Journal of Machine Learning Research 2 (2002) 313-334 Submitted 5/01; Published 2/02

Recommender Systems Using Linear Classifiers

Tong Zhang TZHANG@WATSON.IBM.COM
Vijay S. Iyengar VSI@US.IBM.COM
IBM Research Division, T. J. Watson Research Center
P.O. Box 218, Yorktown Heights, NY 10598, U.S.A.

Editor: Leslie Pack Kaelbling

Abstract

Recommender systems use historical data on user preferences and other available data on users (for example, demographics) and items (for example, taxonomy) to predict items a new user might like. Applications of these methods include recommending items for purchase and personalizing the browsing experience on a web-site. Collaborative filtering methods have focused on using just the history of user preferences to make the recommendations. These methods have been categorized as *memory-based* if they operate over the entire data to make predictions and as *model-based* if they use the data to build a model which is then used for predictions. In this paper, we propose the use of linear classifiers in a model-based recommender system. We compare our method with another model-based method using decision trees and with memory-based methods using data from various domains. Our experimental results indicate that these linear models are well suited for this application. They outperform a commonly proposed memory-based method in accuracy and also have a better tradeoff between off-line and on-line computational requirements.

Keywords: Recommender Systems, Collaborative Filtering, Decision Trees, Linear Models, Unbalanced Data

1. Introduction

Recommender systems use historical data on user preferences and purchases and other available data on users and items to recommend items that might be interesting to a new user. One of the earliest techniques developed for recommendations was based on *nearest-neighbor collaborative filtering* algorithms (Resnick et al., 1994, Shardanand and Maes, 1995) that used just the history of user preferences as input. Sometimes in the literature the term *collaborative filtering* is used to refer to just these methods. However, we will follow the taxonomy introduced by Breese et al. (1998) in which collaborative filtering (*CF*) refers to a broader set of methods that use prior preferences to predict new ones. In this taxonomy, nearest-neighbor collaborative filtering algorithms are categorized as being memory-based *CF*. Nearest-neighbor methods use some notion of similarity between the user for whom predictions are being generated and users in the database. Variations on this notion of similarity and other aspects of memory-based algorithms are discussed by Breese et al. (1998). Scalability is an issue with nearest-neighbor methods. The use of dimension reduction techniques like latent semantic indexing has been proposed to address this issue (Sarwar et al., 2000).

In contrast, model-based *CF* methods use the historical data to build models which are then used for predicting new preferences. A model-based approach using Bayesian networks was found to be comparable to the memory-based approach of Breese et al. (1998). More recently, models based on a newer graphical representation called dependency networks (Hofmann and Tresp, 1997) have been applied to this problem (Heckerman et al., 2000). For this task, dependency network models seem to have slightly poorer accuracy but require significantly less computation when compared to Bayesian network models (Heckerman et al., 2000). Another model-based method uses clustering to group users based on their past preferences. The parameters for this clustering model can be estimated by methods like Gibbs sampling and EM (Ungar and Foster, 1998a,b, Breese et al., 1998). The clustering model explored by Breese et al. (1998) was outperformed by the model-based approach using Bayesian networks and by the memory-based approach CR+ described by Breese et al. (1998).

In this paper, we explore the use of various linear classifiers in a model-based approach to the recommendation task. Linear classifiers have been quite successful in the text classification domain (Joachims, 1998, Yang and Chute, 1994, Zhang and Oles, 2001). Some of the characteristics shared between the text and CF domains include the high dimensionality and sparseness of the data in these domains. The main computational cost of using linear classifiers is in the model building phase, which is an off-line activity. The application of the models is very straightforward especially with sparse data.

Our empirical study will use two data sets that reflect users' browsing behavior and one data set that captures their purchases. Because of its wider applicability, we focus on data that is implicitly gathered, e.g., a boolean flag for each web page representing whether or not it was browsed as in the anonymous-msweb dataset (Blake et al., 1998). This is in contrast with explicitly collected data such as ratings explicitly gotten for movies (Glassman). Section 3 presents results achieved on these datasets by various model-based approaches using linear classifiers. For comparison we also include results achieved by our implementation of the memory-based algorithm CR+ described by Breese et al. (1998) and a model-based approach using decision trees. The linear models studied in this paper are described in the next section.

This paper expands on some early experimental results reported by Iyengar and Zhang (2001). We have derived the algorithms more rigorously and added the naive Bayes model in addition to the regularized linear models considered by Iyengar and Zhang (2001). We have also investigated the differences between the recommendation domain and the text categorization domain. This leads to a study on the impact of loss function on the accuracy of the recommendations. The result of this study gives us some insights on the failure of some linear models in the recommendation domain even though they perform very well in the text categorization domain.

2. Model-Based Approaches

The problem of predicting whether a user (or a customer) will accept a specific recommendation can be modeled as a binary classification problem. However, in a recommender system, we are also interested in the likelihood that a customer will accept a recommendation. This information can be used to rank all of the potential choices according to their likelihoods,

so that we can select the top choices to present to the customer. It is thus necessary that the classifier we use returns a score (or a confidence level), where a higher score corresponds to a higher possibility that the customer will accept the recommendation.

Recommender systems also have characteristics that are similar to those of text categorization. For example, the standard document representation in text categorization is the "bag of words" vector space model. In this model, a text document is represented by a vector of word occurrences in the document. Each vector component represents a word feature, and its value is the number of occurrences of the word in the document. This representation leads to a very large feature space consisting of all possible words in a language. However, since each document to be categorized (such as an e-mail message) is usually relatively short, a vector that represents a document is highly sparse. Therefore text categorization algorithms have to be suitable to problems with large but sparse features.

This characteristic of text categorization is also shared by recommender systems. Typically the set of all possible available merchandises (corresponding to all possible words in text categorization) in a recommender system is very large. However, a customer (corresponding to a document in text categorization) only buys a small number of items. Items that have already been bought by the customer correspond to words appearing in a document. This correspondence leads to a sparse feature representation of the custom profile that is very similar to the vector space model in text categorization. Consequently, it is reasonable to start with the assumption that algorithms that perform well in text categorization may also perform well in the CF domain.

On the other hand, there are also differences between text categorization and CF. There are typically many fewer categories in a text categorization problem than in a recommender system. A text categorization system usually determines whether a document belongs to a certain class or not, while a recommender system provides a ranked list of recommendations indicating the likelihoods of buying certain items. As a result, the evaluation criteria for text categorization and recommender systems are different. In addition, categories in text categorization systems are more likely to be mutually exclusive (that is, a text document usually focuses on a single or very few topics), while a customer may be interested in a relatively large number of different items. Also in a recommender system, the features (items bought) are inter-related to the categories (what item to buy). This is not true in text categorization. A related difference is that when we observe that a person has not bought an item, it can either be the case that the person is not interested in the item, or the case that although the person is interested in the item, (s)he has not yet bought it. Clearly it is difficult to distinguish these two situations — in fact, the purpose of a recommender system is simply to distinguish the two situations. This ambiguity shows that the input to a recommender system is noisy, which makes the CF problem more difficult. On the other hand, this type of noise does not occur in text categorization.

All these differences between the CF problem and the text categorization problem suggest that we may need to modify algorithms used in text categorization. This also suggests that not all algorithms that do well in text categorization will automatically do well in the CF domain. Based on the above discussion, in the following, we shall motivate algorithms considered in this paper from text categorization. However, we also discuss issues specifically related to the CF application as well as necessary modifications of the algorithms.

