International journal of Computer Science & Network Solutions
http://www.ijcsns.com
Jan.2015-Volume 3.No.1
ISSN 2345-3397

Social Networks based e-Learning Systems via Review of Recommender Systems Techniques

A. A. Salama, M.M.Eisa, S.A.EL-Hafeez, M. M. Lotfy

Department of Mathematics and Computer Science, Faculty of Science, Port Said University, Egypt

Department of Computer Science, Higher Institute of Management and Computers, Port Said University, Egypt

Abstract

e-Learning has turned to be a necessity for everyone, as it enables continuous and life-long education. Learners are social by nature. They want to connect to othersand share the same interests. Online communities are important to help and encourage learners to continue education. Learners through social capabilities can share different experiences.Social networks are cornerstone for e-Learning. However, alternatives are many. Learners might get lost in the tremendous learning resources that are available. It is the role of recommender systems to help learners find their own way through e-Learning. We present a review of different recommender system algorithms that are utilized in social networks based e-Learning systems. Future research will include our proposed our e-Learning system that utilizes Recommender System and Social Network.

Keywords: **Social Networks, e-Learning ,Techniques**

I. Introduction

The Internet shows great potential for enhancing collaboration between people and the role of social software has become increasingly relevant in recent years. A vast array of systems exist which employ users' stored profile data, identifying matches for collaboration. Social interaction within an online framework can help university students share experiences and collaborate on relevant topics. As such, social networks can act as a pedagogical agent, for example, with problem-based learning [1].Social networking websites are virtual communities which allow people to connect and interact with each other on a particular subject or to just ‘‘hang out" together online. Membership of online social networks has recently exploded at an exponential rate [2]. Recommender systems cover an important field within collaborative services that are developed in the Web 2.0 environment and enable user-generated opinions to be exploited in a sophisticated and powerful way. Recommender Systems can be considered as social networking tools that provide dynamic and collaborative communication, interaction and knowledge [3].

Course management systems (CMSs) can offer a great variety of channels and workspaces to facilitate information sharing and communication among participants in a course. They let educators distribute information to students, produce content material, prepare assignments and tests, engage in discussions, manage distance classes and enable collaborative learning with forums, chats, file storage areas, news services, etc. Some examples of commercial systems are Blackboard, WebCT and Top Class while some examples of free systems are Moodle, Ilias and Claroline. Nowadays, one of the most commonly used is Moodle (modular object oriented developmental learning environment), a free learning management system enabling the creation of powerful, flexible and engaging online courses and experiences [4].

The new era of e-Learning services is mainly based on ubiquitous learning, mobile technologies, social networks (communities) and personalized knowledge management. “The convergence of e-Learning and knowledge management fosters a constructive, open, dynamic, interconnected, distributed, adaptive, user friendly, socially concerned, and accessible wealth of knowledge”. The knowledge management tools such as community, social software, peer-to-peer and personalized knowledge management and are now commonly are being used in ubiquitous learning. Learners use these tools to generate and share ideas, explore their thinking, and acquire knowledge from other learners. Learners search and navigate the learning objects in this knowledge filled environment. However, due to the failure of indexing methods to provide the anticipated, ubiquitous learning grid for them, learners often fail to reach their desired learning objects [5].

II. Recommender Systems

There is a need for Personal Recommender Systems in Learning Networks in order to provide learners with advice on the suitable learning activities to follow. Learning Networks target lifelong learners in any learning situation, at all educational levels and in all national contexts. They are community-driven because every member is able to contribute to the learning material. Existing Recommender Systems and recommendation techniques used for consumer products and other contexts are assessed on their suitability for providing navigational support in a Learner Networks.

III. 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.

A. Introduction

Finite mixture models are being increasingly used to model the distributions of a wide variety of random phenomena and to cluster data sets [21]. Here we consider their application in the context of cluster analysis.

