Supervised feature extraction for text categorization

Jakob Verbeek

Supervised Feature Extraction for Text Categorization

J.J. Verbeek*†
JVERBEEK@SCIENCE.UVA.NL

Abstract

This paper concerns finding the 'optimal' number of *word groups* for text classification. We present a method to select *which* words to cluster into word groups and *how many* such word groups to use on the basis of a set of pre-classified texts. The method involves a 'greedy' search through the space of possible word groups. The words are grouped according to the 'Jensen-Shannon divergence' between the corresponding distributions over the classes. The criterion to decide *which number* of word groups to use is based on Rissanen's MDL Principle. We present empirical results that indicate that the proposed method performs well. Furthermore, the proposed method outperforms cross-validation in the sense that far fewer word groups are selected while prediction accuracy is just slightly worse. For the experimentation we used a subset of the '20 NEWSGROUP' dataset [10].

keywords: clustering, feature extraction, MDL, naive Bayes, supervised, text categorization

1 Introduction

A problem in the field of text categorization is that of the huge dimensionality of the data when documents are represented by a vector that indicates how often each word occurs in the document. This 'curse of dimensionality' is a well-known phenomenon in pattern recognition problems. As a consequence of the huge dimensionality of the feature space, data sets are often relatively sparse in this space. Sensitivity to noise and high space- and time complexity (both for learning and using the classifier) are a result. Therefore it makes sense to look for reductions of (the dimensions of) the feature space. According to [17, 9] accurate text classifiers *can* be learned even when using rather large feature sets using in the order of several thousands of features. The main benefit of the more 'aggressive' feature reduction (extracting far less features than the several thousands mentioned before) discussed here is therefore the reduction of time and space requirements for the classification process itself (needed in cases where large amounts, or streams, of documents have to be classified).

Several feature *selection* and *extraction* methods have been proposed in the literature. Feature *selection* merely selects a 'good' subset of the original features, whereas feature *extraction* allows for arbitrary new features based on the original ones. Also, specific methods have been developed for cases dealing with textual data. Both supervised and unsupervised methods have been proposed. Supervised methods use a set of pre-classified documents and take into account the labeling of this data (each text belongs to exactly one of a finite number of 'classes', and is 'labeled' to indicate this class). Contrary, unsupervised methods, like Latent Semantic Indexing (LSI) [6], do not take into account such a labeling.

One idea for supervised feature extraction would be to group words together that distribute more or less the same over the classes. Recently a method to find word groups based on this idea was introduced [2, 14]. However, *how many* such word groups to use remained an open question. In the latter publications it is suggested to use cross-validation or to set a bound in terms of mutual information or prediction accuracy to determine the number of word groups. In this paper we propose to use Rissanen's Minimum Description Length (MDL) principle [13] to select the number of word groups.

In the literature several publications [4, 5, 12] report on feature *selection* using the MDL principle to determine the number of features. These papers all concern feature *selection* where we are here concerned with feature *extraction*.

*Intelligent Autonomous Systems Group, Computer Science Institute, Faculty of Science, University of Amsterdam, Kruislaan 403, 1098 SJ Amsterdam, The Netherlands

†In this work the author has been partially supported by the Dutch Foundation of Technical Science STW project AIF 4997. Also, the author thanks Nikos Vlassis and Michiel van Lambalgen for their help.

The rest of this document is organized as follows: First, we give a short introduction to the MDL principle. In Section 3 we discuss the Naive Bayes model we used for text classification. Section 4 deals with applying the MDL principle to the problem at hand. In the subsequent section we discuss a method to decide which word groups we merge. Then, in Section 6, we present the implemented algorithm. Finally, in Section 7, we present our experimental results and the conclusions we draw from them.

2 The Minimum Description Length principle

In this section we give an informal introduction to the Minimum Description Length (MDL) principle. Due to limited space we can not go into technical details here.