2.1 Base-line Systems

As a baseline system, we have implemented a version of the nearest neighbor style algorithm, CR+, described by Breese et al. (1998) and included it in our study. As suggested by Breese et al. (1998), inverse user frequency, case amplification and default voting heuristics are used in our implementation of CR+. CR+ achieves very good performance in the recommender system application. As a comparison, it is also known that nearest neighbor algorithms achieve very good performance in text categorization (Yang, 1999). However, the major disadvantage of this method is the large computational and memory complexity. Consequently, simplifying heuristics have to be used to overcome this problem in practical applications. Such simplifications were not addressed in our baseline implementation.

In the recommender system application, interpretability of the model used is a desirable characteristic to be considered in addition to the accuracy achieved and the computational requirements. For example, in cross-sell applications, interpretability allows marketing analysts to review and possibly modify the generated models.

The interpretability property can be satisfied by using a rule-based system, such as rules obtained from a decision tree. We shall thus include a decision-tree-based recommender system in this empirical study as an example of using an interpretable model. In this decision-tree package, the splitting criterion during tree growth is a modified version of entropy and the tree pruning is done using a Bayesian model combination approach originated from data compression (Willems et al., 1995, Zhang, 1998). A similar approach has been suggested by Kearns and Mansour (1998). One useful aspect of our decision tree is that we take advantage of the sparse data representation. The algorithm has low memory and time complexity for sparse data. A standard decision-tree package such as the C4.5 program (Quinlan, 1993) that does not take advantage of sparsity in the data will not be able to handle our problems.

Although a decision-tree rule-based system is efficient and interpretable, it does not provide the best performance in text categorization. Similarly, experiments in this paper show that our decision tree does not provide the best performance as a recommender system. One remedy is to use the so-called "boosting" procedure, which votes on a large number of decision trees. In fact, the best text categorization result on the standard Reuters evaluation dataset is achieved using boosted decision trees (Weiss et al., 1999). Unfortunately, this approach requires a large number of trees (one hundred in Weiss et al., 1999). The resulting system not only loses the interpretability of a single decision tree, but is also computationally too expensive to be practically interesting for this application. We thus do not consider this method in this paper.

It is known that in text categorization, the same level of performance achieved by boosted decision trees can be achieved by computationally more efficient linear classification methods (Dumais et al., 1998, Joachims, 1998, Zhang and Oles, 2001). It is thus natural to consider linear classification in the context of CF.

2.2 Linear Models

We formally define a two-class categorization problem as one to determine a label $y \in \{-1, 1\}$ associated with a vector x of input variables. A useful method for solving this problem is by linear discriminant functions, which consist of linear combinations of the

input variables. Various techniques have been proposed for determining the weight values for linear discriminant classifiers from a training set of labeled data $(x_1, y_1), \ldots, (x_n, y_n)$. Specifically, we seek a weight vector w and a threshold θ such that $w^T x < \theta$ if its label $y = -1$ and $w^T x \geq \theta$ if its label $y = 1$. A score of value $w^T x - \theta$ can be assigned to each data point as a surrogate for the likelihood of x to be in class.

The problem just described may readily be converted into one in which the threshold θ is taken to be zero. One does this by converting a data point x in the original space into $\tilde{x} = [x, 1]$ in the enlarged space. Each hyperplane w in the original space with threshold θ can then be converted into $[w, -\theta]$ that passes through the origin in the enlarged space. Instead of searching for both an d-dimensional weight vector along with a threshold θ, we can search for an $(d + 1)$-dimensional weight vector along with an anticipated threshold of zero. In the following, unless otherwise indicated, we assume that the vectors of input variables have been suitably transformed so that we may take $\theta = 0$. We also assume that x and w are d-dimensional vectors.

2.2.1 Regularized Discriminative Models

Many algorithms have been proposed for linear classification. We start our discussion with the least squares algorithm, which is based on the following formulation to compute a linear separator $\hat{w}$:

$$\hat{w} = \arg\min_w \frac{1}{n} \sum_{i=1}^{n} (w^T x_i - y_i)^2. \tag{1}$$

The least squares method is extensively used in engineering and statistics. Although the method has mainly been associated with regression problems, it can also be used in classification. Examples include use in text categorization (Yang and Chute, 1994) and uses in combination with neural networks (Ripley, 1996).

The solution of (1) is given by

$$\hat{w} = \left(\sum_{i=1}^{n} x_i x_i^T \right)^{-1} \left(\sum_{i=1}^{n} x_i y_i \right).$$

One problem with the above formulation is that the matrix $\sum_{i=1}^{n} x_i x_i^T$ may be singular or ill-conditioned. This occurs, for example, when n is less than the dimension of x. Note that in this case, for any $\hat{w}$, there exist infinitely many solutions $\tilde{w}$ of $\tilde{w}^T x_i = \hat{w}^T x_i$ for $i = 1, \ldots, n$. This implies that (1) has infinitely many possible solutions $\hat{w}$.

A remedy of this problem is to use a pseudo-inverse (Yang and Chute, 1994). However, one problem of the pseudo-inverse approach is its computational complexity. In order to handle large sparse systems, we need to use iterative algorithms which do not rely on matrix factorization techniques. Therefore in this paper, we use the standard ridge regression method (Hoerl and Kennard, 1970) that adds a regularization term to (1):

$$\hat{w} = \arg\min_w \left[\frac{1}{n} \sum_{i=1}^{n} (w^T x_i y_i - 1)^2 + \lambda w^2 \right], \tag{2}$$

where λ is an appropriately chosen regularization parameter. The solution is given by

$$\hat{w} = \left(\sum_{i=1}^{n} x_i x_i^T + \lambda n I \right)^{-1} \left(\sum_{i=1}^{n} x_i y_i \right),$$

where I denotes the identity matrix. Note that $\sum_{i=1}^{n} x_i x_i^T + \lambda n I$ will always be non-singular, which solves the ill-condition problem. The regularized least squares formulation (2) can be solved by Algorithm 2 in Appendix A. In order to avoid cluttering the main text, we leave the algorithms and their justifications to the appendices.

Another popular linear classification method is the support vector machine, which is a method originally proposed by Cortes and Vapnik (1995) that has nice properties from the sample complexity theory. Slightly different from our approach of forcing threshold $\theta = 0$, and then compensating by appending 1 to each data vector, the standard linear support vector machine (Vapnik, 1998) explicitly includes θ in a quadratic formulation as follows:

$$\begin{aligned} &(\hat{w}, \hat{\theta}) = \arg\inf_{w,\theta} \left[\frac{1}{n} \sum_{i=1}^{n} \xi_i + \lambda w^2 \right], \\ &\text{s.t.} \quad y_i(w^T x_i - \theta) \geq 1 - \xi_i, \quad \xi_i \geq 0, \qquad i = 1, \ldots, n. \end{aligned} \tag{3}$$

By eliminating ξ_i, the above formula is equivalent to the following formulation:

$$(\hat{w}, \hat{\theta}) = \arg\inf_{w,\theta} \frac{1}{n} \left[\sum_{i=1}^{n} g(y_i(w^T x_i - \theta)) + \lambda w^2 \right], \tag{4}$$

where

$$g(z) = \begin{cases} 1 - z & \text{if } z \leq 1, \\ 0 & \text{if } z > 1. \end{cases} \tag{5}$$

The support vector machine method (3) has been applied to text categorization (Joachims, 1998, Dumais et al., 1998). It achieves a performance comparable to the much more complex boosted decision trees of Weiss et al. (1999).