We let the *p*-dimensional vector (**y** = ($y_1,...,y_p$)^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 **y** corresponding to the *j* th entity (*j* = 1, . . . , *n*).With the mixture approach to clustering, $\mathbf{y}_1$, . . . , $\mathbf{y}_n$are assumed to be an observed random sample from mixture of a finite number, say *g*, of groups in some unknown proportions π_1, . . . , π_g.

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

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

where the mixing proportions π_1, . . . , $\pi_g$sum to one and the group-conditional density $f_i(y_j;q_i)$is specified up to a vector q_i of unknown parameters (*i* = 1, . . . , *g*). The vector of all the unknown parameters is given by

$$\Psi=\left(p_1,\ldots,p_{g-1},q_1^T,\ldots,q_g^T\right)^T,$$

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

$$t_i(y_j,\Psi)=\frac{p_i f_i(y_j;q_i)}{f(y_j;\Psi)}, \tag{4}$$

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

The parameter vector Ψ can be estimated by maximum likelihood. The maximum likelihood estimate (MLE) of $\Psi,\hat{\Psi}$, 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 _. Solutions of (6) corresponding to local maximizers can be obtained via the expectation–maximization (EM) algorithm [22].

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,

$$f_i(y_j;q_i) = f(y_j;m_i;\Sigma_i) \tag{7}$$

where $f(y_j;m,\Sigma)$ denotes the p-dimensional multivariate normal distribution with mean vector m 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.

B. *Maximum likelihood estimation of normal mixtures*

McLachlan and Peel [21, Chap. 3] described the E- and M-steps of the EM algorithm for the maximum likelihood (ML) estimation of multivariate normal components; see also [23]. 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 Ψ^k. That is, z_{ij} is replaced by $t_{ij}^{(k)}$, which is the posterior probability that y_j belongs to the ith component of the mixture, using the current fit Ψ^k for Ψ (i= 1, . . . , g; j = 1, . . . , n). It can be expressed as

$$t_{ij}^{(k)} = \frac{p_i^{(k)} f\left(y_j; m_i^{(k)}, \Sigma_i^{(k)}\right)}{f\left(y_j; \Psi^{(k)}\right)}. \tag{8}$$

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

$$p_i^{(k+1)} = \sum_{j=1}^{n} t_{ij}^{(k)} / n, \tag{9}$$

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

and

$$\Sigma_i^{(k+1)} = \frac{\sum_{j=1}^{n} t_{ij}^{(k)} \left(y_j - m_i^{(k+1)}\right)\left(y_j - m_i^{(k+1)}\right)^T}{\sum_{j=1}^{n} t_{ij}^{(k)}} \tag{11}$$

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.

C. *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 re sampling approach as in [24]. 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 $d \log(n)$.

IV. **PageRank**

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.

A. *Overview*

PageRank [25] 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 theWeb. 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
[26].

B. 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 [26] 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

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

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.,

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

Let *A* be the adjacency matrix of our graph with

$$A_{ij} = \begin{cases} \frac{1}{o_i} \; if \; (i, j) \in E \\ o \quad \text{otherwise} \end{cases} \qquad (13)$$

We can write the system of n equations with

$$P = A^{T} P \qquad (14)$$

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

```
**PageRank-Iterate**(G)
$P_0 \leftarrow e/n$
$k \leftarrow 1$
**Repeat**
$P_k \leftarrow (1-d)e + dA^{T} P_{k-1};$
$k \leftarrow k+1;$
**Until** $\|P_k - P_{k-1}\|_1 \langle e$
                    Return $P_k$
```

Fig. 4 The power iteration method for PageRank

the largest eigenvalue and the PageRank vector *P* is the principal eigenvector. A well known mathematical technique called power iteration [27] 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:

$$P = (1-d)e + dA^{T} P_{k-1}, \qquad (15)$$

where **e** is a column vector of all 1's. This gives us the PageRank formula for each page *i* :

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

which is equivalent to the formula given in the original PageRank papers [28]:

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

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 [29].

