutzig, A. Globerson, N. Tishby: Past-future information bottleneck in dynamical systems, 2009, Physical Review E 79, 041925

[5] N Tishby, N Slonim: “Data clustering by Markovian Relaxation and the Information Bottleneck Method” , Neural Information Processing Systems (NIPS) 2000, pp. 640–646

[6] B.W. Silverman: “Density Estimation for Statistical Data Analysis” , Chapman and Hall, 1986.

[7] N. Slonim, N. Tishby: “Document Clustering using Word Clusters via the Information Bottleneck Method” , SIGIR 2000, pp. 208–215

[8] Y. Weiss: “Segmentation using eigenvectors: a unifying view” , Proceedings IEEE International Conference on Computer Vision 1999, pp. 975–982

[9] D. J. Miller, A. V. Rao, K. Rose, A. Gersho: “An Information-theoretic Learning Algorithm for Neural Network Classification” . NIPS 1995: pp. 591–597

[10] P. Harremoes and N. Tishby “The Information Bottleneck Revisited or How to Choose a Good Distortion Measure” . In proceedings of the International Symposium on Information Theory (ISIT) 2007

6.10.4 See also

- Information theory

6.10.5 External links

- Paper by N. Tishby, et al.

6.11 K-SVD

In applied mathematics, **K-SVD** is a dictionary learning algorithm for creating a dictionary for sparse representations, via a singular value decomposition approach. K-SVD is a generalization of the k-means clustering method, and it works by iteratively alternating between sparse coding the input data based on the current dictionary, and updating the atoms in the dictionary to better fit the data.[*][1][*][2] K-SVD can be found widely in use in applications such as image processing, audio processing, biology, and document analysis.

6.11.1 Problem description

Main article: Sparse approximation

The goal of dictionary learning is to learn an overcomplete dictionary matrix $D \in \mathbb{R}^{n\times K}$ that contains K signal-atoms (in this notation, columns of D). A signal vector $y \in \mathbb{R}^n$ can be represented, sparsely, as a linear combination of these atoms; to represent y , the representation vector x should satisfy the exact condition $y = Dx$, or the approximate condition $y \approx Dx$, made precise by requiring that $\|y - Dx\|_p \le \epsilon$ for some small value ε and some Lp norm. The vector $x \in \mathbb{R}^K$ contains the representation coefficients of the signal y . Typically, the norm p is selected as L_1, L_2, or L_∞.

If $n < K$ and D is a full-rank matrix, an infinite number of solutions are available for the representation problem, Hence, constraints should be set on the solution. Also, to ensure sparsity, the solution with the fewest number of nonzero coefficients is preferred. Thus, the sparsity representation is the solution of either

$$(P_0) \quad \min_x \|x\|_0 \qquad \text{to subject} y = Dx$$

or

$$(P_{0,\epsilon}) \quad \min_x \|x\|_0 \qquad \text{to subject} \|y - Dx\|_2 \le \epsilon$$

where the $\|x\|_0$ counts the nonzero entries in the vector x . (See the zero “norm” .)

6.11.2 K-SVD algorithm

K-SVD is a kind of generalization of K-means, as follows. The k-means clustering can be also regarded as a method of sparse representation. That is, finding the best possible codebook to represent the data samples $\{y_i\}_{i=1}^M$ by nearest neighbor, by solving

$$\min_{D,X}\{\|Y-DX\|_F^2\} \qquad \text{to subject} \forall i, x_i = e_k \text{ some for } k.$$

which is quite similar to

$$\min_{D,X}\{\|Y - DX\|_F^2\} \qquad \text{to subject} \quad \forall i, \|x_i\|_0 = 1.$$

The sparse representation term $x_i = e_k$ enforces K-means algorithm to use only one atom (column) in dictionary D. To relax this constraint, the target of the K-SVD algorithm is to represent signal as a linear combination of atoms in D.

The K-SVD algorithm follows the construction flow of K-means algorithm. However, In contrary to K-means, in order to achieve linear combination of atoms in D, sparsity term of the constrain is relaxed so that nonzero entries of each column x_i can be more than 1, but less than a number T_0 .

So, the objective function becomes

$$\min_{D,X}\{\|Y-DX\|_F^2\} \qquad \text{to subject} \quad \forall i\ , \|x_i\|_0 \le T_0.$$

or in another objective form

$$\min_{D,X}\sum_i \|x_i\|_0 \qquad \text{to subject} \quad \forall i\ , \|Y-DX\|_F^2 \le \epsilon.$$

In the K-SVD algorithm, the D is first to be fixed and the best coefficient matrix X . As finding the truly optimal X is impossible, we use an approximation pursuit method. Any such algorithm as OMP, the orthogonal matching pursuit in can be used for the calculation of the coefficients, as long as it can supply a solution with a fixed and predetermined number of nonzero entries T_0 .

After the sparse coding task, the next is to search for a better dictionary D . However, finding the whole dictionary all at a time is impossible, so the process then update only one column of the dictionary D each time while fix X . The update of $k-th$ is done by rewriting the penalty term as

$$\|Y-DX\|_F^2 = \left|Y - \sum_{j=1}^{K} d_j x_T^j\right|_F^2 = \left|\left(Y - \sum_{j\neq k} d_j x_T^j\right) - d_k x_T^k\right|_F^2 = \|E_k - d_k x_T^k\|_F^2$$

where x_T^k denotes the *k*-th row of *X*.

By decomposing the multiplication DX into sum of K rank 1 matrices, we can assume the other $K-1$ terms are assumed fixed, and the $k-th$ remains unknown. After this step, we can solve the minimization problem by approximate the E_k term with a $rank-1$ matrix using singular value decomposition, then update d_k with it. However, the new solution of vector x_T^k is very likely to be filled, because the sparsity constrain is not enforced.

To cure this problem, Define ω_k as

$$\omega_k = \{i \mid 1 \le i \le N, x_T^k(i) \ne 0\}.$$

Which points to examples $\{y_i\}$ that use atom d_k (also the entries of x_i that is nonzero). Then, define Ω_k as a matrix of size $N \times |\omega_k|$, with ones on the $(i\text{-th}, \omega_k(i))$ entries and zeros otherwise. When multiplying $x_R^k = x_T^k\Omega_k$, this shrinks the row vector x_T^k by discarding the nonzero entries. Similarly, the multiplication $Y_k^R = Y\Omega_k$ is the subset of the examples that are current using the d_k atom. The same effect can be seen on $E_k^R = E_k\Omega_k$.

So the minimization problem as mentioned before becomes

$$\|E_k\Omega_k - d_k x_T^k\Omega_k\|_F^2 = \|E_k^R - d_k x_R^k\|_F^2$$

and can be done by directly using SVD. SVD decomposes E_k^R into $U\Delta V^T$. The solution for d_k is the first column of U, the coefficient vector x_R^k as the first column of $V \times \Delta(1,1)$. After updated the whole dictionary, the process then turns to iteratively solve X, then iteratively solve D.

6.11.3 Limitations

Choosing an appropriate "dictionary" for a dataset is a non-convex problem, and K-SVD operates by an iterative update which does not guarantee to find the global optimum.*[2] However, this is common to other algorithms for this purpose, and K-SVD works fairly well in practice.*[2]

6.11.4 See also

- Sparse approximation
- Singular value decomposition
- Matrix norm
- K-means clustering

6.11.5 References

[1] Michal Aharon, Michael Elad, and Alfred Bruckstein (2006), "K-SVD: An Algorithm for Designing Overcomplete Dictionaries for Sparse Representation" (PDF), *IEEE Transactions on Signal Processing* **54** (11): 4311–4322, doi:10.1109/TSP.2006.881199

[2] Rubinstein, R., Bruckstein, A.M., and Elad, M. (2010), "Dictionaries for Sparse Representation Modeling" , *Proceedings of the IEEE* **98** (6): 1045–1057, doi:10.1109/JPROC.2010.2040551

6.12 Linde–Buzo–Gray algorithm

The **Linde–Buzo–Gray algorithm** (introduced by Yoseph Linde, Andrés Buzo and Robert M. Gray in 1980) is a vector quantization algorithm to derive a good codebook.

It is similar to the k-means method in data clustering.

6.12.1 The algorithm

At each iteration, each vector is split into two new vectors.

- A initial state: centroid of the training sequence;
- B initial estimation #1: code book of size 2;
- C final estimation after LGA: Optimal code book with 2 vectors;

- D initial estimation #2: code book of size 4;
- E final estimation after LGA: Optimal code book with 4 vectors;

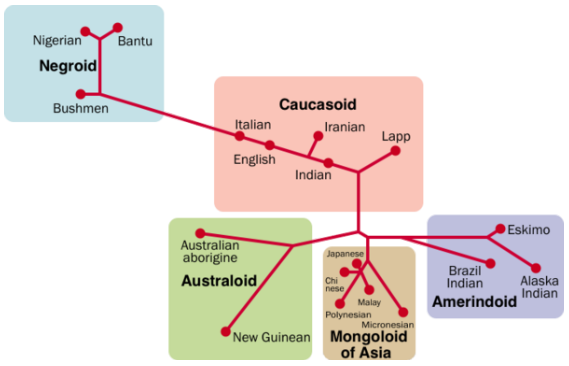
6.12.2 References

- The original paper describing the algorithm, as an extension to Lloyd's algorithm:
  - Linde, Y.; Buzo, A.; Gray, R. (1980). "An Algorithm for Vector Quantizer Design". *IEEE Transactions on Communications* **28**: 84. doi:10.1109/TCOM.1980.1094577.

6.12.3 External links

- http://www.data-compression.com/vq.html#lbg

6.13 Neighbor joining

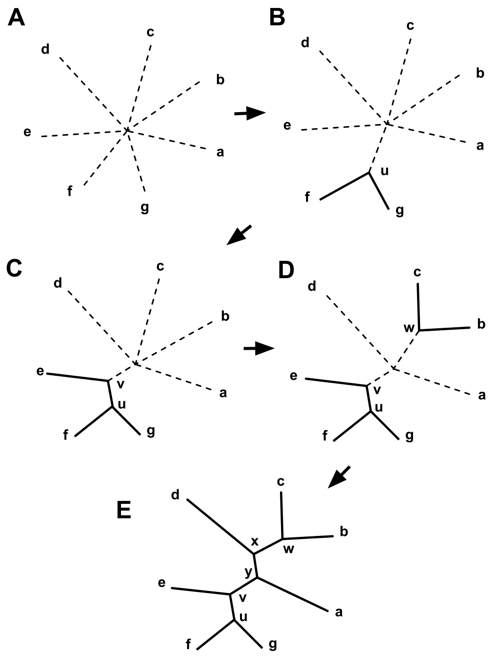


This genetic distance map made in 2002 is an estimate of 18 world human groups by a neighbour-joining method based on 23 kinds of genetic information. It was made by Saitou Naruya (斎藤成也) professor at the (Japanese) National Institute for Genetics.[1]

In bioinformatics, **neighbor joining** is a bottom-up (agglomerative) clustering method for the creation of phylogenetic trees, created by Naruya Saitou and Masatoshi Nei in 1987.[2] Usually used for trees based on DNA or protein sequence data, the algorithm requires knowledge of the distance between each pair of taxa (e.g., species or sequences) to form the tree.[3]

6.13.1 The algorithm

Neighbor joining takes as input a distance matrix specifying the distance between each pair of taxa. The algorithm starts with a completely unresolved tree, whose topology corresponds to that of a star network, and iterates over the following steps until the tree is completely resolved and all branch lengths are known:



Starting with a star tree (A), the Q matrix is calculated and used to choose a pair of nodes for joining, in this case f and g. These are joined to a newly created node, u, as shown in (B). The part of the tree shown as solid lines is now fixed and will not be changed in subsequent joining steps. The distances from node u to the nodes a-e are computed from equation (3). This process is then repeated, using a matrix of just the distances between the nodes, a,b,c,d,e, and u, and a Q matrix derived from it. In this case u and e are joined to the newly created v, as shown in (C). Two more iterations lead first to (D), and then to (E), at which point the algorithm is done, as the tree is fully resolved.

1. Based on the current distance matrix calculate the matrix Q (defined below).
2. Find the pair of distinct taxa i and j (i.e. with $i \neq j$) for which $Q(i, j)$ has its lowest value. These taxa are joined to a newly created node, which is connected to the central node. In the figure at right, f and g are joined to the new node u.
3. Calculate the distance from each of the taxa in the pair to this new node.
4. Calculate the distance from each of the taxa outside of this pair to the new node.
5. Start the algorithm again, replacing the pair of joined neighbors with the new node and using the distances calculated in the previous step.

The Q-matrix

Based on a distance matrix relating the n taxa, calculate Q as follows:

where $d(i,j)$ is the distance between taxa i and j .