Rissanen's MDL [13] principle is one of the many known 'model selection methods'. A *model selection method* is a method to choose among possible models (which may differ in type and 'complexity') to explain some given data. Cross-validation is probably the simplest and best known method, it just selects the model that performs best on an independent test set of data that was not used to fit the models. MDL, however, uses *all* available data for learning and compares models on the basis of a sum of two terms. One term represents how well the model fits the data, the other term represents the 'complexity' of the model. The complexity of a model can be thought of as the number of free parameters in the model. MDL tries to find a good trade-off between complexity and fit. Model selection methods making this trade-off are said to be *penalty based.*

In general, if we want to describe a given body of data, we may do so using a model; the model contains some regularities of the data and we specify how the actual data differs from the model. The MDL principle exploits this idea explicitly. One can encode the data, D, as follows: first encode a model, H, resulting in a code of $L(H)$ bits and next encode the data D *using* the model H with a code of length $L(D|H)$ bits. This results in a total code length given by:

$$L_H(D) = L(H) + L(D|H) \tag{1}$$

The first term in the sum is called the *complexity term* or *penalty term*, the second is called the *error term.* Simple models can be encoded using few bits and for complex models we need many bits. Furthermore, if a model fits the data very well, we need few bits to encode the data using the model. If the model has a bad fit on the data we need many bits to encode the data using the model. The codes used should be prefix codes [3]. As the name already suggests: MDL selects the model, H, with the *minimal total code length* or equivalently the model minimizing equation (1) or equivalently yielding *maximal data compression.* The idea is that this model captures the most regularities of the data.

3 The Text classification model

In the previous section we discussed the principle with which we will choose among possible models. In this section we present the models themselves.

In text classification we want to predict for a document the corresponding class label. It is a common practice in text classification to represent texts as a *bag of words*. The text is treated as a list of word frequencies, therefore the information of the position of the words in the text is discarded. We adopt this practice here too.

We predict the class for a text using a probabilistic model and then simply pick the class with the highest probability. So, if d is a text and $c \in C$ is a class from the set C of classes, we need to compute for each class the probability $p(c|d)$. We do so by computing $p(d|c)$ and using Bayes' Rule to link them together:

$$p(c|d) = \frac{p(c)p(d|c)}{p(d)} \tag{2}$$

We set $p(d|c) = p(|d|)p(d|c,|d|)$ with $|d|$ the length of document d, and let $p(d) = \sum_{c \in C} p(c)p(d|c)$. Since $p(|d|)$ gets divided away in equation (2) there is no need here to define it explicitly and we assume it implicitly throughout. Furthermore, we make an independence assumption between the words in a text. Let w_k be the k^{th} word in the text d, then we set:

$$p(d|c) = \prod_{k=1}^{|d|} p(w_k|c)$$

This is a quite unrealistic assumption. However, we need the assumption to make the number of probabilities we have to estimate manageable. It turns out ([1], page 25) that none of the known text classification approaches shows this assumption to be degrading performance considerably. The name of this type of model, Naive Bayes (NB), reflects the naiveness of this independence assumption. See [7] for a discussion on why the NB assumption works well in many practical applications.

The $p(w|c)$ can be estimated directly from the set of training documents. Take $p(w|c)$ as the fraction of all the words observed in texts from class c being w.[1] The $p(c)$ are directly estimated from the training data, $p(c)$ is just the proportion of the training texts having label c.

Grouping words Up till now we have talked about 'grouping' words in a rather vague way. It is simply regarding the words in a word group as being identical. So the number of occurrences of a word group is the sum of the number of occurrences of its members. Suppose we cluster the words w_s and w_t into the word group w. Then it follows that

$$p(c, w) = p(c, w_t) + p(c, w_s)$$

This fully determines the new distribution.

4 Applying the MDL principle

In this section we discuss in detail the used coding schemes. First, we discuss how we encode the models. Second, the encoding of the class labels *using* a model is discussed.

Encoding models A model consists of a clustering of all n words into k word groups, together with a joint distribution over the classes and word groups. Therefore, we split the code in two: First, we encode which clustering is used and next we encode the corresponding class-distributions. To encode which clustering is used, we code the index in some agreed enumeration of all possible clusterings. To compute the number of bits needed to code the index of the used clustering we need to know how many possible clusterings of n words into k groups there are. This number is known as the Stirling number of the second kind $\{{n \atop k}\}$ with $(n \geq k)$ ([8], Section 6.1). To encode a clustering, we first encode the value of k (now the decoder can compute how many bits will follow to encode the index, assuming n is known) and next we encode the index itself in $\log \{{n \atop k}\}$ bits.

