Text Categorisation: A Survey.

Kjersti Aas and Line Eikvil

June 1999

Contents

Chapter 1

Introduction.

As the volume of information available on the Internet and corporate intranets continues to increase, there is a growing need for tools helping people better find, filter, and manage these resources. Text categorisation, the assignment of free text documents to one or more predefined categories based on their content, is an important component in many information management tasks; real-time sorting of email or files into folder hierarchies, topic identification to support topic-specific processing operations, structured search and/or browsing, or finding documents that match long-term standing interests or more dynamic task-based interests.

In many contexts trained professionals are employed to categorise new items. This process is very time-consuming and costly, thus limiting its applicability. Consequently there is an increasing interest in developing technologies for automatic text categorisation.

A number of statistical classification and machine learning techniques has been applied to text categorisation, including regression models [30], nearest neighbour classifiers [30], decision trees [19], Bayesian classifiers [19], Support Vector Machines [15], rule learning algorithms [6], relevance feedback [23], voted classification [27], and neural networks [28].

In this report we give a survey of the state-of-the-art in text categorisation. To be able to measure progress in this field, it is important to use a standardised collection of documents for analysis and testing. One such data set is the Reuters-21578 collection of newswires for the year 1987, and our survey will focus on the work on text categorisation that have used this collection for testing.

This report is divided into 6 chapters in addition to the introduction. Chapter 2 describes the steps that are needed to transform raw text into a representation suitable for the the classification task. In Chapter 3 we describe 6 methods that have been successfully applied to text categorisation. Chapter 4 introduces performance measures for category ranking evaluation and binary categorisation evaluation. Previous work using the Reuters-21578 collection is surveyed in Chapter 5. In Chapter 6 we describe our own work using the

Reuters collection, while a summary is given in Chapter 7.

Norwegian Computing Center, P.B. 114 Blindern, N-0314 Oslo, Norway Tel.: (+47) 22 85 25 00 Fax: (+47) 22 69 76 60

Chapter 2

Feature Extraction

2.1 Preprocessing

The first step in text categorisation is to transform documents, which typically are strings of characters, into a representation suitable for the learning algorithm and the classification task. The text transformation usually is of the following kind:

- Remove HTML (or other) tags
- Remove stopwords
- Perform word stemming

The **stopwords** are frequent words that carry no information (i.e. pronouns, prepositions, conjunctions etc.).

By **word stemming** we mean the process of suffix removal to generate word stems. This is done to group words that have the same conceptual meaning, such as *walk, walker, walked,* and *walking.* The Porter stemmer [21] is a well-known algorithm for this task.

2.2 Indexing

The perhaps most commonly used document representation is the so called vector space model (SMART) [24]. In the vector space model, documents are represented by vectors of words. Usually, one has a collection of documents which is represented by a word-by-document matrix $\mathbf{A}$, where each entry represents the occurrences of a word in a document, i.e.,

$$\mathbf{A} = (a_{ik}), \tag{2.1}$$

where a_{ik} is the weight of word i in document k. Since every word does not normally appear in each document, the matrix $\mathbf{A}$ is usually sparse. The number of rows, M, of the matrix corresponds to the number of words in the dictionary. M can be very large. Hence, a major characteristic, or difficulty of text categorization problems is the high dimensionality of the feature space. In Section 2.3 we discuss different approaches for dimensionality reduction.

There are several ways of determining the weight a_{ik} of word i in document k, but most of the approaches are based on two empirical observations regarding text:

- The more times a word occurs in a document, the more relevant it is to the topic of the document.
- The more times the word occurs throughout all documents in the collection, the more poorly it discriminates between documents.

Let f_{ik} be the frequency of word i in document k, N the number of documents in the collection, M the number of words in the collection after stopword removal and word stemming, and n_i the total number of times word i occurs in the whole collection. In what follows we describe 6 different weighting schemes that are based on these quantities.

Boolean weighting

The simplest approach is to let the weight be 1 if the word occurs in the document and 0 otherwise:

$$a_{ik} = \begin{cases} 1 & \text{if } f_{ik} > 0 \\ 0 & \text{otherwise} \end{cases} \tag{2.2}$$

Word frequency weighting

Another simple approach is to use the frequency of the word in the document:

$$a_{ik} = f_{ik} \tag{2.3}$$

tf×idf-weighting

The previous two schemes do not take into account the frequency of the word throughout all documents in the collection. A well-known approach for computing word weights is the tf×idf-weighting [24], which assigns the weight to word i in document k in proportion to the number of occurrences of the word in the document, and in inverse proportion to the number of documents in the collection for which the word occurs at least once.

$$a_{ik} = f_{ik} * \log\left(\frac{N}{n_i}\right) \tag{2.4}$$

tfc-weighting

The tf×idf-weighting does not take into account that documents may be of different lengths. The tfc-weighting [25] is similar to the tf×idf-weighting except for the fact that length normalisation is used as part of the word weighting formula.

$$a_{ik} = \frac{f_{ik} * \log\left(\frac{N}{n_i}\right)}{\sqrt{\sum_{j=1}^{M}\left[f_{jk} * \log\left(\frac{N}{n_j}\right)\right]^2}} \tag{2.5}$$

ltc-weighting

A slightly different approach [5] uses the logarithm of the word frequency instead of the raw word frequency, thus reducing the effects of large differences in frequencies.

$$a_{ik} = \frac{\log(f_{ik} + 1.0) * \log\left(\frac{N}{n_i}\right)}{\sqrt{\sum_{j=1}^{M}\left[\log(f_{jk} + 1.0) * \log\left(\frac{N}{n_j}\right)\right]^2}} \tag{2.6}$$

Entropy weighting

Entropy-weighting is based on information theoretic ideas and is the most sophisticated weighting scheme. In [9] it turned out to be the most effective scheme in comparison with 6 others. Averaged over five test collections, it was for instance 40 % more effective than word frequency weighting. In the entropy-weighting scheme, the weight for word i in document k is given by:

$$a_{ik} = \log(f_{ik} + 1.0) * \left(1 + \frac{1}{\log(N)}\sum_{j=1}^{N}\left[\frac{f_{ij}}{n_i}\log\left(\frac{f_{ij}}{n_i}\right)\right]\right) \tag{2.7}$$

where

$$\frac{1}{\log(N)}\sum_{j=1}^{N}\left[\frac{f_{ij}}{n_i}\log\left(\frac{f_{ij}}{n_i}\right)\right]$$

is the average uncertainty or entropy of word i. This quantity is -1 if the word is equally distributed over all documents, and 0 if the word occurs in only one document.

2.3 Dimensionality reduction

A central problem in statistical text classification is the high dimensionality of the feature space. There exists one dimension for each unique word found in the collection of documents, typically hundreds of thousands. Standard classification techniques cannot deal with such a large feature set, since processing is extremely costly in computational terms, and the results become unreliable due to the lack of sufficient training data. Hence, there is a need for a reduction of the original feature set, which is commonly known as dimensionality reduction in the pattern recognition literature. Most of the dimensionality reduction approaches can be classified into one of two categories; feature selection or re-parameterisation.

2.3.1 Feature Selection

Feature selection attempts to remove non-informative words from documents in order to improve categorisation effectiveness and reduce computational complexity. In [30] a thorough evaluation of the five feature selection methods; Document Frequency Thresholding, Information Gain, χ^2-statistic, Mutual Information, and Term Strength is given. In their experiments, the authors found the three first to be the most effective[1]. Below a short description of these methods is given.