Distance from the pair members to the new node

For each of the taxa in the pair being joined, use the following formula to calculate the distance to the new node:

and:

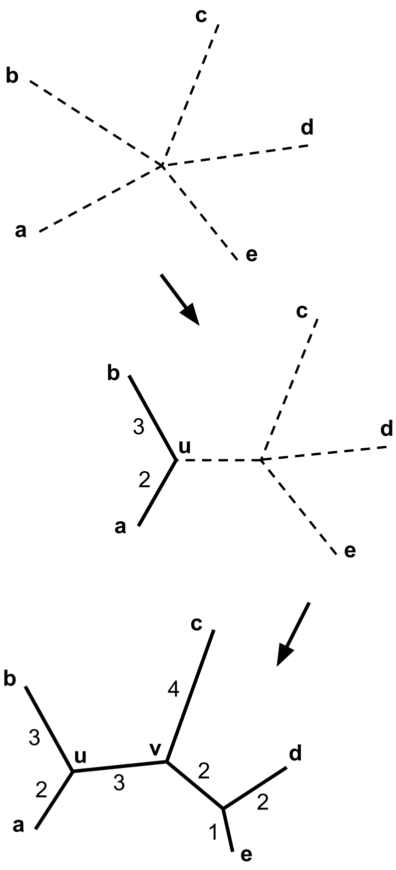
$\delta(g,u) = d(f,g) - \delta(f,u)$

Taxa f and g are the paired taxa and u is the newly created node. The branches joining f and u and g and u , and their lengths, $\delta(f,u)$ and $\delta(g,u)$ are part of the tree which is gradually being created; they neither affect not are affected by later neighbor-joining steps.

Distance of the other taxa from the new node

For each taxon not considered in the previous step, we calculate the distance to the new node as follows:

where u is the new node, k is the node which we want to calculate the distance to and f and g are the members of the pair just joined.



Neighbor joining with 5 taxa. In this case 2 neighbor joining steps give a tree with fully resolved topology. The branches of the resulting tree are labeled with their lengths.

Complexity

Neighbor joining on a set of n taxa requires $n-3$ iterations. At each step one has to build and search a Q matrix. Initially the Q matrix is size $n \times n$, then the next step it is $(n-1) \times (n-1)$, etc. Implementing this in a straightforward way leads to an algorithm with a time complexity of $O(n^3)$; implementations exist which use heuristics to do much better than this on average.

6.13.2 Example

Let us assume that we have five taxa (a,b,c,d,e) and the following distance matrix:

We obtain the following values for the Q matrix (the diagonal elements of the matrix are not used and are omitted here):

In the example above, $Q(a,b) = -50$. This is the smallest value of Q , so we join nodes a and b . Let u denote the new node; the branches joining a and b to u then have lengths $\delta(a,u) = 2$ and $\delta(b,u) = 3$, by equation (2), above.

We then proceed to update the distance matrix; using equation (3) above, we compute the distance from u to each of the other nodes besides a and b . In this case, we obtain $d(u,c) = 7$, $d(u,d) = 7$, and $d(u,e) = 6$. The resulting distance matrix is:

The corresponding Q matrix is:

We may choose either to join u and c , or to join d and e ; both pairs have the minimal Q value of -28 , and either choice leads to the same result. For concreteness, let us join u and c and call the new node v ; this gives branch lengths $\delta(u,v) = 3$ and $\delta(c,v) = 4$ as shown in the figure, and the distance matrix for the remaining 3 nodes, v , d , and e , is:

The tree topology is fully resolved at this point, so we don't need to calculate Q or do any more joining of neighbors. However, we can use these distances to get the remaining 3 branch-lengths, as shown in the figure.

This example represents an idealized case: note that if we move from any taxon to any other along the branches of the tree, and sum the lengths of the branches traversed, the result is equal to the distance between those taxa in the input distance matrix. For example, going from d to b we have $2+2+3+3=10$. A distance matrix whose distances agree in this way with some tree is said to be 'additive', a property which is rare in practice. Nonetheless it is important to note that, given an additive distance matrix as input, neighbor joining is guaranteed to find the tree whose distances between taxa agree with it.

6.13.3 Neighbor joining as minimum evolution

Neighbor joining may be viewed as a greedy algorithm for optimizing a tree according to the 'balanced minimum evolution'*[4] (BME) criterion. For each topology, BME defines the tree length (sum of branch lengths) to be a particular weighted sum of the distances in the distance matrix, with the weights depending on the topology. The BME optimal topology is the one which minimizes this tree length. Neighbor joining at each step greedily joins that pair of taxa which will give the greatest decrease in the estimated tree length. This procedure is not guaranteed to find the topology which is optimal by the BME criterion, although it often does and is usually quite close.

6.13.4 Advantages and disadvantages

The main virtue of NJ is that it is fast, due in part to its being a polynomial-time algorithm. This makes it practical for analyzing large data sets (hundreds or thousands of taxa) and for bootstrapping, for which purposes other means of analysis (e.g. maximum parsimony, maximum likelihood) may be computationally prohibitive.

Neighbor joining has the property that if the input distance matrix is correct, then the output tree will be correct. Furthermore the correctness of the output tree topology is guaranteed as long as the distance matrix is 'nearly additive', specifically if each entry in the distance matrix differs from the true distance by less than half of the shortest branch length in the tree.*[5] In practice the distance matrix rarely satisfies this condition, but neighbor joining often constructs the correct tree topology anyway.*[6] The correctness of neighbor joining for nearly additive distance matrices implies that it is statistically consistent under many models of evolution; given data of sufficient length, neighbor joining will reconstruct the true tree with high probability. Compared with UPGMA, neighbor joining has the advantage that it does not assume all lineages evolve at the same rate (molecular clock hypothesis).

Nevertheless, neighbor joining has been largely superseded by phylogenetic methods that do not rely on distance measures and offer superior accuracy under most conditions. Neighbor joining has the undesirable feature that it often assigns negative lengths to some of the branches.

6.13.5 Implementations and variants

There are many programs available implementing neighbor joining. RapidNJ and NINJA are fast implementations with typical run times proportional to approximately the square of the number of taxa. BIONJ and Weighbor are variants of neighbor joining which improve on its accuracy by making use of the fact that the shorter distances in the distance matrix are generally better known than the longer distances. FastME is an implementation of the closely related balanced minimum evolution method.

6.13.6 See also

- Human genetic clustering
- Nearest neighbor search
- UPGMA

6.13.7 References

[1] Saitou. Kyushu Museum. 2002. February 2, 2007

[2] Saitou N, Nei M. “The neighbor-joining method: a new method for reconstructing phylogenetic trees.” *Molecular Biology and Evolution*, volume 4, issue 4, pp. 406-425, July 1987.

[3] Xavier Didelot (2010). “Sequence-Based Analysis of Bacterial Population Structures” . In D. Ashley Robinson, Daniel Falush, Edward J. Feil. *Bacterial Population Genetics in Infectious Disease*. John Wiley and Sons. pp. 46–47. ISBN 978-0-470-42474-2.

[4] Gascuel O, Steel M (2006). “Neighbor-joining revealed” . *Mol Biol Evol* **23** (11): 1997–2000. doi:10.1093/molbev/msl072. PMID 16877499.

[5] Atteson K (1997). “The performance of neighbor-joining algorithms of phylogeny reconstruction” , pp. 101–110. *In* Jiang, T., and Lee, D., eds., *Lecture Notes in Computer Science, 1276*, Springer-Verlag, Berlin. COCOON '97.

[6] Mihaescu R, Levy D, Pachter L (2009). “Why neighbor-joining works” . *Algorithmica* **54** (1): 1–24. doi:10.1007/s00453-007-9116-4.

Other sources

- Studier JA, Keppler KJ (1988). “A note on the Neighbor-Joining algorithm of Saitou and Nei” (PDF). *Mol Biol Evol* **5** (6): 729–731. PMID 3221794.

- Martin Simonsen, Thomas Mailund, Christian N. S. Pedersen (2008). “Rapid Neighbour Joining” (PDF). *Proceedings of WABI* **5251**: 113–122. doi:10.1007/978-3-540-87361-7_10.

6.13.8 External links

- The Neighbor-Joining Method —a tutorial

6.14 Pitman–Yor process

In probability theory, a **Pitman–Yor process**[1][2][3][4] denoted PY(d, θ, G_0), is a stochastic process whose sample path is a probability distribution. A random sample from this process is an infinite discrete probability distribution, consisting of an infinite set of atoms drawn from G_0, with weights drawn from a two-parameter Poisson–Dirichlet distribution. The process is named after Jim Pitman and Marc Yor.

The parameters governing the Pitman–Yor process are: $0 \leq d < 1$ a discount parameter, a strength parameter $\theta > -d$ and a base distribution G_0 over a probability space X. When $d = 0$, it becomes the Dirichlet process. The discount parameter gives the Pitman–Yor process more flexibility over tail behavior than the Dirichlet process, which has exponential tails. This makes Pitman–Yor process useful for modeling data with power-law tails (e.g., word frequencies in natural language).

The exchangeable random partition induced by the Pitman–Yor process is an example of a Poisson–Kingman partition, and of a Gibbs type random partition.

6.14.1 Naming conventions

The name “Pitman–Yor process” was coined by Ishwaran and James[5] after Pitman and Yor's review on the subject.[2] However the process was originally studied in Perman et al[6][7] so technically it perhaps may have been better named the Perman–Pitman–Yor process.

It is also sometimes referred to as the two-parameter Poisson–Dirichlet process, after the two-parameter generalization of the Poisson–Dirichlet distribution which describes the joint distribution of the sizes of the atoms in the random measure, sorted by strictly decreasing order. However as a name the two-parameter Poisson–Dirichlet process is too long and not very popular. It also emphasizes the strictly decreasing order which is not important in many modeling applications.

6.14.2 See also

- Chinese restaurant process
- Dirichlet distribution
- Latent Dirichlet allocation

6.14.3 References

[1] Ishwaran, H; James, L F (2003). “Generalized weighted Chinese restaurant processes for species sampling mixture models” . *Statistica Sinica* **13**: 1211–1211.

[2] Pitman, Jim; Yor, Marc (1997). “The two-parameter Poisson–Dirichlet distribution derived from a stable subordinator” . *Annals of Probability* **25** (2): 855–900. doi:10.1214/aop/1024404422. MR 1434129. Zbl 0880.60076.

[3] Pitman, Jim (2006). *Combinatorial Stochastic Processes*. Berlin: Springer-Verlag.

[4] Teh, Yee Whye (2006). “A hierarchical Bayesian language model based on Pitman–Yor processes,” . *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics,*.

[5] Ishwaran, H.; James, L. (2001). “Gibbs Sampling Methods for Stick-Breaking Priors” . *Journal of the American Statistical Association*.

[6] Perman, M.; Pitman, J.; Yor, M. (1992). “Size-biased sampling of Poisson point processes and excursions” . *Probability Theory and Related Fields*.

[7] Perman, M. (1990). *Random Discrete Distributions Derived from Subordinators* (Thesis). Department of Statistics, University of California at Berkeley.

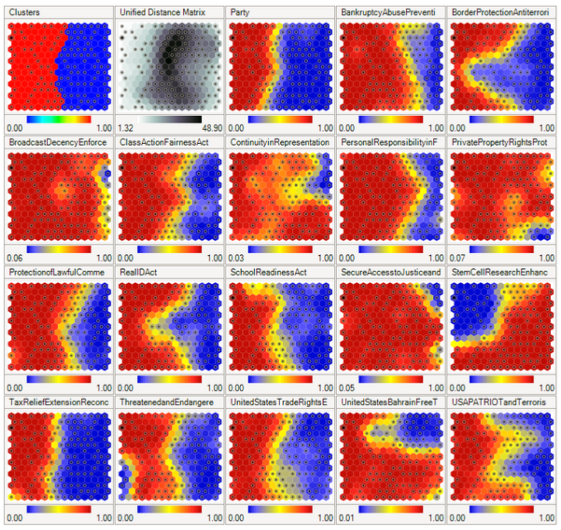
6.15 Self-organizing map

A **self-organizing map** (**SOM**) or **self-organizing feature map** (**SOFM**) is a type of artificial neural network (ANN) that is trained using unsupervised learning to produce a low-dimensional (typically two-dimensional), discretized representation of the input space of the training samples, called a **map**. Self-organizing maps are different from other artificial neural networks in the sense that they use a neighborhood function to preserve the topological properties of the input space.

This makes SOMs useful for visualizing low-dimensional views of high-dimensional data, akin to multidimensional scaling. The artificial neural network introduced by the Finnish professor Teuvo Kohonen in the 1980s is sometimes called a **Kohonen map** or **network**.[1][2] The Kohonen net is a computationally convenient abstraction building on work on biologically neural models from the 1970s[3] and morphogenesis models dating back to Alan Turing in the 1950s[4]

Like most artificial neural networks, SOMs operate in two modes: training and mapping. “Training” builds the map using input examples (a competitive process, also called vector quantization), while “mapping” automatically classifies a new input vector.

A self-organizing map consists of components called nodes or neurons. Associated with each node are a weight



A self-organizing map showing U.S. Congress voting patterns visualized in Synapse. The first two boxes show clustering and distances while the remaining ones show the component planes. Red means a yes vote while blue means a no vote in the component planes (except the party component where red is Republican and blue is Democratic).

vector of the same dimension as the input data vectors, and a position in the map space. The usual arrangement of nodes is a two-dimensional regular spacing in a hexagonal or rectangular grid. The self-organizing map describes a mapping from a higher-dimensional input space to a lower-dimensional map space. The procedure for placing a vector from data space onto the map is to find the node with the closest (smallest distance metric) weight vector to the data space vector.

While it is typical to consider this type of network structure as related to feedforward networks where the nodes are visualized as being attached, this type of architecture is fundamentally different in arrangement and motivation.

Useful extensions include using toroidal grids where opposite edges are connected and using large numbers of nodes.