The computation of PageRank values of the Web pages can be done using the power iteration method [27], 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 [25], it is reported that on a database of 322 million links the algorithm converges to an acceptable tolerance in roughly 52 iterations.

C. *Further references on PageRank*

Since PageRank was presented in [28], researchers have proposed many enhancements to the model, alternative models, improvements for its computation, adding the temporal dimension [30], etc. The books by Liu [29] and by Langville and Meyer [31] contain in-depth analyses of PageRank and several other link-based algorithms.

V. **AdaBoost**

A. *Description of the algorithm*

Ensemble learning [32] 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 [33] 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 X denote the instance space and Y the set of class labels. Assume $Y = \{-1,+1\}$. Given a *weak* or *base learning algorithm* and a training set $\{(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)\}$ where $x_i \in X$ and $y_i \in Y$ $(i = 1, \ldots, m)$, the AdaBoost algorithm works as follows. First, it assigns equal weights to all the training examples $(x_i, y_i)(i \in \{1, \ldots, m\})$. 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: $X \to Y$ 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 Ada-Boost.M1 algorithm [33] 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 [34] 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.

B. *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 [35] 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 [33], 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 $D=\{(x_1,y_1),(x_2,y_2),...,(x_m,y_m)\}$;
        Base learning algorithm $L$;
        Number of learning rounds $T$.