Document Frequency Thresholding

The document frequency for a word is the number of documents in which the word occurs. In Document Frequency Thresholding one computes the document frequency for each word in the training corpus and removes those words whose document frequency is less than some predetermined threshold. The basic assumption is that rare words are either non-informative for category prediction, or not influential in global performance.

Information gain

Information Gain measures the number of bits of information obtained for category prediction by knowing the presence or absence of a word in at document.

[1] Their results are obtained based on a k-nearest neighbour classifier and a regression method.

Let $c_1, ..., c_K$ denote the set of possible categories. The information gain of a word w is defined to be:

$$IG(w) = -\sum_{j=1}^{K} P(c_j) \log P(c_j) + P(w) \sum_{j=1}^{K} P(c_j|w) \log P(c_j|w) + P(\bar{w}) \sum_{j=1}^{K} P(c_j|\bar{w}) \log P(c_j|\bar{w}) \quad (2.8)$$

Here $P(c_j)$ can be estimated from the fraction of documents in the total collection that belongs to class c_j and $P(w)$ from the fraction of documents in which the word w occurs. Moreover, $P(c_j|w)$ can be computed as the fraction of documents from class c_j that have at least one occurrence of word w and $P(c_j|\bar{w})$ as the fraction of documents from class c_j that does not contain word w.

The information gain is computed for each word of the training set, and the words whose information gain is less than some predetermined threshold are removed.

χ^2-statistic

The χ^2-statistic measures the lack of independence between word w and class c_j. It is given by:

$$\chi^2(w, c_j) = \frac{N \times (A\,D - C\,B)^2}{(A+C) \times (B+D) \times (A+B) \times (C+D)} \quad (2.9)$$

Here A is the number of documents from class c_j that contains word w, B is the number of documents that contains w but does not belong to class c_j, C is the number of documents from class c_j that does not contain word w, and D is the number of documents that belongs to class c_j nor contains word w. N is still the total number of documents.

Two different measures can be computed based on the χ^2-statistic:

$$\chi^2(w) = \sum_{j=1}^{K} P(c_j)\, \chi^2(w, c_j) \quad (2.10)$$

or

$$\chi^2_{max}(w) = \max_j \chi^2(w, c_j) \quad (2.11)$$

2.3.2 Re-parameterisation: Latent Semantic Indexing

Re-parameterisation is the process of constructing new features as combinations or transformations of the original features. In this section we describe one such approach; Latent Semantic Indexing (LSI) [2, 7].

LSI is based on the assumption that there is some underlying or latent structure in the pattern of word usage across documents, and that statistical techniques can be used to estimate this structure. LSI uses singular-value decomposition (SVD), a technique closely related to eigenvector decomposition and factor analysis. In what follows we describe the mathematics underlying the particular model of the latent structure; the singular value decomposition.

Singular Value Decomposition

Assume that we have a $M \times N$ word-by-document matrix $\mathbf{A}$, where M is the number of words, and N the number of documents. The singular value decomposition of $\mathbf{A}$ is given by:

$$\mathbf{A} = \mathbf{U\Sigma V^T} \tag{2.12}$$

where $\mathbf{U}$ $(M \times R)$ and $\mathbf{V}$ $(R \times N)$ have orthonormal columns and $\mathbf{\Sigma}$ $(R \times R)$ is the diagonal matrix of singular values. $R \leq \min(M, N)$ is the rank of $\mathbf{A}$. If the singular values of $\mathbf{\Sigma}$ are ordered by size, the K largest may be kept and the remaining smaller ones set to zero. The product of the resulting matrices is a matrix $\mathbf{A}_K$ which is an approximation to $\mathbf{A}$ with rank K.

$$\mathbf{A}_K = \mathbf{U}_K \mathbf{\Sigma}_K \mathbf{V}_K^T \tag{2.13}$$

where $\mathbf{\Sigma}_K$ $(K \times K)$ is obtained by deleting the zero rows and columns of $\mathbf{\Sigma}$, and $\mathbf{U}_K$ $(M \times K)$ and $\mathbf{V}_K$ $(N \times K)$ are obtained by deleting the corresponding rows and columns of $\mathbf{U}$ and $\mathbf{V}$.

$\mathbf{A}_K$ in one sense captures most of the underlying structure in A,yet at the same time removes the noise or variability in word usage. Since the number of dimensions K is much smaller than the number of unique words M, minor differences in terminology will be ignored. Words which occur in similar documents may be near each other in the K-dimensional space even if they never co-occur in the same document. Moreover, documents that do not share any words with each other, may turn out to be similar. In the next section we describe how documents and words may be compared using the SVD.

Computing fundamental comparison quantities from the SVD model

The cosine between two rows in $\mathbf{A}_K$, or equally the cosine between two rows in $\mathbf{U}_K \mathbf{\Sigma}_K$, reflects the extent to which two words have a similar pattern of occurrence across the set of documents. If the cosine is 1, the two words have exactly the same pattern of occurrence, while a cosine of 0 means that the pattern of occurrence is very different for the two words.

The comparison of two documents is similar, but in this case the cosine between two columns in $\mathbf{A}_K$, or equally the cosine between two rows of $\mathbf{V}_K \mathbf{\Sigma}_K$, indicates whether two documents have similar content.

To compare a word i with a document k, one takes the cosine between the ith row of the matrix $\mathbf{U}_K \mathbf{\Sigma}_K^{1/2}$ and the kth row of $\mathbf{V}_K \mathbf{\Sigma}_K^{1/2}$.

To compare a new document with the documents in the training set, one starts with its document vector $\mathbf{d}$ and derive a representation $\hat{\mathbf{d}}$ given by:

$$\hat{\mathbf{d}} = \mathbf{d}^T \, \mathbf{U}_K \boldsymbol{\Sigma}_{\mathbf{K}}{}^{-1}$$

This representation can be used just like a row of $\mathbf{V}_K$ in the comparisons described above, meaning that taking the cosine between $\hat{\mathbf{d}}\boldsymbol{\Sigma}_K$ and rows of $\mathbf{V}_K\boldsymbol{\Sigma}_K$ gives the degree of similarity between the new document and the documents in the training set.

Incorporating new words and documents into an existing SVD model

The most straightforward method of adding more words or documents to an existing SVD model is to re-compute the SVD with the new word-document matrix. However, this approach requires a significant amount of computation time, and for large problems it may be impossible due to memory constraints.

Hence, other methods have been considered. A simple approach is denoted *folding-in* [2]. It follows essentially the process that is described in Section 2.3.2 for comparing a new document with the documents in the training set. To fold in a new document vector $\mathbf{d}$, it is first projected onto the span of the current word vectors (columns of $\mathbf{U}_K$) by:

$$\hat{\mathbf{d}} = \mathbf{d}^T \, \mathbf{U}_K \boldsymbol{\Sigma}_K^{-1}$$

and then appended to the existing document vectors or columns of $\mathbf{V}_K$. Similarly, to fold in a new word vector $\mathbf{w}$, the projection $\hat{\mathbf{w}}$ onto the span of the current document vectors (columns of $\mathbf{V}_K$) is determined by:

$$\hat{\mathbf{w}} = \mathbf{w}^T \, \mathbf{V}_K \boldsymbol{\Sigma}_K^{-1}$$

and then the word vector is appended to the existing word vectors or columns of $\mathbf{U}_K$. Folding-in documents requires less time and memory than the re-computing approach, but can have negative effects on classification if the word usage in the new documents is different from that in the documents that already are in the training set. In this case, the new word usage data may potentially be lost or misrepresented.

A third method, *SVD-updating* that deals with this problem has recently been developed [20]. However, SVD-updating requires slightly more time and memory than the folding-in approach, meaning that neither approach appears to be uniformly superior over the other.