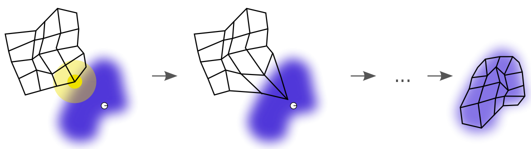
It has been shown that while self-organizing maps with a small number of nodes behave in a way that is similar to K-means, larger self-organizing maps rearrange data in a way that is fundamentally topological in character.*[5]

It is also common to use the U-Matrix.*[6] The U-Matrix value of a particular node is the average distance between the node's weight vector and that of its closest neighbors.*[7] In a square grid, for instance, we might consider the closest 4 or 8 nodes (the Von Neumann and Moore neighborhoods, respectively), or six nodes in a hexagonal grid.

Large SOMs display emergent properties. In maps consisting of thousands of nodes, it is possible to perform cluster operations on the map itself.*[8]

6.15.1 Learning algorithm

The goal of learning in the self-organizing map is to cause different parts of the network to respond similarly to certain input patterns. This is partly motivated by how visual, auditory or other sensory information is handled in separate parts of the cerebral cortex in the human brain.*[9]



An illustration of the training of a self-organizing map. The blue blob is the distribution of the training data, and the small white disc is the current training datum drawn from that distribution. At first (left) the SOM nodes are arbitrarily positioned in the data space. The node (highlighted in yellow) which is nearest to the training datum is selected. It is moved towards the training datum, as (to a lesser extent) are its neighbors on the grid. After many iterations the grid tends to approximate the data distribution (right).

The weights of the neurons are initialized either to small random values or sampled evenly from the subspace spanned by the two largest principal component eigenvectors. With the latter alternative, learning is much faster because the initial weights already give a good approximation of SOM weights.*[10]

The network must be fed a large number of example vectors that represent, as close as possible, the kinds of vectors expected during mapping. The examples are usually administered several times as iterations.

The training utilizes competitive learning. When a training example is fed to the network, its Euclidean distance to all weight vectors is computed. The neuron whose weight vector is most similar to the input is called the best matching unit (BMU). The weights of the BMU and neurons close to it in the SOM lattice are adjusted towards the input vector. The magnitude of the change decreases with time and with distance (within the lattice) from the BMU. The update formula for a neuron v with weight vector $\mathbf{W_v}(s)$ is

$$\mathbf{W_v}(s+1) = \mathbf{W_v}(s) + \Theta(u, v, s)\, \alpha(s)(\mathbf{D}(t) - \mathbf{W_v}(s)),$$

where s is the step index, t an index into the training sample, u is the index of the BMU for $\mathbf{D}(t)$, $\alpha(s)$ is a monotonically decreasing learning coefficient and $\mathbf{D}(t)$ is the input vector; $\Theta(u, v, s)$ is the neighborhood function which gives the distance between the neuron u and the neuron v in step s.*[11] Depending on the implementations, t can scan the training data set systematically (t is 0, 1, 2...T-1, then repeat, T being the training sample's size), be randomly drawn from the data set (bootstrap sampling), or implement some other sampling method (such as jackknifing).

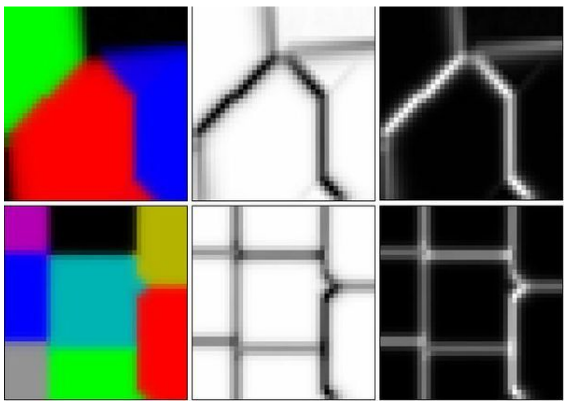
The neighborhood function Θ(u, v, s) depends on the lattice distance between the BMU (neuron *u*) and neuron *v*. In the simplest form it is 1 for all neurons close enough to BMU and 0 for others, but a Gaussian function is a common choice, too. Regardless of the functional form, the neighborhood function shrinks with time.*[9] At the beginning when the neighborhood is broad, the self-organizing takes place on the global scale. When the neighborhood has shrunk to just a couple of neurons, the weights are converging to local estimates. In some implementations the learning coefficient α and the neighborhood function Θ decrease steadily with increasing s, in others (in particular those where t scans the training data set) they decrease in step-wise fashion, once every T steps.

This process is repeated for each input vector for a (usually large) number of cycles **λ**. The network winds up associating output nodes with groups or patterns in the input data set. If these patterns can be named, the names can be attached to the associated nodes in the trained net.

During mapping, there will be one single *winning* neuron: the neuron whose weight vector lies closest to the input vector. This can be simply determined by calculating the Euclidean distance between input vector and weight vector.

While representing input data as vectors has been emphasized in this article, it should be noted that any kind of object which can be represented digitally, which has an appropriate distance measure associated with it, and in which the necessary operations for training are possible can be used to construct a self-organizing map. This includes matrices, continuous functions or even other self-organizing maps.

Preliminary definitions



Self organizing maps (SOM) of three and eight colors with U-Matrix.

Consider an n×m array of nodes, each of which contains a weight vector and is aware of its location in the array. Each weight vector is of the same dimension as the node's input vector. The weights may initially be set to random values.

Now we need input to feed the map —The generated map and the given input exist in separate subspaces. We will create three vectors to represent colors. Colors can be represented by their red, green, and blue components. Consequently our input vectors will have three components, each corresponding to a color space. The input vectors will be:

R = <255, 0, 0>

G = <0, 255, 0>

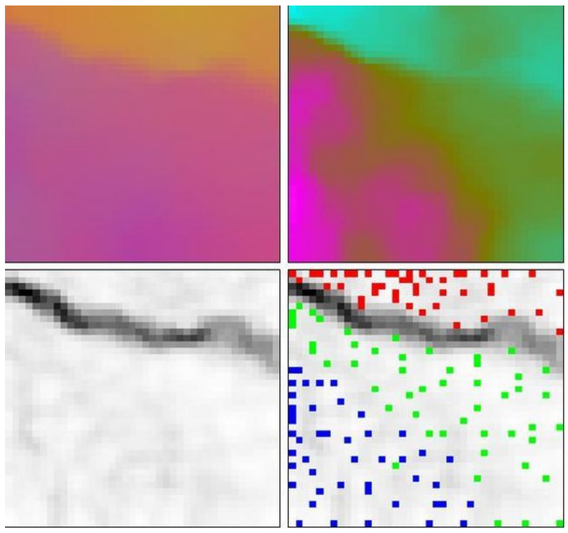
B = <0, 0, 255>

The color training vector data sets used in SOM:

threeColors = [255, 0, 0], [0, 255, 0], [0, 0, 255]

eightColors = [0, 0, 0], [255, 0, 0], [0, 255, 0], [0, 0, 255], [255, 255, 0], [0, 255, 255], [255, 0, 255], [255, 255, 255]

The data vectors should preferably be normalized (vector length is equal to one) before training the SOM.



Self organizing map of Fisher's Iris flower data.

Neurons (40×40 square grid) are trained for 250 iterations with a learning rate of 0.1 using the normalized Iris flower data set which has four-dimensional data vectors. Shown are: a color image formed by the first three dimensions of the four-dimensional SOM weight vectors (top left), a pseudo-color image of the magnitude of the SOM weight vectors (top right), a U-Matrix (Euclidean distance between weight vectors of neighboring cells) of the SOM (bottom left), and an overlay of data points (red: *I. setosa*, green: *I. versicolor* and blue: *I. virginica*) on the U-Matrix based on the minimum Euclidean distance between data vectors and SOM weight vectors (bottom right).

Variables

These are the variables needed, with vectors in bold,

- s is the current iteration
- λ is the iteration limit
- t is the index of the target input data vector in the input data set **D**
- $\mathbf{D}(\mathbf{t})$ is a target input data vector
- v is the index of the node in the map
- $\mathbf{W_v}$ is the current weight vector of node v
- u is the index of the best matching unit (BMU) in the map
- $\Theta(u, v, s)$ is a restraint due to distance from BMU, usually called the neighborhood function, and
- $\alpha(s)$ is a learning restraint due to iteration progress.

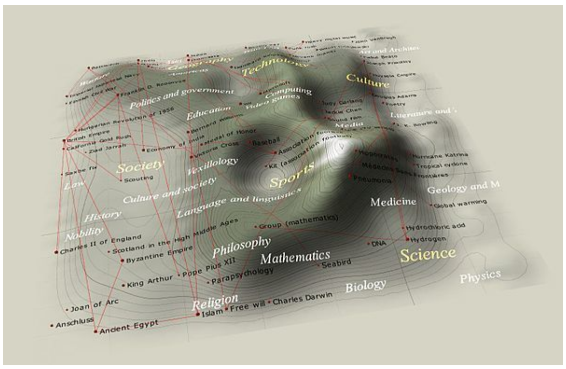
Algorithm

1. Randomize the map's nodes' weight vectors
2. Grab an input vector $\mathbf{D}(\mathbf{t})$
3. Traverse each node in the map
   (a) Use the Euclidean distance formula to find the similarity between the input vector and the map's node's weight vector
   (b) Track the node that produces the smallest distance (this node is the best matching unit, BMU)
4. Update the nodes in the neighborhood of the BMU (including the BMU itself) by pulling them closer to the input vector
   (a) $\mathbf{W_v}(s + 1) = \mathbf{W_v}(s) + \Theta(u, v, s)\ \alpha(s)(\mathbf{D}(t) - \mathbf{W_v}(s))$
5. Increase s and repeat from step 2 while $s < \lambda$

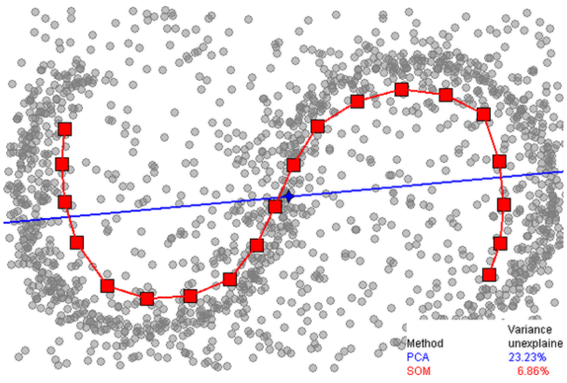
A variant algorithm:

1. Randomize the map's nodes' weight vectors
2. Traverse each input vector in the input data set
   (a) Traverse each node in the map
      i. Use the Euclidean distance formula to find the similarity between the input vector and the map's node's weight vector
      ii. Track the node that produces the smallest distance (this node is the best matching unit, BMU)
   (b) Update the nodes in the neighborhood of the BMU (including the BMU itself) by pulling them closer to the input vector
      i. $\mathbf{W_v}(s + 1) = \mathbf{W_v}(s) + \Theta(u, v, s)\ \alpha(s)(\mathbf{D}(t) - \mathbf{W_v}(s))$
3. Increase s and repeat from step 2 while $s < \lambda$

6.15.2 Interpretation



Cartographical representation of a self-organizing map (U-Matrix) based on Wikipedia featured article data (word frequency). Distance is inversely proportional to similarity. The "mountains" are edges between clusters. The red lines are links between articles.



One-dimensional SOM versus principal component analysis (PCA) for data approximation. SOM is a red broken line with squares, 20 nodes. The first principal component is presented by a blue line. Data points are the small grey circles. For PCA, the fraction of variance unexplained in this example is 23.23%, for SOM it is 6.86%.[12]

There are two ways to interpret a SOM. Because in the training phase weights of the whole neighborhood are moved in the same direction, similar items tend to excite adjacent neurons. Therefore, SOM forms a semantic map where similar samples are mapped close together and dissimilar ones apart. This may be visualized by a U-Matrix (Euclidean distance between weight vectors of neighboring cells) of the SOM.*[6]*[7]*[13]

The other way is to think of neuronal weights as pointers to the input space. They form a discrete approximation of the distribution of training samples. More neurons point to regions with high training sample concentration and fewer where the samples are scarce.

SOM may be considered a nonlinear generalization of Principal components analysis (PCA).*[14] It has been shown, using both artificial and real geophysical data, that SOM has many advantages*[15]*[16] over the conventional feature extraction methods such as Empirical Orthogonal Functions (EOF) or PCA.

Originally, SOM was not formulated as a solution to an optimisation problem. Nevertheless, there have been several attempts to modify the definition of SOM and to formulate an optimisation problem which gives similar results.*[17] For example, Elastic maps use the mechanical metaphor of elasticity to approximate principal manifolds:*[18] the analogy is an elastic membrane and plate.