It is interesting to compare the least squares approach and the support vector machine approach. In the least squares formulation, the loss function $(z-1)^2$ implies that we try to find a weight w such that $w^T x \approx 1$ for an in-class data point x, and $w^T x \approx -1$ for an out-of-class data point x. Although this means that the formulation attempts to separate the in-class data from the out-of-class data, it also penalizes a well behaved data point x such that $w^T xy > 1$. The support vector machine approach remedies this problem by choosing a loss function that does not penalize a well-behaved data point such that $w^T xy > 1$.

Although a support vector machine in (3) is conceptually appealing for classification problems, it performs poorly for CF. This is rather surprising since SVMs perform very well in text categorization. To understand what causes this failure, we have studied histograms of the linear prediction score $w^T x$ from support vector machines (See Section 3). This study revealed that the problem is caused by extremely unbalanced class distributions typically observed in CF. That is, for many items, there are usually only a small percentage of buyers who will be interested in each of them. From the classification point of view, we can achieve

a high confidence by saying that no buyer is interested in the item, although this forfeits the purpose of using a recommender system.

We observe that a major reason that contributes to support vector machines' susceptibility to unbalanced class distribution is the shape of $g(z)$ in (4). Specifically it does not penalize an error ($w^T xy < 0$) significant enough. Consider the initial point at $\tilde{w} = 0$ and $\tilde{\theta} = -1$. In this case, all in-class data contribute $g(y_i(w^T x_i - \theta)) = 0$ in (4), and all out-of-class data contribute $g(y_i(w^T x_i - \theta)) = 2$ in (4). We now consider a change Δw of $\tilde{w}$ that decreases the objective function in (4). For an in-class data point (x_i, y_i) such that $\Delta w^T x_i < 0$, $g(y_i(w^T x_i - \theta))$ is increased by $-\Delta w^T x_i$. For an out-of-class data point (x_i, y_i), $g(y_i(w^T x_i - \theta))$ is decreased when $\Delta w^T x_i < 0$, but no more than $\Delta w^T x_i$ (which may also be an increase). Clearly if there are significantly more in-class data than out-of-class data, it is difficult to find Δw that can result in a net effect of decreasing the right hand side of (4).

The above analysis shows why the standard support vector machine formulation (3) is susceptible to unbalanced class distribution. The trouble is clearly caused by the fact that $g(z)$ has a constant gradient for $z < 1$. This also suggests a simple remedy: we replace the loss term g by a smooth non-increasing function h so that the gradient magnitude $|h'(z)|$ of $h(z)$ is large when $z < 0$ and small when $z > 0$. In this paper, we consider the function $h(z) = g^2(z)$. Since $h'(1) = 0$, our above analysis suggests that at $\tilde{w} = 0$ and $\tilde{\theta} = -1$, a small change Δw leads to an increase of the objective function in the order of $O(\Delta w^2)$ for an in-class data point, and a potential decrease of the objective function in the order of $O(\Delta w)$ for an out-of-class data point. Therefore with the modified loss function $h(z)$, unbalanced class distribution causes significantly fewer problems:

$$\hat{w} = \arg\min_w \left[\frac{1}{n} \sum_{i=1}^{n} h(w^T x_i y_i) + \lambda w^2 \right], \tag{6}$$

where

$$h(z) = \begin{cases} (1-z)^2 & \text{if } z \le 1, \\ 0 & \text{if } z > 1. \end{cases} \tag{7}$$

Another interpretation of (6) is that it is a modification of the least squares algorithm so that the method does not penalize a data point with $w^T xy > 1$. In this sense, it is a mixture of the least squares method and a standard SVM.[1] We thus call it modified least squares. In addition, it has a loss function that is more smooth than that of an SVM, which makes it numerically simpler to solve. A direct numerical optimization of (6) can be performed relatively efficiently. Similar to (2), Algorithm 3 in Appendix A solves (6).

We would like to mention that there are other ways to deal with unbalanced data. One well-known idea is to over-sample the minority class.[2] Although this method works well for binary classification problems, we find for CF, it is difficult to perform this trick in a consistent way for all items without affecting the relative ranking. We should note that

1. It also belongs to an extended family of support vector machines, although such an extension is recommended against by Vapnik. However, in our case, it performs significantly better than a standard SVM for CF problems due to its ability to handle unbalanced class distribution.
2. Equivalently, one may also consider using different loss functions for in-class and out-of-class data so as to penalize errors for the minority class more.

the class distribution itself does contain important information since even the most naive "Popular" method performs reasonably well despite its simplicity (see Section 3). In this sense, the scheme of using (6) is a much better way to handle the unbalanced problem.

To be consistent with other methods described in this paper, we set $\theta = 0$ in (4). As we have mentioned earlier, the effect of θ will be compensated by appending a constant 1 to each data vector. This modification, given below, does not affect the classification performance:

$$\hat{w} = \arg\inf_{w} \left[\frac{1}{n} \sum_{i=1}^{n} g(w^T x_i y_i) + \lambda w^2 \right]. \tag{8}$$

Due to the non-smoothness of g (it has a discontinuous derivative), it is numerically difficult to solve the optimization problem (8) directly in the primal form. We shall introduce an equivalent dual-form of (8) and develop an algorithm to solve the resulting system in Appendix B. Similarly, a dual form can also be obtained for (6), which can be solved by Algorithm 6 in Appendix B.

2.2.2 Naive Bayes Generative Model

Another very popular linear classification algorithm is naive Bayes. It has been applied to text categorization with reasonable performance, although the performance is significantly worse than that achieved by regularized linear classifiers such as support vector machines. Still it is interesting to apply this method for CF due to its simplicity. It is also very suitable for online updating, which could be important in practice. Our experiments indicate that the naive Bayes model outperforms the decision tree model, but is poorer than the regularized linear classification methods.

In this paper, we adopt the multinomial model described by McCallum and Nigam (1998). Instead of taking $\theta = 0$ by embedding the original data vectors into a space of one larger dimension, in the naive Bayes approach, we explicitly compute θ without doing the data transformation. The linear weight w is given by $w = w^1 - w^{-1}$, and $\theta = \theta^1 - \theta^{-1}$. Denote by $x_{i,j}$ the j-th component of the data vector x_i, then the j-th component w_j^c of w^c ($c = \pm 1$) is given by

$$w_j^c = \log \frac{\lambda + \sum_{i:y_i=c} x_{i,j}}{\lambda d + \sum_{j=1}^{d} \sum_{i:y_i=c} x_{i,j}},$$

and θ^c ($c = \pm 1$) is given by $\theta^c = -\log \frac{|\{i:y_i=c\}|}{n}$.

The parameter $\lambda > 0$ in the above formulation is a smoothing (regularization) parameter. McCallum and Nigam (1998) fixed λ to be 1, which corresponds to the Laplacian smoothing. As we shall see from Section 3, the choice of regularization λ can significantly affect the performance.

There is another difference between the naive Bayes method described above and the standard naive Bayes method used in text categorization (McCallum and Nigam, 1998). A standard naive Bayes method would have computed w_j as w_j^1 and θ as θ^1. That is, it only uses the in-class data. Although this approach is reasonable in text categorization (in that the quantity $\exp((w^1)^T x_i - \theta^1)$ is proportional to the probability of the data belonging to the category), it completely fails in CF. The reason is that text categories tend to be more mutually exclusive, while CF recommendations are not—after all, we want to find

associations among different items. Note that $\exp(w^{1T}x_i - \theta^1)$ does not correspond to the likelihood that we buy a particular item, which causes the failure of the standard naive Bayes. The reason for using $w = w^1 - w^{-1}$ and $\theta = \theta^1 - \theta^{-1}$ is now clear: $1/(1+\exp(-(w^T x_i - \theta)))$ is the conditional probability of buying the current item under the observation x_i.