Chapter 3

Methods

Text categorisation is the problem of automatically assigning one or more predefined categories to free text documents. While more and more textual information is available online, effective retrieval is difficult without good indexing and summarisation of document content. Document categorisation is one solution to this problem. A growing number of statistical classification methods and machine learning techniques have been applied to text categorisation in recent years.

Most of the research in text categorisation has been devoted to *binary* problems, where a document is classified as either *relevant* or *not relevant* with respect to a predefined topic. However, there are many sources of textual data, such as Internet News, electronic mail and digital libraries, which are composed of different topics and which therefore pose a *multi-class* categorisation problem.

Moreover, in multi-class problems, it is often the case that documents are relevant to more than one topic. For example, a news article may well be relevant to several topics. While a few methods have been devised for multi-class text categorisation, the *multi-label* case, where a document belong to more than one class, has received very little attention.

The common approach for multi-class, multi-label text categorisation is to break the task into disjoint binary categorisation problems, one for each class. To classify a new document, one needs to apply all the binary classifiers and combine their predictions into a single decision. The end result is a ranking of possible topics. The main drawback with this approach is that it ignores any correlation between the different classes.

In what follows we describe some of the algorithms for text categorisation that have been proposed and evaluated in the past, but first some general notation is given: Let $\mathbf{d} = \{d_1, ..., d_M\}$ be the document vector to be classified and $c_1, ..., c_K$ the possible topics. Further assume that we have a training set consisting of N document vectors $\mathbf{d}_1, ..., \mathbf{d}_N$ with true classes $y_1, ..., y_N$. N_j is then the number of training documents for which the true class is c_j.

3.1 Rocchio's algorithm

Rocchio is the classic method for document routing or filtering in information retrieval. In this method, a prototype vector is built for each class c_j, and a document vector $\mathbf{d}$ is classified by calculating the distance between $\mathbf{d}$ and each of the prototype vectors. The distance can be computed by for instance the dot product or by using the Jaccard similarity measure.

The protype vector for class c_j is computed as the average vector over all training document vectors that belong to class c_j. This means that learning is very fast for this method.

3.2 Naive Bayes

The naive Bayes classifier [14] is constructed by using the training data to estimate the probability of each class given the document feature values of a new instance. We use Bayes theorem to estimate the probabilities:

$$P(c_j|\mathbf{d}) = \frac{P(c_j)\,P(\mathbf{d}|c_j)}{P(\mathbf{d})} \tag{3.1}$$

The denominator in the above equation does not differ between categories and can be left out. Moreover, the naive part of such a model is the assumption of word independence, i.e. we assume that the features are conditionally independent, given the class variable. This simplifies the computations yielding

$$P(c_j|\mathbf{d}) = P(c_j) \prod_{i=1}^{M} P(d_i|c_j) \tag{3.2}$$

An estimate $\hat{P}(c_j)$ for $P(c_j)$ can be calculated from the fraction of training documents that is assigned to class c_j:

$$\hat{P}(C = c_j) = \frac{N_j}{N} \tag{3.3}$$

Moreover, an estimate $\hat{P}(d_i|c_j)$ for $P(d_i|c_j)$ is given by:

$$\hat{P}(d_i|c_j) = \frac{1 + N_{ij}}{M + \sum_{k=1}^{M} N_{kj}} \tag{3.4}$$

where N_{ij} is the number of times word i occurred within documents from class c_j in the training set.

Despite the fact that the assumption of conditional independence is generally not true for word appearance in documents, the Naive Bayes classifier is surprisingly effective.

3.3 K-nearest neighbour

To classify an unknown document vector **d**, the k-nearest neighbour (kNN) algorithm [8] ranks the document's neighbours among the training document vectors, and use the class labels of the k most similar neighbours to predict the class of the input document. The classes of these neighbours are weighted using the similarity of each neighbour to **d**, where similarity may be measured by for example the Euclidean distance or the cosine between the two document vectors.

kNN is a lazy learning instance-based method that does not have a off-line training phase. The main computation is the on-line scoring of training documents given a test document in order to find the k nearest neighbours. Using an inverted-file indexing of training documents, the time complexity is $O(L * N/M)$ where L is the number of elements of the document vector that are greater than zero, M is the length of the document vector, and N is the number of training samples.

3.4 Decision Trees

In this approach, the document vector **d** is matched against a decision tree to determine whether the document is relevant to the user or not. The decision tree is constructed from the training samples, and one the most popular approaches for this task is the *CART* [3] algorithm that will be described here.

3.4.1 Creating the tree (CART)

CART builds a binary decision tree by splitting the set of training vectors at each node according to a function of one single vector element. The first task is therefore to decide which of the vector elements that makes the best splitter, i.e. the one that partitions the set in as homogenous subsets as possible. This means that the best splitter is the one that decreases the diversity of the set of training samples by the greatest amount, i.e. one wants to maximise:

$$\text{diversity(before split)-[diversity(left child)+diversity(right child)]} \tag{3.5}$$

One of the commonly used diversity measures is *entropy*:

$$\sum_{j=1}^{K} p(c_j|t) \log p(c_j|t) \tag{3.6}$$

where $p(c_j|t)$ is the probability of a training sample being in class c_j given that it falls into node t. This probability can be estimated by:

$$p(c_j|t) = \frac{N_j(t)}{N(T)} \tag{3.7}$$

where $N_j(t)$ and $N(t)$ are the number of samples of class c_j and the total number of samples at node t, respectively.

To choose the best splitter at a node in the tree, each component of the document vector is considered in turn. A binary search is performed to determine the best split value for the component, using the decrease in diversity as the measure of goodness. Having found the best split value, one compares the decrease in diversity to that provided by the current best splitter. The component that corresponds to the largest decrease in diversity is chosen as the splitter for the node.

This procedure is repeated until no sets can be partitioned any further, i.e. until no split can be found that significantly decreases the diversity at any node. The nodes at the bottom of the tree are denoted leaf nodes, and at the end of the tree-growing process, every sample of the training set has been assigned to some leaf of the full decision tree.

Each leaf can now be assigned a class. This does not mean, however, that all training samples reaching this leaf actually have the same class. The error rate of a leaf measures the probability of samples reaching this leaf being misclassified. The error rate, $E(T)$, of the whole tree is the weighted sum of the error rates of all the leaves.

3.4.2 Pruning the tree

Even if the full tree resulting from the procedure described above minimises the error rate using the training data for evaluation, it will probably not do the best job of classifying new data sets. This is due to the reason that it has been over-fitted to the training set. In order to get more accurate predictions in the general case, we need to prune the tree.

The goal of the pruning is to remove the branches which provide the least additional predictive power per leaf. To identify these branches we use the adjusted error rate of a tree T:

$$E_\alpha(T) = E(T) + \alpha\, N_{leaves}(T) \tag{3.8}$$

where $N_{leaves}(T)$ is the number of leaves of the tree, and α is a parameter. This means that we put a penalty on the number of leaves in the tree. Let T_α be the subtree of T for which $E_\alpha(\cdot)$ is minimal. When α increases from 0 to infinity, there are only a finite number of values of α for which T_α is different. Let these values be denoted $0 = \alpha_1 < .... < \alpha_k$. The corresponding subtrees $T_0, , , , , , T_{\alpha_k}$ form a decreasing sequence of trees, where T_0 is the original tree and T_{α_k} consists of the root node only.

The final task of the pruning process is to select the tree T_{α_i} that will best classify new data. For this purpose, we use another training set. Each of the candidate trees is used to classify the vectors in the new training set, and the tree that performs this task with the lowest overall error rate is declared the winner.