6.15.3 Alternatives

- The **generative topographic map** (GTM) is a potential alternative to SOMs. In the sense that a GTM explicitly requires a smooth and continuous mapping from the input space to the map space, it is topology preserving. However, in a practical sense, this measure of topological preservation is lacking.*[19]
- The **time adaptive self-organizing map** (TASOM) network is an extension of the basic SOM. The TASOM employs adaptive learning rates and neighborhood functions. It also includes a scaling parameter to make the network invariant to scaling, translation and rotation of the input space. The TASOM and its variants have been used in several applications including adaptive clustering, multilevel thresholding, input space approximation, and active contour modeling.*[20] Moreover, a Binary Tree TASOM or BTASOM, resembling a binary natural tree having nodes composed of TASOM networks has been proposed where the number of its levels and the number of its nodes are adaptive with its environment.*[21]
- The **growing self-organizing map** (GSOM) is a growing variant of the self-organizing map. The GSOM was developed to address the issue of identifying a suitable map size in the SOM. It starts with a minimal number of nodes (usually four) and grows new nodes on the boundary based on a heuristic. By using a value called the *spread factor*, the data analyst has the ability to control the growth of the GSOM.
- The **elastic maps** approach*[22] borrows from the spline interpolation the idea of minimization of the elastic energy. In learning, it minimizes the sum of quadratic bending and stretching energy with the least squares approximation error.
- The conformal approach *[23]*[24] that uses conformal mapping to interpolate each training sample between grid nodes in a continuous surface. An one-to-one smooth mapping is possible in this approach.

6.15.4 Applications

- Meteorology and oceanography*[25]
- Project prioritization and selection *[26]

6.15.5 See also

- Neural gas
- Liquid state machine
- Large Memory Storage and Retrieval (LAMSTAR) neural networks (See: Graupe D, Kordylewski H, (1996), "A Large-Memory Storage and Retrieval Neural Network for Browsing and Medical Diagnosis" , Proc. 6th ANNIE Conf., St. Louis, Missouri, ASME Press, 711-716; Graupe D, (2013), "Principles of Artificial Neural Networks" , 3rd Edition, World Scientific Publishing)
- Hybrid Kohonen SOM
- Sparse coding
- Sparse distributed memory
- Deep learning
- Neocognitron
- Topological data analysis

6.15.6 References

[1] Kohonen, Teuvo; Honkela, Timo (2007). "Kohonen Network" . *Scholarpedia*.

[2] Kohonen, Teuvo (1982). "Self-Organized Formation of Topologically Correct Feature Maps" . *Biological Cybernetics* **43** (1): 59–69. doi:10.1007/bf00337288.

[3] Von der Malsburg, C (1973). "Self-organization of orientation sensitive cells in the striate cortex" . *Kybernetik* **14**: 85–100. doi:10.1007/bf00288907.

[4] Turing, Alan (1952). "The chemical basis of morphogenesis" . *Phil. Trans. Of the Royal Society* **237**: 5–72.

[5] "Self-organizing map" .

[6] Ultsch, Alfred; Siemon, H. Peter (1990). "Kohonen's Self Organizing Feature Maps for Exploratory Data Analysis" . In Widrow, Bernard; Angeniol, Bernard. *Proceedings of the International Neural Network Conference (INNC-90), Paris, France, July 9–13, 1990* **1**. Dordrecht, Netherlands: Kluwer. pp. 305–308. ISBN 978-0-7923-0831-7.

[7] Ultsch, Alfred (2003); *U*-Matrix: A tool to visualize clusters in high dimensional data*, Department of Computer Science, University of Marburg, Technical Report Nr. 36:1-12

[8] Ultsch, Alfred (2007). "Emergence in Self-Organizing Feature Maps". In Ritter, H.; Haschke, R. *Proceedings of the 6th International Workshop on Self-Organizing Maps (WSOM '07)*. Bielefeld, Germany: Neuroinformatics Group. ISBN 978-3-00-022473-7.

[9] Haykin, Simon (1999). "9. Self-organizing maps". *Neural networks - A comprehensive foundation* (2nd ed.). Prentice-Hall. ISBN 0-13-908385-5.

[10] Kohonen, Teuvo (2005). "Intro to SOM". *SOM Toolbox*. Retrieved 2006-06-18.

[11] Kohonen, Teuvo; Honkela, Timo (2011). "Kohonen network". *Scholarpedia*. Retrieved 2012-09-24.

[12] Illustration is prepared using free software: Mirkes, Evgeny M.; *Principal Component Analysis and Self-Organizing Maps: applet*, University of Leicester, 2011

[13] Saadatdoost, Robab, Alex Tze Hiang Sim, and Jafarkarimi, Hosein. "Application of self organizing map for knowledge discovery based in higher education data." Research and Innovation in Information Systems (ICRIIS), 2011 International Conference on. IEEE, 2011.

[14] Yin, Hujun; *Learning Nonlinear Principal Manifolds by Self-Organising Maps*, in Gorban, Alexander N.; Kégl, Balázs; Wunsch, Donald C.; and Zinovyev, Andrei (Eds.); *Principal Manifolds for Data Visualization and Dimension Reduction*, Lecture Notes in Computer Science and Engineering (LNCSE), vol. 58, Berlin, Germany: Springer, 2008, ISBN 978-3-540-73749-0

[15] Liu, Yonggang; and Weisberg, Robert H. (2005); *Patterns of Ocean Current Variability on the West Florida Shelf Using the Self-Organizing Map*, Journal of Geophysical Research, 110, C06003, doi:10.1029/2004JC002786

[16] Liu, Yonggang; Weisberg, Robert H.; and Mooers, Christopher N. K. (2006); *Performance Evaluation of the Self-Organizing Map for Feature Extraction*, Journal of Geophysical Research, 111, C05018, doi:10.1029/2005jc003117

[17] Heskes, Tom; *Energy Functions for Self-Organizing Maps*, in Oja, Erkki; and Kaski, Samuel (Eds.), *Kohonen Maps*, Elsevier, 1999

[18] Gorban, Alexander N.; Kégl, Balázs; Wunsch, Donald C.; and Zinovyev, Andrei (Eds.); *Principal Manifolds for Data Visualization and Dimension Reduction*, Lecture Notes in Computer Science and Engineering (LNCSE), vol. 58, Berlin, Germany: Springer, 2008, ISBN 978-3-540-73749-0

[19] Kaski, Samuel (1997). "Data Exploration Using Self-Organizing Maps". *Acta Polytechnica Scandinavica*. Mathematics, Computing and Management in Engineering Series No. 82 (Espoo, Finland: Finnish Academy of Technology). ISBN 952-5148-13-0.

[20] Shah-Hosseini, Hamed; Safabakhsh, Reza (April 2003). "TASOM: A New Time Adaptive Self-Organizing Map". *IEEE Transactions on Systems, Man, and Cybernetics—Part B: Cybernetics* **33** (2): 271–282. doi:10.1109/tsmcb.2003.810442.

[21] Shah-Hosseini, Hamed (May 2011). "Binary Tree Time Adaptive Self-Organizing Map". *Neurocomputing* **74** (11): 1823–1839. doi:10.1016/j.neucom.2010.07.037.

[22] A. N. Gorban, A. Zinovyev, Principal manifolds and graphs in practice: from molecular biology to dynamical systems, International Journal of Neural Systems, Vol. 20, No. 3 (2010) 219–232.

[23] Liou, C.-Y.; Kuo, Y.-T. (2005). "Conformal Self-organizing Map for a Genus Zero Manifold". *The Visual Computer* **21** (5): 340–353. doi:10.1007/s00371-005-0290-6.

[24] Liou, C.-Y.; Tai, W.-P. (2000). "Conformality in the self-organization network". *Artificial Intelligence* **116**: 265–286. doi:10.1016/S0004-3702(99)00093-4.

[25] Liu, Y.,and R.H. Weisberg (2011) A review of self-organizing map applications in meteorology and oceanography. In: Self-Organizing Maps-Applications and Novel Algorithm Design, 253-272.

[26] Zheng, G. and Vaishnavi, V. (2011) "A Multidimensional Perceptual Map Approach to Project Prioritization and Selection," AIS Transactions on Human-Computer Interaction (3) 2, pp. 82-103

Application of self-organising maps and multi-layer perceptron-artificial neural networks for streamflow and water level forecasting in data-poor catchments: the case of the Lower Shire floodplain, Malawi. http://www.iwaponline.com/nh/up/nh2014168.htm

6.15.7 External links

- Self-organizing maps for WEKA: Implementation of a self-organizing maps in Java, for the WEKA Machine Learning Workbench.
- Self-organizing maps for Ruby: Implementation of self-organizing maps in Ruby, for the AI4R project.
- Self-organizing map for JavaScript: An open-source implementation of a self-organizing map in JavaScript for node.js from Lucid Technics, LLC.
- Self-organizing map for Python: An open-source implementation of a self-organizing map in python. The SOM structure and training procedure is similar to som toolbox for Matlab
- Self-organizing map for Haskell: An open-source implementation of a self-organising map in Haskell.
- A Self-organizing Map implementation for PHP An open-source implementation of a self-organizing map in PHP.

- Spice-SOM: A free GUI application of self-organizing map
- IFCSoft: An open-source Java platform for generating self-organizing maps
- DemoGNG: Java applet implementing self-organizing maps and other network models (neural gas, growing neural gas, growing grid etc.)
- kohonen An open source Supervised and unsupervised self-organising maps package for R.
- supraHex A supra-hexagonal map for analysing high-dimensional omics data.

6.16 SUBCLU

SUBCLU is an algorithm for clustering high-dimensional data by Karin Kailing, Hans-Peter Kriegel and Peer Kröger.*[1] It is a subspace clustering algorithm that builds on the density-based clustering algorithm DBSCAN. SUBCLU can find clusters in axis-parallel subspaces, and uses a bottom-up, greedy strategy to remain efficient.

6.16.1 Approach

SUBCLU uses a monotonicity criteria: if a cluster is found in a subspace S , then each subspace $T \subseteq S$ also contains a cluster. However, a cluster $C \subseteq DB$ in subspace S is not necessarily a cluster in $T \subseteq S$, since clusters are required to be maximal, and more objects might be contained in the cluster in T that contains C . However, a density-connected set in a subspace S is also a density-connected set in $T \subseteq S$.

This *downward-closure property* is utilized by SUBCLU in a way similar to the Apriori algorithm: first, all 1-dimensional subspaces are clustered. All clusters in a higher-dimensional subspace will be subsets of the clusters detected in this first clustering. SUBCLU hence recursively produces $k+1$ -dimensional candidate subspaces by combining k -dimensional subspaces with clusters sharing $k-1$ attributes. After pruning irrelevant candidates, DBSCAN is applied to the candidate subspace to find out if it still contains clusters. If it does, the candidate subspace is used for the next combination of subspaces. In order to improve the runtime of DBSCAN, only the points known to belong to clusters in one k -dimensional subspace (which is chosen to contain as little clusters as possible) are considered. Due to the downward-closure property, other point cannot be part of a $k+1$ -dimensional cluster anyway.

6.16.2 Pseudocode

SUBCLU takes two parameters, ϵ and $MinPts$, which serve the same role as in DBSCAN. In a first step, DBSCAN is used to find 1D-clusters in each subspace spanned by a single attribute:

```
$SUBCLU(DB, eps, MinPts)$
    $S_1 := \emptyset$
    $C_1 := \emptyset$
    $for\, each\, a \in Attributes$
        $C^{\{a\}} = DBSCAN(DB, \{a\}, eps, MinPts)$
        $if (C^{\{a\}} \neq \emptyset)$
        $S_1 := S_1 \cup \{a\}$
        $C_1 := C_1 \cup C^{\{a\}}$
        $end\, if$
    $end\, for$
```

In a second step, $k+1$ -dimensional clusters are built from k -dimensional ones:

```
    $k := 1$
    $while (C_k \neq \emptyset)$
        $CandS_{k+1} := GenerateCandidateSubspaces(S_k)$
        $for\, each\, cand \in CandS_{k+1}$
            $bestSubspace := \min_{s \in S_k \wedge s \subset cand} \sum_{C_i \in C^s} |C_i|$
            $C^{cand} := \emptyset$
            $for\, each\, cluster\, cl \in C^{bestSubspace}$
                $C^{cand} := C^{cand} \cup DBSCAN(cl, cand, eps, MinPts)$
                $if\, (C^{cand} \neq \emptyset)$
                $S_{k+1} := S_{k+1} \cup cand$
                $C_{k+1} := C_{k+1} \cup C^{cand}$
                $end\, if$
            $end\, for$
        $end\, for$
        $k := k+1$
    $end\, while$
$end$
```

The set S_k contains all the k -dimensional subspaces that are known to contain clusters. The set C_k contains the sets of clusters found in the subspaces. The

$bestSubspace$ is chosen to minimize the runs of DBSCAN (and the number of points that need to be considered in each run) for finding the clusters in the candidate subspaces.

Candidate subspaces are generated much alike the Apriori algorithm generates the frequent itemset candidates: Pairs of the k -dimensional subspaces are compared, and if they differ in one attribute only, they form a $k+1$ -dimensional candidate. However, a number of irrelevant candidates are found as well; they contain a k -dimensional subspace that does not contain a cluster. Hence, these candidates are removed in a second step:

```
$GenerateCandidateSubspaces(S_k)$
    $CandS_{k+1} := \emptyset$
    $for\ each\ s_1 \in S_k$
        $for\ each\ s_2 \in S_k$
            $if\ (s_1\ and\ s_2\ differ\ in\ exactely\ one\ attribute)$
            $CandS_{k+1} := CandS_{k+1} \cup \{s_1 \cup s_2\}$
            $end\ if$
        $end\ for$
    $end\ for$
    // Pruning of irrelevant candidate subspaces
    $for\ each\ cand \in CandS_{k+1}$
        $for\ each\ k-element\ s \subset cand$
            $if\ (s \not\in S_k)$
            $CandS_{k+1} = CandS_{k+1} \setminus \{cand\}$
            $end\ if$
        $end\ for$
    $end\ for$
$end$
```

6.16.3 Availability

An example implementation of SUBCLU is available in the ELKI framework.