Due to the above mentioned differences, we call the naive Bayes method used in our study modified naive Bayes.

3. Experiments

The true value of a recommender system can only be measured by controlled experiments with actual users. Such an experiment could measure the improvement achieved by a specific recommendation algorithm when compared to, say, recommending the most popular item. Experiments with historical data have been used to estimate the value of recommender algorithms in the absence of controlled live experiments (Breese et al., 1998, Sarwar et al., 2000). In this paper we will follow experimental procedures similar to those introduced by Breese et al. (1998).

3.1 Data Sets

Characteristics	Dataset		
	msweb	*pageweb*	*wine*
Training cases	32711	9195	13103
Total test cases	5000	1804	2610
Test cases with at least 2 items (*All But 1*)	3453	1243	1770
Test cases with at least 3 items (*Given 2*)	2213	932	1280
Test cases with at least 6 items (*Given 5*)	657	455	624
Test cases with at least 11 items(*Given 10*)	102	168	268
Total items	294	5781	663
Mean items per case in training set	3.02	4.36	4.60

Table 1: Description of the data sets

Characteristics of the data sets used in our experiments are given in Table 1. The first dataset *msweb* was introduced by Breese et al. (1998) and added to the UCI repository under the name *anonymous-msweb*. As described by Breese et al. (1998), this dataset contains for each user the web page groups (called vroots) that were visited in a fixed time period. The total number of items is relatively small (around 300) for this dataset and this can be attributed to the fact that an item refers to a group of web pages.

The second dataset *pageweb* also captures visits by users to a different web site but at the individual page level (with about 6000 total items). Intuitively, one might expect the task of recommending specific pages to be more difficult than that of recommending page groups. But the other factor to be considered is that we also have more fine-grained information at the individual page level about user preferences that can be used by the models. This

dataset will be useful in evaluating how the various algorithms handle recommending from a large number of items.

The third dataset *wine* represents wine purchases made by customers of a leading supermarket chain store within a specified period. The dataset captures for each customer the wines purchased in this period as a binary value (purchased versus not purchased).

We have chosen to use a binary representation of the item/page variables in all the experiments. An alternative representation would be use more information like the number of visits to a web page or the time spent viewing a web page or the quantity of wine purchased.

3.2 Experimental Setup

Following the experimental setup introduced by Breese et al. (1998), the datasets are split into two disjoint sets (training and test) of users. The entire set of visits (or purchases) for users in the training set is available for the model building process. The known visits (purchases) for users in the test set are split into two disjoint sets: given and hidden. The given set is used by the recommender methods to rank all the remaining items in the order of predicted preference to the user. This ranked list is evaluated by using the hidden set as the reference indicating what should have been predicted.

The evaluation metric R, proposed by Breese et al. (1998), is based on the assumption that each successive item in a list is less likely to be interesting to the user with an exponential decay. This metric uses a parameter, α, which is the position in the ranked list which has a 50-50 chance of being considered by the user. Following Breese et al. (1998), we will set α such that the fifth position has a 50-50 chance of being considered.

The exponential decay in interest, which forms the basis for the R-metric, is a plausible behavior model for consumers. However, this metric is not easy to interpret. Also, the number of allowed recommendations can also be constrained by the environment. For example, the form factor of a hand-held device might restrict the number of recommendations on it to a small number. Hence, we will also report results using a simpler metric which measures the fraction of users for whom at least one valid recommendation (according to the hidden set as reference) was given in the top K items of the ranked list. In particular, we will report this metric for K values 1, 3 and 10.

The split of the test set data into the given and hidden sets is done as suggested by Breese et al. (1998). Three of these splits are denoted as *Given 2*, *Given 5*, and *Given 10*. These have 2, 5, and 10 items chosen into the given sets, respectively. The fourth split is denoted as *All But 1* because one item for each test user is randomly chosen to be hidden in this scenario. These scenarios can be used to assess how each recommender system handles different amounts of information being known and to be hidden (predicted) for each test user. Table 1 provides for each dataset the number of test users that are included in each of these scenarios.

Each scenario will be run five times with different random choices for the split between given and hidden subsets in the test data. Mean values and standard deviations are computed over these five experiments. We have adopted this approach to be compatible with the prior literature with regard to the training/test splits. A more traditional approach would have been to use n-fold cross validation where both training and test sets are differ-

ent in the experiments. However, given the compatibility constraints, performing multiple experiments with the given/hidden splits provides some information on the experimental variability.

3.3 Results

The results achieved for all the four scenarios (*Given 2, Given 5, Given 10, All But 1*) are given in Tables 3, 4 and 5 for the datasets *msweb*, *pageweb* and *wine*, respectively. The format in which these results are provided in these tables for each combination of algorithm and scenario is explained in Table 2. The mean and the standard deviation for the R-metric (expressed as percentage) are given on top. The three numbers below indicate the percentage of test users that had at least one successful recommendation in the top 1, 3 and 10 positions of the ranked list. The least squares and the modified least squares (primal and dual) linear models were generated using 25 iterations with the regularization parameter λ set at 0.001. The naive Bayes model used the value of 1 for the smoothing parameter λ (as in McCallum and Nigam, 1998).

R-metric $\pm$ std. dev.		
Success within top 1	Success within top 3	Success within top 10

Table 2: Explanation of the entries in the Tables 3, 4 and 5

The baseline approach of recommending popular items does significantly poorer when compared to the other algorithms on datasets with more items. The decision tree model also exhibits this pattern of not performing as well on datasets with more items.

As mentioned earlier, one advantage of model-based methods is that the model building is done off-line. The model building times for the dataset *msweb* were around 500 seconds for linear least squares and modified linear (primal) models and around 200 seconds for modified linear (dual) model. These times were recorded using our prototype implementation on an IBM RISC System/6000, Model 43P-140 using a 332 MHz PowerPC 604e processor. These model build times can be compared to those reported by Heckerman et al. (2000) for Bayesian networks (144.65 seconds) and for dependency networks (98.31 seconds) on a 300 MHz Pentium system. Applying our linear models to compute the scores is comparable in simplicity to using models like decision trees and rule systems. Hence, recommendations can be generated quickly in our system using these computed scores.

