Top 10 algorithms in data mining

Article *in* Knowledge and Information Systems · December 2007

DOI: 10.1007/s10115-007-0114-2 · Source: OAI

14 authors, including:

Xindong Wu
UVM
339 PUBLICATIONS **24,700** CITATIONS
SEE PROFILE

Vipin Kumar
University of Minnesota Twin Cities
732 PUBLICATIONS **81,850** CITATIONS
SEE PROFILE

Ross Quinlan
rulequest research
97 PUBLICATIONS **72,778** CITATIONS
SEE PROFILE

Joydeep Ghosh
University of Texas at Austin
386 PUBLICATIONS **26,469** CITATIONS
SEE PROFILE

Knowl Inf Syst (2008) 14:1–37
DOI 10.1007/s10115-007-0114-2

SURVEY PAPER

Top 10 algorithms in data mining

Xindong Wu · Vipin Kumar · J. Ross Quinlan · Joydeep Ghosh · Qiang Yang · Hiroshi Motoda · Geoffrey J. McLachlan · Angus Ng · Bing Liu · Philip S. Yu · Zhi-Hua Zhou · Michael Steinbach · David J. Hand · Dan Steinberg

Received: 9 July 2007 / Revised: 28 September 2007 / Accepted: 8 October 2007
Published online: 4 December 2007

Abstract This paper presents the top 10 data mining algorithms identified by the IEEE International Conference on Data Mining (ICDM) in December 2006: C4.5, *k*-Means, SVM, Apriori, EM, PageRank, AdaBoost, *k*NN, Naive Bayes, and CART. These top 10 algorithms are among the most influential data mining algorithms in the research community. With each algorithm, we provide a description of the algorithm, discuss the impact of the algorithm, and review current and further research on the algorithm. These 10 algorithms cover classification,

X. Wu (✉)
Department of Computer Science, University of Vermont, Burlington, VT, USA
e-mail: xwu@cs.uvm.edu

V. Kumar
Department of Computer Science and Engineering,
University of Minnesota, Minneapolis, MN, USA
e-mail: kumar@cs.umn.edu

J. Ross Quinlan
Rulequest Research Pty Ltd,
St Ives, NSW, Australia
e-mail: quinlan@rulequest.com

J. Ghosh
Department of Electrical and Computer Engineering,
University of Texas at Austin, Austin, TX 78712, USA
e-mail: ghosh@ece.utexas.edu

Q. Yang
Department of Computer Science,
Hong Kong University of Science and Technology,
Honkong, China
e-mail: qyang@cs.ust.hk

H. Motoda
AFOSR/AOARD and Osaka University,
7-23-17 Roppongi, Minato-ku, Tokyo 106-0032, Japan
e-mail: motoda@ar.sanken.osaka-u.ac.jp

clustering, statistical learning, association analysis, and link mining, which are all among the most important topics in data mining research and development.

0 Introduction

In an effort to identify some of the most influential algorithms that have been widely used in the data mining community, the IEEE International Conference on Data Mining (ICDM, http://www.cs.uvm.edu/~icdm/) identified the top 10 algorithms in data mining for presentation at ICDM '06 in Hong Kong.

As the first step in the identification process, in September 2006 we invited the ACM KDD Innovation Award and IEEE ICDM Research Contributions Award winners to each nominate up to 10 best-known algorithms in data mining. All except one in this distinguished set of award winners responded to our invitation. We asked each nomination to provide the following information: (a) the algorithm name, (b) a brief justification, and (c) a representative publication reference. We also advised that each nominated algorithm should have been widely cited and used by other researchers in the field, and the nominations from each nominator as a group should have a reasonable representation of the different areas in data mining.

G. J. McLachlan
Department of Mathematics,
The University of Queensland, Brisbane, Australia
e-mail: gjm@maths.uq.edu.au

A. Ng
School of Medicine, Griffith University,
Brisbane, Australia

B. Liu
Department of Computer Science,
University of Illinois at Chicago, Chicago, IL 60607, USA

P. S. Yu
IBM T. J. Watson Research Center,
Hawthorne, NY 10532, USA
e-mail: psyu@us.ibm.com

Z.-H. Zhou
National Key Laboratory for Novel Software Technology,
Nanjing University, Nanjing 210093, China
e-mail: zhouzh@nju.edu.cn

M. Steinbach
Department of Computer Science and Engineering,
University of Minnesota, Minneapolis, MN 55455, USA
e-mail: steinbac@cs.umn.edu

D. J. Hand
Department of Mathematics,
Imperial College, London, UK
e-mail: d.j.hand@imperial.ac.uk

D. Steinberg
Salford Systems,
San Diego, CA 92123, USA
e-mail: dsx@salford-systems.com

After the nominations in Step 1, we verified each nomination for its citations on Google Scholar in late October 2006, and removed those nominations that did not have at least 50 citations. All remaining (18) nominations were then organized in 10 topics: association analysis, classification, clustering, statistical learning, bagging and boosting, sequential patterns, integrated mining, rough sets, link mining, and graph mining. For some of these 18 algorithms such as *k*-means, the representative publication was not necessarily the original paper that introduced the algorithm, but a recent paper that highlights the importance of the technique. These representative publications are available at the ICDM website (http://www.cs.uvm.edu/~icdm/algorithms/CandidateList.shtml).

In the third step of the identification process, we had a wider involvement of the research community. We invited the Program Committee members of KDD-06 (the 2006 ACM SIGKDD International Conference on Knowledge Discovery and Data Mining), ICDM '06 (the 2006 IEEE International Conference on Data Mining), and SDM '06 (the 2006 SIAM International Conference on Data Mining), as well as the ACM KDD Innovation Award and IEEE ICDM Research Contributions Award winners to each vote for up to 10 well-known algorithms from the 18-algorithm candidate list. The voting results of this step were presented at the ICDM '06 panel on Top 10 Algorithms in Data Mining.

At the ICDM '06 panel of December 21, 2006, we also took an open vote with all 145 attendees on the top 10 algorithms from the above 18-algorithm candidate list, and the top 10 algorithms from this open vote were the same as the voting results from the above third step. The 3-hour panel was organized as the last session of the ICDM '06 conference, in parallel with 7 paper presentation sessions of the Web Intelligence (WI '06) and Intelligent Agent Technology (IAT '06) conferences at the same location, and attracting 145 participants to this panel clearly showed that the panel was a great success.

1 C4.5 and beyond

1.1 Introduction

Systems that construct classifiers are one of the commonly used tools in data mining. Such systems take as input a collection of cases, each belonging to one of a small number of classes and described by its values for a fixed set of attributes, and output a classifier that can accurately predict the class to which a new case belongs.

These notes describe C4.5 [64], a descendant of CLS [41] and ID3 [62]. Like CLS and ID3, C4.5 generates classifiers expressed as decision trees, but it can also construct classifiers in more comprehensible ruleset form. We will outline the algorithms employed in C4.5, highlight some changes in its successor See5/C5.0, and conclude with a couple of open research issues.

1.2 Decision trees

Given a set S of cases, C4.5 first grows an initial tree using the divide-and-conquer algorithm as follows:

- If all the cases in S belong to the same class or S is small, the tree is a leaf labeled with the most frequent class in S.
- Otherwise, choose a test based on a single attribute with two or more outcomes. Make this test the root of the tree with one branch for each outcome of the test, partition S into corresponding subsets $S_1, S_2, \ldots$ according to the outcome for each case, and apply the same procedure recursively to each subset.

There are usually many tests that could be chosen in this last step. C4.5 uses two heuristic criteria to rank possible tests: information gain, which minimizes the total entropy of the subsets $\{S_i\}$ (but is heavily biased towards tests with numerous outcomes), and the default gain ratio that divides information gain by the information provided by the test outcomes.

Attributes can be either numeric or nominal and this determines the format of the test outcomes. For a numeric attribute A they are $\{A \leq h, A > h\}$ where the threshold h is found by sorting S on the values of A and choosing the split between successive values that maximizes the criterion above. An attribute A with discrete values has by default one outcome for each value, but an option allows the values to be grouped into two or more subsets with one outcome for each subset.

The initial tree is then pruned to avoid overfitting. The pruning algorithm is based on a pessimistic estimate of the error rate associated with a set of N cases, E of which do not belong to the most frequent class. Instead of E/N, C4.5 determines the upper limit of the binomial probability when E events have been observed in N trials, using a user-specified confidence whose default value is 0.25.

Pruning is carried out from the leaves to the root. The estimated error at a leaf with N cases and E errors is N times the pessimistic error rate as above. For a subtree, C4.5 adds the estimated errors of the branches and compares this to the estimated error if the subtree is replaced by a leaf; if the latter is no higher than the former, the subtree is pruned. Similarly, C4.5 checks the estimated error if the subtree is replaced by one of its branches and when this appears beneficial the tree is modified accordingly. The pruning process is completed in one pass through the tree.

C4.5's tree-construction algorithm differs in several respects from CART [9], for instance:

- Tests in CART are always binary, but C4.5 allows two or more outcomes.
- CART uses the Gini diversity index to rank tests, whereas C4.5 uses information-based criteria.
- CART prunes trees using a cost-complexity model whose parameters are estimated by cross-validation; C4.5 uses a single-pass algorithm derived from binomial confidence limits.
- This brief discussion has not mentioned what happens when some of a case's values are unknown. CART looks for surrogate tests that approximate the outcomes when the tested attribute has an unknown value, but C4.5 apportions the case probabilistically among the outcomes.

1.3 Ruleset classifiers

Complex decision trees can be difficult to understand, for instance because information about one class is usually distributed throughout the tree. C4.5 introduced an alternative formalism consisting of a list of rules of the form "if A and B and C and ... then class X", where rules for each class are grouped together. A case is classified by finding the first rule whose conditions are satisfied by the case; if no rule is satisfied, the case is assigned to a default class.

C4.5 rulesets are formed from the initial (unpruned) decision tree. Each path from the root of the tree to a leaf becomes a prototype rule whose conditions are the outcomes along the path and whose class is the label of the leaf. This rule is then simplified by determining the effect of discarding each condition in turn. Dropping a condition may increase the number N of cases covered by the rule, and also the number E of cases that do not belong to the class nominated by the rule, and may lower the pessimistic error rate determined as above. A hill-climbing algorithm is used to drop conditions until the lowest pessimistic error rate is found.

To complete the process, a subset of simplified rules is selected for each class in turn. These class subsets are ordered to minimize the error on the training cases and a default class is chosen. The final ruleset usually has far fewer rules than the number of leaves on the pruned decision tree.

The principal disadvantage of C4.5's rulesets is the amount of CPU time and memory that they require. In one experiment, samples ranging from 10,000 to 100,000 cases were drawn from a large dataset. For decision trees, moving from 10 to 100K cases increased CPU time on a PC from 1.4 to 61 s, a factor of 44. The time required for rulesets, however, increased from 32 to 9,715 s, a factor of 300.

1.4 See5/C5.0

C4.5 was superseded in 1997 by a commercial system See5/C5.0 (or C5.0 for short). The changes encompass new capabilities as well as much-improved efficiency, and include:

- A variant of boosting [24], which constructs an ensemble of classifiers that are then voted to give a final classification. Boosting often leads to a dramatic improvement in predictive accuracy.
- New data types (e.g., dates), "not applicable" values, variable misclassification costs, and mechanisms to pre-filter attributes.
- Unordered rulesets—when a case is classified, all applicable rules are found and voted. This improves both the interpretability of rulesets and their predictive accuracy.
- Greatly improved scalability of both decision trees and (particularly) rulesets. Scalability is enhanced by multi-threading; C5.0 can take advantage of computers with multiple CPUs and/or cores.

More details are available from http://rulequest.com/see5-comparison.html.

1.5 Research issues

We have frequently heard colleagues express the view that decision trees are a "solved problem." We do not agree with this proposition and will close with a couple of open research problems.

Stable trees. It is well known that the error rate of a tree on the cases from which it was constructed (the resubstitution error rate) is much lower than the error rate on unseen cases (the predictive error rate). For example, on a well-known letter recognition dataset with 20,000 cases, the resubstitution error rate for C4.5 is 4%, but the error rate from a leave-one-out (20,000-fold) cross-validation is 11.7%. As this demonstrates, leaving out a single case from 20,000 often affects the tree that is constructed!

Suppose now that we could develop a non-trivial tree-construction algorithm that was hardly ever affected by omitting a single case. For such stable trees, the resubstitution error rate should approximate the leave-one-out cross-validated error rate, suggesting that the tree is of the "right" size.

Decomposing complex trees. Ensemble classifiers, whether generated by boosting, bagging, weight randomization, or other techniques, usually offer improved predictive accuracy. Now, given a small number of decision trees, it is possible to generate a single (very complex) tree that is exactly equivalent to voting the original trees, but can we go the other way? That is, can a complex tree be broken down to a small collection of simple trees that, when voted together, give the same result as the complex tree? Such decomposition would be of great help in producing comprehensible decision trees.

C4.5 Acknowledgments

Research on C4.5 was funded for many years by the Australian Research Council.

C4.5 is freely available for research and teaching, and source can be downloaded from http://rulequest.com/Personal/c4.5r8.tar.gz.

2 The *k*-means algorithm

2.1 The algorithm

The `k-means` algorithm is a simple iterative method to partition a given dataset into a user-specified number of clusters, k. This algorithm has been discovered by several researchers across different disciplines, most notably Lloyd (1957, 1982) [53], Forgey (1965), Friedman and Rubin (1967), and McQueen (1967). A detailed history of `k-means` alongwith descriptions of several variations are given in [43]. Gray and Neuhoff [34] provide a nice historical background for `k-means` placed in the larger context of hill-climbing algorithms.