6.16.4 References

[1] Karin Kailing, Hans-Peter Kriegel and Peer Kröger. *Density-Connected Subspace Clustering for High-Dimensional Data*. In: *Proc. SIAM Int. Conf. on Data Mining (SDM'04)*, pp. 246-257, 2004.

6.17 Ward's method

In statistics, **Ward's method** is a criterion applied in hierarchical cluster analysis. **Ward's minimum variance method** *inaccurate, see talk is a special case of the objective function approach originally presented by Joe H. Ward, Jr.*[1] Ward suggested a general agglomerative hierarchical clustering procedure, where the criterion for choosing the pair of clusters to merge at each step is based on the optimal value of an objective function. This objective function could be "any function that reflects the investigator's purpose." Many of the standard clustering procedures are contained in this very general class. To illustrate the procedure, Ward used the example where the objective function is the error sum of squares, and this example is known as *Ward's method* or more precisely *Ward's minimum variance method*.

6.17.1 The minimum variance criterion

Ward's minimum variance criterion minimizes the total within-cluster variance. At each step the pair of clusters with minimum between-cluster distance are merged. To implement this method, at each step find the pair of clusters that leads to minimum increase in total within-cluster variance after merging. This increase is a weighted squared distance between cluster centers. At the initial step, all clusters are singletons (clusters containing a single point). To apply a recursive algorithm under this objective function, the initial distance between individual objects must be (proportional to) squared Euclidean distance.

The initial cluster distances in Ward's minimum variance method are therefore defined to be the squared Euclidean distance between points:

$$d_{ij} = d(\{X_i\}, \{X_j\}) = \|X_i - X_j\|^2.$$

Note: In software that implements Ward's method, it is important to check whether the function arguments should specify Euclidean distances or squared Euclidean distances.

6.17.2 Lance–Williams algorithms

Ward's minimum variance method can be defined and implemented recursively by a Lance–Williams algorithm.[2] The Lance–Williams algorithms are an infinite family of agglomerative hierarchical clustering algorithms which are represented by a recursive formula for updating cluster distances at each step (each time a pair of clusters is merged). At each step, it is necessary to optimize the objective function (find the optimal pair of clusters to merge). The recursive formula simplifies finding the optimal pair.

Suppose that clusters C_i and C_j were next to be merged. At this point all of the current pairwise cluster distances are known. The recursive formula gives the updated cluster distances following the pending merge of clusters C_i and C_j . Let

- d_{ij} , d_{ik} , and d_{jk} be the pairwise distances between clusters C_i , C_j , and C_k , respectively,
- $d_{(ij)k}$ be the distance between the new cluster $C_i \cup C_j$ and C_k .

An algorithm belongs to the Lance-Williams family if the updated cluster distance $d_{(ij)k}$ can be computed recursively by

$$d_{(ij)k} = \alpha_i d_{ik} + \alpha_j d_{jk} + \beta d_{ij} + \gamma |d_{ik} - d_{jk}|,$$

where $\alpha_i, \alpha_j, \beta,$ and γ are parameters, which may depend on cluster sizes, that together with the cluster distance function d_{ij} determine the clustering algorithm. Several standard clustering algorithms such as single linkage, complete linkage, and group average method have a recursive formula of the above type. A table of parameters for standard methods is given by several authors.*[2]*[3]*[4]

Ward's minimum variance method can be implemented by the Lance–Williams formula. For disjoint clusters $C_i, C_j,$ and C_k with sizes $n_i, n_j,$ and n_k respectively:

$$d(C_i \cup C_j, C_k) = \frac{n_i + n_k}{n_i + n_j + n_k}\, d(C_i, C_k) + \frac{n_j + n_k}{n_i + n_j + n_k}\, d(C_j, C_k) - \frac{n_k}{n_i + n_j + n_k}\, d(C_i, C_j).$$

Hence Ward's method can be implemented as a Lance–Williams algorithm with

$$\alpha_l = \frac{n_i + n_k}{n_i + n_j + n_k}, \qquad \beta = \frac{-n_k}{n_i + n_j + n_k}, \qquad \gamma = 0.$$

6.17.3 References

[1] Ward, J. H., Jr. (1963), "Hierarchical Grouping to Optimize an Objective Function" , *Journal of the American Statistical Association*, 58, 236–244.

[2] Cormack, R. M. (1971), "A Review of Classification" , *Journal of the Royal Statistical Society*, Series A, *134(3), 321-367.*

[3] Gordon, A. D. (1999), *Classification, 2nd Edition*, Chapman and Hall, Boca Raton.

[4] Milligan, G. W. (1979), "Ultrametric Hierarchical Clustering Algorithms" , *Psychometrika*, 44(3), 343–346.

6.17.4 Further reading

- Everitt, B. S., Landau, S. and Leese, M. (2001), *Cluster Analysis, 4th Edition*, Oxford University Press, Inc., New York; Arnold, London. ISBN 0340761199
- Hartigan, J. A. (1975), *Clustering Algorithms*, New York: Wiley.
- Jain, A. K. and Dubes, R. C. (1988), *Algorithms for Clustering Data*, New Jersey: Prentice–Hall.
- Kaufman, L. and Rousseeuw, P. J. (1990), *Finding Groups in Data: An Introduction to Cluster Analysis*, New York: Wiley.

Chapter 7

Text and image sources, contributors, and licenses

7.1 Text

Hardy, Lexor, Kku, Delirium, Dmb000006, Ketil, Pearle, TheParanoidOne, Oleg Alexandrov, Graham87, Rjwilmsi, Bluebot, AnAj, Alexbateman, WatsonCN, Jonesey95, Robertcedgar, Math-ghamhainn, Zvrkast, BG19bot, Liwz, ChrisGualtieri, InsightSeeker, Phleg1 and Anonymous: 29

- **Data stream clustering** *Source:* http://en.wikipedia.org/wiki/Data%20stream%20clustering?oldid=655963874 *Contributors:* Phil Boswell, Bearcat, Rjwilmsi, Malcolma, SmackBot, Rettetast, Melcombe, Yobot, Bfoteini, SporkBot, Fazlican, Frietjes, Nachklang and Anonymous: 4
- **Constrained clustering** *Source:* http://en.wikipedia.org/wiki/Constrained%20clustering?oldid=582374369 *Contributors:* Danski14, Gary, Ruud Koot, SmackBot, CmdrObot, Jmacglashan, Lamro, Melcombe, AnomieBOT, Ergosys, SporkBot and Anonymous: 6
- **Fuzzy clustering** *Source:* http://en.wikipedia.org/wiki/Fuzzy%20clustering?oldid=660200389 *Contributors:* Behnam, Andreas Kaufmann, Flammifer, Alai, DESiegel, Light current, Koblentz, Moxon, Mcld, Bluebot, Bairam, Abhineetnazi, Shmlchr, Phoolimin, McSly, Coffee, ClueBot, 1ForTheMoney, XLinkBot, P.r.newman, Addbot, Themfromspace, AnomieBOT, Lynxoid84, Dan Golding, WadiEgg, Chire, Rafnuss, Helpful Pixie Bot, BG19bot, BattyBot, ChrisGualtieri and Anonymous: 29
- **Spectral clustering** *Source:* http://en.wikipedia.org/wiki/Spectral%20clustering?oldid=654148039 *Contributors:* Michael Hardy, Qwertyus, Rjwilmsi, Naught101, Took, Melcombe, Sameer0s, Yobot, AnomieBOT, Trappist the monk, RjwilmsiBot, John of Reading, Chire, Tdietterich, Habil zare, Bluesky234, Tokekar, Zoratao, Andy Allinger, Mark viking, PapercoreEdit, Motrom, Monkbot and Anonymous: 13
- **Determining the number of clusters in a data set** *Source:* http://en.wikipedia.org/wiki/Determining%20the%20number%20of%20clusters%20in%20a%20data%20set?oldid=661006527 *Contributors:* Fnielsen, Janka~enwiki, Cfp, Zaslav, Comtebenoit, Qwertyus, Rjwilmsi, Stephenb, Cyocum, Talgalili, BossOfTheGame, David Eppstein, StevenBell, PerryTachett, JL-Bot, SpikeToronto, Yobot, Citation bot, Rainbowgoblin, Erik9bot, JohnMeier, DrilBot, Trappist the monk, N.maisonneuve, Chire, Fazlican, Srueter, BattyBot, Pratyya Ghosh, Monkbot and Anonymous: 18
- **Expectation–maximization algorithm** *Source:* http://en.wikipedia.org/wiki/Expectation%E2%80%93maximization%20algorithm?oldid=662388702 *Contributors:* Rodrigob, Michael Hardy, Karada, Jrauser, BAxelrod, Hike395, Phil Boswell, Owenman, Robbyjo~enwiki, Benwing, Wile E. Heresiarch, Giftlite, Paisa, Vadmium, Onco p53, MarkSweep, Piotrus, Cataphract, Rama, MisterSheik, Alex Kosorukoff, O18, John Vandenberg, Jjmerelo~enwiki, 3mta3, Terrycojones, B k, Eric Kvaalen, Cburnett, Finfobia, Jheald, Forderud, Sergey Dmitriev, Igny, Bkkbrad, Bluemoose, Btyner, Qwertyus, Rjwilmsi, KYPark, Salix alba, Hild, Mathbot, Glopk, Kri, BradBeattie, YurikBot, Nils Grimsmo, Schmock, Régis B., Klutzy, Hakeem.gadi, Maechler, Ladypine, M.A.Dabbah, SmackBot, Mcld, Nbarth, Tekhnofiend, Iwaterpolo, Bilgrau, Joeyo, Raptur, Derek farn, Jrouquie, Dicklyon, Alex Selby, Saviourmachine, Lavaka, Requestion, Cydebot, A876, Kallerdis, Libro0, Blaisorblade, Skittleys, Andyrew609, Talgalili, Tiedyeina, Rusmike, Headbomb, RobHar, LachlanA, AnAj, Zzpmarco, Dekimasu, JamesBWatson, Richard Bartholomew, Livingthingdan, Nkwatra, User A1, Edratzer, Osquar F, Numbo3, Salih, GongYi, Douglas-Lanman, Bigredbrain, Market Efficiency, Lamro, Daviddoria, Pine900, Tambal, Mosaliganti1.1, Melcombe, Sitush, Pratx, Alexbot, Hbeigi, Jakarr, Jwmarck, XLinkBot, Jamshidian, Addbot, Sunjuren, Fgnievinski, LaaknorBot, Aanthony1243, Peni, Luckas-bot, Yobot, LeonardoWeiss, AnomieBOT, Citation bot, TechBot, Chuanren, FrescoBot, Nageh, Erhanbas, Nocheenlatierra, Qiemem, Kiefer.Wolfowitz, Jmc200, Stpasha, Jszymon, GeypycGn, Trappist the monk, Thái Nhi, Ismailari, Dropsciencenotbombs, RjwilmsiBot, Slon02, EmausBot, Mikealandewar, John of Reading, Ш, Chire, Statna, ClueBot NG, Rezabot, Meea, Qwerty9967, Helpful Pixie Bot, Rxnt, Bibcode Bot, BG19bot, Chafe66, Whym, Lvilnis, BattyBot, Yasuo2, Illia Connell, JYBot, Blegat, Yogtad, Tentinator, Marko0991, Ginsuloft, Wccsnow, Ronniemaor, Monkbot, Nboley, Faror91, DilumA, Rider ranger47, Velvel2, Crimsonslide, Megadata tensor, Surbut and Anonymous: 149
- **Clustering high-dimensional data** *Source:* http://en.wikipedia.org/wiki/Clustering%20high-dimensional%20data?oldid=657315593 *Contributors:* Alexrexpvt, Qwertyus, Rjwilmsi, Bgwhite, Wavelength, Melcombe, Iohannes Animosus, Yobot, Slowmo0815, RjwilmsiBot, Chire and Anonymous: 12
- **Canopy clustering algorithm** *Source:* http://en.wikipedia.org/wiki/Canopy%20clustering%20algorithm?oldid=624511369 *Contributors:* Michael Hardy, Giftlite, Rich Farmbrough, Pol098, Wavelength, SmackBot, Sadads, Endpoint, BenJWoodcroft, TheNewPhobia, Eve Teschlemacher, Miniapolis, Melcombe, XLinkBot, Yobot, RobinK, Chire, Helpful Pixie Bot and Anonymous: 11
- **Single-linkage clustering** *Source:* http://en.wikipedia.org/wiki/Single-linkage%20clustering?oldid=613585768 *Contributors:* Michael Hardy, Zeno Gantner, 3mta3, Gary, Rjwilmsi, XLerate, Alaibot, BetacommandBot, Headbomb, Fabrictramp, David Eppstein, Cindamuse, Melcombe, Manuel freire, Wosamy, Pot, Ankit Rakha, AnomieBOT, Lynxoid84, Dan Golding, RjwilmsiBot, Chire, Wcherowi, Cricetus, BattyBot, Roblehall1234 and Anonymous: 7
- **Complete-linkage clustering** *Source:* http://en.wikipedia.org/wiki/Complete-linkage%20clustering?oldid=625941679 *Contributors:* Bearcat, Rjwilmsi, Iae, Cindamuse, Melcombe, Iceblock, Aquila78, AnomieBOT, Vromascanu, RjwilmsiBot, Fæ, AvicAWB, Chire, Marcusogden, Mathstat, Cricetus, Roblehall1234 and Anonymous: 5
- **Nearest-neighbor chain algorithm** *Source:* http://en.wikipedia.org/wiki/Nearest-neighbor%20chain%20algorithm?oldid=595759604 *Contributors:* Edward, Mandarax, Seaphoto, David Eppstein, Ojdo, RjwilmsiBot, Helpful Pixie Bot, Tycho Bray, Matus Telgarsky, Illia Connell and Anonymous: 1
- **UPGMA** *Source:* http://en.wikipedia.org/wiki/UPGMA?oldid=615035513 *Contributors:* AdamRetchless, Edward, Lexor, Cyan, Samsara, Alan filipski, Thorwald, Rajah, Jonsafari, TheParanoidOne, Minority Report, Aranae, YurikBot, Dysmorodrepanis~enwiki, SmackBot, Wzhao553, Paalexan, BroodKiller, TXiKiBoT, Tomfy, Springbok26, Dave noise~enwiki, Fixtgear, Johnuniq, Addbot, Archy33, Lynxoid84, Citation bot, DSisyphBot, RjwilmsiBot, EmausBot, Chire, Djamesb, Rezabot, Cricetus, Upperala, Roblehall1234 and Anonymous: 18
- **BIRCH** *Source:* http://en.wikipedia.org/wiki/BIRCH?oldid=636772083 *Contributors:* Rich Farmbrough, Dmol, Qwertyus, Lockley, Nikkimaria, SmackBot, Odd bloke, SMasters, KimChee, Fabrictramp, JaGa, Katharineamy, Melcombe, AnomieBOT, LilHelpa, Varmanitw, Chire, Captnbunny, Nlskrg, BG19bot, DoctorKubla and Anonymous: 13
- **K-nearest neighbors algorithm** *Source:* http://en.wikipedia.org/wiki/K-nearest%20neighbors%20algorithm?oldid=661965137 *Contributors:* The Anome, B4hand, Michael Hardy, Ronz, Charles Matthews, Topbanana, AnonMoos, Pakaran, Robbot, Altenmann, DHN, Adam McMaster, Pgan002, Dan aka jack, Thorwald, Rama, Slambo, Barro~enwiki, BlueNovember, Caesura, GiovanniS, RHaworth, SQFreak, Btyner, Marudubshinki, BD2412, Qwertyus, Rjwilmsi, Stoph, Debivort, Wavelength, Janto, Garion96, SmackBot, CommodiCast, Mdd4696, Stimpy, Mcld, DHN-bot~enwiki, Hongooi, MisterHand, Joerite, Memming, Gnack, Hu12, Atreys, Ogerard, Kozuch, AnAj, MER-C, Olaf, Jbom1, Peteymills, Dustinsmith, User A1, Mach7, McSly, AntiSpamBot, RJASE1, Joeoettinger, TXiKiBoT, ITurtle, Mpx, SieBot, Prakash Nadkarni, Flyer22, Narasimhanator, AlanUS, Melcombe, Eamon Nerbonne, Svante1, Cibi3d, ClueBot, JP.Martin-Flatin,