The accuracy achieved on the public dataset *msweb* cannot be directly compared with the results of Breese et al. (1998) and Heckerman et al. (2000) because of random choices made in the given/hidden sets. The results in Tables 3, 4 and 5 suggest that linear least squares and the primal and dual forms of the modified version fare well in comparison with our implementation of CR+. For example, the linear (dual) model is more accurate than CR+ in 11 out of the 12 experimental setups (3 datasets with 4 scenarios in each) using the success in the top 3 metric. If we use the R-metric the linear (dual) model beats CR+ in 8 out of the 12 experimental setups.

algorithm	Given2			Given5			Given10			AllBut1		
Popular	46.5 ± 0.2			43.7 ± 0.5			41.6 ± 2.0			46.5 ± 0.6		
	33.9	55.8	82.0	29.0	55.6	80.6	32.2	57.1	79.8	22.7	38.3	63.8
CR+	56.7 ± 0.1			54.2 ± 0.6			51.5 ± 1.9			60.8 ± 0.6		
	45.0	70.5	88.7	39.9	68.3	88.1	43.7	67.7	87.8	34.6	54.8	76.2
Decision Tree	53.4 ± 0.3			54.3 ± 0.7			53.0 ± 1.0			62.3 ± 0.5		
	46.6	71.3	87.4	48.0	73.9	88.5	51.6	72.9	87.8	38.4	58.4	74.9
Least Squares	55.7 ± 0.3			57.5 ± 0.7			57.0 ± 1.5			64.1 ± 0.5		
	46.9	72.4	89.6	49.9	75.0	90.9	55.5	77.1	91.0	38.5	58.8	79.2
Mod LS Primal	55.6 ± 0.3			57.7 ± 0.8			56.9 ± 1.4			64.4 ± 0.5		
	46.9	72.6	89.8	50.3	75.2	91.0	56.1	76.9	91.8	38.9	59.1	79.6
Mod LS Dual	55.2 ± 0.2			57.5 ± 0.9			56.7 ± 1.4			64.4 ± 0.6		
	46.5	72.9	89.7	50.5	75.6	90.5	57.1	77.3	91.8	39.0	59.0	79.4
Modified Naive Bayes	57.8 ± 0.3			57.0 ± 0.9			52.1 ± 0.8			62.5 ± 0.5		
	48.4	72.0	89.4	48.7	71.8	89.5	49.4	71.8	88.2	37.5	57.1	77.0

Table 3: Results on dataset *msweb*. For explanation of entries refer to Table 2.

algorithm	Given2			Given5			Given10			AllBut1		
Popular	8.3 ± 0.3			7.0 ± 0.3			6.2 ± 0.7			7.6 ± 0.5		
	7.5	14.5	29.5	6.0	12.4	27.9	6.0	11.6	29.5	2.3	5.1	11.1
CR+	29.3 ± 0.8			31.9 ± 1.1			32.5 ± 1.4			33.3 ± 0.3		
	28.8	47.6	67.2	30.2	50.1	68.7	33.5	55.7	74.2	15.2	27.5	44.9
Decision Tree	16.2 ± 0.1			19.9 ± 0.6			23.3 ± 1.2			22.7 ± 0.7		
	23.3	33.5	47.0	28.9	41.6	52.7	36.9	50.7	63.2	13.5	20.2	28.4
Least Squares	27.7 ± 0.3			32.5 ± 0.9			35.4 ± 0.8			34.9 ± 0.6		
	30.1	48.4	66.8	33.7	53.2	71.6	41.7	62.9	77.6	17.1	29.8	46.6
Mod LS Primal	28.3 ± 0.3			33.0 ± 0.9			35.7 ± 1.1			35.5 ± 0.5		
	30.3	48.7	67.7	33.8	53.3	73.4	41.1	61.4	77.9	17.2	30.2	47.4
Mod LS Dual	27.8 ± 0.3			32.9 ± 0.9			35.5 ± 1.2			35.2 ± 0.5		
	29.9	48.7	67.1	34.6	53.4	73.1	41.8	62.3	77.6	17.4	30.0	46.7
Modified Naive Bayes	26.1 ± 0.4			29.1 ± 0.8			30.2 ± 1.3			27.7 ± 0.4		
	25.9	43.4	60.9	28.8	47.4	63.9	32.5	52.5	69.9	12.3	22.3	37.6

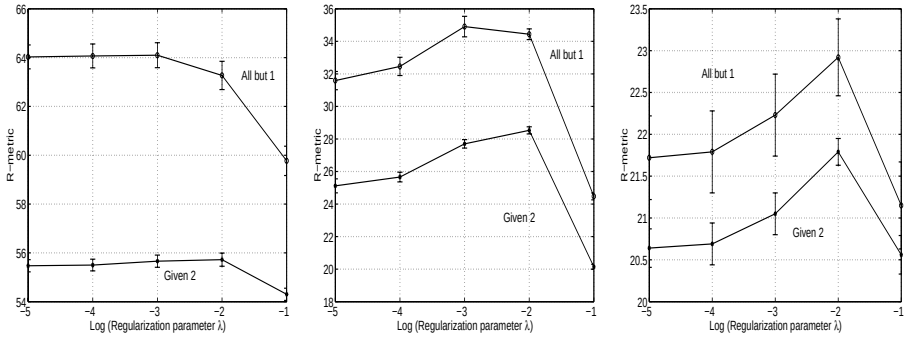
Table 4: Results on dataset *pageweb*. For explanation of entries refer to Table 2.

The impact of the regularization parameter λ on the accuracy of three of the models is shown in Figures 1, 2 and 3. The choice of λ makes a non-negligible difference for all algorithms. The value for this parameter could be chosen using cross-validation experiments with the training data, though this was not done in our study. The figures also suggest

algorithm	Given2			Given5			Given10			AllBut1		
Popular	15.3 ± 0.2			14.5 ± 0.2			14.2 ± 0.4			13.6 ± 0.8		
	10.1	21.7	47.6	6.8	20.4	50.0	5.4	19.9	51.2	5.3	9.2	19.8
CR+	23.7 ± 0.3			24.6 ± 0.4			26.7 ± 0.8			21.4 ± 0.5		
	20.3	37.3	60.3	21.6	38.7	61.6	25.5	42.5	65.3	8.5	16.7	29.8
Decision Tree	16.9 ± 0.4			18.6 ± 0.4			22.1 ± 0.5			17.9 ± 0.4		
	16.8	27.8	49.0	18.9	33.9	52.8	21.4	41.3	60.1	7.6	14.3	24.1
Least Squares	21.1 ± 0.3			24.7 ± 0.4			28.1 ± 0.3			22.2 ± 0.5		
	19.4	35.8	57.1	23.5	43.2	63.8	25.9	46.3	68.1	8.8	17.5	31.3
Mod LS Primal	20.8 ± 0.2			24.4 ± 0.4			27.8 ± 0.3			22.3 ± 0.4		
	19.2	35.7	56.9	23.9	42.8	63.6	25.7	46.9	67.2	8.7	17.5	31.2
Mod LS Dual	19.7 ± 0.2			23.4 ± 0.4			27.1 ± 0.5			21.8 ± 0.4		
	18.0	34.0	55.6	23.1	41.8	62.9	26.6	46.2	66.9	8.6	17.3	30.2
Modified Naive Bayes	24.0 ± 0.3			26.7 ± 0.5			29.0 ± 0.6			22.0 ± 0.4		
	20.1	37.6	59.9	23.7	42.4	65.1	25.5	45.8	68.2	8.3	16.4	31.4

Table 5: Results on dataset *wine*. For explanation of entries refer to Table 2.

that in practice, one may choose a fixed λ with reasonable performance across a number of datasets, without any cross-validation λ selection. We would like to mention that for all algorithms, the value of λ should be same for every potential recommendation item. Otherwise, the computed scores $w^T x$ will not be comparable for different items. A side effect is that we only have a single λ to determine for each algorithm. Therefore a cross-validation procedure can be used to determine this value stably.



msweb *pageweb* *wine*

Figure 1: Linear least squares classifier accuracy vs. regularization parameter λ