Next, we encode the distribution. It turns out [16] that if we use the NB model, it suffices to have $p(c)$ and $p(c|w)$ to compute $p(c|d)$. Furthermore, the encoding of the distribution $p(c)$ over the classes will take the same number of bits for all models. Therefore, the contribution to the total code length of the encoding of $p(c)$ is ignored in finding the model yielding minimum description length.

Note that the distributions $p(c|w)$ are induced by frequency lists we obtained from the sample. Therefore, we can also encode these frequency lists instead. Now, we need to encode a sequence, of known length, of integers. This can be done using a standard way to encode integers. However, this code scheme is sub-optimal here. Below we describe a more efficient coding scheme in four steps:

1. **Scaling** Scale all rows in the original word-class frequency table (there is a row for each word group and a column for each class) to sum up to some fixed integer $\rho \geq 1$. So, ρ determines the *precision* with which we will approximate the original distributions over the classes for each word group. We will still use only integers to represent the distribution.

2. **Pruning** We can use the fact that all rows in the scaled table sum up to ρ. We know that all other entries on a certain row are zero, if ρ is the sum of all entries up that point and we know ρ. In the same manner, we always know the value of the last entry on a row if we have seen the other entries. We do not have to encode the values of these entries that can be derived from previous entries. We can 'prune' the table to exclude those entries. Say that the pruned table has l entries.

3. **Histogram** We make a histogram of all *values* occurring in the pruned table. This histogram can be seen as an empirical distribution q on the set $\{0, 1, \ldots, \rho\}$. We encode q using the standard coding scheme for integers.

4. **Simple encoding** Now, we approximate the code length needed to encode the l *values* in the pruned table by $l\mathcal{H}(q)$.[2] See ([3], Section 5.4) for a justification of this approximation.

[1] We use Laplace Smoothing ([3],Section 7.10) on these distributions to avoid probabilities of zero or one.

[2] $\mathcal{H}(\cdot)$ denotes the entropy function.

It turns out that this coding scheme produces code words much shorter than those from the standard code scheme. Empirically we found that a value of $\rho = 25$ is (close to) optimal. In our experimentation, prediction performance was reduced by less than one percent if we used $\rho = 25$. For smaller ρ, performance started to drop significantly.

Encoding labels using a model Since we are using a 'probabilistic model' we can use the Kraft Inequality [3], which states that each probability distribution defines a prefix code and vice versa, to arrive at the code length needed to encode the labels. This way, we obtain code lengths given by $-\log p(c|d)$ to encode the class label c of text d.

The probabilities the model assigns to the classes differ heavily, this is due to the structure of our model. Therefore, if the model makes a wrong prediction, the -log of the probability of the correct class can be a rather large number. As a result, the curve of the code length needed to encode the class labels as a function of the number of word groups used can behave rather wildly, i.e. not smooth at all. This in turn, may result in bad performance of the algorithm. Below, we discuss an alternative coding scheme that yields a smoother curve for the code length. Instead of the log-loss, the alternative encoding implements a loss resembling the zero-one loss, which seems more appropriate for classification problems.

The alternative consists simply of encoding for which of the m training texts the model did not predict the correct class and specifying for those texts what the correct class is. Say that a fraction ϵ of the documents was misclassified by the model. We encode the correct classes by first encoding the number $m\epsilon$ of misclassified texts. Next we encode *which* $m\epsilon$ labels were predicted incorrectly, note that there are $\binom{m}{m\epsilon}$ possibilities. We do so by means of specifying an index (using $\log \binom{m}{m\epsilon}$ bits) in an enumeration of these possibilities. Next we encode the correct class labels using $m\epsilon \log(|C| - 1)$ bits[3]. We refer to this coding scheme as 'coding exceptions'.