3.4.3 Other algorithms

Two other well-known decision tree algorithms are *C4.5* [22] and *CHAID* [16]. The differences between these algorithms and CART will be briefly described in this section.

C4.5 is the most recent version of the decision-tree algorithm that Quinlan has been evolving and refining for many years (an earlier version of it was denoted ID3). C4.5 differs from CART in that it produces trees with varying numbers of branches per node. For instance for categorical variables there will be one branch for each value taken on by a categorical variable. Another area in which C4.5 differs from CART is the approach to pruning.

CHAID differs from CART and C4.5 in that rather than first overfitting the data, then pruning, CHAID attempts to stop growing the tree before overfitting occurs. Another difference is that CHAID is restricted to categorical variables, meaning that continuous variables must be broken into ranges or replaced by classes before the tree is built.

3.5 Support Vector Machines

Support Vector Machines (SVMs) have shown to yield good generalisation performance on a wide variety of classification problems, most recently text categorisation [15, 17]. The SVM integrates dimension reduction and classification. It is only applicable for binary classification tasks, meaning that, using this method text categorisation has to be treated as a series of dichotomous classification problems.

The SVM classifies a vector $\mathbf{d}$ to either -1 or 1 using

$$s = \mathbf{w}^T \phi(\mathbf{d}) + b = \sum_{i=1}^{N} \alpha_i \, y_i \, K(\mathbf{d}, \mathbf{d}_i) + b \tag{3.9}$$

and

$$y = \begin{cases} 1 & \text{if } s > s_0 \\ -1 & \text{otherwise} \end{cases} \tag{3.10}$$

Here $\{\mathbf{d}_i\}_{i=1}^N$ is the set of training vectors as before and $\{y_i\}_{i=1}^N$ are the corresponding classes ($y_i \in -1, 1$). $K(\mathbf{d}_i, \mathbf{d}_j)$ is denoted a kernel and is often chosen as a polynom of degree d, i.e.

$$K(\mathbf{d}, \mathbf{d}_i) = (\mathbf{d}^T \mathbf{d}_i + 1)^d \tag{3.11}$$

The training of the SVM consists of determining the $\mathbf{w}$ that maximises the distance between the training samples from the two classes. In what follows we describe how this is achieved.

3.5.1 Training the SVM - the separable case

First, consider the case for which the data is linearly separable. That means that there exists a vector $\mathbf{w}$ and a scalar b such that

$$\mathbf{w}^T \phi(\mathbf{d}_i) + b \geq 1 \quad \text{if } y_i = 1 \tag{3.12}$$

$$\mathbf{w}^T \phi(\mathbf{d}_i) + b \leq -1 \quad \text{if } y_i = -1 \tag{3.13}$$

for all $\{\mathbf{d}_i\}_{i=1}^N$. The SVM constructs a hyperplane $\mathbf{w}^T \phi(\mathbf{d}) + b$ for which the separation between the two classes is maximised. The $\mathbf{w}$ for the optimal hyperplane can be found by minimising

$$||\mathbf{w}||^2 \tag{3.14}$$

The optimal $\mathbf{w}$ can be written as a linear combination of $\phi(\mathbf{d})$'s, i.e.,

$$\sum_{i=1}^{N} \alpha_i \, y_i \, \phi(\mathbf{d}_i) \tag{3.15}$$

where $\{\alpha_i\}_{i=1}^N$ can by found by maximising:

$$\mathbf{\Lambda}^T \mathbf{1} - \frac{1}{2} \mathbf{\Lambda}^T Q \, \mathbf{\Lambda} \tag{3.16}$$

with respect to $\mathbf{\Lambda}$, under the constraints

$$\mathbf{\Lambda} \geq 0 \quad \text{and} \quad \mathbf{\Lambda}^T \mathbf{Y} = 0 \tag{3.17}$$

Here $\mathbf{Y} = (y_1, ...y_N)$ and $\mathbf{Q}$ is a symmetric matrix with elements

$$Q_{ij} = y_i \, y_j \, K(\mathbf{d}_i, \mathbf{d}_j) = y_i \, y_j \, \phi(\mathbf{d}_i)^T \phi(\mathbf{d}_j) \tag{3.18}$$

Only the α_i's corresponding to the training examples along the margins of the decision boundary (i.e. the support vectors) are greater than zero.

3.5.2 Training the SVM - the non-separable case

In the non-separable case, Equation 3.14 is altered to:

$$\frac{1}{2}||\mathbf{w}||^2 + C \sum_{i=1}^{N} \xi_i \tag{3.19}$$

where ξ_i satisfy the constraints:

$$\mathbf{w}^T\phi(\mathbf{d}_i) + b \geq 1 - \xi_i \quad \text{if } y_i = 1 \tag{3.20}$$

$$\mathbf{w}^T\phi(\mathbf{d}_i) + b \leq -1 + \xi_i \quad \text{if } y_i = -1 \tag{3.21}$$

The user-defined parameter C balances the contributions from the first and second terms. Minimisation of Equation 3.19 can be achieved by maximising Equation 3.16 under the constraints

$$0 \leq \mathbf{\Lambda} \leq C\,\mathbf{1} \quad \text{and} \quad \mathbf{\Lambda}^T\,\mathbf{Y} = 0 \tag{3.22}$$

3.5.3 Ability to incorporate new documents

The optimisation problem described above is very challenging when the data set is large, as the memory requirement grows with the square of the size of the data set. The computational and storage complexities can be reduced by dividing the training set into a number of *chunks* and extracting support vectors from each of these. The support vectors can later be combined together.

The same procedure can be used for incorporating new documents into the existing set of support vectors. It can be shown that the final solution is as good as if all documents were processed together.

3.6 Voted Classification

Many researchers have investigated the technique of combining the predictions of multiple classifiers to produce a single classifier. This process is often denoted voting. Voting algorithms takes a classifier and a training set as input and trains the classifier multiple times on different versions of the training set. The generated classifiers are then combined to create a final classifier that is used to classify the test set.

Voting algorithms can be divided into two types: *bagging* algorithms and *boosting* algorithms. The main difference between the two types is the way the different versions of the training set are created. In what follows a closer description of the two types of algorithms is given.

3.6.1 The Bagging Algorithm

Bagging [4] takes as input a classification algorithm $f(\cdot)$ and a training set T and returns a set of classifiers $f^*(\cdot) = \{f_1(\cdot), \ldots.., f_R(\cdot)\}$. Here $f_r(\cdot)$ is a classifier that is learned from a bootstrap sample T_r of the training set. The bootstrap sample is formed by uniform probability random selection from T with replacement N times, where N is the size of the training set. This will create a training set with the same number of samples as the original, but some cases may be represented more than once, while others may be not be represented at all. The expected frequency with which the cases from T are represented in a single bootstrap sample T_r is described by the discrete Poisson distribution.

To classify a new sample $\mathbf{d}$, each classifier $f_r(\cdot)$ from $f^*(\cdot)$ is applied to $\mathbf{d}$ resulting in labels $f_1(\mathbf{d}), f_2(\mathbf{d}), ..., f_R(\mathbf{d})$. The result of the voting classifier is the class that obtains the most votes from the single classifiers when applied to $\mathbf{d}$:

$$f^*(\mathbf{d}) = \text{argmax}_y \sum_{r:f_r(\mathbf{d})=y} 1 \tag{3.23}$$

3.6.2 Boosting

Boosting [11] encompasses a family of methods. Like bagging, these methods choose a training set of size N for classifier f_r by randomly selecting with replacement examples from the original training set. Unlike bagging, however, the probability of selecting a sample is not the same for all samples of the training set. It depends instead on how often that sample was misclassified by the previous $k-1$ classifiers. Thus boosting attempts to produce new classifiers that are better able to correctly classify examples for which the performance of the current classifiers are poor.