The algorithm operates on a set of d-dimensional vectors, $D = \{\mathbf{x_i} \mid i = 1, \ldots, N\}$, where $\mathbf{x_i} \in \Re^d$ denotes the ith data point. The algorithm is initialized by picking k points in $\Re^d$ as the initial k cluster representatives or "centroids". Techniques for selecting these initial seeds include sampling at random from the dataset, setting them as the solution of clustering a small subset of the data or perturbing the global mean of the data k times. Then the algorithm iterates between two steps till convergence:

Step 1: Data Assignment. Each data point is assigned to its *closest* centroid, with ties broken arbitrarily. This results in a partitioning of the data.

Step 2: Relocation of "means". Each cluster representative is relocated to the center (mean) of all data points assigned to it. If the data points come with a probability measure (weights), then the relocation is to the expectations (weighted mean) of the data partitions.

The algorithm converges when the assignments (and hence the $\mathbf{c_j}$ values) no longer change. The algorithm execution is visually depicted in Fig. 1. Note that each iteration needs $N \times k$ comparisons, which determines the time complexity of one iteration. The number of iterations required for convergence varies and may depend on N, but as a first cut, this algorithm can be considered linear in the dataset size.

One issue to resolve is how to quantify "closest" in the assignment step. The default measure of closeness is the Euclidean distance, in which case one can readily show that the non-negative cost function,

$$\sum_{i=1}^{N} \left(\operatorname*{argmin}_{j} ||\mathbf{x_i} - \mathbf{c_j}||_2^2 \right) \tag{1}$$

will decrease whenever there is a change in the assignment or the relocation steps, and hence convergence is guaranteed in a finite number of iterations. The greedy-descent nature of `k-means` on a non-convex cost also implies that the convergence is only to a local optimum, and indeed the algorithm is typically quite sensitive to the initial centroid locations. Figure 2 [1] illustrates how a poorer result is obtained for the same dataset as in Fig. 1 for a different choice of the three initial centroids. The local minima problem can be countered to some

[1] Figures 1 and 2 are taken from the slides for the book, *Introduction to Data Mining*, Tan, Kumar, Steinbach, 2006.

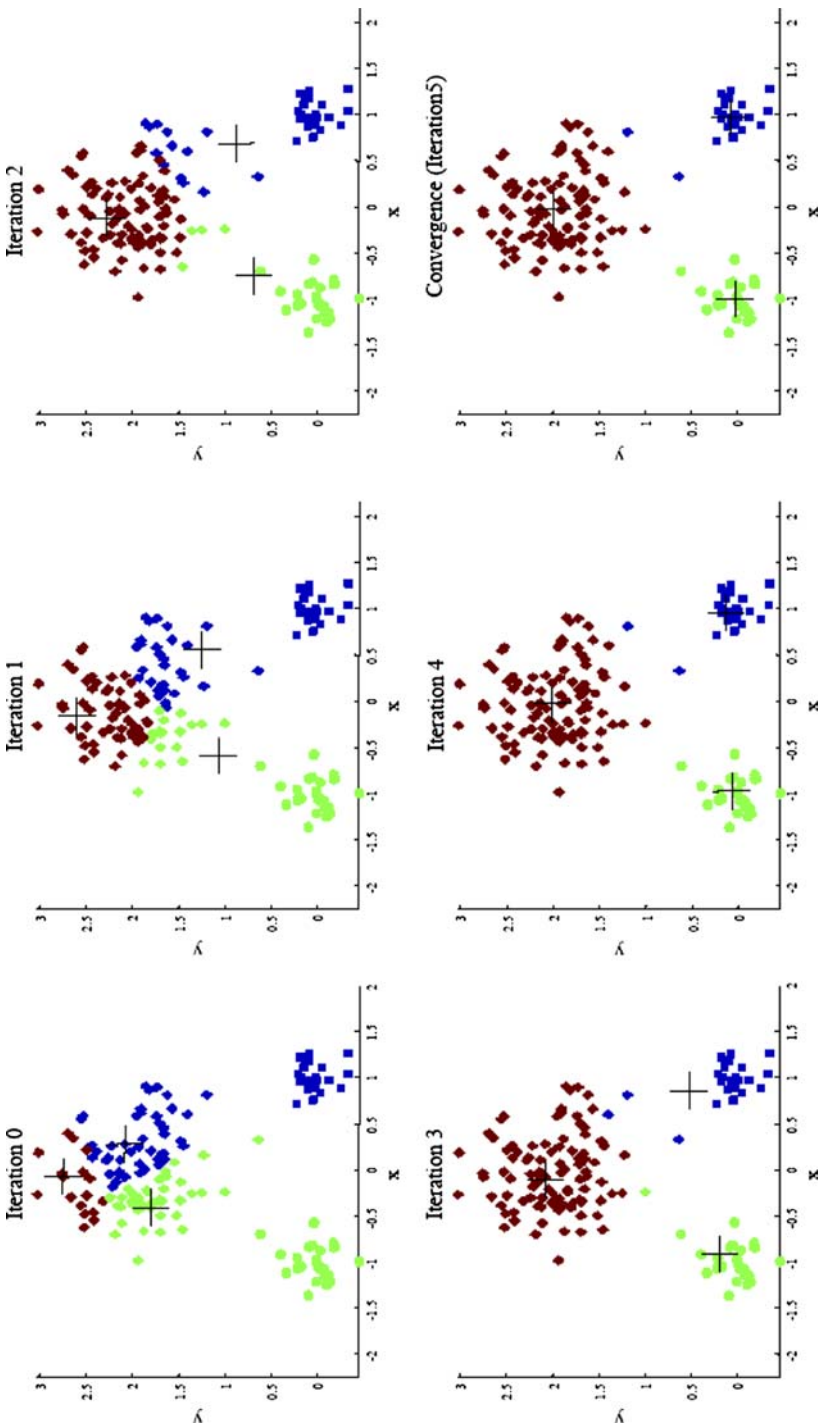

Fig. 1 Changes in cluster representative locations (indicated by '+' signs) and data assignments (indicated by color) during an execution of the `k-means` algorithm

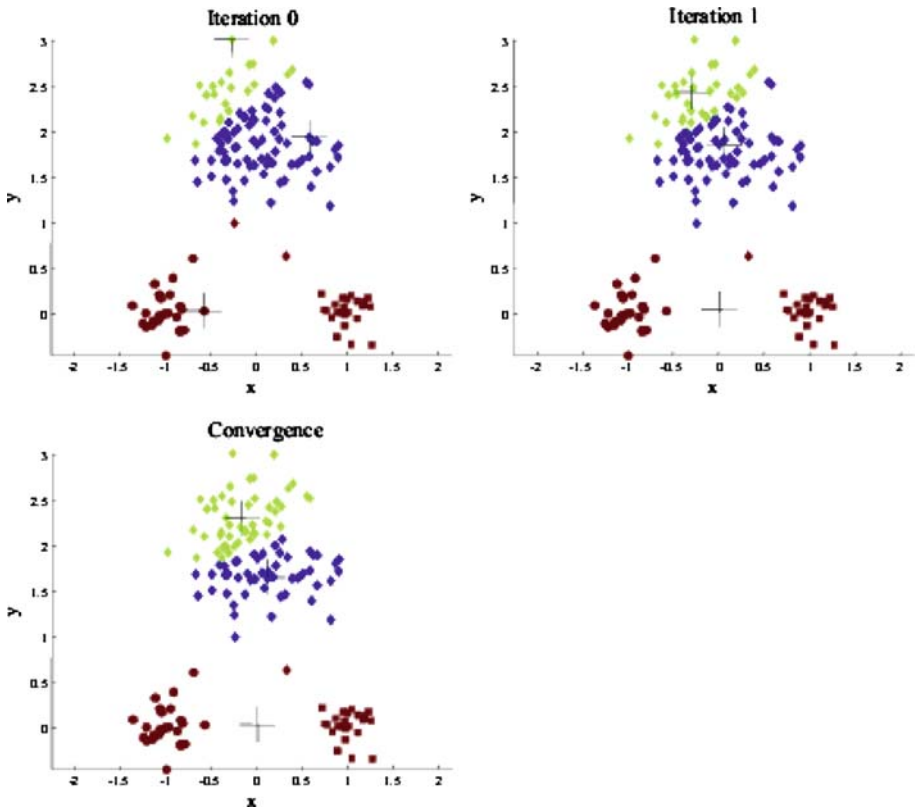

Fig. 2 Effect of an inferior initialization on the `k-means` results

extent by running the algorithm multiple times with different initial centroids, or by doing limited local search about the converged solution.

2.2 Limitations

In addition to being sensitive to initialization, the `k-means` algorithm suffers from several other problems. First, observe that `k-means` is a limiting case of fitting data by a mixture of k Gaussians with identical, isotropic covariance matrices ($\Sigma = \sigma^2\mathbf{I}$), when the soft assignments of data points to mixture components are hardened to allocate each data point solely to the most likely component. So, it will falter whenever the data is not well described by reasonably separated spherical balls, for example, if there are non-covex shaped clusters in the data. This problem may be alleviated by rescaling the data to "whiten" it before clustering, or by using a different distance measure that is more appropriate for the dataset. For example, information-theoretic clustering uses the KL-divergence to measure the distance between two data points representing two discrete probability distributions. It has been recently shown that if one measures distance by selecting any member of a very large class of divergences called Bregman divergences during the assignment step and makes no other changes, the essential properties of `k-means`, including guaranteed convergence, linear separation boundaries and scalability, are retained [3]. This result makes `k-means` effective for a much larger class of datasets so long as an appropriate divergence is used.

k-means can be paired with another algorithm to describe non-convex clusters. One first clusters the data into a large number of groups using k-means. These groups are then agglomerated into larger clusters using single link hierarchical clustering, which can detect complex shapes. This approach also makes the solution less sensitive to initialization, and since the hierarchical method provides results at multiple resolutions, one does not need to pre-specify k either.

The cost of the optimal solution decreases with increasing k till it hits zero when the number of clusters equals the number of distinct data-points. This makes it more difficult to (a) directly compare solutions with different numbers of clusters and (b) to find the optimum value of k. If the desired k is not known in advance, one will typically run k-means with different values of k, and then use a suitable criterion to select one of the results. For example, SAS uses the cube-clustering-criterion, while X-means adds a complexity term (which increases with k) to the original cost function (Eq. 1) and then identifies the k which minimizes this adjusted cost. Alternatively, one can progressively increase the number of clusters, in conjunction with a suitable stopping criterion. Bisecting k-means [73] achieves this by first putting all the data into a single cluster, and then recursively splitting the least compact cluster into two using 2-means. The celebrated LBG algorithm [34] used for vector quantization doubles the number of clusters till a suitable code-book size is obtained. Both these approaches thus alleviate the need to know k beforehand.

The algorithm is also sensitive to the presence of outliers, since "mean" is not a robust statistic. A preprocessing step to remove outliers can be helpful. Post-processing the results, for example to eliminate small clusters, or to merge close clusters into a large cluster, is also desirable. Ball and Hall's ISODATA algorithm from 1967 effectively used both pre- and post-processing on k-means.

2.3 Generalizations and connections

As mentioned earlier, k-means is closely related to fitting a mixture of k isotropic Gaussians to the data. Moreover, the generalization of the distance measure to all Bregman divergences is related to fitting the data with a mixture of k components from the exponential family of distributions. Another broad generalization is to view the "means" as probabilistic models instead of points in R^d. Here, in the assignment step, each data point is assigned to the most likely model to have generated it. In the "relocation" step, the model parameters are updated to best fit the assigned datasets. Such *model-based* k-means allow one to cater to more complex data, e.g. sequences described by Hidden Markov models.

One can also "kernelize" k-means [19]. Though boundaries between clusters are still linear in the implicit high-dimensional space, they can become non-linear when projected back to the original space, thus allowing kernel k-means to deal with more complex clusters. Dhillon et al. [19] have shown a close connection between kernel k-means and spectral clustering. The *K-medoid* algorithm is similar to k-means except that the centroids have to belong to the data set being clustered. Fuzzy c-means is also similar, except that it computes fuzzy membership functions for each clusters rather than a hard one.

Despite its drawbacks, k-means remains the most widely used partitional clustering algorithm in practice. The algorithm is simple, easily understandable and reasonably scalable, and can be easily modified to deal with streaming data. To deal with very large datasets, substantial effort has also gone into further speeding up k-means, most notably by using kd-trees or exploiting the triangular inequality to avoid comparing each data point with all the centroids during the assignment step. Continual improvements and generalizations of the

basic algorithm have ensured its continued relevance and gradually increased its effectiveness as well.

3 Support vector machines

In today's machine learning applications, support vector machines (SVM) [83] are considered a must try—it offers one of the most robust and accurate methods among all well-known algorithms. It has a sound theoretical foundation, requires only a dozen examples for training, and is insensitive to the number of dimensions. In addition, efficient methods for training SVM are also being developed at a fast pace.

In a two-class learning task, the aim of SVM is to find the best classification function to distinguish between members of the two classes in the training data. The metric for the concept of the "best" classification function can be realized geometrically. For a linearly separable dataset, a linear classification function corresponds to a separating hyperplane $f(x)$ that passes through the middle of the two classes, separating the two. Once this function is determined, new data instance x_n can be classified by simply testing the sign of the function $f(x_n)$; x_n belongs to the positive class if $f(x_n) > 0$.

Because there are many such linear hyperplanes, what SVM additionally guarantee is that the best such function is found by maximizing the margin between the two classes. Intuitively, the margin is defined as the amount of space, or separation between the two classes as defined by the hyperplane. Geometrically, the margin corresponds to the shortest distance between the closest data points to a point on the hyperplane. Having this geometric definition allows us to explore how to maximize the margin, so that even though there are an infinite number of hyperplanes, only a few qualify as the solution to SVM.