Algomaster, Alexbot, Agor153, El bot de la dieta, Rubybrian, Pradtke, XLinkBot, Ploptimist, Addbot, MrOllie, Protonk, Luckas-bot, Yobot, AnomieBOT, Tappoz, Citation bot, Megatang, Miym, PM800, Leonid Volnitsky, FrescoBot, Paine Ellsworth, X7q, Citation bot 1, Rickyphyllis, Emslo69, Lars Washington, Dinamik-bot, Bracchesimo, Delmonde, Geomwiz, Sideways713, DARTH SIDIOUS 2, Thedwards, RjwilmsiBot, Larry.europe, GodfriedToussaint, Nikolaosvasiloglou, EmausBot, Logical Cowboy, Fly by Night, Wijobs, Microfries, Slightsmile, Manyu aditya, Meng6, Chire, Yc319, Mlguy, Vedantkumar, Lovasoa, Dennis97519, Chafe66, Hipponix, Luvegood, ChrisGualtieri, Vbaculum, Jamesx12345, Joeinwiki, Sevamoo, TejDham, Comp.arch, LokeshRavindranathan, Skr15081997, Monkbot, Niraj Aher, Moshe.benyamin, Vermelhomarajó, Sachith500 and Anonymous: 113

- **K-means clustering** *Source:* http://en.wikipedia.org/wiki/K-means%20clustering?oldid=660233806 *Contributors:* Fnielsen, Michael Hardy, Ixfd64, Den fjättrade ankan~enwiki, Charles Matthews, Phil Boswell, Ashwin, Pengo, Giftlite, BenFrantzDale, Duncharris, Soren.harward, WorldsApart, Ratiocinate, Gazpacho, Rich Farmbrough, Mathiasl26, Greenleaf~enwiki, 3mta3, Jonsafari, Andkaha, Ricky81682, Jnothman, Alai, Robert K S, Qwertyus, Rjwilmsi, Hgkamath, Miserlou, Gringer, Mathbot, Mahlon, Chobot, Bgwhite, UkPaolo, YurikBot, Wavelength, SpuriousQ, Annabel, Hakkinen, SamuelRiv, Leishi, Killerandy, SmackBot, Zanetu, Mauls, Mcld, Memming, Cronholm144, Barabum, Denshade, Mauro Bieg, CBM, Chrike, Chrisahn, Talgalili, Thijs!bot, June8th, N5iln, Headbomb, Nick Number, Phoolimin, Sanchom, Charibdis, Smartcat, Magioladitis, David Eppstein, Kzafer, Gfxguy, Turketwh, Stimpak, Mati22081979, Alirn, JohnBlackburne, TXiKiBoT, FedeLebron, ChrisDing, Corvus cornix, Ostrouchov, Yannis1962, Billinghurst, Maxlittle2007, Erniepan, Illuminated, Strife911, Weston.pace, Ntvuok, AlanUS, Melcombe, PerryTachett, MenoBot, DEEJAY JPM, DragonBot, Alexbot, Pot, Tbmurphy, Rcalhoun, Agor153, Qwfp, Niteskate, Avoided, Addbot, DOI bot, Foma84, Fgnievinski, Homncruse, Wfolta, AndresH, Yobot, AnomieBOT, Jim1138, Materialscientist, Citation bot, LilHelpa, Honkkis, Gtfjbl, Gilo1969, Simeon87, Woolleynick, Wonderful597, Dpf90, Foobarhoge, Dan Golding, Phillipe Israel, Jonesey95, Cincoutprabu, Amkilpatrick, NedLevine, Ranumao, Larry.europe, Helwr, EmausBot, John of Reading, Lessbread, Manyu aditya, ZéroBot, Sgoder, Chire, Toninowiki, 0sm0sm0, Helpsome, ClueBot NG, Mathstat, Jack Greenmaven, Railwaycat, BlueScreenD, Jsanchezalmeida, BG19bot, MusikAnimal, Mark Arsten, SciCompTeacher, Chmarkine, Utacsecd, Amritamaz, EdwardH, Sundirac, BattyBot, Illia Connell, MarkPundurs, MindAfterMath, Jamesx12345, MEmreCelebi, Jcallega, Watarok, E8xE8, Quenhitran, Anrnusna, MSheshera, Monkbot, Mazumdarparijat, Joma.huguet, Niraj Aher, Alvisedt, Laiwoonsiu, Eyurtsev and Anonymous: 198
- **K-medians clustering** *Source:* http://en.wikipedia.org/wiki/K-medians%20clustering?oldid=641320815 *Contributors:* Michael Hardy, Bearcat, WorldsApart, Mdd, Talgalili, David Eppstein, Lasselm, Melcombe, Niceguyedc, Deltaway, Addbot, Wfolta, Yobot, LucienBOT, WadiEgg, Chire, TransportObserver, Megalibgwilia and Anonymous: 5
- **K-means++** *Source:* http://en.wikipedia.org/wiki/K-means%2B%2B?oldid=646445061 *Contributors:* Michael Hardy, 3mta3, Rjwilmsi, Jugander, Renaud.richardet, Hakkinen, SmackBot, Melcombe, The Thing That Should Not Be, Sun Creator, Wogone, Sameer0s, AnomieBOT, Citation bot, Thewayiseeit, Shadowjams, Foobarhoge, Citation bot 1, MetaphysicalBrian, Chire, Shaddim, Chafe66, Msto12, Melonkelon and Anonymous: 31
- **K-medoids** *Source:* http://en.wikipedia.org/wiki/K-medoids?oldid=662136504 *Contributors:* Dina, Giftlite, WorldsApart, Andreas Kaufmann, 3mta3, Tommytao, Mandarax, Oliekirk, SmackBot, Mauls, Cronholm144, Jonnat, Anthony Bradbury, Alaibot, Talgalili, Magioladitis, Subail, Justin wen, Shrikantnangare, Turketwh, AlleborgoBot, Vkotor, Tosanjay, Xinunus, Bradpitcher~enwiki, Agor153, XLinkBot, Addbot, Yobot, Surturpain, Kxx, FrescoBot, Dan Golding, Ismailari, EmausBot, Sgoder, AvicAWB, Chire, Donner60, Pokbot, Cheater no1, Gongzhitaao, Chimpoop, Wesamaaa, Eracle~enwiki, Khirodkantnaik and Anonymous: 51
- **K q-flats** *Source:* http://en.wikipedia.org/wiki/K%20q-flats?oldid=610217968 *Contributors:* Andrewman327, Wavelength, JamesBWatson, Niceguyedc, Addbot, Yobot, WikiDan61, AnomieBOT, Chire, Puffin, Ohandyya, Sundirac and Anonymous: 4
- **Voronoi diagram** *Source:* http://en.wikipedia.org/wiki/Voronoi%20diagram?oldid=662636919 *Contributors:* Zundark, Edemaine, Patrick, Michael Hardy, Shyamal, TakuyaMurata, Cyp, Evercat, MichaK, Charles Matthews, Aarontay, Dcoetzee, Jitse Niesen, Smacdonald, Hyacinth, AndrewKepert, Phil Boswell, Drxenocide, Kaol, Altenmann, Gandalf61, Phatsphere, Bkell, Giftlite, Gwalla, Gene Ward Smith, Nadavspi, MSGJ, Edcolins, Joseph Myers, Baba, AmarChandra, Kelson, MichaelMcGuffin, Murtasa, Elwikipedista~enwiki, C S, El oso, Mdd, Gwendal (usurped), River., BRW, Axeman89, Oleg Alexandrov, Firsfron, Linas, Daira Hopwood, Qwertyus, Rjwilmsi, Koavf, Raincomplex, Lordsatri, Salix alba, Gareth McCaughan, Dianelos, Mathbot, Dylan Thurston, Bgwhite, WijzeWillem, Wavelength, Sasuke Sarutobi, BlackShift, KSchutte, Ino5hiro, Zwobot, Mysid, Avraham, Hirak 99, Deville, PTSE, Zerodamage, Th1rt3en, Nekura, SmackBot, RDBury, Ddcampayo, Mcld, Andreas Fabri, Thumperward, Jweimar, Martin.duke, Nbarth, DHN-bot~enwiki, Tamfang, Berland, M0nkey, Adrian bowyer, Firefox13, דניאל צבי, George100, Mikiemike, CRGreathouse, Nczempin, Stebulus, Gfonsecabr, Www.cgal.org, Thijs!bot, Salgueiro~enwiki, JAnDbot, Matthew Fennell, Lopkiol, A3nm, David Eppstein, Chkno, Lvwarren, Glrx, Obscurans, Bhudson, InTheZone~enwiki, Squids and Chips, Pleasantville, Akramm1, Lambyte, Spinningspark, AlleborgoBot, CVT0, GirasoleDE, TomC phys, Sean.hoyland, Krefts, Babakathy, Tkreuz, Yasmar, Laudak, Jwz, Algomaster, PixelBot, Datamajor, RuppertsAlgorithm, Zo1980, SchreiberBike, DumZiBoT, Vog, Addbot, Download, RhinosoRoss, Jarble, Cesaar, Yobot, AnomieBOT, Ciphers, Cleroth, Rui mu, Vanished user 04, Citation bot, ArthurBot, Xqbot, Citation bot 1, Fama Clamosa, Balu.ertl, Avdbits, RjwilmsiBot, Twastvedt, WikitanvirBot, Ben Shamos, Revolution Rock, DanielPerrine, Silicaslip, Baruch60610, Helpful Pixie Bot, HMSSolent, Asmallworld, BG19bot, Smitty121981, Franklin Yu, M.jooyandeh, RedBlackTreeNode, Shiny giratina, Augochy, Jcurrie33, RolandGeraerts, 7nail, Nelg, Zbjornson, CarrieVS, Ven Seyranyan., Lowthy75, ZX95, BackFire, Mark viking, Roman Bazylevych, Zbigniew stachurski, नितीश् चन्द्र, VoroFan and Anonymous: 114
- **DBSCAN** *Source:* http://en.wikipedia.org/wiki/DBSCAN?oldid=655024819 *Contributors:* Fnielsen, Ijon, Wombat~enwiki, Phil Boswell, Angelo.romano, Mathiasl26, 3mta3, Hpxchan, Qwertyus, Rjwilmsi, Gareth Jones, SmackBot, Evert Mouw, Dicklyon, Nurnur, Pgr94, AlexAlex, Nick Number, R'n'B, Lewian, AlanUS, Peterbvolk, WikHead, Addbot, Pyfan, Lightbot, Luckas-bot, Citation bot, Fortdj33, DrilBot, Hbisgin, Lingeek, Hirsutism, Arrakis3k, Rrobby86, ZéroBot, Chire, ClueBot NG, Jumanbar, Helpful Pixie Bot, Justinbozonier, BethNaught, BenLinus1214, Pogomaniac, Saurabhmaurya06 and Anonymous: 84
- **OPTICS algorithm** *Source:* http://en.wikipedia.org/wiki/OPTICS%20algorithm?oldid=639021401 *Contributors:* Bolinstephen, Gareth Jones, MatthewDaniel, Pgr94, Mild Bill Hiccup, Ivionday, 1ForTheMoney, Addbot, Luckas-bot, Yobot, AnomieBOT, Alvin Seville, Citation bot 2, Slowmo0815, Citation bot 1, RjwilmsiBot, John of Reading, Hirsutism, Arrakis3k, Chire, ClueBot NG, Wikikoff and Anonymous: 24
- **Mean shift** *Source:* http://en.wikipedia.org/wiki/Mean%20shift?oldid=658784648 *Contributors:* The Anome, Stevenj, LukeSurl, Qwertyus, Conradl, SmackBot, Dicklyon, Inglada, NE2, Sanchom, Magioladitis, Subail, Dindon~enwiki, Bilal.alsallakh, Maxlittle2007, Iknowyourider, Ichbinder, Qwfp, XLinkBot, DOI bot, Yobot, GrantIngersoll, Citation bot, Czecj Kazinski, Omnipaedista, FrescoBot, RjwilmsiBot, Akats, ClueBot NG, Kkddkkdd, Cheater no1, Karl 334, Barakafrit, ChrisGualtieri, Shaibagon, Colin Sell at the piano, Joeinwiki, Paolor82, Monkbot, Palmoghe, Yanivgwiki and Anonymous: 27