Process
$D_1(i)=1/m$.        % Initialize the weight distribution
  For $t=1,...,T$:
$h_t=L(D,D_t)$;     %Train a week learner $h_t$ form D using distribution $D_t$
$e_t=\Pr_{i\sim D_i}[h_t(x_i\neq y_i)]$;     % Measure the error of $h_t$
```

$$a_t = \frac{1}{2}\ln\left(\frac{1\text{-}e_t}{e_t}\right); \qquad \%\text{Determine the Weight of } h_t$$

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

$$= \frac{D_t(i)\exp(-a_t y_i h_t(x_i))}{} \qquad \%\text{update the distribution , where } Z_t \text{ is}$$

$$\%\text{a normalization factor which enables } D_{t+1} \text{ be a distributed}$$

End.

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

Fig. 5 The AdaBoost algorithm

AdaBoost and its variants have been applied to diverse domains with great success. Forexample, Viola and Jones [36] 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 466MHz 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 buzzword in computer vision and many other application areas.

C. *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 over fit, 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. [37]. Schapire et al.[38] presented amargin-based explanation. They argued that AdaBoost is able to increase the margins even after the training error is zero, and thus it does not over fit even after a large number of rounds. However, Breiman [39] indicated that larger margin does not necessarily mean better generalization, which seriously challenged the margin-based explanation. Recently,Reyzin and Schapire [40] 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 [36], 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.

VI. *k*NN: *k*-nearest neighbor classification

A. *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. Amore sophisticated approach, *k*-nearest neighbor (*k*NN) classification [41], 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 $x = (x', y')$, the algorithm computes the distance (or similarity) between *z* and all the training objects *(*x, *y) ∈D* to determine its nearest-neighbor list, D_z. (x is the data of a training object, while *y* is its class. Likewise, 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:

$$\textit{Majority}\ \text{Voting:}\ y' = \underset{v}{\operatorname{argmax}} \sum_{(x_i, y_i) \in D_z} I(v = y_i), \qquad (18)$$

where *v* is a class label, $y_i$is the class label for the *i*th nearest neighbors, and *I (·)* 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 $z = (x', y')$

Process

Computed (x', x), the distance between z and every object, $(x, y) \in D$.
Select $D_z \subseteq D$, the set of k closest training objects to z.

Output: $y' = \underset{v}{\operatorname{argmax}} \sum (x_i, y_i) \in D \; I(v = y_i)$

Fig. 6 The *k*-nearest neighbor classification algorithm

B. 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 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: $w_i = 1/d(x', x_i)^2$. This amounts to replacing the last step of the kNN algorithm with the following:

$$Distance\text{-}\mathrm{Weightted\ Voting:}\ y' = \underset{v}{\operatorname{argmax}} \sum_{(x_i, y_i) \in D_z} w_i \times I(v = y_i).$$
(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 1.8m, 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 [42].
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 [43] 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.

C. *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 [44] shows that 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.

D. *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. Thesis known as ‘condensing’ and can greatly speed up the classification of new objects [46]. In addition, data objects can be removed to improve classification accuracy, a process known as “editing” [47]. 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 [48,49], 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 [50] and the book by Devroyeet al. [51]. Finally, a fuzzy approach to *k*NN can be found in the work of Bezdek [52].

VII. **Naive Bayes**

A. *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 [53].

B. *The basic principle*

For convenience of exposition here, we will assume just two classes, labeled i = 0, 1. Ouraim 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 objectwill belong to class i if we know nothing further about it (the 'prior' probability of class i).This means that the ratio becomes

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

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

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

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

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

If we define $w_j = 1n\left(f(x_j \mid 1) / f(x_j \mid 0)\right)$ and a constant $k = 1n\left(\mathrm{p}(1) / \mathrm{p}(0)\right)$ we see that (22) takes the form of a simple sum

$$1n \frac{P(1 \mid x)}{P(0 \mid x)} = k + \sum_{j=1}^{p} w_{\mathrm{j}} . (23)$$

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.

C. *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*j* th discrete predictor variable, x_j, has $c_r$categories, and if $n_{jr}$of the total of *n* objects fall into the *r* th 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 $\left(n_{jr}+c_r^{-1}\right)/(\mathrm{n}+1)$. 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 naïve Bayes model. Within either (or any, more generally) class, the joint distribution of *x* is

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