Different forms of boosting generate the probabilities for selecting samples in different ways. In this report, we describe the approach used in the AdaBoost algorithm [11].

AdaBoost

Let the probability of selecting the sample $\mathbf{d}_i$ for training set T_r be p_{ir}. Initially, all the probabilities are equal, i.e. $p_{i1} = 1/N$ for all samples $\mathbf{d}_i$. To determine the p_{ir}'s for classifier $f_{r+1}(\cdot)$ AdaBoost first computes the sum, ϵ_r, of the probabilities corresponding to the samples that were misclassified using classifier $f_r(\cdot)$:

$$\epsilon_r = \sum_{i:f_r(\mathbf{d_i})\neq y_i} p_{ir} \tag{3.24}$$

Next, AdaBoost updates the probabilities of all the training samples in such a way that the correctly classified samples get a lower weight and the misclassified samples get a higher weight. To be more specific, the probability of each sample correctly classified by $f_r(\cdot)$ is multiplied by $e^{-\alpha_r}$ and the probability of each incorrectly classified sample is multiplied by e^{α_r}. Here α_r is given by:

$$\alpha_r = \frac{1}{2}\log(\frac{1-\epsilon_r}{\epsilon_r}) \tag{3.25}$$

Finally, the probabilities are renormalized so that they again sum up to 1.

After this procedure has been repeated for R iterations, R classifiers $f_1(\cdot), ..., f_r(\cdot)$ and R values $\alpha_1, ..., \alpha_R$ remain. To classify a new sample $\mathbf{d}$, each classifier $f_r(\cdot)$ from $f^*(\cdot)$ is applied to $\mathbf{d}$ resulting in labels $f_1(\mathbf{d}), f_2(\mathbf{d}), ..., f_r(\mathbf{d})$. Unlike bagging, one does not assign equal importance to each of the classification results, but instead weight the results using the α_r values that were previously used to update the probabilities p_{ir}. This means that the final class of $\mathbf{d}$ is given by:

$$f^*(\mathbf{d}) = \text{argmax}_y \sum_{k:f_r(\mathbf{d})=y} \alpha_r \tag{3.26}$$

A main disadvantage with AdaBoost is that it is not very good at solving multi-class problems. In addition to that, it doesn't handle cases where a document may belong to more than one class. In this following section we present an extension of the original algorithm, *the AdaBoost.MH algorithm* [26], that can efficiently handle multi-label problems.

AdaBoost.MH

Let the weight of sample $\mathbf{d}_i$ and label c_k in iteration r be p_{ikr}. Initially, all weights are equal, i.e. $p_{ik1} = 1/N$ for all samples $\mathbf{d}_i$ and all labels c_k. For each round, the AdaBoost.MH algorithm estimates K classifiers $f_r(\mathbf{d}, k)$. The sign of $f_r(\mathbf{d}_i, k)$ reflects whether the label c_k is or is not assigned to training sample $\mathbf{d}_i$, while the magnitude of $f_r(\mathbf{d}_i, k)$ is interpreted as a measure of the confidence in the prediction. The weights are updated using the following formula:

$$p_{ik(r+1)} = p_{ikr} \exp(-y_{ik} f_r(\mathbf{d}_i, k)) \tag{3.27}$$

Here y_{ik} is 1 if label c_k is among the possible true labels of sample $\mathbf{d}_i$ and -1 otherwise. After updating the weights they are renormalised so that $\sum_i \sum_k p_{ik(r+1)} = 1$.

After this procedure has been repeated for R iterations, one has $R \times K$ classifiers $f_r(\mathbf{d}, k)$. To classify a new sample $\mathbf{d}$, each classifier is applied to $\mathbf{d}$ and the final class of $\mathbf{d}$ is given by:

$$f^*(\mathbf{d}, k) = \sum_{r=1}^{R} f_r(\mathbf{d}, k) \tag{3.28}$$

Chapter 4

Performance Measures

An important issue of text categorisation is how to measure the performance of the classifiers. Many measures have been used, each of which has been designed to evaluate some aspect of the categorisation performance of a system. In this chapter we describe some of the measures that have been reported in the literature; Section 4.1 treats the multiple binary classification task, while Section 4.2 describes measures that have been used for evaluating the performance of the multi-class methods.

4.1 Multiple binary classification tasks

A common approach for multi-class categorisation is to break the task into disjoint binary categorisation problems. For each category and each document one determines whether the document belongs to the category or not. When evaluating the performance of the classifiers, four quantities are of interest for each category:

- a - the number of documents *correctly assigned* to this category.
- b - the number of documents *incorrectly assigned* to this category.
- c - the number of documents *incorrectly rejected* from this category.
- d - the number of documents *correctly rejected* from this category.

From these quantities, we define the following performance measures:

$$\text{recall} = \frac{a}{a+c} \tag{4.1}$$

$$\text{precision} = \frac{a}{a+b} \tag{4.2}$$

$$\text{fallout} = \frac{b}{b+d} \tag{4.3}$$

$$\text{accuracy} = \frac{a+d}{a+b+c+d} \tag{4.4}$$

$$\text{error} = \frac{b+c}{a+b+c+d} \tag{4.5}$$

Micro- and macro-averaging For evaluating performance average across categories, there are two conventional methods, namely *macro-averaging* and *micro-averaging*. Macro-averaged performance scores are determined by first computing the performance measures per category and then averaging these to compute the global means. Micro-average performance scores are determined by first computing the totals of a, b, c, and d for all categories and then use these totals to compute the performance measures. There is an important distinction between the two types of averaging. Micro-averaging gives equal weight to every document, while macro-averaging gives equal weight to each category.

Break-even point The performance measures above may be misleading when examined alone. Usually a classifier exhibits a trade-off between recall and precision, to obtain a high recall usually means sacrificing precision and vice versa. If the recall and precision are tuned to have an equal value, then this value is called the *break-even point* of the system. The break-even point has been commonly used in text categorisation evaluations.

F-measure Another evaluation criterion that combines recall and precision is the *F-measure* [18]:

$$F_\beta = \frac{(\beta^2+1) * \text{precision} * \text{recall}}{\beta^2 * \text{precision} + \text{recall}} \tag{4.6}$$

where β is a parameter allowing different weighting of recall and precision.

Interpolation For some methods, the category assignments are made by thresholding a confidence value. For instance the Bayes classifier computes the probability that a document is in the current category, and one has to decide how large this probability must be (specified by a threshold) for the document to be assigned to the category. By adjusting this threshold, one can achieve different levels of recall and precision. In [13] the results for different thresholds are combined using interpolation.

4.2 Multi-class and multi-label classification

To measure the performance of a classifier that produces a ranked list of categories for each test document, with a confidence score for each candidate, an approach called *interpolated*

11-point average precision [29] may be used. In this approach the recall for one specific document is defined to be:

$$\text{recall} = \frac{\text{Number of categories found that are correct}}{\text{Total number of true categories}} \tag{4.7}$$

For each of 11 values 0.0,....,1.0 of this fraction, the system decides how far down the ranked list one has to go (i.e. the size of the numerator) to achieve the specified recall. The precision is then computed for this number of categories by:

$$\text{precision} = \frac{\text{Number of categories found that are correct}}{\text{Total number of categories found}} \tag{4.8}$$

The resulting 11 precision values are averaged to obtain a single number-measure of performance for the document. For a test set of documents, the average precision values of individual documents are further averaged to obtain a global measure of system performance over the entire set.