It is easy to show [16] that this encoding yields exactly the same code lengths as when we use the following method: We assign to the class predicted by the model (i.e. the one that was assigned highest probability) probability $1 - \epsilon$ and distribute the remaining probability mass uniformly over the remaining $|C| - 1$ classes. This is intuitively characterized as:

> *"If we know our predictions were correct* p *percent of the previous cases, then we assign* p *percent of confidence to our new predictions."*

5 A distance measure between word groups

Since there are in general many ways to cluster n words into k word groups, we cannot consider all possible clusterings of the words.[4] Therefore, we perform a greedy search through the space of all clusterings. At each step of the algorithm the two 'closest' (in a sense we will discuss below) word groups are merged together. The algorithm starts with each word being a word group.

The algorithm we propose here uses the same criterion as proposed in [2, 14] to decide which word groups we will merge. We will call this quantity the *merge cost*. At each step of the algorithm the two word groups with the minimum merge cost are merged. Since we want to retain good predictive performance while merging word groups, it makes sense to base the merge cost on how the conditional distributions $p(c|w)$ differ for two word groups. The Jensen-Shannon (JS) divergence measures how much r distributions $p_1, \ldots, p_r$ differ from their *weighted* mean:

$$JS_{\pi_1,\ldots,\pi_r}[p_1, \ldots, p_r] = \mathcal{H}(\sum_i \pi_i p_i) - \sum_i \pi_i \mathcal{H}(p_i) = \sum_i \pi_i D(p_i \parallel \sum_j \pi_j p_j)$$

Where $D(\cdot \parallel \cdot)$ denotes the Kullback-Leibler divergence [3] and the $\pi_1, \ldots, \pi_r$ are the weighting factors. In our case we use the JS divergence between two distributions, so $r = 2$. The effectiveness of this measure to decide which word groups to merge is illustrated in [2, 14]. The use of the JS divergence can be motived by showing [14] that the JS divergence measures the loss in the Mutual Information $I(C;\ W) = \mathcal{H}(C) - \mathcal{H}(C|W)$ between the variables C and W due to a merge of word groups. Where C can take as values the different classes and W can take values in the set of words observed in the training documents. The joint distribution used to define the mutual information is the empirical distribution obtained from the training texts. Also, when using the log-loss discussed above, the JS divergence can be shown to express the *expected* change in code length if documents consisted of just one word. Furthermore,

[3] We use $|C|$ to denote the cardinality of the set C.

[4] For example if $n = 9$ and $k = 4$ then there are 7770 possible clusterings.

INPUT: A set of pre-classified texts
OUTPUT: A clustering of words and distributions $p(c|w_i)$ for all word groups w_i
begin
- estimate empirical distribution over classes
- estimate empirical distributions over classes given words
for each pair of words **do**
    - compute JS divergence of pair
end
while there is more than one word group **do**
        - determine pair with minimum JS divergence
        - compute change in code length and store it
        - record word clustering if shortest code length up to now is reached
        - adapt probability distributions to implement the word merge
end
- output word clustering and distributions $p(c|w)$ yielding minimum code length

end

Figure 1: The algorithm in pseudo code

it provides (quite loose) upper and lower bounds for the change in code length when documents consist of more than one word [16].

6 The algorithm

In Figure 1 the algorithm is outlined in pseudo code. Our program takes as input a set of pre-classified documents that are represented as word-text frequency lists. First, we compute the empirical joint distribution over the classes and words. Then, for each pair of word groups (originally each word group consists of just one word) we compute the JS divergence. As long as there are still word groups to merge, we merge the word groups with minimal JS divergence. Then, we compute how many bits are needed to encode the data (encoding the model plus encoding the data using the model). Finally, we compute the JS divergence between the old word groups that were not merged and the new word group. Now, again the word groups with minimum JS divergence are merged, etc. Finally, we look which number of word groups yielded the minimum description length and return the corresponding model (clustering plus joint distribution on the classes and word groups).

Let $|V|$ denote the number of words in the lexicon, i.e. the number of distinct words occurring in the sample. Let $|C|$ denote the number of classes and let $|D|$ denote the number of documents in the sample. To store the word-class data we need $|V||C|$ integer positions. To compare all the possible merges of clusters at each step, we need to compute for each pair of clusters the merge cost. There are originally $|V|(|V|-1)/2 \in \mathcal{O}(|V|^2)$ word pairs. We also need to store all the frequency lists corresponding to the training texts, this costs $|V||D|$ integer positions. Note that $|D| \gg |C|$, therefore, the space complexity is $\mathcal{O}(|V||D| + |V|^2)$.