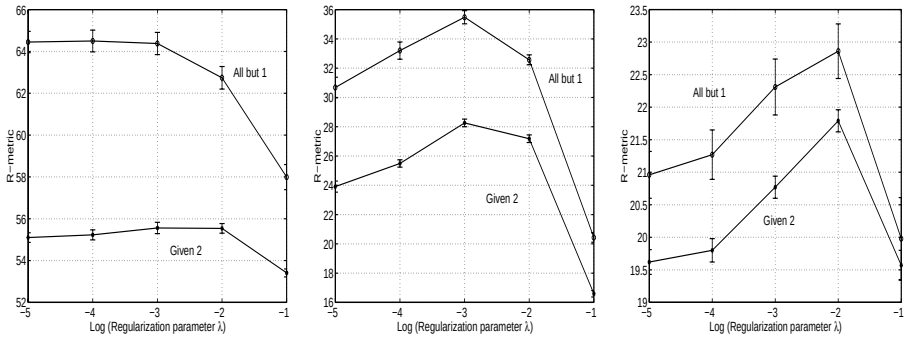


Figure 2: Modified linear least squares (primal) classifier accuracy vs. regularization parameter λ

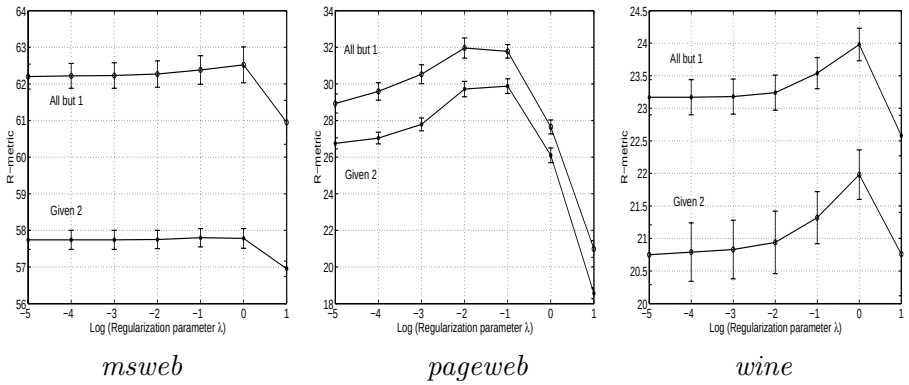


Figure 3: Modified naive Bayes classifier accuracy vs. regularization parameter λ

3.4 Impact Of Loss Function

To study the impact of the loss functions we will consider the classifier performance for one particular item in the *msweb* dataset. The chosen item has an in-class probability of 0.0205. The impact of loss functions on the classifier for this item is shown in Figure 4. Each plot is a histogram of the projected data of $w^T x - \theta$, either for in-class, or for out-of-class, or for combined in-class and out-of-class data. The top row employs formulation (6) with loss function $h(z)$. The middle row employs formulation (8) with loss function $g(z)$. The bottom row employs a balancing technique in (8), where we duplicate each in-class data point five times, which gives an effective 10% in-class population.

Clearly, from the results, we note that without balancing (i.e., over sampling the minority in-class data), the SVM formulation (8) performs very poorly since it does not separate the in-class data from the out-of-class data at all. However, with balancing, it correctly classifies

some in-class data, but also misclassified some out-of-class data. In this experiment, we oversample the in-class data five times. However, it is not clearly what is the best trade-off. We can also see that the modified least squares formulation (6) manages to partially separate in-class data from out-of-class data even without balancing. Although the binary classification error is still poor since the majority of in-class data are centered around $w^T x - \theta \approx -0.5$, the resulting partial class separation is sufficient to yield useful ranking information when we compare different items. Table 6 compares the performance of the SVM and the modified least squares (dual) formulations on the *msweb* dataset.

algorithm	Given2			Given5			Given10			AllBut1		
Mod LS	55.2 ± 0.2			57.5 ± 0.9			56.7 ± 1.4			64.4 ± 0.6		
Dual	46.5	72.9	89.7	50.5	75.6	90.5	57.1	77.3	91.8	39.0	59.0	79.4
SVM	33.6 ± 0.4			42.1 ± 0.9			43.6 ± 0.8			46.2 ± 0.3		
	41.3	53.9	66.0	44.6	65.0	76.8	48.8	66.3	81.0	32.7	42.5	53.7

Table 6: Comparing modified least squares (dual) and SVM on dataset *msweb*. For explanation of entries refer to Table 2.

In addition, the histogram plots can also be used to partially explain why in this particular application, the standard least squares method does as well as the more complicated modified least squares method. We simply notice that histograms of projections resulted from (6) are concentrated in the interval $[-1, 1]$. Consequently there is virtually no difference between the standard least squares loss and the modified least squares loss. However, for other applications such as text categorization, the data projection is typically not contained in $[-1, 1]$. Consequently, the modified least squares method performs better than the standard least squares method in those applications.

4. Conclusion

This paper presents a model-based approach to recommender systems using linear classification models. We have explored various linear formulations and the corresponding algorithms for building the models. Experiments are performed with three datasets and recommendation accuracies compared using two different metrics. The experiments indicate that the linear models are more accurate than a memory-based collaborative filtering approach reported earlier. This improved accuracy in combination with the better computational characteristics makes these linear models very attractive for this application.

Acknowledgments

We would like to thank Murray Campbell, Richard Lawrence and George Almasi for their help during this work.