Chapter 5

Previous work using Reuters-21578 collection

5.1 The Reuters collection

The Reuters-21578 collection is publicly available at:

`http://www.research.att.com/~lewis/reuters21578.html`

In this chapter we only give a short description of this data set, a more thorough description is given in Appendix A.

The documents in the Reuters collection were collected from Reuters newswire in 1987. To divide the collection into a training and a test set, the modified Apte ("ModApte") split has been most frequently used, including the work that is described in this chapter. According to the documentation, using the ModApte split leads to a training set consisting of 9,603 stories, and a test set consisting of 3,299 stories.

135 different categories have been assigned to the Reuter documents. These are listed in Appendix A. While some documents have up to 14 assigned categories, the mean is only 1.24 categories per document. The frequency of occurrence varies greatly from category to category; *earnings*, for example, appears in 2709 documents, while 75 of the categories (i.e. more than 50 %) have been assigned to fewer than 10 training documents. Actually, 21 of the categories have not been assigned to any training document.

5.2 Previous work

In this section we describe previous work on text categorisation where the Reuters-21578 collection has been used to evaluate the methods. The papers that are described are the works of Dumais et al. [10], Joachims [13], Shapire et al. [26], Weiss et al. [27] and Yang [29]. Tables 5.1 and 5.2 summarise the papers. All authors have used the ModApte split. The first table contain the number of training and test documents, and the number of categories that were used by each of the authors. Moreover it specifies the approaches for indexing and feature reduction. Finally, the type of classification (i.e. multiple binary or multi-class) and the evaluation criteria is given.

As it can be seen from Table 5.1, all authors except Yang have used training and test sets of sizes 9,603 and 3,299, respectively. It is not clear how they have treated the stories for which either the topic was missing, or there was no text enclosed by the `<BODY>`, `</BODY>` tags. The number of categories varies from 90 to 118. The question mark in the row corresponding to Shapire et al. is due to the fact that they have not used the standard set of categories.

All authors have used different indexing methods. The names refer to Section 2.2 and a closer description can be found here. Three of the authors have used feature selection. Dumais et al. have used Mutual Information (MI), Joachims has used Information Gain (IG), while Yang has used the χ^2-statistic. The last two approaches are described in Section 2.3, while a description of the MI-approach can be found in [30].

Most of the research in text categorisation has been devoted to binary problems, where a document is classified as either relevant or not relevant with respect to a predefined topic. As it can be seen from Table 5.1, this is also the truth for the papers surveyed here. Only Shapire et al. have considered multi-class and multi-label problems. As far as performance measures are concerned, the precision/recall break-even point has been used by all the authors that have treated the text categorisation as a multiple binary problem. Some of these authors have also used other performance measures, but for the sake of simplicity, we have only considered the break-even point here. Shapire et al.. have used three evaluation measures that are suited for multi-class problems. A further description of these measures can be found in [26].

Table 5.2 lists the methods used by the authors, and the break-even points that were achieved. It should be noted that some of the authors have tested more methods than those that are listed in the table, but we have chosen to compare only the methods that were described in Chapter 3.

An "x" in the table means that the method was tested, but that the performance measure was not the break-even point. Hence, it is difficult to compare the results to those that the other authors have achieved. However, the following gives an indication of the results obtained by Shapire et al.. They report that the error rate of their voting algorithm (AdaBoost.MH) was almost 50 % smaller than the error rate of the best competing method,

Author	*Train*	*Test*	*Topics*	*Indexing*	*Reduc.*	*Method*	*Measure*
Dumais	9,603	3,299	118	boolean	MI	Binary	Break-even
Joachims	9,603	3,299	90	tfc	IG	Binary	Break-even
Shapire	9,603	3,299	?	tf x idf	None	Multiclass	Precision
Weiss	9,603	3,299	95	frequency	?	Binary	Break-even
Yang	7,789	3,309	93	ltc	χ^2	Binary	Break-even

Table 5.1: *Summary of previous work. The three first columns contain the number of training documents, the number of test documents and the number of categories that were used. The fourth column specifies the indexing approach, while the fifth gives the feature selection method, if any. Finally, the last two columns contain the classification method and the performance measures that were used.*

which was Naive Bayes.

All the authors Dumais, Joachims, Weiss and Yang, use the break-even point as their performance measure, and hence, the results may be compared directly. For the Naive Bayes method, the k-NN method, the decision tree, and the SVM method there are small variations in the results obtained by the different authors. The average break-points are 72.9, 84.5, 79.1, and 86.4, respectively, for the four methods. As far as Rocchio's algorithm is concerned there are greater variations. The break-even point reported by Dumais et al. is as low as 61.7, while the average of the values reported by the others, is 77.9. The overall greatest break-even point, 87.8, has been achieved by Weiss et al. for their voting algorithm.

Author	*Rocchio*	*Bayes*	*k-NN*	*Tree*	*SVM*	*Voting*
Dumais	61.7	75.2	-	-	87.0	-
Joachims	79.9	72.0	82.3	79.4	86.0	-
Shapire	x	x	-	-	-	x
Weiss	78.7	73.4	86.3	78.9	86.3	87.8
Yang	75.0	71.0	85.0	79.0	-	

Table 5.2: *Summary of previous work (2). The numbers in the table are the precision/recall break-even points. The "-" means that the method was not tested by the author. "x" means that the method was tested, but that the evaluation criterion was not the break-even point.*

Chapter 6

Experiments

6.1 Data set

Like the previous work described in Chapter 5, we used the ModApte split to divide the Reuters collection into a training set and a test set. According to the documentation, this should lead to a training set consisting of 9,603 stories, and test set consisting of 3,299 stories. However, for a lot of these stories either the topic was missing (even if TOPICS="YES"), or there was no text enclosed within the `<BODY>`, `</BODY>` tags. We chose not to use these stories in our experiments. Hence, they were removed from the data set, leaving us with 7,063 training documents and 2,658 test documents.

6.2 Feature Extraction

The bodies of all documents were converted from the original format (i.e. strings of characters) to a word vector. In what follows we describe the steps of this procedure.

6.2.1 Preprocessing

First the individual words were extracted and the stop words removed using a list consisting of 525 frequent English words. Then word stemming was performed using the Porter stemmer [21]. This procedure resulted in 15,247 unique words.

6.2.2 Indexing

The entropy-weighting (see Section 2.2) was used for word indexing. In this scheme, the weight a_{ik} for word i in document k is given by:

$$a_{ik} = \log(f_{ik} + 1.0) * \left(1 + \frac{1}{\log(N)} \sum_{j=1}^{N} \left[\frac{f_{ij}}{n_i} \log\left(\frac{f_{ij}}{n_i}\right)\right]\right) \tag{6.1}$$

where f_{ik} is the frequency of word i in document k, N is the number of training documents and n_i is the total number of times word i occurs in the whole collection.

6.2.3 Dimensionality reduction

Both feature selection and re-parameterisation were performed. First we removed those words which occurred in only one document, leaving us with 8,315 words. Then latent semantic indexing was performed on the resulting 8,315 $\times$ 7,063 word-document matrix, giving the best "reduced-dimension" approximation to this matrix. To perform the singular value decomposition, we used the Single-Vector Lanczos Method from the SVDPACKC library [1]. The number of calculated singular values (dimensions) was 200.