- **Biclustering** *Source:* http://en.wikipedia.org/wiki/Biclustering?oldid=660907620 *Contributors:* Fnielsen, Zeno Gantner, Delirium, Charles Matthews, Everyking, Klemen Kocjancic, Zy26, Jonsafari, Linas, Rjwilmsi, Yhchung, Gwernol, Ms2ger, Took, Bluebot, Radagast83, SMasters, Beefyt, Thijs!bot, Xiaowei JIANG, Alphachimpbot, Magioladitis, David Eppstein, Gwern, R'n'B, Jia.meng, Cobi, Innar, Seemu, Ykluger, Ahsan1010, Muhandes, Addbot, DOI bot, TutterMouse, Luckas-bot, Yobot, Bunnyhop11, Citation bot, Hazyhxj, Citation bot 1, Chenopodiaceous, Trappist the monk, Amkilpatrick, RjwilmsiBot, Akhil 0950, Chire, Toninowiki, EdoBot, Fazlican, Wikikoff, Bct.x42, Delusion23, Helpful Pixie Bot, Smadeira, DoctorKubla, Ashleyleia, ArtfulVampire, Monkbot, Mytsf and Anonymous: 35
- **Clique (graph theory)** *Source:* http://en.wikipedia.org/wiki/Clique%20(graph%20theory)?oldid=655155440 *Contributors:* The Anome, Dominus, Rp, Eric119, Angela, Schneelocke, Dcoetzee, Dysprosia, Robbot, MathMartin, Giftlite, Dbenbenn, Bfinn, Mellum, Tyir, Neilc, Mormegil, 3mta3, Obradovic Goran, Oleg Alexandrov, Shreevatsa, Fredao, GregorB, Joerg Kurt Wegner, Margosbot~enwiki, Salvatore Ingala, RussBot, Samuel Huang, Ott2, SmackBot, Maksim-e~enwiki, Davepape, Zanetu, Chris the speller, Lesnail, Sabik, Thijs!bot, Headbomb, JAnDbot, Deflective, David Eppstein, Ztobor, R'n'B, VolkovBot, Singleheart, Jamelan, Lourakis, Justin W Smith, Ideal gas equation, Dmitrey, Aitias, C. lorenz, Addbot, Peti610botH, Yobot, Amirobot, Rubinbot, Citation bot, Twri, Miym, Bigweeboy, Citation bot 1, At-par, RobinK, GustavLa, Helpful Pixie Bot, Wlxcr, Mark viking, K9re11 and Anonymous: 36
- **Affinity propagation** *Source:* http://en.wikipedia.org/wiki/Affinity%20propagation?oldid=655243792 *Contributors:* Tobias Bergemann, Qwertyus, Rjwilmsi, Yobot, Sgoder, Ecashin, Lovro Ilijasic, RCGuan and Anonymous: 8
- **Basic sequential algorithmic scheme** *Source:* http://en.wikipedia.org/wiki/Basic%20sequential%20algorithmic%20scheme?oldid=626186416 *Contributors:* Michael Hardy, A. Parrot, Funandtrvl, Ferred, Hahc21, Pinuo, AnomieBOT, Jonpatterns, Wgolf, Jamesmcmahon0, AvNiElNi-nA and Anonymous: 2
- **Binarization of consensus partition matrices** *Source:* http://en.wikipedia.org/wiki/Binarization%20of%20consensus%20partition%20matrices?oldid=655185322 *Contributors:* Michael Hardy, Bearcat, Rjwilmsi, Basel1988, Yobot, AnomieBOT, Lightlowemon, BG19bot, Mark viking and Monkbot
- **Cluster-weighted modeling** *Source:* http://en.wikipedia.org/wiki/Cluster-weighted%20modeling?oldid=648332389 *Contributors:* Michael Hardy, Bender235, Rjwilmsi, Hakkinen, SmackBot, Od Mishehu, FlyingToaster, R'n'B, Enigmaman, Melcombe, Chire, Monkbot and Anonymous: 3
- **Cobweb (clustering)** *Source:* http://en.wikipedia.org/wiki/Cobweb%20(clustering)?oldid=636823068 *Contributors:* Rjwilmsi, Bask, Od Mishehu, Object01, Bluebot, Melcombe, AnomieBOT, FrescoBot, Citation bot 1, Trappist the monk, Ismailari, RjwilmsiBot, Cmaclell, BG19bot and Anonymous: 5
- **CURE data clustering algorithm** *Source:* http://en.wikipedia.org/wiki/CURE%20data%20clustering%20algorithm?oldid=617190604 *Contributors:* Malcolma, SmackBot, Sadads, KimChee, Fabrictramp, Katharineamy, Joseph A. Spadaro, Melcombe, Mild Bill Hiccup, Qwfp, Yobot, Erik9bot, Varmanitw, PigFlu Oink, John of Reading, Chire, Nehchal, Helpful Pixie Bot, Nightshift Bagel cart, APerson, Monkbot and Anonymous: 7
- **FLAME clustering** *Source:* http://en.wikipedia.org/wiki/FLAME%20clustering?oldid=537298419 *Contributors:* Michael Hardy, Giftlite, Terrycojones, SmackBot, Phoolimin, Qwfp, Lynxoid84, Chire and Anonymous: 6
- **Information bottleneck method** *Source:* http://en.wikipedia.org/wiki/Information%20bottleneck%20method?oldid=594578601 *Contributors:* Edward, Michael Hardy, Willem, Hike395, Itai, Merovingian, VladShchogolev, TonyW, Rich Farmbrough, Oleg Alexandrov, Bgwhite, SmackBot, Commander Keane bot, Mcld, Tmg1165, Sohale, Tishby, Melcombe, Therustyone, Yobot, Felix Creutzig, WhatWasDone, Chire, DoctorKubla and Anonymous: 9
- **K-SVD** *Source:* http://en.wikipedia.org/wiki/K-SVD?oldid=662069237 *Contributors:* Michael Hardy, Andrewman327, Qwertyus, Mcld, Sheridp, Dthomsen8, Ironholds, Yobot, AnomieBOT, Elsdrm, Iohannez, Erickraz, Hueihan Jhuang, FooCow, Pllull1, Caesar cai, Tadeocorradi and Anonymous: 6
- **Linde–Buzo–Gray algorithm** *Source:* http://en.wikipedia.org/wiki/Linde%E2%80%93Buzo%E2%80%93Gray%20algorithm?oldid=656355601 *Contributors:* Michael Hardy, CesarB, Glenn, Uzume, GregorB, SudoMonas, Enjoy~enwiki, SmackBot, Bluebot, Raharu~enwiki, Memming, Endpoint, Blaisorblade, Alaibot, David Eppstein, Melcombe, Unbuttered Parsnip, Xiawi, RuppertsAlgorithm, Yobot, AnomieBOT, JensKohl, Omnipaedista, RobinK, Chire and Anonymous: 8
- **Neighbor joining** *Source:* http://en.wikipedia.org/wiki/Neighbor%20joining?oldid=617224399 *Contributors:* Josh Grosse, Edward, Michael Hardy, Cyan, Charles Matthews, Glimz~enwiki, Samsara, Alan filipski, Thorwald, Rajah, Aranae, Gringer, Rell Canis, Debivort, Wavelength, Zwobot, Wzhao553, Paalexan, Matanninio, Dicklyon, Glor, Gioto, Aarem, Magioladitis, Ling.Nut, David Eppstein, Lafw, VolkovBot, TXiKiBoT, Tamorlan, Jochgem, Tomfy, Springbok26, Der Golem, Dthomsen8, Addbot, Yobot, Themfromspace, AnomieBOT, Citation bot, Rickproser, Lothar von Richthofen, Pairwise, Traviswheeler, RjwilmsiBot, Somme89, Smirarab~enwiki, S.MahdiRazavi, Helpful Pixie Bot, Ephert, BG19bot, MatthewIreland, ChrisGualtieri, Faizan, Upperala and Anonymous: 35
- **Pitman–Yor process** *Source:* http://en.wikipedia.org/wiki/Pitman%E2%80%93Yor%20process?oldid=600613642 *Contributors:* Michael Hardy, 3mta3, Schmock, SmackBot, Took, Rdds, Xtaty~enwiki, Headbomb, Magioladitis, Maghnus, Yobot, Jpillow, Trappist the monk, Drbayes and Anonymous: 6
- **Self-organizing map** *Source:* http://en.wikipedia.org/wiki/Self-organizing%20map?oldid=662182366 *Contributors:* Ap, Mrwojo, Michael Hardy, Shyamal, Delirium, Pieter Suurmond, Ronz, Glenn, AugPi, Rotem Dan, Hike395, Guaka, Psychonaut, Rholton, Ojigiri~enwiki, Ancheta Wis, Chinasaur, Alensha, Mboverload, Daniel Brockman, Pgan002, Gene s, Urhixidur, JimQ, Thorwald, Rich Farmbrough, Iainscott, ZeroOne, Alex Kosorukoff, ThruTheLukinGlas, Onay, Janna Isabot, .:Ajvol:., Denoir, Freshraisin, Oleg Alexandrov, Ruud Koot, Male1979, Kbdank71, Rjwilmsi, Miserlou, Itkovian, Hansamurai, Chobot, Wavelength, SpuriousQ, Sanguinity, Aktech, Hakeem.gadi, Ms2ger, Jgzheng, Dq1, Mebden, KnightRider~enwiki, SmackBot, KocjoBot~enwiki, CommodiCast, Mcld, Bluebot, Mgeorg~enwiki, Conway71, Miquonranger03, Jasonb05, Goodale, JonHarder, Mitar, Dangraupe, WMod-NS, CRGreathouse, CmdrObot, Zarex, Yarnalgo, Lachambre, Bediako, Galeth, Pihka, JamesBrownJr, Thijs!bot, Jojan, Rasikaa, Lotif, DorisH, Liquid-aim-bot, AnAj, Geo.per, Ninjakannon, Magioladitis, Vernanimalcula, Jqshenker, Schumi555, Andre.holzner, Jorgenumata, Marllenc, Timohonkela, Arkadi kagan, Chilti, Anoko moonlight, SieBot, Kylemew, Melcombe, A udachny, ClueBot, Rakeshchalasani, Francisco Albani, Agor153, ElectricTypist, Kwantum, FORTRANslinger, Addbot, MrVanBot, Tide rolls, Middayexpress, Tedtoal, Depuarg, Yobot, Mmmcalzones, Citation bot, ArthurBot, Obersachsebot, Xqbot, Ocean518, Breezest, Aultsch, Citation bot 1, Tinton5, Wondigoma, Ismailari, RjwilmsiBot, Ripchip Bot, Helwr, EmausBot, RaoInWiki, Daryakav, MrHedless85, Techteachermayank, Chire, AManWithNoPlan, Hfang80, Shinosin, Imprecisekludge, Kleinash, Zackron, Mfemi, Meatsgains, Volkoo, Jsalatas, ÄDA - DÄP, Mansoorexpert, Sevamoo, Hfang bristol, R saadat, Monkbot, Taozaa007, Oyinda8 and Anonymous: 150