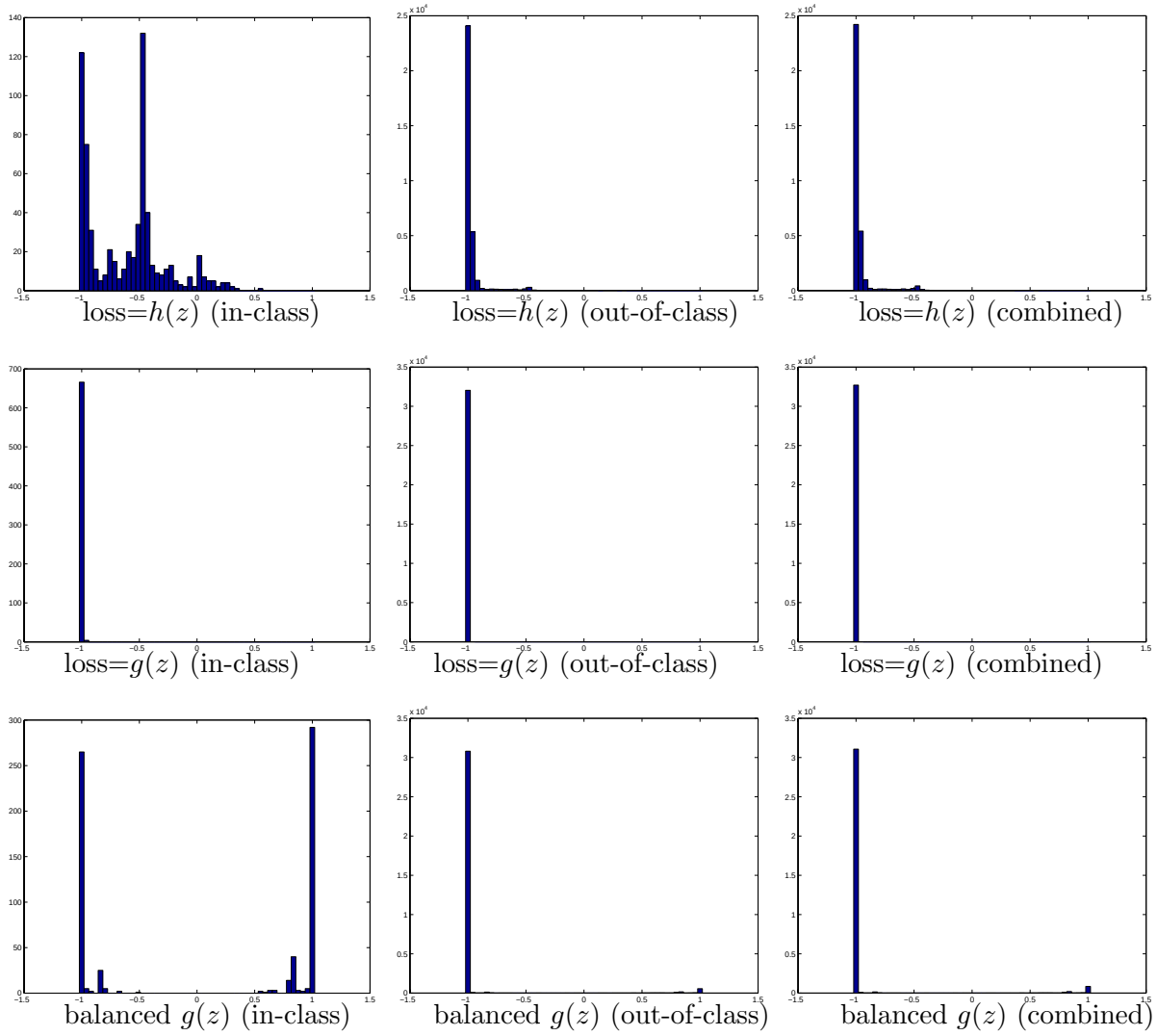


Figure 4: Impact of different loss functions (2% in-class population)

Appendix A. Primal Algorithms

Algorithms considered here are modified from those of Zhang and Oles (2001) for text categorization. We include them here for completeness. We consider a more general formulation

$$\hat{w} = \arg\inf_w \left[\frac{1}{n} \sum_{i=1}^{n} f(w^T x_i y_i) + \lambda w^2 \right], \tag{9}$$

where f is a relatively smooth function which has a continuous first order derivative and a non-negative piecewise continuous second order derivative. In this case, the formulation in (9) is convex, thus it has a unique local minimum which is also the global minimum. Methods investigated in this section are based on the following generic relaxation algorithm (see Golub and Van Loan, 1996):

Algorithm 1 *(Primal Gauss-Seidel)*

let $w = 0$ and $r_i = w^T x_i$
for $k = 1, 2, \ldots$
  for $j = 1, \ldots, d$
    find Δw_j by approximately minimizing
      $\frac{1}{n} \sum_i f(r_i + \Delta w_j x_{ij} y_i) + \lambda (w_j + \Delta w_j)^2$ $\qquad (*)$
    update r: $r_i = r_i + \Delta w_j x_{ij} y_i$ $\qquad (i = 1, \ldots, n)$
    update w: $w_j = w_j + \Delta w_j$
  end
end

The above relaxation method is often called the Gauss-Seidel procedure in numerical optimization. The algorithm cycles through components of w, and optimizes one component at a time (while keeping others fixed). For our problems, this method converges under quite moderate conditions as long as we reduce the objective function value in $(*)$ at each step.

We can now apply the above procedure to the regularized linear least squares fit formulation (2), and obtain Algorithm 2 where $(*)$ is solved exactly.

Algorithm 2 *(Least Squares Primal)*

let $w = 0$ and $r_i = 0$
for $k = 1, 2, \ldots$
  for $j = 1, \ldots, d$
    $\Delta w_j = -\left(\sum_i (r_i - 1) x_{ij} y_i + \lambda n w_j\right) / \left(\sum_i x_{ij}^2 + \lambda n\right)$
    update r: $r_i = r_i + \Delta w_j x_{ij} y_i$ $\qquad (i = 1, \ldots, n)$
    update w: $w_j = w_j + \Delta w_j$
  end
end

To apply Algorithm 1 to (6), it is helpful to further enhance the smoothness of h by introducing a continuation parameter $c \in [0,1]$, so that $h(x) = h_0(x)$ and the smoothness of $h_c''(x)$ decreases as c decreases:

$$h_c(x) = \begin{cases} (x-1)^2 & x \le 1 \\ c(x-1)^2 & x > 1. \end{cases}$$

Algorithm 1 should then be modified so that at each step k, a different c_k is chosen (so that $1 = c_1 \ge c_2 \ge \cdots \ge c_K = 0$), and the function f shall be replaced by f_{c_k}. Note that this introduction of a continuation parameter is not required in order for Algorithm 1 to converge. However, in our experience, this simple modification accelerates the rate of convergence. In the following, we compute a Newton update to approximately solve $(*)$. However, to make the method more robust, we use a step size Δw_j in $(*)$ that is half of the computed Newton update.

Algorithm 3 *(Modified Least Squares Primal)*

let $w = 0$ and $r_i = 0$
pick a decreasing sequence of $1 = c_1 \ge c_2 \ge \cdots \ge c_K = 0$
for $k = 1, 2, \ldots, K$
  define function $C_k(r_i) = 1$ if $r_i \le 1$ and $C_k(r_i) = c_k$ otherwise
  for $j = 1, \ldots, d$
    $\Delta w_j = -0.5 \left[\sum_i C_k(r_i)(r_i - 1)x_{ij}y_i + \lambda n w_j\right] / \left[\sum_i C_k(r_i)x_{ij}^2 + \lambda n\right]$
    update r: $r_i = r_i + \Delta w_j x_{ij} y_i \qquad (i = 1, \ldots, n)$
    update w: $w_j = w_j + \Delta w_j$
  end
end

Appendix B. Dual Algorithms

As we have mentioned, it is inappropriate to solve (8) directly using the primal Gauss-Seidel method in Appendix A, due to the non-smoothness of h. We need to introduce a dual form of (9) and use a dual form of the Gauss-Seidel method to solve the resulting system.

To obtain a dual form of (9), we consider an auxiliary variable ζ_i for each data point x_i:

$$(\hat{w}, \hat{\zeta}) = \arg \inf_w \sup_\zeta \left[\frac{1}{n} \sum_{i=1}^n [-k(\zeta_i) + \zeta_i w^T x_i y_i] + \lambda w^2 \right],$$

where $k(\cdot)$ is the Legendre transform of $f(\cdot)$: $k(v) = \sup_u(uv - f(u))$. It is well known that k is convex. By switching the order of $\inf_w$ and $\sup_\zeta$, which is valid for the above minimax convex programming problem (see Zhang and Oles, 2001, for proof), we obtain

$$\hat{\zeta} = \arg \sup_\zeta \left[\frac{1}{n} \sum_{i=1}^n [-k(\zeta_i) + \zeta_i w^T x_i y_i] + \lambda w^2 \right],$$

where w is minimized at $w = -\frac{1}{2\lambda n}\sum_i \zeta_i x_i y_i$. Substituting into the above equation, we obtain

$$\hat{\zeta} = \arg\inf_{\zeta}\left[\sum_{i=1}^{n} k(\zeta_i) + \frac{1}{4\lambda n}(\sum_{i=1}^{n}\zeta_i x_i y_i)^2\right], \tag{10}$$

and

$$\hat{w} = -\frac{1}{2\lambda n}\sum_i \hat{\zeta}_i x_i y_i.$$

Similar to Algorithm 1 which solves the primal problem (9), the following generic relaxation algorithm solves the dual problem (10):

Algorithm 4 *(Dual Gauss-Seidel)*

let $\zeta = 0$ and $v_j = 0$ for $j = 1, \ldots, d$
for $k = 1, 2, \ldots$
  for $i = 1, \ldots, n$
    find $\Delta\zeta_i$ by approximately minimizing
      $k(\zeta_i + \Delta\zeta_i) + \frac{1}{4\lambda n}(2\Delta\zeta_i v^T x_i y_i + \Delta\zeta_i^2 x_i^2)$ $\quad (**)$
    update v: $v_j = v_j + \Delta\zeta_i x_{ij} y_i$ $\quad (j = 1, \ldots, d)$
    update ζ: $\zeta_i = \zeta_i + \Delta\zeta_i$
  end
end
let $w = -\frac{1}{2\lambda n} v$.