The reason why SVM insists on finding the maximum margin hyperplanes is that it offers the best generalization ability. It allows not only the best classification performance (e.g., accuracy) on the training data, but also leaves much room for the correct classification of the future data. To ensure that the maximum margin hyperplanes are actually found, an SVM classifier attempts to maximize the following function with respect to $\vec{\mathbf{w}}$ and b:

$$L_P = \frac{1}{2} \parallel \vec{\mathbf{w}} \parallel - \sum_{i=1}^{t} \alpha_i y_i (\vec{\mathbf{w}} \cdot \vec{\mathbf{x}}_\mathbf{i} + b) + \sum_{i=1}^{t} \alpha_i \tag{2}$$

where t is the number of training examples, and α_i, $i = 1, \ldots, t$, are non-negative numbers such that the derivatives of L_P with respect to α_i are zero. α_i are the Lagrange multipliers and L_P is called the Lagrangian. In this equation, the vectors $\vec{\mathbf{w}}$ and constant b define the hyperplane.

There are several important questions and related extensions on the above basic formulation of support vector machines. We list these questions and extensions below.

1. Can we understand the meaning of the SVM through a solid theoretical foundation?
2. Can we extend the SVM formulation to handle cases where we allow errors to exist, when even the best hyperplane must admit some errors on the training data?
3. Can we extend the SVM formulation so that it works in situations where the training data are not linearly separable?
4. Can we extend the SVM formulation so that the task is to predict numerical values or to rank the instances in the likelihood of being a positive class member, rather than classification?

5. Can we scale up the algorithm for finding the maximum margin hyperplanes to thousands and millions of instances?

Question 1 Can we understand the meaning of the SVM through a solid theoretical foundation?

Several important theoretical results exist to answer this question.

A learning machine, such as the SVM, can be modeled as a function class based on some parameters α. Different function classes can have different capacity in learning, which is represented by a parameter h known as the VC dimension [83]. The VC dimension measures the maximum number of training examples where the function class can still be used to learn perfectly, by obtaining zero error rates on the training data, for any assignment of class labels on these points. It can be proven that the actual error on the future data is bounded by a sum of two terms. The first term is the training error, and the second term if proportional to the square root of the VC dimension h. Thus, if we can minimize h, we can minimize the future error, as long as we also minimize the training error. In fact, the above maximum margin function learned by SVM learning algorithms is one such function. Thus, theoretically, the SVM algorithm is well founded.

Question 2 Can we extend the SVM formulation to handle cases where we allow errors to exist, when even the best hyperplane must admit some errors on the training data?

To answer this question, imagine that there are a few points of the opposite classes that cross the middle. These points represent the training error that existing even for the maximum margin hyperplanes. The "soft margin" idea is aimed at extending the SVM algorithm [83] so that the hyperplane allows a few of such noisy data to exist. In particular, introduce a slack variable ξ_i to account for the amount of a violation of classification by the function $f(x_i)$; ξ_i has a direct geometric explanation through the distance from a mistakenly classified data instance to the hyperplane $f(x)$. Then, the total cost introduced by the slack variables can be used to revise the original objective minimization function.

Question 3 Can we extend the SVM formulation so that it works in situations where the training data are not linearly separable?

The answer to this question depends on an observation on the objective function where the only appearances of $\vec{\mathbf{x}}_\mathbf{i}$ is in the form of a dot product. Thus, if we extend the dot product $\vec{\mathbf{x}}_\mathbf{i} \cdot \vec{\mathbf{x}}_\mathbf{j}$ through a functional mapping $\Phi(\vec{\mathbf{x}}_\mathbf{i})$ of each $\vec{\mathbf{x}}_\mathbf{i}$ to a different space $\mathcal{H}$ of larger and even possibly infinite dimensions, then the equations still hold. In each equation, where we had the dot product $\vec{\mathbf{x}}_\mathbf{i} \cdot \vec{\mathbf{x}}_\mathbf{j}$, we now have the dot product of the transformed vectors $\Phi(\vec{\mathbf{x}}_\mathbf{i}) \cdot \Phi(\vec{\mathbf{x}}_\mathbf{j})$, which is called a kernel function.

The kernel function can be used to define a variety of nonlinear relationship between its inputs. For example, besides linear kernel functions, you can define quadratic or exponential kernel functions. Much study in recent years have gone into the study of different kernels for SVM classification [70] and for many other statistical tests. We can also extend the above descriptions of the SVM classifiers from binary classifiers to problems that involve more than two classes. This can be done by repeatedly using one of the classes as a positive class, and the rest as the negative classes (thus, this method is known as the one-against-all method).

Question 4 Can we extend the SVM formulation so that the task is to learn to approximate data using a linear function, or to rank the instances in the likelihood of being a positive class member, rather a classification?

SVM can be easily extended to perform numerical calculations. Here we discuss two such extensions. The first is to extend SVM to perform regression analysis, where the goal is to produce a linear function that can approximate that target function. Careful consideration goes into the choice of the error models; in support vector regression, or SVR, the error is defined to be zero when the difference between actual and predicted values are within a epsilon amount. Otherwise, the *epsilon insensitive* error will grow linearly. The support vectors can then be learned through the minimization of the Lagrangian. An advantage of support vector regression is reported to be its insensitivity to outliers.

Another extension is to learn to rank elements rather than producing a classification for individual elements [39]. Ranking can be reduced to comparing pairs of instances and producing a $+1$ estimate if the pair is in the correct ranking order, and -1 otherwise. Thus, a way to reduce this task to SVM learning is to construct new instances for each pair of ranked instance in the training data, and to learn a hyperplane on this new training data.

This method can be applied to many areas where ranking is important, such as in document ranking in information retrieval areas.

Question 5 Can we scale up the algorithm for finding the maximum margin hyperplanes to thousands and millions of instances?

One of the initial drawbacks of SVM is its computational inefficiency. However, this problem is being solved with great success. One approach is to break a large optimization problem into a series of smaller problems, where each problem only involves a couple of carefully chosen variables so that the optimization can be done efficiently. The process iterates until all the decomposed optimization problems are solved successfully. A more recent approach is to consider the problem of learning an SVM as that of finding an approximate minimum enclosing ball of a set of instances.

These instances, when mapped to an N-dimensional space, represent a core set that can be used to construct an approximation to the minimum enclosing ball. Solving the SVM learning problem on these core sets can produce a good approximation solution in very fast speed. For example, the core-vector machine [81] thus produced can learn an SVM for millions of data in seconds.

4 The Apriori algorithm

4.1 Description of the algorithm

One of the most popular data mining approaches is to find frequent itemsets from a transaction dataset and derive association rules. Finding frequent itemsets (itemsets with frequency larger than or equal to a user specified minimum support) is not trivial because of its combinatorial explosion. Once frequent itemsets are obtained, it is straightforward to generate association rules with confidence larger than or equal to a user specified minimum confidence.

Apriori is a seminal algorithm for finding frequent itemsets using candidate generation [1]. It is characterized as a level-wise complete search algorithm using anti-monotonicity of itemsets, "if an itemset is not frequent, any of its superset is never frequent". By convention, Apriori assumes that items within a transaction or itemset are sorted in lexicographic order. Let the set of frequent itemsets of size k be F_k and their candidates be C_k. Apriori first scans the database and searches for frequent itemsets of size 1 by accumulating the count for each item and collecting those that satisfy the minimum support requirement. It then iterates on the following three steps and extracts all the frequent itemsets.

1. Generate C_{k+1}, candidates of frequent itemsets of size $k+1$, from the frequent itemsets of size k.
2. Scan the database and calculate the support of each candidate of frequent itemsets.
3. Add those itemsets that satisfies the minimum support requirement to F_{k+1}.

The Apriori algorithm is shown in Fig. 3. Function apriori-gen in line 3 generates C_{k+1} from F_k in the following two step process:

1. Join step: Generate R_{K+1}, the initial candidates of frequent itemsets of size $k+1$ by taking the union of the two frequent itemsets of size k, P_k and Q_k that have the first $k-1$ elements in common.

$$R_{K+1} = P_k \cup Q_k = \{item_l, \ldots, item_{k-1}, item_k, item_{k'}\}$$
$$P_k = \{item_l, item_2, \ldots, item_{k-1}, item_k\}$$
$$Q_k = \{item_l, item_2, \ldots, item_{k-1}, item_{k'}\}$$

   where, $iteml < item2 < \cdots < item_k < item_{k'}$.
2. Prune step: Check if all the itemsets of size k in R_{k+1} are frequent and generate C_{k+1} by removing those that do not pass this requirement from R_{k+1}. This is because any subset of size k of C_{k+1} that is not frequent cannot be a subset of a frequent itemset of size $k+1$.

Function subset in line 5 finds all the candidates of the frequent itemsets included in transaction t. Apriori, then, calculates frequency only for those candidates generated this way by scanning the database.

It is evident that Apriori scans the database at most $k_{\max+1}$ times when the maximum size of frequent itemsets is set at $k_{\max}$.

The Apriori achieves good performance by reducing the size of candidate sets (Fig. 3). However, in situations with very many frequent itemsets, large itemsets, or very low minimum support, it still suffers from the cost of generating a huge number of candidate sets

Algorithm 1 *Apriori*

F_l=(Frequent itemsets of cardinality 1);
for($k = 1; F_k \neq \phi; k++$) do begin
  C_{k+1} = apriori-gen(F_k); //New candidates
  for all transactions $t \in$ Database do begin
    C'_t = subset(C_{k+1}, t); //Candidates contained in t
    for all candidate $c \in C'_t$ do
      $c.count++$;
    end
    $F_{k+1} = \{C \in C_{k+1} \quad | c.count \geq$ minimum support $\}$
  end
end
Answer $\cup_k F_k$;

Fig. 3 Apriori algorithm

and scanning the database repeatedly to check a large set of candidate itemsets. In fact, it is necessary to generate 2^{100} candidate itemsets to obtain frequent itemsets of size 100.

4.2 The impact of the algorithm

Many of the pattern finding algorithms such as decision tree, classification rules and clustering techniques that are frequently used in data mining have been developed in machine learning research community. Frequent pattern and association rule mining is one of the few exceptions to this tradition. The introduction of this technique boosted data mining research and its impact is tremendous. The algorithm is quite simple and easy to implement. Experimenting with Apriori-like algorithm is the first thing that data miners try to do.

4.3 Current and further research

Since Apriori algorithm was first introduced and as experience was accumulated, there have been many attempts to devise more efficient algorithms of frequent itemset mining. Many of them share the same idea with Apriori in that they generate candidates. These include hash-based technique, partitioning, sampling and using vertical data format. Hash-based technique can reduce the size of candidate itemsets. Each itemset is hashed into a corresponding bucket by using an appropriate hash function. Since a bucket can contain different itemsets, if its count is less than a minimum support, these itemsets in the bucket can be removed from the candidate sets. A partitioning can be used to divide the entire mining problem into n smaller problems. The dataset is divided into n non-overlapping partitions such that each partition fits into main memory and each partition is mined separately. Since any itemset that is potentially frequent with respect to the entire dataset must occur as a frequent itemset in at least one of the partitions, all the frequent itemsets found this way are candidates, which can be checked by accessing the entire dataset only once. Sampling is simply to mine a random sampled small subset of the entire data. Since there is no guarantee that we can find all the frequent itemsets, normal practice is to use a lower support threshold. Trade off has to be made between accuracy and efficiency. Apriori uses a horizontal data format, i.e. frequent itemsets are associated with each transaction. Using vertical data format is to use a different format in which transaction IDs (TIDs) are associated with each itemset. With this format, mining can be performed by taking the intersection of TIDs. The support count is simply the length of the TID set for the itemset. There is no need to scan the database because TID set carries the complete information required for computing support.

The most outstanding improvement over Apriori would be a method called FP-growth (frequent pattern growth) that succeeded in eliminating candidate generation [36]. It adopts a divide and conquer strategy by (1) compressing the database representing frequent items into a structure called FP-tree (frequent pattern tree) that retains all the essential information and (2) dividing the compressed database into a set of conditional databases, each associated with one frequent itemset and mining each one separately. It scans the database only twice. In the first scan, all the frequent items and their support counts (frequencies) are derived and they are sorted in the order of descending support count in each transaction. In the second scan, items in each transaction are merged into a prefix tree and items (nodes) that appear in common in different transactions are counted. Each node is associated with an item and its count. Nodes with the same label are linked by a pointer called node-link. Since items are sorted in the descending order of frequency, nodes closer to the root of the prefix tree are shared by more transactions, thus resulting in a very compact representation that stores all the necessary information. Pattern growth algorithm works on FP-tree by choosing an

item in the order of increasing frequency and extracting frequent itemsets that contain the chosen item by recursively calling itself on the conditional FP-tree. FP-growth is an order of magnitude faster than the original Apriori algorithm.

There are several other dimensions regarding the extensions of frequent pattern mining. The major ones include the followings: (1) incorporating taxonomy in items [72]: Use of taxonomy makes it possible to extract frequent itemsets that are expressed by higher concepts even when use of the base level concepts produces only infrequent itemsets. (2) incremental mining: In this setting, it is assumed that the database is not stationary and a new instance of transaction keeps added. The algorithm in [12] updates the frequent itemsets without restarting from scratch. (3) using numeric valuable for item: When the item corresponds to a continuous numeric value, current frequent itemset mining algorithm is not applicable unless the values are discretized. A method of subspace clustering can be used to obtain an optimal value interval for each item in each itemset [85]. (4) using other measures than frequency, such as information gain or χ^2 value: These measures are useful in finding discriminative patterns but unfortunately do not satisfy anti-monotonicity property. However, these measures have a nice property of being convex with respect to their arguments and it is possible to estimate their upperbound for supersets of a pattern and thus prune unpromising patterns efficiently. AprioriSMP uses this principle [59]. (5) using richer expressions than itemset: Many algorithms have been proposed for sequences, tree and graphs to enable mining from more complex data structure [90,42]. (6) closed itemsets: A frequent itemset is closed if it is not included in any other frequent itemsets. Thus, once the closed itemsets are found, all the frequent itemsets can be derived from them. LCM is the most efficient algorithm to find the closed itemsets [82].