6.3 Methods

The kNN method is a very simple approach that has previously shown very good performance on text categorisation tasks (See e.g. Chapter 5). Hence, we decided to use this method for classification. Thorough investigations on suitable choices of the value of k for the Reuters collection have been reported in previous papers [29], and the main observation was that the performance of kNN is relatively stable for a large range of k values. We used $k = 30$ in our experiments. It should be noted that for the categories that have very few training samples assigned to it, this value of k will not work very well.

To classify an unknown document vector $\mathbf{d}$, it was first projected onto the span of the training set word vectors. Then, the k nearest training set document vectors were found, using the cosine between two document vectors as a similarity measure. The rank r_c of category c for the input document was computed by:

$$r_c(\mathbf{d}) = \frac{\sum_{i:y_i=c} cos(\mathbf{d_i}, \mathbf{d})}{\sum_i cos(\mathbf{d_i}, \mathbf{d})} \tag{6.2}$$

Here $cos(\mathbf{d_i}, \mathbf{d})$ is the cosine similarity between neighbour i and the input document. If r_c was higher than a threshold value the category c was assigned to the input document, otherwise not. Different values of the threshold gives different results. When the threshold increases, the recall decreases and the precision increases.

6.4 Results

To evaluate the performance of our method, we used precision and recall (See Section 4.1). Table 6.1 shows the summary of recall and precision for the ten most frequent categories. Both precision and recall decreases when the number of training (and the number of test) samples decreases.

For evaluating performance average across categories, we used micro-averaging (See Section 4.1). The overall recall and precision were 79.2 % and 81.8 %, indicating that the recall/precision breakeven point is approximately 80 %. The breakeven point was achieved when using a threshold of 0.2 (see the previous section). Our results are comparable to the results described in Chapter 5. When considering only the 10 largest categories that are listed in Table 6.1, the recall/precision breakeven point was 89 %, and it was obtained using a threshold value of 0.3.

Topic	*Nof train*	*Nof test*	*Recall*	*Precision*
earnings	2709	1014	0.95	0.92
aqusitions	1488	630	1.00	0.91
money-fx	460	133	0.92	0.65
crude	349	160	0.96	0.70
grain	394	130	0.82	0.75
trade	337	106	0.89	0.66
interest	289	95	0.80	0.71
ship	191	82	0.85	0.77
wheat	198	65	0.69	0.73
corn	160	46	0.35	0.76

Table 6.1: *Summary of recall and precision for the 10 most frequent topics using a threshold value of 0.2. The two leftmost columns give the number of training and the number of test samples, respectively.*

Chapter 7

Summary

With the dramatic rise in the use of the Internet, there has been an explosion in the volume of online documents and electronic mail. Text categorisation, the assignment of free text documents to one or more predefined categories based on their content, is an important component in many information management tasks, some examples are real-time sorting of email into folder hierarchies and topic identification to support topic-specific processing operations. This report has reviewed progress in the field with particular emphasis on the work where the Reuters-21578 collection has been used for evaluation.

A typical text categorisation process consists of the following steps; preprocessing, indexing, dimensionality reduction, and classification. In this report different approaches for all these steps have been described. Moreover, a summary of the results from previous work on text categorisation where the Reuters-21578 collection has been used for evaluation is given. The following methods were evaluated: Rocchio's algorithm, Naive Bayes, K-nearest neighbour, Decision Trees, Support Vector Machines, and Voted Classification. They all seemed to perform reasonably well, and neither approach appears to be superior over others.

In this report, we also describe results from our own experiments using the Reuters-21578 collection. In these experiments, the entropy-weighting was used for word indexing. Dimensionality reduction was performed by first removing rare words and then by re-parameterising using latent semantic indexing. The K-nearest neighbour algorithm was used for classification. It is a very simple approach, but has shown to have approximately the same performance as more complicated methods. Our results gave a precision/recall breakeven point of approximately 80 % which is comparable to other studies reported for the Reuters-21578 collection.

Appendix A

Reuters-21578 collection

The Reuters-21578 collection is distributed in 22 files. Each of the first 21 files (reut2-000.sgm through reut2-020.sgm) contain 1000 documents, while the last (reut2-021.sgm) contains 578 documents.

A.1 File format

Each of the 22 files begins with a document type declaration line:

```
<!DOCTYPE lewis SYSTEM "lewis.dtd">
```

Each article starts with an "open tag" of the form

```
<REUTERS TOPICS=?? LEWISSPLIT=?? CGISPLIT=?? OLDID=?? NEWID=??>
```

where the ?? are filled in an appropriate fashion. Each article ends with a "close tag" of the form:

```
</REUTERS>
```

Each REUTERS tag contains explicit specifications of the values of five attributes; TOPICS, LEWISSPLIT, CGISPLIT, OLDID, and NEWID. These attributes are meant to identify documents and groups of documents. The values of the attributes determine how the documents are divided into a training set and a test set. In the experiments described in this report, we used *the modified Apte split*, which is the one that is most used in the literature. This split is achieved by the following choice of parameters:

```
Training Set (9,603 docs): LEWISSPLIT="TRAIN";    TOPICS="YES"
Test Set     (3,299 docs): LEWISSPLIT="TEST";     TOPICS="YES"
Unused       (8,676 docs): LEWISSPLIT="NOT-USED"; TOPICS="YES"
                           or TOPICS="NO"
                           or TOPICS="BYPASS"
```

The attributes CGISPLIT, OLDID, and NEWID was ignored in our experiments.

A.2 Document internal tags

Just as the <REUTERS> and </REUTERS> tags serve to delimit documents within a file, other tags are used to delimit elements within a document. These are <DATE>,<MKNOTE>,<TOPICS>, <PLACES>, <PEOPLE>,<ORGS>,<EXCHANGES>, <COMPANIES>, <UNKNOWN>,<TEXT>. Of these only <TOPICS> and <TEXT> were used in our experiments.

<TOPICS>, </TOPICS>

These tags enclose the list of TOPICS categories, if any, for the document. If TOPICS categories are present, each will be delimited by the tags <D> and </D>.

<TEXT>, </TEXT>

All the textual material of each story is delimited between a pair of these tags. Some control characters and other "junk" material may also be included. The white space structure of the text has been preserved. The following tags optionally delimit elements inside the TEXT element: <AUTHOR>, </AUTHOR>, <DATELINE>, </DATELINE>, <TITLE>, </TITLE>, <BODY>, </BODY>. Of these, we only used the last two (the title is also extracted, but not used for text categorisation), which enclose the main text of the document.

A.3 Example

The first part of a Reuters file will be as follows:

```
<!DOCTYPE lewis SYSTEM "lewis.dtd">
<REUTERS TOPICS="YES" LEWISSPLIT="TRAIN" CGISPLIT="TRAINING-SET" OLDID="5544" NEWID="1">
<DATE>26-FEB-1987 15:01:01.79</DATE>
<TOPICS><D>cocoa</D></TOPICS>
```

```
<PLACES><D>el-salvador</D><D>usa</D><D>uruguay</D></PLACES>
<PEOPLE></PEOPLE>
<ORGS></ORGS>
<EXCHANGES></EXCHANGES>
<COMPANIES></COMPANIES>
<UNKNOWN>
&#5;&#5;&#5;C T
&#22;&#22;&#1;f0704&#31;reute
u f BC-BAHIA-COCOA-REVIEW   02-26 0105</UNKNOWN>
<TEXT>&#2;
<TITLE>BAHIA COCOA REVIEW</TITLE>
<DATELINE>    SALVADOR, Feb 26 - </DATELINE><BODY>Showers continued throughout the week in
the Bahia cocoa zone, alleviating the drought since early
January and improving prospects for the coming temporao,
although normal humidity levels have not been restored,
Comissaria Smith said in its weekly review.
    The dry period means the temporao will be late this year.
    Arrivals for the week ended February 22 were 155,221 bags
of 60 kilos making a cumulative total for the season of 5.93
mln against 5.81 at the same stage last year. Again it seems
that cocoa delivered earlier on consignment was included in the
arrivals figures.
.
.
    Final figures for the period to February 28 are expected to
be published by the Brazilian Cocoa Trade Commission after
carnival which ends midday on February 27.
 Reuter
&#3;</BODY></TEXT>
</REUTERS>
<REUTERS TOPICS="NO" LEWISSPLIT="TRAIN" CGISPLIT="TRAINING-SET" OLDID="5545" NEWID="2">
.
.
.
```