- **SUBCLU** *Source:* http://en.wikipedia.org/wiki/SUBCLU?oldid=532412734 *Contributors:* Malcolma, Gelingvistoj, Mild Bill Hiccup, Erik9bot, Slowmo0815, Chire, BG19bot and Anonymous: 3
- **Ward's method** *Source:* http://en.wikipedia.org/wiki/Ward'{}s%20method?oldid=661724473 *Contributors:* Michael Hardy, Andrewman327, Mdd, Bgwhite, Avraham, Melcombe, Niceguyedc, Qwfp, Addbot, Yobot, Ptbotgourou, Chire, Mathstat, BG19bot, Megalibgwilia, MKKowalczyk and Anonymous: 11

7.2 Images

- **File:2C_3_1979.JPG** *Source:* http://upload.wikimedia.org/wikipedia/commons/4/4c/2C_3_1979.JPG *License:* CC BY-SA 3.0 *Contributors:* Gerard Caris *Original artist:* Gerard Caris
- **File:Ambox_important.svg** *Source:* http://upload.wikimedia.org/wikipedia/commons/b/b4/Ambox_important.svg *License:* Public domain *Contributors:* Own work, based off of Image:Ambox scales.svg *Original artist:* Dsmurat (talk · contribs)
- **File:Ambox_wikify.svg** *Source:* http://upload.wikimedia.org/wikipedia/commons/e/e1/Ambox_wikify.svg *License:* Public domain *Contributors:* Own work *Original artist:* penubag
- **File:Approximate_Voronoi_Diagram.svg** *Source:* http://upload.wikimedia.org/wikipedia/commons/3/37/Approximate_Voronoi_Diagram.svg *License:* CC BY-SA 4.0 *Contributors:* Own work *Original artist:* Balu Ertl
- **File:Bellcurve.svg** *Source:* http://upload.wikimedia.org/wikipedia/commons/d/df/Bellcurve.svg *License:* Copyrighted free use *Contributors:* ? *Original artist:* ?
- **File:Binary_tree.svg** *Source:* http://upload.wikimedia.org/wikipedia/commons/f/f7/Binary_tree.svg *License:* Public domain *Contributors:* Own work *Original artist:* Derrick Coetzee
- **File:Blue_morpho_butterfly.jpg** *Source:* http://upload.wikimedia.org/wikipedia/commons/6/65/Blue_morpho_butterfly.jpg *License:* CC-BY-SA-3.0 *Contributors:* Own work *Original artist:* Gregory Phillips
- **File:BorderRAtio.PNG** *Source:* http://upload.wikimedia.org/wikipedia/commons/e/e6/BorderRAtio.PNG *License:* CC BY 3.0 *Contributors:* The illustration from the software description E. M. Mirkes, KNN and Potential Energy: applet. University of Leicester, 2011. Published under Attribution 3.0 Unported (CC BY 3.0) licence. *Original artist:* E. M. Mirkes
- **File:BottleCateg_1.jpg** *Source:* http://upload.wikimedia.org/wikipedia/en/1/18/BottleCateg_1.jpg *License:* Cc-by-sa-3.0 *Contributors:*
  Own work
  Original artist:
  Therustyone (talk) (Uploads)
- **File:Cluster-2.svg** *Source:* http://upload.wikimedia.org/wikipedia/commons/c/c8/Cluster-2.svg *License:* Public domain *Contributors:*
- Cluster-2.gif *Original artist:* Cluster-2.gif: hellisp
- **File:ClusterAnalysis_Mouse.svg** *Source:* http://upload.wikimedia.org/wikipedia/commons/0/09/ClusterAnalysis_Mouse.svg *License:* Public domain *Contributors:* Own work *Original artist:* Chire
- **File:Clusters.svg** *Source:* http://upload.wikimedia.org/wikipedia/commons/b/b5/Clusters.svg *License:* Public domain *Contributors:*
- Clusters.PNG *Original artist:* Clusters.PNG:
- **File:Coloured_Voronoi_3D_slice.svg** *Source:* http://upload.wikimedia.org/wikipedia/commons/8/84/Coloured_Voronoi_3D_slice.svg *License:* CC-BY-SA-3.0 *Contributors:* ? *Original artist:* ?
- **File:Commons-logo.svg** *Source:* http://upload.wikimedia.org/wikipedia/en/4/4a/Commons-logo.svg *License:* ? *Contributors:* ? *Original artist:* ?
- **File:Concept_tree.png** *Source:* http://upload.wikimedia.org/wikipedia/en/9/94/Concept_tree.png *License:* PD *Contributors:*
  Own work
  Original artist:
  Dfass (talk) (Uploads)
- **File:Constructing_phylogenetic_tree_using_neighbor-joining_5_taxa_improved.svg** *Source:* http://upload.wikimedia.org/wikipedia/en/2/26/Constructing_phylogenetic_tree_using_neighbor-joining_5_taxa_improved.svg *License:* CC-BY-SA-3.0 *Contributors:*
  Google drive drawing.
  Previously published: -
  Original artist:
  Tomfy
- **File:Copyright-problem.svg** *Source:* http://upload.wikimedia.org/wikipedia/en/c/cf/Copyright-problem.svg *License:* PD *Contributors:* ? *Original artist:* ?
- **File:DBSCAN-Illustration.svg** *Source:* http://upload.wikimedia.org/wikipedia/commons/a/af/DBSCAN-Illustration.svg *License:* CC BY-SA 3.0 *Contributors:* Own work *Original artist:* Chire
- **File:DBSCAN-density-data.svg** *Source:* http://upload.wikimedia.org/wikipedia/commons/0/05/DBSCAN-density-data.svg *License:* CC BY-SA 3.0 *Contributors:* Own work *Original artist:* Chire
- **File:DataClustering_ElbowCriterion.JPG** *Source:* http://upload.wikimedia.org/wikipedia/commons/c/cd/DataClustering_ElbowCriterion.JPG *License:* CC-BY-SA-3.0 *Contributors:* ? *Original artist:* ?
- **File:EM_Clustering_of_Old_Faithful_data.gif** *Source:* http://upload.wikimedia.org/wikipedia/commons/6/69/EM_Clustering_of_Old_Faithful_data.gif *License:* CC BY-SA 3.0 *Contributors:* Own work *Original artist:* Chire

- **File:Edit-clear.svg** *Source:* http://upload.wikimedia.org/wikipedia/en/f/f2/Edit-clear.svg *License:* Public domain *Contributors:* The Tango! Desktop Project. *Original artist:*

  The people from the Tango! project. And according to the meta-data in the file, specifically: "Andreas Nilsson, and Jakub Steiner (although minimally)."
- **File:Em_old_faithful.gif** *Source:* http://upload.wikimedia.org/wikipedia/commons/a/a7/Em_old_faithful.gif *License:* CC BY-SA 3.0 *Contributors:* Own work *Original artist:* 3mta3 (<a href='//commons.wikimedia.org/wiki/User_talk:3mta3' title='User talk: 3mta3'>talk</a>) 16:55, 23 March 2009 (UTC)
- **File:Euclidean_Voronoi_diagram.svg** *Source:* http://upload.wikimedia.org/wikipedia/commons/5/54/Euclidean_Voronoi_diagram.svg *License:* CC BY-SA 4.0 *Contributors:* Own work *Original artist:* Balu Ertl
- **File:FLAME_Demo.png** *Source:* http://upload.wikimedia.org/wikipedia/en/2/26/FLAME_Demo.png *License:* GFDL *Contributors:* ? *Original artist:* ?
- **File:Fisher_iris_versicolor_sepalwidth.svg** *Source:* http://upload.wikimedia.org/wikipedia/commons/4/40/Fisher_iris_versicolor_sepalwidth.svg *License:* CC BY-SA 3.0 *Contributors:* en:Image:Fisher iris versicolor sepalwidth.png *Original artist:* en:User:Qwfp (original); Pbroks13 (talk) (redraw)
- **File:Folder_Hexagonal_Icon.svg** *Source:* http://upload.wikimedia.org/wikipedia/en/4/48/Folder_Hexagonal_Icon.svg *License:* Cc-by-sa-3.0 *Contributors:* ? *Original artist:* ?
- **File:Hierarchical_clustering_simple_diagram.svg** *Source:* http://upload.wikimedia.org/wikipedia/commons/a/ad/Hierarchical_clustering_simple_diagram.svg *License:* CC-BY-SA-3.0 *Contributors:*

  Original artist:
- derivative work: Mhbrugman (<a href='//commons.wikimedia.org/wiki/User_talk:Mhbrugman' title='User talk:Mhbrugman'>talk</a>)
- **File:IdentifiabilityZones.pdf** *Source:* http://upload.wikimedia.org/wikipedia/commons/7/77/IdentifiabilityZones.pdf *License:* CC BY-SA 3.0 *Contributors:* Own work *Original artist:* Shenbaba
- **File:Integrated_circuit_icon.svg** *Source:* http://upload.wikimedia.org/wikipedia/commons/8/8c/Integrated_circuit_icon.svg *License:* LGPL *Contributors:* All Crystal icons were posted by the author as LGPL on kde-look *Original artist:* Everaldo Coelho (YellowIcon);
- **File:Internet_map_1024.jpg** *Source:* http://upload.wikimedia.org/wikipedia/commons/d/d2/Internet_map_1024.jpg *License:* CC BY 2.5 *Contributors:* Originally from the English Wikipedia; description page is/was here. *Original artist:* The Opte Project
- **File:Iris_Flowers_Clustering_kMeans.svg** *Source:* http://upload.wikimedia.org/wikipedia/commons/1/10/Iris_Flowers_Clustering_kMeans.svg *License:* Public domain *Contributors:* Own work *Original artist:* Chire
- **File:K-means_convergence_to_a_local_minimum.png** *Source:* http://upload.wikimedia.org/wikipedia/commons/7/7c/K-means_convergence_to_a_local_minimum.png *License:* CC BY-SA 3.0 *Contributors:* Own work *Original artist:* Agor153
- **File:K-means_versus_k-medoids.png** *Source:* http://upload.wikimedia.org/wikipedia/commons/b/b3/K-means_versus_k-medoids.png *License:* CC BY-SA 3.0 *Contributors:* Own work *Original artist:* Agor153
- **File:Kmedoid1.jpg** *Source:* http://upload.wikimedia.org/wikipedia/commons/e/e1/Kmedoid1.jpg *License:* Public domain *Contributors:* en:image:Kmedoid1.jpg *Original artist:* en:User:Shrikantnangare
- **File:Kmedoid2.jpg** *Source:* http://upload.wikimedia.org/wikipedia/commons/8/82/Kmedoid2.jpg *License:* Public domain *Contributors:* en:image:Kmedoid2.jpg *Original artist:* en:User:Shrikantnangare
- **File:Kmedoid3.jpg** *Source:* http://upload.wikimedia.org/wikipedia/commons/a/a4/Kmedoid3.jpg *License:* Public domain *Contributors:* en:image:Kmedoid3.jpg *Original artist:* en:User:Shrikantnangare
- **File:KnnClassification.svg** *Source:* http://upload.wikimedia.org/wikipedia/commons/e/e7/KnnClassification.svg *License:* CC-BY-SA-3.0 *Contributors:* Own work *Original artist:* Antti Ajanki AnAj
- **File:LampFlowchart.svg** *Source:* http://upload.wikimedia.org/wikipedia/commons/9/91/LampFlowchart.svg *License:* CC-BY-SA-3.0 *Contributors:* vector version of Image:LampFlowchart.png *Original artist:* svg by Booyabazooka
- **File:Manhattan_Voronoi_Diagram.svg** *Source:* http://upload.wikimedia.org/wikipedia/commons/6/6d/Manhattan_Voronoi_Diagram.svg *License:* CC BY-SA 1.0 *Contributors:* Own work *Original artist:* Balu Ertl
- **File:Marble_floor_mosaic_Basilica_of_St_Mark_Vencice.jpg** *Source:* http://upload.wikimedia.org/wikipedia/commons/9/95/Marble_floor_mosaic_Basilica_of_St_Mark_Vencice.jpg *License:* Public domain *Contributors:* http://www.georgehart.com/virtual-polyhedra/uccello.html *Original artist:* Paolo Uccello
- **File:Nearest-neighbor_chain_algorithm_animated.gif** *Source:* http://upload.wikimedia.org/wikipedia/commons/d/db/Nearest-neighbor_chain_algorithm_animated.gif *License:* Public domain *Contributors:* Own work *Original artist:* Matus Telgarsky
- **File:Neighbor-joining_Tree-2.png** *Source:* http://upload.wikimedia.org/wikipedia/commons/7/77/Neighbor-joining_Tree-2.png *License:* Public domain *Contributors:* Neighbor-joining Tree.svg *Original artist:* Ephert
- **File:Neighbor_joining_7_taxa_start_to_finish_diagram.svg** *Source:* http://upload.wikimedia.org/wikipedia/en/c/cf/Neighbor_joining_7_taxa_start_to_finish_diagram.svg *License:* CC-BY-SA-3.0 *Contributors:*

  Google doc drawing. Edited drawing also created by me in early 2012.

  Original artist:

  Tomfy
- **File:OPTICS.svg** *Source:* http://upload.wikimedia.org/wikipedia/commons/f/f9/OPTICS.svg *License:* Public domain *Contributors:* Own work *Original artist:* Chire
- **File:PACexplained.png** *Source:* http://upload.wikimedia.org/wikipedia/commons/1/14/PACexplained.png *License:* CC BY-SA 3.0 *Contributors:* Own work *Original artist:* Shenbaba

7.3 Content license