At the start of the algorithm, we compute merge costs for all pairs. If we merge two clusters, we have to update the merge costs only for at most $|V|$ pairs. Since there are originally $\mathcal{O}(|V|^2)$ pairs of words and we perform $|V|-1$ merges, we conclude that we have to compute $\mathcal{O}(|V|^2)$ merge costs. Looking at the definition of the JS divergence we see that each merge cost can be computed in $\mathcal{O}(|C|)$ operations. The change in code length for a sample of size $|D|$ can be computed in $\mathcal{O}(|C||D||V|)$ operations. Note that we compute the change in code length after each of the $|V|-1$ merges. Hence the algorithm has time complexity $\mathcal{O}(|C||D||V|^2)$.

7 Results and conclusions

We used the '20 NEWSGROUP' dataset [10] for text classification. This dataset contains texts from newsgroups that are very much like each other: i.e. `comp.os.ms-windows.misc` and `comp.windows.x`. There-

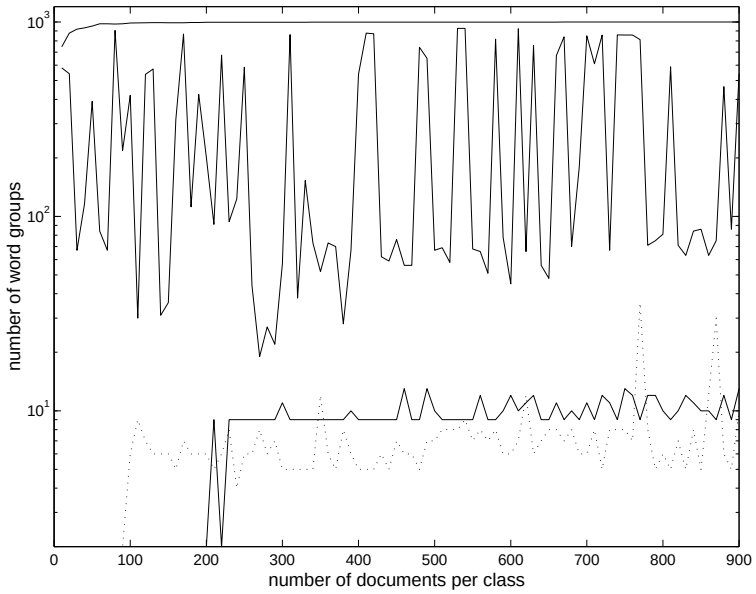

Figure 2: Number of word groups against sample size. Highest, smooth, curve: lexicon size. The other curves represent three model selection methods: Heavily oscillating curve: cross-validation. Dotted curve: MDL with probabilistic model as code scheme for the data. Lowest solid curve: MDL with 'coding exceptions' code scheme.

fore, we used a subset of the data set containing eight more clearly separated newsgroups: `rec.autos`, `rec.motorcycles` `rec.sport.baseball`, `rec.sport.hockey`, `talk.politics.guns`, `misc.forsale`, `talk.politics.mideast` and `talk.religion.misc`. To extract the necessary statistics from the data set we made gratefully use of the RAINBOW program, part of the BOW TOOLKIT by McCallum [11]. To measure predictive performance we used a test set of fifty documents per class. It turned out that the variance in predictive performance on different smaller test sets was negligible, therefore we decided to use just one test set. As preprocessing we used stemming, a stop list and removed words that occur in less than 5% of the documents. To make the experiments manageable on the available equipment we decided to prune the lexicon to the 1000 words having the highest mutual information with the text labels.

7.1 Discussion of results

We found that the JS divergence is a good measure to merge word groups. This conforms with the results presented in [2]. There, the same clustering method is also compared with LSI (using the '20 NEWSGROUP' dataset) and feature *selection* based on information-gain (using the '20 NEWSGROUP' dataset, the 'REUTERS-21578' dataset and the 'YAHOO!' dataset). It turns out that the clustering method outperforms LSI, which could be expected since LSI is unsupervised. Furthermore, it also outperforms information-gain feature selection for small number of features (less that 100) and performs comparable for larger number of features.