A.4 Categories

The TOPICS categories are economic subject categories. Examples include *coconut*, *gold*, *inventories*, and *money-supply*. This set of 135 categories is the one that has been used in almost all previous research with the Reuters data. Below all the categories are listed:

```
acq   alum   austdlr   austral   barley   bfr   bop   can   carcass   castor-meal
castor-oil   castorseed   citruspulp   cocoa   coconut   coconut-oil   coffee
copper   copra-cake   corn   corn-oil   cornglutenfeed   cotton    cotton-meal
cotton-oil   cottonseed   cpi   cpu   crude   cruzado   dfl   dkr   dlr   dmk
drachma earn   escudo   f-cattle   ffr   fishmeal   flaxseed   fuel gas   gnp
gold   grain groundnut   groundnut-meal   groundnut-oil   heat   hk   hog   housing
income instal-debt   interest   inventories   ipi   iron-steel   jet   jobs
l-cattle   lead lei   lin-meal   lin-oil   linseed   lit   livestock lumber
lupin   meal-feed mexpeso   money-fx   money-supply   naphtha   nat-gas   nickel
nkr   nzdlr   oat   oilseed   orange   palladium   palm-meal palm-oil   palmkernel
peseta   pet-chem   platinum   plywood   pork-belly   potato   propane
rand rape-meal rape-oil   rapeseed   red-bean   reserves   retail   rice   ringgit
rubber   rupiah   rye   saudriyal   sfr   ship   silk   silver   singdlr   skr
sorghum   soy-meal   soy-oil   soybean   stg   strategic-metal   sugar
sun-meal   sun-oil   sunseed   tapioca   tea   tin   trade   tung   tung-oil
veg-oil   wheat   wool   wpi   yen   zinc
```

Bibliography

[1] M. Berry, T. Do, G. O'Brien, V. Krishna, and S. Varadhan, *SVDPACKC (version 1.0) user's guide.*, Technical Report CS-93-194, University of Tennessee, Department of Computer Science, 1993.

[2] M. W. Berry, S. T. Dumais and G. W. O'Brien, *Using linear algebra for intelligent information retrieval*, SIAM Review, Vol. 37, pp. 573–595, December 1995.

[3] L. Breiman, J. H. Friedman, R. A. Olshen and C. J. Stone, *Classification and Regression Trees*, Belmont, CA: Wadsworth, 1984.

[4] L. Breiman, *Bagging predictors*, Machine Learning, Vol. 24, pp.123–140, 1996.

[5] C. Buckley, G. Salton, J. Allan and A. Singhal, *Automatic Query Expansion Using SMART: TREC 3*, In Proc. 3rd Text Retrieval Conference, NIST, 1994.

[6] W. J. Cohen and Y. Singer, *Context-sensitive learning methods for text categorization*, In SIGIR'96: Proc. 19th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, pp. 307–315, 1996.

[7] S. Deerwester, S. Dumais, G. Furnas, T. Landauer and R. Harshman, *Indexing by Latent Semantic Analysis*, Journal of the American Society for Information Science, Vol. 41, No. 6, pp. 391–407, 1990.

[8] R. O. Duda and P. E. Hart, *Pattern Classification and Scene analysis*, John Wiley & Sons, 1973.

[9] S. T. Dumais, *Improving the retrieval information from external sources*, Behaviour Research Methods, Instruments and Computers, Vol. 23, No. 2, pp. 229–236, 1991.

[10] S. Dumais, J. Platt, D. Heckerman, and M. Sahami *Inductive Learning Algorithms and Representations for Text Categorization*, Technical Report, Microsoft Research, 1998.

[11] Y. Freund and R. E. Shapire, *Experiments with a new boosting algorithm.*, in Proc. 13th Int. Conf. on Machine Learning, pp.148–156, 1996.

[12] J. Friedman, T. Hastie and R. Tibshirani, *Additive Logistic Regression: a Statistical View of Boosting.* Tech. report Stanford University, July 23, 1998

[13] T. Joachims, *Text Categorization with Support Vector Machines: Learning with many relevant features*, Technical report, University of Dortmund, 1997.

[14] T. Joachims, *A probabilistic analysis of the rocchio algorithm with TFIDF for text categorization*, In Int. Conf. Machine Learning, 1997.

[15] T. Joachims, *Text categorization with support vector machines: Learning with many relevant features.*, In Proc. 10th European Conference on Machine Learning (ECML), Springer Verlag, 1998.

[16] G. V. Kass, *An exploratory technique for investigating large quantities of data*, Applied Statistics, Vol. 29, pp. 119–127, 1980.

[17] J. T.-Y. Kwok, *Automatic Text Categorization Using Support Vector Machine*, Proc. Int. Conf. on Neural Information Processing, pp. 347–351, October 1998.

[18] D. D. Lewis and W. A. Gale, *A sequential algorithm for training text classifiers*, In SIGIR'94, pp. 3–12, 1994.

[19] D. Lewis and M. Ringuette, *A comparison of two learning algorithms for text classification*, In Third Annual Symposium on Document Analysis and Information Retrieval, pp. 81–93, 1994.

[20] G. W. O'Brien, *Information Management Tools for Updating an SVD-Encoded Indexing Scheme*, Master's thesis, The University of Knoxwille, Tennessee, Knoxwille, 1994.

[21] M. F. Porter, *An Algorithm for Suffix Stripping*, Program 14 (3), pp. 130–137, July 1980.

[22] J. R. Quinlan, *C4.5: Programming for machine learning*, Morgan Kaumann, 1993.

[23] J. Rocchio, *Relevance feedback in information retrieval*, In The SMART Retrival System: Experiments in Automatic Document Processing, pp. 313–323, Prentice-Hall Inc., 1971.

[24] G. Salton and M. J. McGill, *An Introduction to Modern Information Retrieval*, McGraw-Hill, 1983.

[25] G. Salton and C. Buckley, *Term weighting approaches in automatic text retrieval*, Information Processing and Management, Vol. 24, No. 5, pp. 513–523, 1988.

[26] R. E. Shapire and Y. Singer, *BoosTexter: A System for Multi-Label Text Categorization*, Draft, Mars 1998.

[27] S. M. Weiss, C. Apte and F. J. Damerau, *Maximizing Text-Mining Performance*, Will be appearing in IEEE Intelligent Systems 1999.

[28] E. Wiener, J. O. Pedersen and A. S. Weigend, *A neural network approach to topic spotting*, In Proc. 4th annual symposium on document analysis and information retrieval, pp. 22–34, 1993.

[29] Y. Yang, *An Evaluation of Statistical Approaches to Text Categorization*, Technical Report CMU-CS-97-127, Computer Science Department, Carnegie Mellon University, 1997.

[30] Y. Yang and J. P. Pedersen, Feature selection in statistical learning of text categorization, In the 14th Int. Conf. on Machine Learning, pp. 412–420, 1997.