An important implementation issue for Algorithm 4 is that the data ordering can have a significant effect on the rate of convergence (the feature ordering does not appear to have a very noticeable effect on Algorithm 4). More specifically, if we order the data points such that members in each class are grouped together, then we may experience a very slow convergence. On the other hand, the dual algorithm appears to work very well with a random data ordering.

For the modified SVM formulation in (8), we obtain

$$k(z) = \begin{cases} z & -1 \le z \le 0, \\ +\infty & \text{otherwise.} \end{cases}$$

The $+\infty$ value is effectively a simple constraint on each dual variable: $-1 \le z \le 0$. Algorithm 4 can be applied to this formulation. The optimization step $(**)$ in Algorithm 4 can be solved exactly, and this leads to the following method:

Algorithm 5 *(Modified SVM Dual)*

let $\zeta = 0$ and $v_j = 0$ for $j = 1, \ldots, d$
for $k = 1, 2, \ldots$
  for $i = 1, \ldots, n$
    $\Delta\zeta_i = \min\left(-\zeta_i, \max\left(-1 - \zeta_i, -\frac{2\lambda n + v^T x_i y_i}{x_i^2}\right)\right)$
    update v: $v_j = v_j + \Delta\zeta_i x_{ij} y_i \qquad (j = 1, \ldots, d)$
    update ζ: $\zeta_i = \zeta_i + \Delta\zeta_i$
  end
end
let $w = -\frac{1}{2\lambda n} v$.

For the modified least squares formulation (6), $k(z)$ is defined only for $z \leq 0$ ($k(z) = +\infty$ for $z > 0$): $k(z) = z^2/4 + z$. We thus obtain an instance of Algorithm 4 with $(**)$ solved exactly:

Algorithm 6 *(Modified Least Squares Dual)*

let $\zeta = 0$ and $v_j = 0$ for $j = 1, \ldots, d$
for $k = 1, 2, \ldots$
  for $i = 1, \ldots, n$
    $\Delta\zeta_i = \min\left(-\zeta_i, -\frac{(2+\zeta_i)\lambda n + v^T x_i y_i}{\lambda n + x_i^2}\right)$
    update v: $v_j = v_j + \Delta\zeta_i x_{ij} y_i \qquad (j = 1, \ldots, d)$
    update ζ: $\zeta_i = \zeta_i + \Delta\zeta_i$
  end
end
let $w = -\frac{1}{2\lambda n} v$.

References

C. Blake, E. Keogh, and C.J. Merz. UCI repository of machine learning databases. University of California, Irvine, Dept. of Information and Computer Science, URL=http://www.ics.uci.edu/∼mlearn/MLRespository.html, 1998.

J.S. Breese, D. Heckerman, and C. Kadie. Emperical analysis of predictive algorithms for collaborative filtering. In *Proceedings of Fourteenth Conference on Uncertainty in Artificial Intelligence*. Morgan Kaufmann, 1998.

C. Cortes and V.N. Vapnik. Support vector networks. *Machine Learning*, 20:273–297, 1995.

S. Dumais, J. Platt, D. Heckerman, and M. Sahami. Inductive learning algorithms and representations for text categorization. In *Proceedings of the 1998 ACM 7th International Conference on Information and Knowledge Management*, pages 148–155, 1998.

S. Glassman. Eachmovie data set. URL=http://research.compaq.com/SRC-/eachmovie/.

G.H. Golub and C.F. Van Loan. *Matrix computations*. Johns Hopkins University Press, Baltimore, MD, third edition, 1996.

David Heckerman, David Maxwell Chickering, Christopher Meek, Robert Rounthwaite, and Carl Kadie. Dependency networks for inference, collaborative filtering, and data visualization. *Journal of Machine Learning Research*, 1:49–75, 2000.

A. E. Hoerl and R. W. Kennard. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12(1):55–67, 1970.

R. Hofmann and V. Tresp. Nonlinear markov networks for continuous variables. In *Advances in Neural Information Processing Systems 9, editors: M. Mozer, M. Jordan and T. Petsche*. MIT Press, 1997.

V.S. Iyengar and T. Zhang. Empirical study of recommender systems using linear classifiers. In *Proceedings of the Fifth Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 16–27, 2001.

T. Joachims. Text categorization with support vector machines: learning with many relevant features. In *European Conference on Machine Learing, ECML-98*, pages 137–142, 1998.

M. Kearns and Y. Mansour. A fast, bottom-up decision tree pruning algorithm with near optimal generalization. In *Proceedings of the 15th International Conference on Machine Learning*, pages 269–277, 1998.

Andrew McCallum and Kamal Nigam. A comparison of event models for naive bayes text classification. In *AAAI/ICML-98 Workshop on Learning for Text Categorization*, pages 41–48, 1998.

J. Quinlan. *C4.5 programs for machine learning*. Morgan Kaufmann Publishers, 1993.

P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl. Grouplens: An open architecture for collaborative filtering of netnews. In *Proceedings of CSCW*, 1994.

B.D. Ripley. *Pattern recognition and neural networks*. Cambridge University Press, 1996.

B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Analysis of recommendation algorithms for e-commerce. In *Proceedings ACM E-Commerce 2000 Conference*, 2000.

U. Shardanand and P. Maes. Social information filtering: Algorithms for automating word of mouth. In *Proceedings of CHI'95*, 1995.

L. Ungar and D. Foster. Clustering methods for collaborative filtering. In *Workshop on Recommendation Systems at the Fifteenth National Conference on AI*, 1998a.

L. Ungar and D. Foster. A formal statistical approach to collaborative filtering. In *CONALD'98*, 1998b.

V.N. Vapnik. *Statistical learning theory*. John Wiley & Sons, New York, 1998.

S.M. Weiss, C. Apte, F. Damerau, D.E. Johnson, F. Oles, T. Goetz, and T. Hampp. Maximizing text-mining performance. *IEEE Intelligent Systems and their applications*, 14(4): 63–69, July/August 1999.

F.M.J. Willems, Y.M. Shtarkov, and T.J. Tjalkens. The context tree weighting method: basic properties. *IEEE Trans. on Inform. Theory*, 41(3):653–664, 1995.

Yiming Yang. An evaluation of statistical approaches to text categorization. *Information Retrieval*, 1:69–90, 1999.

Yiming Yang and Christopher G. Chute. An example-based mapping method for text categorization and retrieval. *ACM Transactions on Information Systems*, 12:252–277, 1994.

Tong Zhang. Compression by model combination. In *Proceedings of IEEE Data Compression Conference, DCC'98*, pages 319–328, 1998.

Tong Zhang and Frank J. Oles. Text categorization based on regularized linear classification methods. *Information Retrieval*, 4:5–31, 2001.