Until a very small number of word groups (in the order of ten) is reached predictive performance is hardly lost. If, however, even less word groups are used predictive performance is dropping dramatically. This phenomenon is more often observed and referred to as the 'elbow phenomenon' in pattern recognition literature (due to the shape of a plot of prediction accuracy against the number of word groups used).

Figure 2 shows how the number of word groups proposed by the algorithm as a function of the sample size, i.e. the number of documents per class in the sample. For this experiment as well as the one depicted in Figure 3, sample sizes of ten up to 900 (with an interval of ten) documents per class were used. Note the logarithmic scale on the vertical axis! The plot contains four curves: (i) The highest (smooth) curve depicts the number of different words present in the sample. (ii) The heavily oscillating curve below the latter curve shows which number of word groups yielded best performance on the test set, i.e. when using cross-validation as model selection method. The heavy oscillation can be explained by the marginal loss in prediction performance for high numbers of word groups, as discussed above. (iii) and (iv) The remaining two curves show how many word groups were proposed by the algorithm using the two different

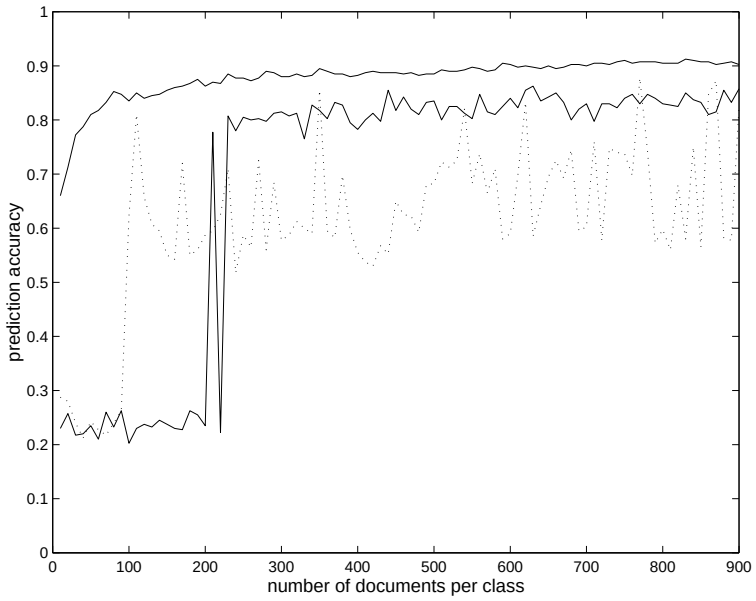

Figure 3: Prediction accuracy against sample size. The curves represent the three model selection methods. Highest curve: cross-validation. Dotted curve: MDL with probabilistic model as code scheme for the data. Lowest solid curve: MDL with 'coding exceptions' code scheme.

coding schemes. The dotted line corresponds to using the probabilistic model directly to code the labels, where the solid line corresponds to coding exceptions. Note that the algorithm suffers from a *start-up period*, in which the algorithm proposes using just one or two word groups. Such a start up period was also encountered in the experiments reported in [15]. It is caused by the domination of the error term in equation (1) by the complexity term for small samples, due to the fixed precision with which the models are encoded.

In Figure 3 we see three lines corresponding to the three lower curves in Figure 2. Here, the prediction accuracy of the proposed models on the test set is depicted. Note the dramatic performance during the start-up period. Furthermore, note the sub-optimal performance when using the probabilistic model directly to encode the class labels. We see that the 'coding exceptions' code scheme, by favoring just slightly more word groups than the direct code scheme, yields far better results in terms of predictive performance on the test set. We can see that the difference between the performance of cross-validation and that of the 'coding exceptions' code scheme differ in general between five an ten percent. More precisely: outside the 'start-up' period, the mean difference is 7.16% with a standard deviation of 1.70% This conforms to our observations of the difference in optimal performance and the performance at the 'elbow' point discussed above.