5 The EM algorithm

Finite mixture distributions provide a flexible and mathematical-based approach to the modeling and clustering of data observed on random phenomena. We focus here on the use of normal mixture models, which can be used to cluster continuous data and to estimate the underlying density function. These mixture models can be fitted by maximum likelihood via the EM (Expectation–Maximization) algorithm.

5.1 Introduction

Finite mixture models are being increasingly used to model the distributions of a wide variety of random phenomena and to cluster data sets [57]. Here we consider their application in the context of cluster analysis.

We let the p-dimensional vector ($\mathbf{y} = (y_1, \ldots, y_p)^{\mathrm{T}}$) contain the values of p variables measured on each of n (independent) entities to be clustered, and we let $\mathbf{y}_j$ denote the value of $\mathbf{y}$ corresponding to the jth entity ($j = 1, \ldots, n$). With the mixture approach to clustering, $\mathbf{y}_1, \ldots, \mathbf{y}_n$ are assumed to be an observed random sample from mixture of a finite number, say g, of groups in some unknown proportions $\pi_1, \ldots, \pi_g$.

The mixture density of $\mathbf{y}_j$ is expressed as

$$f(y_i; \Psi) = \sum_{i=1}^{g} \pi_i f_i(y_j; \theta_i) \quad (j = 1, \ldots, n), \tag{3}$$

where the mixing proportions $\pi_1, \ldots, \pi_g$ sum to one and the group-conditional density $f_i(y_j; \theta_i)$ is specified up to a vector θ_i of unknown parameters ($i = 1, \ldots, g$). The vector of all the unknown parameters is given by

$$\Psi = (\pi_1, \ldots, \pi_{g-1}, \theta_1^{\mathrm{T}}, \ldots, \theta_g^{\mathrm{T}})^{\mathrm{T}},$$

where the superscript "T" denotes vector transpose. Using an estimate of Ψ, this approach gives a probabilistic clustering of the data into g clusters in terms of estimates of the posterior probabilities of component membership,

$$\tau_i(y_j, \Psi) = \frac{\pi_i f_i(y_j; \theta_i)}{f(y_j; \Psi)}, \tag{4}$$

where $\tau_i(\mathbf{y}_j)$ is the posterior probability that y_j (really the entity with observation y_j) belongs to the ith component of the mixture ($i = 1, \ldots, g$; $j = 1, \ldots, n$).

The parameter vector Ψ can be estimated by maximum likelihood. The maximum likelihood estimate (MLE) of Ψ, $\hat{\Psi}$, is given by an appropriate root of the likelihood equation,

$$\partial \log L(\Psi)/\partial \Psi = 0, \tag{5}$$

where

$$\log L(\Psi) = \sum_{j=1}^{n} \log f(y_j; \Psi) \tag{6}$$

is the log likelihood function for Ψ. Solutions of (6) corresponding to local maximizers can be obtained via the expectation–maximization (EM) algorithm [17].

For the modeling of continuous data, the component-conditional densities are usually taken to belong to the same parametric family, for example, the normal. In this case,

$$f_i(y_j; \theta_i) = \phi(y_j; \mu_i, \Sigma_i), \tag{7}$$

where $\phi(y_j; \mu, \Sigma)$ denotes the p-dimensional multivariate normal distribution with mean vector μ and covariance matrix Σ.

One attractive feature of adopting mixture models with elliptically symmetric components such as the normal or t densities, is that the implied clustering is invariant under affine transformations of the data (that is, under operations relating to changes in location, scale, and rotation of the data). Thus the clustering process does not depend on irrelevant factors such as the units of measurement or the orientation of the clusters in space.

5.2 Maximum likelihood estimation of normal mixtures

McLachlan and Peel [57, Chap. 3] described the E- and M-steps of the EM algorithm for the maximum likelihood (ML) estimation of multivariate normal components; see also [56]. In the EM framework for this problem, the unobservable component labels z_{ij} are treated as being the "missing" data, where z_{ij} is defined to be one or zero according as y_j belongs or does not belong to the ith component of the mixture ($i = 1, \ldots, g;$, $j = 1, \ldots, n$).

On the $(k+1)$th iteration of the EM algorithm, the E-step requires taking the expectation of the complete-data log likelihood $\log L_c(\Psi)$, given the current estimate Ψ^k for Ψ. As is linear in the unobservable z_{ij}, this E-step is effected by replacing the z_{ij} by their conditional expectation given the observed data y_j, using $\Psi^{(k)}$. That is, z_{ij} is replaced by $\tau_{ij}^{(k)}$, which is the posterior probability that y_j belongs to the ith component of the mixture, using the

current fit $\Psi^{(k)}$ for Ψ ($i = 1, \ldots, g$; $j = 1, \ldots, n$). It can be expressed as

$$\tau_{ij}^{(k)} = \frac{\pi_i^{(k)} \phi(y_j; \mu_i^{(k)}, \Sigma_i^{(k)})}{f(y_j; \Psi^{(k)})}. \tag{8}$$

On the M-step, the updated estimates of the mixing proportion π_j, the mean vector μ_i, and the covariance matrix Σ_i for the ith component are given by

$$\pi_i^{(k+1)} = \sum_{j=1}^{n} \tau_{ij}^{(k)} \Big/ n, \tag{9}$$

$$\mu_i^{(k+1)} = \sum_{j=1}^{n} \tau_{ij}^{(k)} y_j \Big/ \sum_{j=1}^{n} \tau_{ij}^{(k)} \tag{10}$$

and

$$\Sigma_i^{(k+1)} = \frac{\sum_{j=1}^{n} \tau_{ij}^{(k)} (y_j - \mu_i^{(k+1)})(y_j - \mu_i^{(k+1)})^T}{\sum_{j=1}^{n} \tau_{ij}^{(k)}}. \tag{11}$$

It can be seen that the M-step exists in closed form.

These E- and M-steps are alternated until the changes in the estimated parameters or the log likelihood are less than some specified threshold.

5.3 Number of clusters

We can make a choice as to an appropriate value of g by consideration of the likelihood function. In the absence of any prior information as to the number of clusters present in the data, we monitor the increase in the log likelihood function as the value of g increases.

At any stage, the choice of $g = g_0$ versus $g = g_1$, for instance $g_1 = g_0 + 1$, can be made by either performing the likelihood ratio test or by using some information-based criterion, such as BIC (Bayesian information criterion). Unfortunately, regularity conditions do not hold for the likelihood ratio test statistic λ to have its usual null distribution of chi-squared with degrees of freedom equal to the difference d in the number of parameters for $g = g_1$ and $g = g_0$ components in the mixture models. One way to proceed is to use a resampling approach as in [55]. Alternatively, one can apply BIC, which leads to the selection of $g = g_1$ over $g = g_0$ if $-2 \log \lambda$ is greater than $d \log(n)$.

6 PageRank

6.1 Overview

PageRank [10] was presented and published by Sergey Brin and Larry Page at the Seventh International World Wide Web Conference (WWW7) in April 1998. It is a search ranking algorithm using hyperlinks on the Web. Based on the algorithm, they built the search engine Google, which has been a huge success. Now, every search engine has its own hyperlink based ranking method.

PageRank produces a static ranking of Web pages in the sense that a PageRank value is computed for each page off-line and it does not depend on search queries. The algorithm relies on the democratic nature of the Web by using its vast link structure as an indicator of an individual page's quality. In essence, PageRank interprets a hyperlink from page x to page y as a vote, by page x, for page y. However, PageRank looks at more than just the sheer

number of votes, or links that a page receives. It also analyzes the page that casts the vote. Votes casted by pages that are themselves "important" weigh more heavily and help to make other pages more "important". This is exactly the idea of rank prestige in social networks [86].

6.2 The algorithm

We now introduce the PageRank formula. Let us first state some main concepts in the Web context.

In-links of page i: These are the hyperlinks that point to page i from other pages. Usually, hyperlinks from the same site are not considered.

Out-links of page i: These are the hyperlinks that point out to other pages from page i. Usually, links to pages of the same site are not considered.

The following ideas based on rank prestige [86] are used to derive the PageRank algorithm:

1. A hyperlink from a page pointing to another page is an implicit conveyance of authority to the target page. Thus, the more in-links that a page i receives, the more prestige the page i has.
2. Pages that point to page i also have their own prestige scores. A page with a higher prestige score pointing to i is more important than a page with a lower prestige score pointing to i. In other words, a page is important if it is pointed to by other important pages.

According to rank prestige in social networks, the importance of page i (i's PageRank score) is determined by summing up the PageRank scores of all pages that point to i. Since a page may point to many other pages, its prestige score should be shared among all the pages that it points to.

To formulate the above ideas, we treat the Web as a directed graph $G = (V, E)$, where V is the set of vertices or nodes, i.e., the set of all pages, and E is the set of directed edges in the graph, i.e., hyperlinks. Let the total number of pages on the Web be n (i.e., $n = |V|$). The PageRank score of the page i (denoted by $P(i)$) is defined by

$$P(i) = \sum_{(j,i)\in E} \frac{P(j)}{O_j}, \tag{12}$$

where O_j is the number of out-links of page j. Mathematically, we have a system of n linear equations (12) with n unknowns. We can use a matrix to represent all the equations. Let P be a n-dimensional column vector of PageRank values, i.e.,

$$P = (P(1), P(2), \ldots, P(n))T.$$

Let A be the adjacency matrix of our graph with

$$A_{ij} = \begin{cases} \frac{1}{O_i} & \text{if } (i, j) \in E \\ 0 & \text{otherwise} \end{cases} \tag{13}$$

We can write the system of n equations with

$$\mathbf{P} = \mathbf{A}^{\mathrm{T}}\mathbf{P}. \tag{14}$$

This is the characteristic equation of the *eigensystem*, where the solution to P is an *eigenvector* with the corresponding *eigenvalue* of 1. Since this is a circular definition, an iterative algorithm is used to solve it. It turns out that if some conditions are satisfied, 1 is

Fig. 4 The power iteration method for PageRank

PageRank-Iterate(G)
  $\boldsymbol{P}_0 \leftarrow \boldsymbol{e}/n$
  $k \leftarrow 1$
  repeat
    $\boldsymbol{P}_k \leftarrow (1-d)\boldsymbol{e} + d\boldsymbol{A}^T\boldsymbol{P}_{k-1}$;
    $k \leftarrow k+1$;
  until $||\boldsymbol{P}_k - \boldsymbol{P}_{k-1}||_1 < \varepsilon$
  return $\boldsymbol{P}_k$

the largest eigenvalue and the PageRank vector P is the principal eigenvector. A well known mathematical technique called power iteration [30] can be used to find P.

However, the problem is that Eq. (14) does not quite suffice because the Web graph does not meet the conditions. In fact, Eq. (14) can also be derived based on the Markov chain. Then some theoretical results from Markov chains can be applied. After augmenting the Web graph to satisfy the conditions, the following PageRank equation is produced:

$$\mathbf{P} = (1-d)\mathbf{e} + d\mathbf{A}^{\mathrm{T}}\mathbf{P}, \tag{15}$$

where $\mathbf{e}$ is a column vector of all 1's. This gives us the PageRank formula for each page i:

$$P(i) = (1-d) + d\sum_{j=1}^{n} A_{ji}P(j), \tag{16}$$

which is equivalent to the formula given in the original PageRank papers [10,61]:

$$P(i) = (1-d) + d\sum_{(j,i)\in E} \frac{P(j)}{O_j}. \tag{17}$$

The parameter d is called the *damping factor* which can be set to a value between 0 and 1. $d = 0.85$ is used in [10,52].

The computation of PageRank values of the Web pages can be done using the power iteration method [30], which produces the principal eigenvector with the eigenvalue of 1. The algorithm is simple, and is given in Fig. 1. One can start with any initial assignments of PageRank values. The iteration ends when the PageRank values do not change much or converge. In Fig. 4, the iteration ends after the 1-norm of the residual vector is less than a pre-specified threshold e.

Since in Web search, we are only interested in the ranking of the pages, the actual convergence may not be necessary. Thus, fewer iterations are needed. In [10], it is reported that on a database of 322 million links the algorithm converges to an acceptable tolerance in roughly 52 iterations.

6.3 Further references on PageRank

Since PageRank was presented in [10,61], researchers have proposed many enhancements to the model, alternative models, improvements for its computation, adding the temporal dimension [91], etc. The books by Liu [52] and by Langville and Meyer [49] contain in-depth analyses of PageRank and several other link-based algorithms.

7 AdaBoost

7.1 Description of the algorithm

Ensemble learning [20] deals with methods which employ multiple learners to solve a problem. The generalization ability of an ensemble is usually significantly better than that of a single learner, so ensemble methods are very attractive. The AdaBoost algorithm [24] proposed by Yoav Freund and Robert Schapire is one of the most important ensemble methods, since it has solid theoretical foundation, very accurate prediction, great simplicity (Schapire said it needs only "just 10 lines of code"), and wide and successful applications.