and this can be approximated by simplifying the conditional probabilities. The extreme arises with $f(x_i\mid x_1,\ldots,\mathrm{x}_{\mathrm{i}-1})=f(x_i)$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

$$f(x)=f(x_1)f(x_2\mid x_1)f(x_3\mid x_2)\ldots f(x_p\mid x_{p-1}). \quad (25)$$

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 $f(x\mid 1)$by $g(x)\prod_{\mathrm{j=1}}^{\mathrm{p}}\mathrm{h}_1(x_j)$ and $f(x\mid 0)$by $g(x)\prod_{\mathrm{j=1}}^{\mathrm{p}}\mathrm{h}_0(x_j)$, where the function *g(x)* is the same in each model. The ratio is thus

$$\frac{P(1\mid x)}{P(0\mid x)}=\frac{P(1)g(x)\prod_{\mathrm{j=1}}^{\mathrm{p}}\mathrm{h}_1(x_j)}{P(0)g(x)\prod_{\mathrm{j=1}}^{\mathrm{p}}\mathrm{h}_0(x_j)}=\frac{P(1)}{P(0)}\cdot\frac{\prod_{\mathrm{j=1}}^{\mathrm{p}}\mathrm{h}_1(x_j)}{\prod_{\mathrm{j=1}}^{\mathrm{p}}\mathrm{h}_0(x_j)}. \quad (26)$$

Here, the $\mathrm{h}_\mathrm{i}(x_i)$do not even have to be probability density functions—it is sufficient that the $g(x)\prod_{\mathrm{j=1}}^{\mathrm{p}}\mathrm{h}_\mathrm{i}(x_i)$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.

D. *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 [54].

VIII. **Cart**

The 1984 monograph, “CART: Classification and Regression Trees,” co-authored by Leo Breiman, Jerome Friedman, Richard Olshen, and Charles Stone, [55] 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.

A. *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. Thesis in sharp contrast to methods such as C4.5 that generate preferred models on the basis of training data measures.

B. *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 “Ginimeasure of impurity” of a node *t* is

$$G(t)=1-p(t)^2-\left(1-p(t)^2\right),(27)$$

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

$$I(p)=G(p)\ -qG(L)-(1-q)G(R).(28)$$

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 towing rule, which is based on a direct comparison of the target attribute distribution in two child nodes:

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

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-trollablepenalty on splits generating unequal-sized child nodes. (This splitter is a modified version of Messenger and Mandell [56].) They also introduce a variant of the towing split criterion that treats the classes of the target as ordered; ordered towing attempts to ensure target classes represented on the left of a split are ranked below those represented on the right. In our experience the towing 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.

C. *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,

$$\mathrm{N}_1(node)/\,\mathrm{N}_1(root) \rangle\ \mathrm{N}_0(node)/\,\mathrm{N}_0(root). \quad (30)$$

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.

D. *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 [57] 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 amusingness. 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 [58] 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[57] 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 [57].Friedman et al. [59] 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 andZ >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.

E. *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.

F. *Dynamic feature construction*

Friedman [58] 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 discriminate). 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.

G. *Cost-sensitive learning*

Costs are central to statistical decision theory but cost-sensitive learning received only modest attention before Domingos [60]. 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; $C(i, j) = 0$ 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 [61]. 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.

H. *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

$$R_a(T) = R(T) + a|T|, \quad (31)$$

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 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.

I. *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

$$r^*(t) = r(t) + e / (q(t) + S) \quad (32)$$

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 over fit.

Bloch et al. [62] reported very good performance for the Breiman adjustment in a series of empirical experiments.

J. *Theoretical foundations*

The earliest work on decision trees was entirely theoretical. 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.

K. *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) [63] 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 [64].

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.

IX. Conclusion and Future Work

In this paper, we presented the importance of social networks in e-Learning systems. Recommender systems play important roles in e-Learning as they help students to chose among different learning objects to study and activities to participate in. Among the different objects and activities available, recommender systems can chose between different algorithms. Presented algorithms in this paper are: C4.5, K-Means, Support Vector Machine, and Apriori algorithms. Each of those algorithms fit into a certain functionality of the recommender system. Future work will include comparison between other important machine learning algorithms, and our proposed e-Learning model that utilizes different machine learning algorithms for social network supported e-Learning.

References