7.2 Conclusions

Based on the experiments reported here and the matters encountered during this research, we are able to draw the following conclusions:

1. We implemented our algorithm using two code schemes. It became clear that using the 'coding exceptions' code scheme yields significantly better performance of our algorithm.

2. The experimental results suggest the proposed method as an adequate method to determine, in a supervised setting, the number of features for a classification task.

3. We had originally hoped to be able to use the conditional entropy as an useful estimate of the code length needed to encode the class labels. This would have been nice, since then we could use the JS-divergence (which we compute anyway to decide which word groups to merge) as the change in (expected) code length. However, using conditional entropy as an estimate for the code length turned out to yield very poor performance. This is due to an overestimation of the actual code length.

4. One of the main problems we encountered during this research was to find efficient code schemes to encode the models, we also encountered this problem in [15]. We are led to conclude that this can form serious problems in applications of the MDL principle. It is also possible to use a second interpretation of the MDL principle, called 'predictive MDL', where we do not need to encode the models explicitly. This method, however, is very time consuming and therefore not appropriate here.

Open Issues Due to limited time, this research did not involve extensive comparison of this method to other known methods. Also, it would be interesting to apply the same method to cluster *documents* instead of words and determine automatically *how many* documents clusters to use. We feel that both issues deserve some attention.

References

[1] Ricardo Baeza-Yates and Berthier Ribeiro-Neto. *Modern Information Retrieval.* Addison-Wesley-Longman, 1999.

[2] D. Baker and A. McCallum. Distributional clustering of words for text classification. In *SIGIR 1998*, 1998.

[3] T. Cover and J. Thomas. *Elements of Information Theory.* Wiley, 1991.

[4] M. Derthick. A minimal encoding approach to feature discovery. In *Proceedings of AAAI-91*, pages 565–571, Anaheim, CA, 1991.

[5] B. Dom, W. Niblack, and J.Sheinvald. Feature selection for classification using the MDL principle. Computer Science RJ-7117, IBM Almaden Research Center, March 1989.

[6] S.T. Dumaisand, G.W. Furnas, T.K. Landauer, S. Deerwester, and R. Harshman. Using latent semantic analysis to improve access to textual information. In *Conference proceedings on Human factors in computing systems*, pages 281–285, 1998.

[7] J.H. Friedman. On bias, viariance 0/1 loss, and the curse-of-dimensionality. *Data Mining and Knowledge Discovery*, 1(1):55–77, April 1997.

[8] R. Graham, D. Knuth, and O. Patashnik. *Concrete Mathematics, a Foundation for Computer Science.* Addison Wesley, 1989.

[9] T. Joachims. Text Categorization with Support Vector Machines: Learning with Many Relevant Features. In C. Nédellec and C. Rouveirol, editors, *Proceedings of the 10^{th} European Conference on Machine Learning*, volume 1398 of *Lecture Notes in Computer Science*, pages 137–142. Springer, 1998.

[10] K. Lang. 20 newsgroups data set. fetched May 24 2000, `http://www.ai.mit.edu/~jrennie/20_newsgroups/`.

[11] A.K. McCallum. Bow: A toolkit for statistical language modeling, text retrieval, classification and clustering. fetched May 24 2000 `http://www.cs.cmu.edu/~mccallum/bow`, 1996.

[12] J. Rissanen. On optimal number of features in classification. Technical Report RJ-6471, IBM Thomas J. Watson Research Division, 1988.

[13] J. Rissanen. *Stochastic Complexity in Statistical Inquiry.* World Scientific, 1989.

[14] N. Slonim and N. Tishby. Agglomerative information bottleneck. *Advances in Neural Information Processing Systems*, pages 617–623, 2000.

[15] J.J. Verbeek. Overfitting using the MDL Principle. Master's thesis, University of Amsterdam, 1998. fetched November 9 2000, `http://www.science.uva.nl/~jverbeek`.

[16] J.J. Verbeek. An information theoretic approach to finding word groups for text classification. Master's thesis, Institute for Logic, Language and Computation (ILLC-MoL-2000-03), Amsterdam, The Netherlands, September 2000.

[17] Y. Yang. An evaluation of statistical approaches to text categorization. *Journal of Information Retrieval*, 1(1/2):67–88, 1999.