Let $\mathcal{X}$ denote the instance space and $\mathcal{Y}$ the set of class labels. Assume $\mathcal{Y} = \{-1, +1\}$. Given a *weak* or *base learning algorithm* and a training set $\{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_m, y_m)\}$ where $\boldsymbol{x}_i \in \mathcal{X}$ and $y_i \in \mathcal{Y}$ $(i = 1, \ldots, m)$, the AdaBoost algorithm works as follows. First, it assigns equal weights to all the training examples $(\boldsymbol{x}_i, y_i)(i \in \{1, \ldots, m\})$. Denote the distribution of the weights at the t-th learning round as D_t. From the training set and D_t the algorithm generates a *weak* or *base learner* $h_t : \mathcal{X} \to \mathcal{Y}$ by calling the base learning algorithm. Then, it uses the training examples to test h_t, and the weights of the incorrectly classified examples will be increased. Thus, an updated weight distribution D_{t+1} is obtained. From the training set and D_{t+1} AdaBoost generates another weak learner by calling the base learning algorithm again. Such a process is repeated for T rounds, and the final model is derived by weighted majority voting of the T weak learners, where the weights of the learners are determined during the training process. In practice, the base learning algorithm may be a learning algorithm which can use weighted training examples directly; otherwise the weights can be exploited by sampling the training examples according to the weight distribution D_t. The pseudo-code of AdaBoost is shown in Fig. 5.

In order to deal with multi-class problems, Freund and Schapire presented the AdaBoost.M1 algorithm [24] which requires that the weak learners are strong enough even on hard distributions generated during the AdaBoost process. Another popular multi-class version of AdaBoost is AdaBoost.MH [69] which works by decomposing multi-class task to a series of binary tasks. AdaBoost algorithms for dealing with regression problems have also been studied. Since many variants of AdaBoost have been developed during the past decade, *Boosting* has become the most important "family" of ensemble methods.

7.2 Impact of the algorithm

As mentioned in Sect. 7.1, AdaBoost is one of the most important ensemble methods, so it is not strange that its high impact can be observed here and there. In this short article we only briefly introduce two issues, one theoretical and the other applied.

In 1988, Kearns and Valiant posed an interesting question, i.e., whether a *weak* learning algorithm that performs just slightly better than random guess could be "boosted" into an arbitrarily accurate *strong* learning algorithm. In other words, whether two complexity classes, *weakly learnable* and *strongly learnable* problems, are equal. Schapire [67] found that the answer to the question is "yes", and the proof he gave is a construction, which is the first Boosting algorithm. So, it is evident that AdaBoost was born with theoretical significance. AdaBoost has given rise to abundant research on theoretical aspects of ensemble methods, which can be easily found in machine learning and statistics literature. It is worth mentioning that for their AdaBoost paper [24], Schapire and Freund won the Godel Prize, which is one of the most prestigious awards in theoretical computer science, in the year of 2003.

Input: Data set $\mathcal{D} = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \cdots, (\boldsymbol{x}_m, y_m)\}$;

Base learning algorithm $\mathcal{L}$;

Number of learning rounds T.

Process:

$D_1(i) = 1/m$. % Initialize the weight distribution

for $t = 1, \cdots, T$:

$h_t = \mathcal{L}(\mathcal{D}, D_t)$; % Train a weak learner h_t from $\mathcal{D}$ using distribution D_t

$\epsilon_t = \Pr_{i \sim D_i}[h_t(\boldsymbol{x}_i \neq y_i)]$; % Measure the error of h_t

$\alpha_t = \frac{1}{2}\ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$; % Determine the weight of h_t

$$D_{t+1}(i) = \frac{D_t(i)}{Z_t} \times \begin{cases} \exp(-\alpha_t) & \text{if } h_t(\boldsymbol{x}_i) = y_i \\ \exp(\alpha_t) & \text{if } h_t(\boldsymbol{x}_i) \neq y_i \end{cases}$$

$$= \frac{D_t(i)\exp(-\alpha_t y_i h_t(\boldsymbol{x}_i))}{Z_t}$$ % Update the distribution, where Z_t is

% a normalization factor which enables D_{t+1} be a distribution

end.

Output: $H(\boldsymbol{x}) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(\boldsymbol{x})\right)$

Fig. 5 The AdaBoost algorithm

AdaBoost and its variants have been applied to diverse domains with great success. For example, Viola and Jones [84] combined AdaBoost with a cascade process for face detection. They regarded rectangular features as weak learners, and by using AdaBoost to weight the weak learners, they got very intuitive features for face detection. In order to get high accuracy as well as high efficiency, they used a cascade process (which is beyond the scope of this article). As the result, they reported a very strong face detector: On a 466 MHz machine, face detection on a 384×288 image cost only 0.067 seconds, which is 15 times faster than state-of-the-art face detectors at that time but with comparable accuracy. This face detector has been recognized as one of the most exciting breakthroughs in computer vision (in particular, face detection) during the past decade. It is not strange that "Boosting" has become a buzzword in computer vision and many other application areas.

7.3 Further research

Many interesting topics worth further studying. Here we only discuss on one theoretical topic and one applied topic.

Many empirical study show that AdaBoost often does not overfit, i.e., the test error of AdaBoost often tends to decrease even after the training error is zero. Many researchers have studied this and several theoretical explanations have been given, e.g. [38]. Schapire et al. [68] presented a margin-based explanation. They argued that AdaBoost is able to increase the margins even after the training error is zero, and thus it does not overfit even after a large number of rounds. However, Breiman [8] indicated that larger margin does not necessarily mean

better generalization, which seriously challenged the margin-based explanation. Recently, Reyzin and Schapire [65] found that Breiman considered minimum margin instead of average or median margin, which suggests that the margin-based explanation still has chance to survive. If this explanation succeeds, a strong connection between AdaBoost and SVM could be found. It is obvious that this topic is well worth studying.

Many real-world applications are born with high dimensionality, i.e., with a large amount of input features. There are two paradigms that can help us to deal with such kind of data, i.e., dimension reduction and feature selection. Dimension reduction methods are usually based on mathematical projections, which attempt to transform the original features into an appropriate feature space. After dimension reduction, the original meaning of the features is usually lost. Feature selection methods directly select some original features to use, and therefore they can preserve the original meaning of the features, which is very desirable in many applications. However, feature selection methods are usually based on heuristics, lacking solid theoretical foundation. Inspired by Viola and Jones's work [84], we think AdaBoost could be very useful in feature selection, especially when considering that it has solid theoretical foundation. Current research mainly focus on images, yet we think general AdaBoost-based feature selection techniques are well worth studying.

8 *k*NN: *k*-nearest neighbor classification

8.1 Description of the algorithm

One of the simplest, and rather trivial classifiers is the Rote classifier, which memorizes the entire training data and performs classification only if the attributes of the test object match one of the training examples exactly. An obvious drawback of this approach is that many test records will not be classified because they do not exactly match any of the training records. A more sophisticated approach, k-nearest neighbor (kNN) classification [23,75], finds a group of k objects in the training set that are closest to the test object, and bases the assignment of a label on the predominance of a particular class in this neighborhood. There are three key elements of this approach: a set of labeled objects, e.g., a set of stored records, a distance or similarity metric to compute distance between objects, and the value of k, the number of nearest neighbors. To classify an unlabeled object, the distance of this object to the labeled objects is computed, its k-nearest neighbors are identified, and the class labels of these nearest neighbors are then used to determine the class label of the object.

Figure 6 provides a high-level summary of the nearest-neighbor classification method. Given a training set D and a test object $x = (\mathbf{x}', y')$, the algorithm computes the distance (or similarity) between z and all the training objects $(\mathbf{x}, y) \in D$ to determine its nearest-neighbor list, D_z. ($\mathbf{x}$ is the data of a training object, while y is its class. Likewise, $\mathbf{x}'$ is the data of the test object and y' is its class.)

Once the nearest-neighbor list is obtained, the test object is classified based on the majority class of its nearest neighbors:

$$\text{Majority Voting: } y' = \operatorname*{argmax}_{v} \sum_{(\mathbf{x}_i, y_i) \in D_z} I(v = y_i), \tag{18}$$

where v is a class label, y_i is the class label for the ith nearest neighbors, and $I(\cdot)$ is an indicator function that returns the value 1 if its argument is true and 0 otherwise.

Input: D, the set of k training objects, and test object $z = (\mathbf{x}', y')$

Process:

Compute $d(\mathbf{x}', \mathbf{x})$, the distance between z and every object, $(\mathbf{x}, y) \in D$.

Select $D_z \subseteq D$, the set of k closest training objects to z.

Output: $y' = \underset{v}{\operatorname{argmax}} \sum_{(\mathbf{x}_i, y_i) \in D_z} I(v = y_i)$

Fig. 6 The k-nearest neighbor classification algorithm

8.2 Issues

There are several key issues that affect the performance of *k*NN. One is the choice of k. If k is too small, then the result can be sensitive to noise points. On the other hand, if k is too large, then the neighborhood may include too many points from other classes.

Another issue is the approach to combining the class labels. The simplest method is to take a majority vote, but this can be a problem if the nearest neighbors vary widely in their distance and the closer neighbors more reliably indicate the class of the object. A more sophisticated approach, which is usually much less sensitive to the choice of k, weights each object's vote by its distance, where the weight factor is often taken to be the reciprocal of the squared distance: $w_i = 1/d(\mathbf{x}', \mathbf{x}_i)^2$. This amounts to replacing the last step of the kNN algorithm with the following:

$$\text{Distance-Weighted Voting: } y' = \underset{v}{\operatorname{argmax}} \sum_{(\mathbf{x}_i, y_i) \in D_z} w_i \times I(v = y_i). \tag{19}$$

The choice of the distance measure is another important consideration. Although various measures can be used to compute the distance between two points, the most desirable distance measure is one for which a smaller distance between two objects implies a greater likelihood of having the same class. Thus, for example, if *k*NN is being applied to classify documents, then it may be better to use the cosine measure rather than Euclidean distance. Some distance measures can also be affected by the high dimensionality of the data. In particular, it is well known that the Euclidean distance measure become less discriminating as the number of attributes increases. Also, attributes may have to be scaled to prevent distance measures from being dominated by one of the attributes. For example, consider a data set where the height of a person varies from 1.5 to 1.8 m, the weight of a person varies from 90 to 300 lb, and the income of a person varies from $10,000 to $1,000,000. If a distance measure is used without scaling, the income attribute will dominate the computation of distance and thus, the assignment of class labels. A number of schemes have been developed that try to compute the weights of each individual attribute based upon a training set [32].

In addition, weights can be assigned to the training objects themselves. This can give more weight to highly reliable training objects, while reducing the impact of unreliable objects. The PEBLS system by by Cost and Salzberg [14] is a well known example of such an approach.

*K*NN classifiers are lazy learners, that is, models are not built explicitly unlike eager learners (e.g., decision trees, SVM, etc.). Thus, building the model is cheap, but classifying unknown objects is relatively expensive since it requires the computation of the k-nearest neighbors of the object to be labeled. This, in general, requires computing the distance of the unlabeled object to all the objects in the labeled set, which can be expensive particularly for large training sets. A number of techniques have been developed for efficient computation

of k-nearest neighbor distance that make use of the structure in the data to avoid having to compute distance to all objects in the training set. These techniques, which are particularly applicable for low dimensional data, can help reduce the computational cost without affecting classification accuracy.

8.3 Impact

*K*NN classification is an easy to understand and easy to implement classification technique. Despite its simplicity, it can perform well in many situations. In particular, a well known result by Cover and Hart [15] shows that the the error of the nearest neighbor rule is bounded above by twice the Bayes error under certain reasonable assumptions. Also, the error of the general *k*NN method asymptotically approaches that of the Bayes error and can be used to approximate it.

*K*NN is particularly well suited for multi-modal classes as well as applications in which an object can have many class labels. For example, for the assignment of functions to genes based on expression profiles, some researchers found that *k*NN outperformed SVM, which is a much more sophisticated classification scheme [48].

8.4 Current and future research

Although the basic *k*NN algorithm and some of its variations, such as weighted *k*NN and assigning weights to objects, are relatively well known, some of the more advanced techniques for *k*NN are much less known. For example, it is typically possible to eliminate many of the stored data objects, but still retain the classification accuracy of the *k*NN classifier. This is known as 'condensing' and can greatly speed up the classification of new objects [35]. In addition, data objects can be removed to improve classification accuracy, a process known as "editing" [88]. There has also been a considerable amount of work on the application of proximity graphs (nearest neighbor graphs, minimum spanning trees, relative neighborhood graphs, Delaunay triangulations, and Gabriel graphs) to the *k*NN problem. Recent papers by Toussaint [79,80], which emphasize a proximity graph viewpoint, provide an overview of work addressing these three areas and indicate some remaining open problems. Other important resources include the collection of papers by Dasarathy [16] and the book by Devroye et al. [18]. Finally, a fuzzy approach to *k*NN can be found in the work of Bezdek [4].

9 Naive Bayes

9.1 Introduction

Given a set of objects, each of which belongs to a known class, and each of which has a known vector of variables, our aim is to construct a rule which will allow us to assign future objects to a class, given only the vectors of variables describing the future objects. Problems of this kind, called problems of supervised classification, are ubiquitous, and many methods for constructing such rules have been developed. One very important one is the naive Bayes method—also called idiot's Bayes, simple Bayes, and independence Bayes. This method is important for several reasons. It is very easy to construct, not needing any complicated iterative parameter estimation schemes. This means it may be readily applied to huge data sets. It is easy to interpret, so users unskilled in classifier technology can understand why it is making the classification it makes. And finally, it often does surprisingly well: it may not

be the best possible classifier in any particular application, but it can usually be relied on to be robust and to do quite well. General discussion of the naive Bayes method and its merits are given in [22,33].