1. IlariaLiccardi, AsmaOunnas, Reena Pau, Elizabeth Massey, PäiviKinnunen, Sarah Lewthwaite, Marie-Anne Midy, and ChandanSarkar. 2007. The role of social networks in students' learning experiences. SIGCSE Bull. 39, 4 (December 2007), 224-237.

2. Christy M.K. Cheung, Pui-Yee Chiu, Matthew K.O. Lee, Online social networks: Why do students use facebook?, Computers in Human Behavior, Volume 27, Issue 4, July 2011, Pages 1337-1343.

3. J. Bobadilla, F. Serradilla, A. Hernando, MovieLens, Collaborative filtering adapted to recommender systems of e-learning, Knowledge-Based Systems, Volume 22, Issue 4, May 2009, Pages 261-265.

4. Cristóbal Romero, Sebastián Ventura, Enrique García, Data mining in course management systems: Moodle case study and tutorial, Computers & Education, Volume 51, Issue 1, August 2008, Pages 368-384.

5. Adela Lau, Eric Tsui, Knowledge management perspective on e-learning effectiveness, Knowledge-Based Systems, Volume 22, Issue 4, May 2009, Pages 324-325.

6. McLachlan GJ, Peel D (2000) Finite Mixture Models. Wiley, New York.

7. 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.
8. McLachlan GJ, Krishnan T (1997) The EM algorithm and extensions. Wiley, New York.

9. McLachlan GJ (1987) On bootstrapping the likelihood ratio test statistic for the number of components in a normal mixture..Appl Stat 36:318–324.

10. Brin S, Page L (1998) The anatomy of a large-scale hypertextualWeb Search Sngine. Comput Networks30(1–7):107–117.

11. Wasserman S, Raust K (1994) Social network analysis. Cambridge University Press, Cambridge.
12. Golub GH, Van Loan CF (1983) Matrix computations. The Johns Hopkins University Press.

13. Page L, Brin S, Motwami R, Winograd T (1999) The PageRank citation ranking: bringing order to theWeb. Technical Report 1999–0120, Computer Science Department, Stanford University.

14. Liu B (2007) Web data mining: exploring hyperlinks, contents and usage Data. Springer, Heidelberg.

15. 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).

16. Langville AN, Meyer CD (2006) Google's PageRank and beyond: the science of search engine rankings.Princeton University Press, Princeton Dietterich TG (1997) Machine learning: Four current directions. AI Mag 18(4):97–136**.**

17. 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.

18. Schapire RE, Singer Y (1999) Improved boosting algorithms using confidence-rated predictions. Mach Learn 37(3):297–336.

19. Schapire RE (1990) The strength of weak learnability. Mach Learn 5(2):197–227.

20. Viola P, JonesM(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.

21. Friedman J, Hastie T, Tibshirani R (2000) Additive logistic regression: a statistical view of boosting with discussions. Ann Stat 28(2):337–407.

22. 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.
23. Breiman L (1999) Prediction games and arcing classifiers. Neural Comput 11(7):1493–1517.

24. 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.
25. Tan P-N, Steinbach M, Kumar V (2006) Introduction to data mining. Pearson Addison-Wesley.

26. Han E (1999) Text categorization using weight adjusted k-nearest neighbor classification. PhD thesis, University of Minnesota, October 1999.

27. 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).

28. Cover T, Hart P (1967) Nearest neighbor pattern classification. IEEE Trans Inform Theory 13(1):21–27.

29. Kuramochi M, Karypis G (2005) Gene Classification using Expression Profiles: A Feasibility Study. Int J Artif Intell Tools 14(4):641–660.

30. Hart P (1968) The condensed nearest neighbor rule. IEEE Trans Inform Theory 14:515–516.

31. Wilson DL (1972) Asymptotic properties of nearest neighbor rules using edited data. IEEE Trans Syst Man Cyberne 2:408–420.

32. 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.

33. Toussaint GT (2002) Open problems in geometric methods for instance-based learning. JCDCG 273–283.

34. Dasarathy BV (ed) (1991) Nearest neighbor (NN) norms: NN pattern classification techniques. IEEE Computer Society Press.

35. Devroye L, Gyorfi L, Lugosi G (1996) A probabilistic theory of pattern recognition. Springer, New York. ISBN 0-387-94618-7.

36. 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.
37. Hand DJ, Yu K (2001) Idiot's Bayes—not so stupid after all?. Int Stat Rev 69:385–398.

38. 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.

39. Breiman L, Friedman JH, Olshen RA, Stone CJ (1984) Classification and regression trees. Wadsworth, Belmont.

40. Messenger RC, Mandell ML (1972) A model search technique for predictive nominal scale multivariate analysis. J Am Stat Assoc 67:768–772.
41. Quinlan R (1989) Unknown attribute values in induction. In: Proceedings of the sixth international workshop on machine learning, pp. 164–168.

42. Friedman JH, Bentley JL, Finkel RA (1977) An algorithm for finding best matches in logarithmic time. ACMTrans.Math. Software 3, 209. Also available as Stanford Linear Accelerator Center Rep. SIX-PUB- 1549, February 1975.

43. 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.

44. 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.

45. Ting KM (2002) An instance-weighting method to induce cost-sensitive trees. IEEE Trans Knowl Data Eng 14:659–665.

46. Bloch DA, Olshen RA, Walker MG (2002) Risk estimation for classification trees. J Comput Graph Stat 11:263–288.

47. Breiman L (1968) Probability theory. Addison-Wesley, Reading. Republished (1991) in Classics of mathematics. SIAM, Philadelphia.

48. Olshen R (2001) A conversation with Leo Breiman. Stat Sci 16(2):184–198.