9.2 The basic principle

For convenience of exposition here, we will assume just two classes, labeled $i = 0, 1$. Our aim is to use the initial set of objects with known class memberships (the training set) to construct a score such that larger scores are associated with class 1 objects (say) and smaller scores with class 0 objects. Classification is then achieved by comparing this score with a threshold, t. If we define $P(i|x)$ to be the probability that an object with measurement vector $x = (x_1, \ldots, x_p)$ belongs to class i, then any monotonic function of $P(i|x)$ would make a suitable score. In particular, the ratio $P(1|x)/P(0|x)$ would be suitable. Elementary probability tells us that we can decompose $P(i|x)$ as proportional to $f(x|i)P(i)$, where $f(x|i)$ is the conditional distribution of x for class i objects, and $P(i)$ is the probability that an object will belong to class i if we know nothing further about it (the 'prior' probability of class i). This means that the ratio becomes

$$\frac{P(1|x)}{P(0|x)} = \frac{f(x|1)P(1)}{f(x|0)P(0)}. \tag{20}$$

To use this to produce classifications, we need to estimate the $f(x|i)$ and the $P(i)$. If the training set was a random sample from the overall population, the $P(i)$ can be estimated directly from the proportion of class i objects in the training set. To estimate the $f(x|i)$, the naive Bayes method assumes that the components of x are independent, $f(x|i) = \prod_{j=1}^{p} f(x_j|i)$, and then estimates each of the univariate distributions $f(x_j|i)$, $j = 1, \ldots, p$; $i = 0, 1$, separately. Thus the p dimensional multivariate problem has been reduced to p univariate estimation problems. Univariate estimation is familiar, simple, and requires smaller training set sizes to obtain accurate estimates. This is one of the particular, indeed unique attractions of the naive Bayes methods: estimation is simple, very quick, and does not require complicated iterative estimation schemes.

If the marginal distributions $f(x_j|i)$ are discrete, with each x_j taking only a few values, then the estimate $\hat{f}(x_j|i)$ is a multinomial histogram type estimator (see below)—simply counting the proportion of class i objects which fall into each cell. If the $f(x_j|i)$ are continuous, then a common strategy is to segment each of them into a small number of intervals and again use multinomial estimator, but more elaborate versions based on continuous estimates (e.g. kernel estimates) are also used.

Given the independence assumption, the ratio in (20) becomes

$$\frac{P(1|x)}{P(0|x)} = \frac{\prod_{j=1}^{p} f(x_j|1)P(1)}{\prod_{j=1}^{p} f(x_j|0)P(0)} = \frac{P(1)}{P(0)} \prod_{j=1}^{p} \frac{f(x_j|1)}{f(x_j|0)}. \tag{21}$$

Now, recalling that our aim was merely to produce a score which was monotonically related to $P(i|x)$, we can take logs of (21)—log is a monotonic increasing function. This gives an alternative score

$$\ln \frac{P(1|x)}{P(0|x)} = \ln \frac{P(1)}{P(0)} + \sum_{j=1}^{p} \ln \frac{f(x_j|1)}{f(x_j|0)}. \tag{22}$$

If we define $w_j = \ln(f(x_j|1)/f(x_j|0))$ and a constant $k = \ln(P(1)/P(0))$ we see that (22) takes the form of a simple sum

$$\ln\frac{P(1|x)}{P(0|x)} = k + \sum_{j=1}^{p} w_j, \tag{23}$$

so that the classifier has a particularly simple structure.

The assumption of independence of the x_j within each class implicit in the naive Bayes model might seem unduly restrictive. In fact, however, various factors may come into play which mean that the assumption is not as detrimental as it might seem. Firstly, a prior variable selection step has often taken place, in which highly correlated variables have been eliminated on the grounds that they are likely to contribute in a similar way to the separation between classes. This means that the relationships between the remaining variables might well be approximated by independence. Secondly, assuming the interactions to be zero provides an implicit regularization step, so reducing the variance of the model and leading to more accurate classifications. Thirdly, in some cases when the variables are correlated the optimal decision surface coincides with that produced under the independence assumption, so that making the assumption is not at all detrimental to performance. Fourthly, of course, the decision surface produced by the naive Bayes model can in fact have a complicated nonlinear shape: the surface is linear in the w_j but highly nonlinear in the original variables x_j, so that it can fit quite elaborate surfaces.

9.3 Some extensions

Despite the above, a large number of authors have proposed modifications to the naive Bayes method in an attempt to improve its predictive accuracy.

One early proposed modification was to shrink the simplistic multinomial estimate of the proportions of objects falling into each category of each discrete predictor variable. So, if the jth discrete predictor variable, x_j, has c_r categories, and if n_{jr} of the total of n objects fall into the rth category of this variable, the usual multinomial estimator of the probability that a future object will fall into this category, n_{jr}/n, is replaced by $(n_{jr} + c_r^{-1})/(n + 1)$. This shrinkage also has a direct Bayesian interpretation. It leads to estimates which have lower variance.

Perhaps the obvious way of easing the independence assumption is by introducing extra terms in the models of the distributions of x in each class, to allow for interactions. This has been attempted in a large number of ways, but we must recognize that doing this necessarily introduces complications, and so sacrifices the basic simplicity and elegance of the naive Bayes model. Within either (or any, more generally) class, the joint distribution of x is

$$f(x) = f(x_1)f(x_2|x_1)f(x_3|x_1, x_2)\cdots f(x_p|x_1, x_2, \ldots, x_{p-1}), \tag{24}$$

and this can be approximated by simplifying the conditional probabilities. The extreme arises with $f(x_i|x_1, \ldots, x_{i-1}) = f(x_i)$ for all i, and this is the naive Bayes method. Obviously, however, models between these two extremes can be used. For example, one could use the Markov model

$$f(x) = f(x_1)f(x_2|x_1)f(x_3|x_2)\cdots f(x_p|x_{p-1}). \tag{25}$$

This is equivalent to using a subset of two way marginal distributions instead of the univariate marginal distributions in the naive Bayes model.

Another extension to the naive Bayes model was developed entirely independently of it. This is the logistic regression model. In the above we obtained the decomposition (21) by

adopting the naive Bayes independence assumption. However, exactly the same structure for the ratio results if we model $f(x|1)$ by $g(x)\prod_{j=1}^{p} h_1(x_j)$ and $f(x|0)$ by $g(x)\prod_{j=1}^{p} h_0(x_j)$, where the function $g(x)$ is the same in each model. The ratio is thus

$$\frac{P(1|x)}{P(0|x)} = \frac{P(1)g(x)\prod_{j=1}^{p} h_1(x_j)}{P(0)g(x)\prod_{j=1}^{p} h_0(x_j)} = \frac{P(1)}{P(0)} \cdot \frac{\prod_{j=1}^{p} h_1(x_j)}{\prod_{j=1}^{p} h_0(x_j)}. \tag{26}$$

Here, the $h_i(x_j)$ do not even have to be probability density functions—it is sufficient that the $g(x)\prod_{j=1}^{p} h_i(x_j)$ are densities. The model in (26) is just as simple as the naive Bayes model, and takes exactly the same form—take logs and we have a sum as in (23)—but it is much more flexible because it does not assume independence of the x_j in each class. In fact, it permits arbitrary dependence structures, via the $g(x)$ function, which can take any form. The point is, however, that this dependence is the same in the two classes, so that it cancels out in the ratio in (26). Of course, this considerable extra flexibility of the logistic regression model is not obtained without cost. Although the resulting model form is identical to the naive Bayes model form (with different parameter values, of course), it cannot be estimated by looking at the univariate marginals separately: an iterative procedure has to be used.

9.4 Concluding remarks on naive Bayes

The naive Bayes model is tremendously appealing because of its simplicity, elegance, and robustness. It is one of the oldest formal classification algorithms, and yet even in its simplest form it is often surprisingly effective. It is widely used in areas such as text classification and spam filtering. A large number of modifications have been introduced, by the statistical, data mining, machine learning, and pattern recognition communities, in an attempt to make it more flexible, but one has to recognize that such modifications are necessarily complications, which detract from its basic simplicity. Some such modifications are described in [27,66].

10 CART

The 1984 monograph, "CART: Classification and Regression Trees," co-authored by Leo Breiman, Jerome Friedman, Richard Olshen, and Charles Stone, [9] represents a major milestone in the evolution of Artificial Intelligence, Machine Learning, non-parametric statistics, and data mining. The work is important for the comprehensiveness of its study of decision trees, the technical innovations it introduces, its sophisticated discussion of tree-structured data analysis, and its authoritative treatment of large sample theory for trees. While CART citations can be found in almost any domain, far more appear in fields such as electrical engineering, biology, medical research and financial topics than, for example, in marketing research or sociology where other tree methods are more popular. This section is intended to highlight key themes treated in the CART monograph so as to encourage readers to return to the original source for more detail.

10.1 Overview

The CART decision tree is a binary recursive partitioning procedure capable of processing continuous and nominal attributes both as targets and predictors. Data are handled in their raw form; no binning is required or recommended. Trees are grown to a maximal size without the use of a stopping rule and then pruned back (essentially split by split) to the root via cost-complexity pruning. The next split to be pruned is the one contributing least to the

overall performance of the tree on training data (and more than one split may be removed at a time). The procedure produces trees that are invariant under any order preserving transformation of the predictor attributes. The CART mechanism is intended to produce not one, but a sequence of nested pruned trees, all of which are candidate optimal trees. The "right sized" or "honest" tree is identified by evaluating the predictive performance of every tree in the pruning sequence. CART offers no internal performance measures for tree selection based on the training data as such measures are deemed suspect. Instead, tree performance is always measured on independent test data (or via cross validation) and tree selection proceeds only after test-data-based evaluation. If no test data exist and cross validation has not been performed, CART will remain agnostic regarding which tree in the sequence is best. This is in sharp contrast to methods such as C4.5 that generate preferred models on the basis of training data measures.

The CART mechanism includes automatic (optional) class balancing, automatic missing value handling, and allows for cost-sensitive learning, dynamic feature construction, and probability tree estimation. The final reports include a novel attribute importance ranking. The CART authors also broke new ground in showing how cross validation can be used to assess performance for every tree in the pruning sequence given that trees in different CV folds may not align on the number of terminal nodes. Each of these major features is discussed below.

10.2 Splitting rules

CART splitting rules are always couched in the form

An instance goes left if CONDITION, and goes right otherwise,

where the CONDITION is expressed as "attribute $X_i <= C$" for continuous attributes. For nominal attributes the CONDITION is expressed as membership in an explicit list of values. The CART authors argue that binary splits are to be preferred because (1) they fragment the data more slowly than multi-way splits, and (2) repeated splits on the same attribute are allowed and, if selected, will eventually generate as many partitions for an attribute as required. Any loss of ease in reading the tree is expected to be offset by improved performance. A third implicit reason is that the large sample theory developed by the authors was restricted to binary partitioning.

The CART monograph focuses most of its discussion on the Gini rule, which is similar to the better known entropy or information-gain criterion. For a binary (0/1) target the "Gini measure of impurity" of a node t is

$$G(t) = 1 - p(t)^2 - (1 - p(t))^2, \tag{27}$$

where $p(t)$ is the (possibly weighted) relative frequency of class 1 in the node, and the improvement (gain) generated by a split of the parent node P into left and right children L and R is

$$I(P) = G(P) - qG(L) - (1 - q)G(R). \tag{28}$$

Here, q is the (possibly weighted) fraction of instances going left. The CART authors favor the Gini criterion over information gain because the Gini can be readily extended to include symmetrized costs (see below) and is computed more rapidly than information gain. (Later versions of CART have added information gain as an optional splitting rule.) They introduce the modified twoing rule, which is based on a direct comparison of the target attribute

distribution in two child nodes:

$$I(split) = \left[.25(q(1-q))^u \sum_k |p_L(k) - p_R(k)| \right]^2, \quad (29)$$

where k indexes the target classes, $p_L()$ and $p_R()$ are the probability distributions of the target in the left and right child nodes respectively, and the power term u embeds a user-trollable penalty on splits generating unequal-sized child nodes. (This splitter is a modified version of Messenger and Mandell [58].) They also introduce a variant of the twoing split criterion that treats the classes of the target as ordered; ordered twoing attempts to ensure target classes represented on the left of a split are ranked below those represented on the right. In our experience the twoing criterion is often a superior performer on multi-class targets as well as on inherently difficult-to-predict (e.g. noisy) binary targets. For regression (continuous targets), CART offers a choice of Least Squares (LS) and Least Absolute Deviation (LAD) criteria as the basis for measuring the improvement of a split. Three other splitting rules for cost-sensitive learning and probability trees are discussed separately below.

10.3 Prior probabilities and class balancing

In its default classification mode CART always calculates class frequencies in any node relative to the class frequencies in the root. This is equivalent to automatically reweighting the data to balance the classes, and ensures that the tree selected as optimal minimizes balanced class error. The reweighting is implicit in the calculation of all probabilities and improvements and requires no user intervention; the reported sample counts in each node thus reflect the unweighted data. For a binary (0/1) target any node is classified as class 1 if, and only if,

$$N_1(node)/N_1(root) > N_0(node)/N_0(root). \quad (30)$$

This default mode is referred to as "priors equal" in the monograph. It has allowed CART users to work readily with any unbalanced data, requiring no special measures regarding class rebalancing or the introduction of manually constructed weights. To work effectively with unbalanced data it is sufficient to run CART using its default settings. Implicit reweighting can be turned off by selecting the "priors data" option, and the modeler can also elect to specify an arbitrary set of priors to reflect costs, or potential differences between training data and future data target class distributions.

10.4 Missing value handling

Missing values appear frequently in real world, and especially business-related databases, and the need to deal with them is a vexing challenge for all modelers. One of the major contributions of CART was to include a fully automated and highly effective mechanism for handling missing values. Decision trees require a missing value-handling mechanism at three levels: (a) during splitter evaluation, (b) when moving the training data through a node, and (c) when moving test data through a node for final class assignment. (See [63] for a clear discussion of these points.) Regarding (a), the first version of CART evaluated each splitter strictly on its performance on the subset of data for which the splitter is available. Later versions offer a family of penalties that reduce the split improvement measure as a function of the degree of missingness. For (b) and (c), the CART mechanism discovers "surrogate" or substitute splitters for every node of the tree, whether missing values occur in the training data or not. The surrogates are thus available should the tree be applied to new data that

does include missing values. This is in contrast to machines that can only learn about missing value handling from training data that include missing values. Friedman [25] suggested moving instances with missing splitter attributes into both left and right child nodes and making a final class assignment by pooling all nodes in which an instance appears. Quinlan [63] opted for a weighted variant of Friedman's approach in his study of alternative missing value-handling methods. Our own assessments of the effectiveness of CART surrogate performance in the presence of missing data are largely favorable, while Quinlan remains agnostic on the basis of the approximate surrogates he implements for test purposes [63]. Friedman et al. [26] noted that 50% of the CART code was devoted to missing value handling; it is thus unlikely that Quinlan's experimental version properly replicated the entire CART surrogate mechanism.

In CART the missing value handling mechanism is fully automatic and locally adaptive at every node. At each node in the tree the chosen splitter induces a binary partition of the data (e.g., X1 <= c1 and X1 > c1). A surrogate splitter is a single attribute Z that can predict this partition where the surrogate itself is in the form of a binary splitter (e.g., Z <= d and Z > d). In other words, every splitter becomes a new target which is to be predicted with a single split binary tree. Surrogates are ranked by an association score that measures the advantage of the surrogate over the default rule predicting that all cases go to the larger child node. To qualify as a surrogate, the variable must outperform this default rule (and thus it may not always be possible to find surrogates). When a missing value is encountered in a CART tree the instance is moved to the left or the right according to the top-ranked surrogate. If this surrogate is also missing then the second ranked surrogate is used instead, (and so on). If all surrogates are missing the default rule assigns the instance to the larger child node (possibly adjusting node sizes for priors). Ties are broken by moving an instance to the left.

10.5 Attribute importance

The importance of an attribute is based on the sum of the improvements in all nodes in which the attribute appears as a splitter (weighted by the fraction of the training data in each node split). Surrogates are also included in the importance calculations, which means that even a variable that never splits a node may be assigned a large importance score. This allows the variable importance rankings to reveal variable masking and nonlinear correlation among the attributes. Importance scores may optionally be confined to splitters and comparing the splitters-only and the full importance rankings is a useful diagnostic.

10.6 Dynamic feature construction

Friedman [25] discussed the automatic construction of new features within each node and, for the binary target, recommends adding the single feature

$$x * w,$$

where x is the original attribute vector and w is a scaled difference of means vector across the two classes (the direction of the Fisher linear discriminant). This is similar to running a logistic regression on all available attributes in the node and using the estimated logit as a predictor. In the CART monograph, the authors discuss the automatic construction of linear combinations that include feature selection; this capability has been available from the first release of the CART software. BFOS also present a method for constructing Boolean

combinations of splitters within each node, a capability that has not been included in the released software.

10.7 Cost-sensitive learning

Costs are central to statistical decision theory but cost-sensitive learning received only modest attention before Domingos [21]. Since then, several conferences have been devoted exclusively to this topic and a large number of research papers have appeared in the subsequent scientific literature. It is therefore useful to note that the CART monograph introduced two strategies for cost-sensitive learning and the entire mathematical machinery describing CART is cast in terms of the costs of misclassification. The cost of misclassifying an instance of class i as class j is $C(i, j)$ and is assumed to be equal to 1 unless specified otherwise; $C(i, i) = 0$ for all i. The complete set of costs is represented in the matrix C containing a row and a column for each target class. Any classification tree can have a total cost computed for its terminal node assignments by summing costs over all misclassifications. The issue in cost-sensitive learning is to induce a tree that takes the costs into account during its growing and pruning phases.

The first and most straightforward method for handling costs makes use of weighting: instances belonging to classes that are costly to misclassify are weighted upwards, with a common weight applying to all instances of a given class, a method recently rediscovered by Ting [78]. As implemented in CART. the weighting is accomplished transparently so that all node counts are reported in their raw unweighted form. For multi-class problems BFOS suggested that the entries in the misclassification cost matrix be summed across each row to obtain relative class weights that approximately reflect costs. This technique ignores the detail within the matrix but has now been widely adopted due to its simplicity. For the Gini splitting rule the CART authors show that it is possible to embed the entire cost matrix into the splitting rule, but only after it has been symmetrized. The "symGini" splitting rule generates trees sensitive to the difference in costs $C(i, j)$ and $C(i, k)$, and is most useful when the symmetrized cost matrix is an acceptable representation of the decision maker's problem. In contrast, the instance weighting approach assigns a single cost to all misclassifications of objects of class i. BFOS report that pruning the tree using the full cost matrix is essential to successful cost-sensitive learning.

10.8 Stopping rules, pruning, tree sequences, and tree selection

The earliest work on decision trees did not allow for pruning. Instead, trees were grown until they encountered some stopping condition and the resulting tree was considered final. In the CART monograph the authors argued that no rule intended to stop tree growth can guarantee that it will not miss important data structure (e.g., consider the two-dimensional XOR problem). They therefore elected to grow trees without stopping. The resulting overly large tree provides the raw material from which a final optimal model is extracted.

The pruning mechanism is based strictly on the training data and begins with a cost-complexity measure defined as

$$R_a(T) = R(T) + a|T|, \tag{31}$$

where $R(T)$ is the training sample cost of the tree, $|T|$ is the number of terminal nodes in the tree and a is a penalty imposed on each node. If $a = 0$ then the minimum cost-complexity tree is clearly the largest possible. If a is allowed to progressively increase the minimum cost-complexity tree will become smaller since the splits at the bottom of the tree that reduce

$R(T)$ the least will be cut away. The parameter a is progressively increased from 0 to a value sufficient to prune away all splits. BFOS prove that any tree of size Q extracted in this way will exhibit a cost $R(Q)$ that is minimum within the class of all trees with Q terminal nodes.

The optimal tree is defined as that tree in the pruned sequence that achieves minimum cost on test data. Because test misclassification cost measurement is subject to sampling error, uncertainty always remains regarding which tree in the pruning sequence is optimal. BFOS recommend selecting the "1 SE" tree that is the smallest tree with an estimated cost within 1 standard error of the minimum cost (or "0 SE") tree.

10.9 Probability trees

Probability trees have been recently discussed in a series of insightful articles elucidating their properties and seeking to improve their performance (see Provost and Domingos 2000). The CART monograph includes what appears to be the first detailed discussion of probability trees and the CART software offers a dedicated splitting rule for the growing of "class probability trees." A key difference between classification trees and probability trees is that the latter want to keep splits that generate terminal node children assigned to the same class whereas the former will not (such a split accomplishes nothing so far as classification accuracy is concerned). A probability tree will also be pruned differently than its counterpart classification tree, therefore, the final structure of the two optimal trees can be somewhat different (although the differences are usually modest). The primary drawback of probability trees is that the probability estimates based on training data in the terminal nodes tend to be biased (e.g., towards 0 or 1 in the case of the binary target) with the bias increasing with the depth of the node. In the recent ML literature the use of the LaPlace adjustment has been recommended to reduce this bias (Provost and Domingos 2002). The CART monograph offers a somewhat more complex method to adjust the terminal node estimates that has rarely been discussed in the literature. Dubbed the "Breiman adjustment", it adjusts the estimated misclassification rate $r^*(t)$ of any terminal node upwards by

$$r^*(t) = r(t) + e/(q(t) + S) \tag{32}$$

where $r(t)$ is the train sample estimate within the node, $q(t)$ is the fraction of the training sample in the node and S and e are parameters that are solved for as a function of the difference between the train and test error rates for a given tree. In contrast to the LaPlace method, the Breiman adjustment does not depend on the raw predicted probability in the node and the adjustment can be very small if the test data show that the tree is not overfit. Bloch et al. [5] reported very good performance for the Breiman adjustment in a series of empirical experiments.

10.10 Theoretical foundations

The earliest work on decision trees was entirely atheoretical. Trees were proposed as methods that appeared to be useful and conclusions regarding their properties were based on observing tree performance on a handful of empirical examples. While this approach remains popular in Machine Learning, the recent tendency in the discipline has been to reach for stronger theoretical foundations. The CART monograph tackles theory with sophistication, offering important technical insights and proofs for several key results. For example, the authors derive the expected misclassification rate for the maximal (largest possible) tree, showing that it is bounded from above by twice the Bayes rate. The authors also discuss the bias variance tradeoff in trees and show how the bias is affected by the number of attributes.

Based largely on the prior work of CART co-authors Richard Olshen and Charles Stone, the final three chapters of the monograph relate CART to theoretical work on nearest neighbors and show that as the sample size tends to infinity the following hold: (1) the estimates of the regression function converge to the true function, and (2) the risks of the terminal nodes converge to the risks of the corresponding Bayes rules. In other words, speaking informally, with large enough samples the CART tree will converge to the true function relating the target to its predictors and achieve the smallest cost possible (the Bayes rate). Practically speaking. such results may only be realized with sample sizes far larger than in common use today.

10.11 Selected biographical details

CART is often thought to have originated from the field of Statistics but this is only partially correct. Jerome Friedman completed his PhD in Physics at UC Berkeley and became leader of the Numerical Methods Group at the Stanford Linear Accelerator Center in 1972, where he focused on problems in computation. One of his most influential papers from 1975 presents a state-of-the-art algorithm for high speed searches for nearest neighbors in a database. Richard Olshen earned his BA at UC Berkeley and PhD in Statistics at Yale and focused his earliest work on large sample theory for recursive partitioning. He began his collaboration with Friedman after joining the Stanford Linear Accelerator Center in 1974. Leo Breiman earned his BA in Physics at the California Institute of Technology, his PhD in Mathematics at UC Berkeley, and made notable contributions to pure probability theory (Breiman, 1968) [7] while a Professor at UCLA. In 1967 he left academia for 13 years to work as an industrial consultant; during this time he encountered the military data analysis problems that inspired his contributions to CART. An interview with Leo Breiman discussing his career and personal life appears in [60].

Charles Stone earned his BA in mathematics at the California Institute of Technology, and his PhD in Statistics at Stanford. He pursued probability theory in his early years as an academic and is the author of several celebrated papers in probability theory and nonparametric regression. He worked with Breiman at UCLA and was drawn by Breiman into the research leading to CART in the early 1970s. Breiman and Friedman first met at an Interface conference in 1976, which shortly led to collaboration involving all four co-authors. The first outline of their book was produced in a memo dated 1978 and the completed CART monograph was published in 1984.

The four co-authors have each been distinguished for their work outside of CART. Stone and Breiman were elected to the National Academy of Sciences (in 1993 and 2001, respectively) and Friedman was elected to the American Academy of Arts and Sciences in 2006. The specific work for which they were honored can be found on the respective academy websites. Olshen is a Fellow of the Institute of Mathematical Statistics, a Fellow of the IEEE, and Fellow, American Association for the Advancement of Science.

11 Concluding remarks

Data mining is a broad area that integrates techniques from several fields including machine learning, statistics, pattern recognition, artificial intelligence, and database systems, for the analysis of large volumes of data. There have been a large number of data mining algorithms rooted in these fields to perform different data analysis tasks. The 10 algorithms identified by the IEEE International Conference on Data Mining (ICDM) and presented in

this article are among the most influential algorithms for classification [47,51,77], clustering [11,31,40,44–46], statistical learning [28,76,92], association analysis [2,6,13,50,54,74], and link mining.

With a formal tie with the ICDM conference, *Knowledge and Information Systems* has been publishing the best papers from ICDM every year, and several of the above papers cited for classification, clustering, statistical learning, and association analysis were selected by the previous years' ICDM program committees for journal publication in *Knowledge and Information Systems* after their revisions and expansions. We hope this survey paper can inspire more researchers in data mining to further explore these top-10 algorithms, including their impact and new research issues.

Acknowledgments The initiative of identifying the top-10 data mining algorithms started in May 2006 out of a discussion between Dr. Jiannong Cao in the Department of Computing at the Hong Kong Polytechnic University (PolyU) and Dr. Xindong Wu, when Dr. Wu was giving a seminar on 10 Challenging Problems in Data Mining Research [89] at PolyU. Dr. Wu and Dr. Kumar continued this discussion at KDD-06 in August 2006 with various people, and received very enthusiastic support. Naila Elliott in the Department of Computer Science and Engineering at the University of Minnesota collected and compiled the algorithm nominations and voting results in the 3-step identification process. Yan Zhang in the Department of Computer Science at the University of Vermont converted the 10 section submissions in different formats into the same LaTeX format, which was a time-consuming process.

References

1. Agrawal R, Srikant R (1994) Fast algorithms for mining association rules. In: Proceedings of the 20th VLDB conference, pp 487–499
2. Ahmed S, Coenen F, Leng PH (2006) Tree-based partitioning of date for association rule mining. Knowl Inf Syst 10(3):315–331
3. Banerjee A, Merugu S, Dhillon I, Ghosh J (2005) Clustering with Bregman divergences. J Mach Learn Res 6:1705–1749
4. Bezdek JC, Chuah SK, Leep D (1986) Generalized k-nearest neighbor rules. Fuzzy Sets Syst 18(3):237–256. http://dx.doi.org/10.1016/0165-0114(86)90004-7
5. Bloch DA, Olshen RA, Walker MG (2002) Risk estimation for classification trees. J Comput Graph Stat 11:263–288
6. Bonchi F, Lucchese C (2006) On condensed representations of constrained frequent patterns. Knowl Inf Syst 9(2):180–201
7. Breiman L (1968) Probability theory. Addison-Wesley, Reading. Republished (1991) in Classics of mathematics. SIAM, Philadelphia
8. Breiman L (1999) Prediction games and arcing classifiers. Neural Comput 11(7):1493–1517
9. Breiman L, Friedman JH, Olshen RA, Stone CJ (1984) Classification and regression trees. Wadsworth, Belmont
10. Brin S, Page L (1998) The anatomy of a large-scale hypertextual Web Search Sngine. Comput Networks 30(1–7):107–117
11. Chen JR (2007) Making clustering in delay-vector space meaningful. Knowl Inf Syst 11(3):369–385
12. Cheung DW, Han J, Ng V, Wong CY (1996) Maintenance of discovered association rules in large databases: an incremental updating technique. In: Proceedings of the ACM SIGMOD international conference on management of data, pp. 13–23
13. Chi Y, Wang H, Yu PS, Muntz RR (2006) Catch the moment: maintaining closed frequent itemsets over a data stream sliding window. Knowl Inf Syst 10(3):265–294
14. Cost S, Salzberg S (1993) A weighted nearest neighbor algorithm for learning with symbolic features. Mach Learn 10:57.78 (PEBLS: Parallel Examplar-Based Learning System)
15. Cover T, Hart P (1967) Nearest neighbor pattern classification. IEEE Trans Inform Theory 13(1):21–27
16. Dasarathy BV (ed) (1991) Nearest neighbor (NN) norms: NN pattern classification techniques. IEEE Computer Society Press
17. Dempster AP, Laird NM, Rubin DB (1977) Maximum likelihood from incomplete data via the EM algorithm (with discussion). J Roy Stat Soc B 39:1–38

18. Devroye L, Gyorfi L, Lugosi G (1996) A probabilistic theory of pattern recognition. Springer, New York. ISBN 0-387-94618-7
19. Dhillon IS, Guan Y, Kulis B (2004) Kernel k-means: spectral clustering and normalized cuts. KDD 2004, pp 551–556
20. Dietterich TG (1997) Machine learning: Four current directions. AI Mag 18(4):97–136
21. Domingos P (1999) MetaCost: A general method for making classifiers cost-sensitive. In: Proceedings of the fifth international conference on knowledge discovery and data mining, pp 155–164
22. Domingos P, Pazzani M (1997) On the optimality of the simple Bayesian classifier under zero-one loss. Mach Learn 29:103–130
23. Fix E, Hodges JL, Jr (1951) Discriminatory analysis, nonparametric discrimination. USAF School of Aviation Medicine, Randolph Field, Tex., Project 21-49-004, Rept. 4, Contract AF41(128)-31, February 1951
24. Freund Y, Schapire RE (1997) A decision-theoretic generalization of on-line learning and an application to boosting. J Comput Syst Sci 55(1):119–139
25. Friedman JH, Bentley JL, Finkel RA (1977) An algorithm for finding best matches in logarithmic time. ACM Trans. Math. Software 3, 209. Also available as Stanford Linear Accelerator Center Rep. SIX-PUB-1549, February 1975
26. Friedman JH, Kohavi R, Yun Y (1996) Lazy decision trees. In: Proceedings of the thirteenth national conference on artificial intelligence, San Francisco, CA. AAAI Press/MIT Press, pp. 717–724
27. Friedman N, Geiger D, Goldszmidt M (1997) Bayesian network classifiers. Mach Learn 29:131–163
28. Fung G, Stoeckel J (2007) SVM feature selection for classification of SPECT images of Alzheimer's disease using spatial information. Knowl Inf Syst 11(2):243–258
29. Gates GW (1972) The reduced nearest neighbor rule. IEEE Trans Inform Theory 18:431–433
30. Golub GH, Van Loan CF (1983) Matrix computations. The Johns Hopkins University Press
31. Gondek D, Hofmann T (2007) Non-redundant data clustering. Knowl Inf Syst 12(1):1–24
32. Han E (1999) Text categorization using weight adjusted k-nearest neighbor classification. PhD thesis, University of Minnesota, October 1999
33. Hand DJ, Yu K (2001) Idiot's Bayes—not so stupid after all?. Int Stat Rev 69:385–398
34. Gray RM, Neuhoff DL (1998) Quantization. IEEE Trans Inform Theory 44(6):2325–2384
35. Hart P (1968) The condensed nearest neighbor rule. IEEE Trans Inform Theory 14:515–516
36. Han J, Pei J, Yin Y (2000) Mining frequent patterns without candidate generation. In: Proceedings of ACM SIGMOD international conference on management of data, pp 1–12
37. Hastie T, Tibshirani R (1996) Discriminant adaptive nearest neighbor classification. IEEE Trans Pattern Anal Mach Intell 18(6):607–616
38. Friedman J, Hastie T, Tibshirani R (2000) Additive logistic regression: a statistical view of boosting with discussions. Ann Stat 28(2):337–407
39. Herbrich R, Graepel T, Obermayer K (2000) Rank boundaries for ordinal regression. Adv Mar Classif pp 115–132
40. Hu T, Sung SY (2006) Finding centroid clusterings with entropy-based criteria. Knowl Inf Syst 10(4):505–514
41. Hunt EB, Marin J, Stone PJ (1966) Experiments in induction. Academic Press, New York
42. Inokuchi A, Washio T, Motoda H (2005) General framework for mining frequent subgraphs from labeled graphs. Fundament Inform 66(1-2):53–82
43. Jain AK, Dubes RC (1988) Algorithms for clustering data. Prentice-Hall, Englewood Cliffs
44. Jin R, Goswami A, Agrawal G (2006) Fast and exact out-of-core and distributed *k*-means clustering. Knowl Inf Syst 10(1):17–40
45. Kobayashi M, Aono M (2006) Exploring overlapping clusters using dynamic re-scaling and sampling. Knowl Inf Syst 10(3):295–313
46. Koga H, Ishibashi T, Watanabe T (2007) Fast agglomerative hierarchical clustering algorithm using Locality-Sensitive Hashing. Knowl Inf Syst 12(1):25–53
47. Kukar M (2006) Quality assessment of individual classifications in machine learning and data mining. Knowl Inf Syst 9(3):364–384
48. Kuramochi M, Karypis G (2005) Gene Classification using Expression Profiles: A Feasibility Study. Int J Artif Intell Tools 14(4):641–660
49. Langville AN, Meyer CD (2006) Google's PageRank and beyond: the science of search engine rankings. Princeton University Press, Princeton
50. Leung CW-k, Chan SC-f, Chung F-L (2006) A collaborative filtering framework based on fuzzy association rules and multiple-level similarity. Knowl Inf Syst 10(3):357–381
51. Li T, Zhu S, Ogihara M (2006) Using discriminant analysis for multi-class classification: an experimental investigation. Knowl Inf Syst 10(4):453–472

52. Liu B (2007) Web data mining: exploring hyperlinks, contents and usage Data. Springer, Heidelberg
53. Lloyd SP (1957) Least squares quantization in PCM. Unpublished Bell Lab. Tech. Note, portions presented at the Institute of Mathematical Statistics Meeting Atlantic City, NJ, September 1957. Also, IEEE Trans Inform Theory (Special Issue on Quantization), vol IT-28, pp 129–137, March 1982
54. Leung CK-S, Khan QI, Li Z, Hoque T (2007) CanTree: a canonical-order tree for incremental frequent-pattern mining. Knowl Inf Syst 11(3):287–311
55. McLachlan GJ (1987) On bootstrapping the likelihood ratio test statistic for the number of components in a normal mixture.. Appl Stat 36:318–324
56. McLachlan GJ, Krishnan T (1997) The EM algorithm and extensions. Wiley, New York
57. McLachlan GJ, Peel D (2000) Finite Mixture Models. Wiley, New York
58. Messenger RC, Mandell ML (1972) A model search technique for predictive nominal scale multivariate analysis. J Am Stat Assoc 67:768–772
59. Morishita S, Sese J (2000) Traversing lattice itemset with statistical metric pruning. In: Proceedings of PODS'00, pp 226–236
60. Olshen R (2001) A conversation with Leo Breiman. Stat Sci 16(2):184–198
61. Page L, Brin S, Motwami R, Winograd T (1999) The PageRank citation ranking: bringing order to the Web. Technical Report 1999–0120, Computer Science Department, Stanford University
62. Quinlan JR (1979) Discovering rules by induction from large collections of examples. In: Michie D (ed), Expert systems in the micro electronic age. Edinburgh University Press, Edinburgh
63. Quinlan R (1989) Unknown attribute values in induction. In: Proceedings of the sixth international workshop on machine learning, pp. 164–168
64. Quinlan JR (1993) C4.5: Programs for machine learning. Morgan Kaufmann Publishers, San Mateo
65. Reyzin L, Schapire RE (2006) How boosting the margin can also boost classifier complexity. In: Proceedings of the 23rd international conference on machine learning. Pittsburgh, PA, pp. 753–760
66. Ridgeway G, Madigan D, Richardson T (1998) Interpretable boosted naive Bayes classification. In: Agrawal R, Stolorz P, Piatetsky-Shapiro G (eds) Proceedings of the fourth international conference on knowledge discovery and data mining.. AAAI Press, Menlo Park pp 101–104
67. Schapire RE (1990) The strength of weak learnability. Mach Learn 5(2):197–227
68. Schapire RE, Freund Y, Bartlett P, Lee WS (1998) Boosting the margin: A new explanation for the effectiveness of voting methods. Ann Stat 26(5):1651–1686
69. Schapire RE, Singer Y (1999) Improved boosting algorithms using confidence-rated predictions. Mach Learn 37(3):297–336
70. Scholkopf B, Smola AJ (2002) Learning with kernels. MIT Press
71. Seidl T, Kriegel H (1998) Optimal multi-step k-nearest neighbor search. In: Tiwary A, Franklin M (eds) Proceedings of the 1998 ACM SIGMOD international conference on management of data, Seattle, Washington, United States, 1–4 June, 1998. ACM Press, New York pp 154–165
72. Srikant R, Agrawal R (1995) Mining generalized association rules. In: Proceedings of the 21st VLDB conference. pp. 407–419
73. Steinbach M, Karypis G, Kumar V (2000) A comparison of document clustering techniques. In: Proceedings of the KDD Workshop on Text Mining
74. Steinbach M, Kumar V (2007) Generalizing the notion of confidence. Knowl Inf Syst 12(3):279–299
75. Tan P-N, Steinbach M, Kumar V (2006) Introduction to data mining. Pearson Addison-Wesley
76. Tao D, Li X, Wu X, Hu W, Maybank SJ (2007) Supervised tensor learning. Knowl Inf Syst 13(1):1–42
77. Thabtah FA, Cowling PI, Peng Y (2006) Multiple labels associative classification. Knowl Inf Syst 9(1):109–129
78. Ting KM (2002) An instance-weighting method to induce cost-sensitive trees. IEEE Trans Knowl Data Eng 14:659–665
79. Toussaint GT (2002) Proximity graphs for nearest neighbor decision rules: recent progress. In: Interface-2002, 34th symposium on computing and statistics (theme: Geoscience and Remote Sensing). Ritz-Carlton Hotel, Montreal, Canada, 17–20 April, 2002
80. Toussaint GT (2002) Open problems in geometric methods for instance-based learning. JCDCG 273–283
81. Tsang IW, Kwok JT, Cheung P-M (2005) Core vector machines: Fast SVM training on very large data sets. J Mach Learn Res 6:363–392
82. Uno T, Asai T, Uchida Y, Arimura H (2004) An efficient algorithm for enumerating frequent closed patterns in transaction databases. In: Proc. of the 7th international conference on discovery science. LNAI vol 3245, Springe, Heidelberg, pp 16–30
83. Vapnik V (1995) The nature of statistical learning theory. Springer, New York
84. Viola P, Jones M (2001) Rapid object detection using a boosted cascade of simple features. In: Proceedings of the IEEE computer society conference on computer vision and pattern recognition. pages 511–518, Kauai, HI

85. Washio T, Nakanishi K, Motoda H (2005) Association rules based on levelwise subspace clustering. In: Proceedings. of 9th European conference on principles and practice of knowledge discovery in databases. LNAI, vol 3721, pp. 692–700 Springer, Heidelberg
86. Wasserman S, Raust K (1994) Social network analysis. Cambridge University Press, Cambridge
87. Wettschereck D, Aha D, Mohri T (1997) A review and empirical evaluation of feature weighting methods for a class of lazy learning algorithms. Artif Intell Rev 11:273–314
88. Wilson DL (1972) Asymptotic properties of nearest neighbor rules using edited data. IEEE Trans Syst Man Cyberne 2:408–420
89. Yang Q, Wu X (2006) 10 challenging problems in data mining research. Int J Inform Technol Decis Making 5(4):597–604
90. Yan X, Han J (2002) gSpan: Graph-based substructure pattern mining. In: Proceedings of ICDM'02, pp 721–724
91. Yu PS, Li X, Liu B (2005) Adding the temporal dimension to search—a case study in publication search. In: Proceedings of Web Intelligence (WI'05)
92. Zhang J, Kang D-K, Silvescu A, Honavar V (2006) Learning accurate and concise naïve Bayes classifiers from attribute value taxonomies and data. Knowl Inf Syst 9(2):157–179