PERGAMON

Information Processing and Management 38 (2002) 305–323

INFORMATION PROCESSING & MANAGEMENT

www.elsevier.com/locate/infoproman

Support vector machines: relevance feedback and information retrieval ☆

Harris Drucker [a,*], Behzad Shahrary [b], David C. Gibbon [b]

[a] *AT&T Research and Monmouth University, West Long Branch, NJ 07764, USA*
[b] *AT&T Research, 200 Laurel Ave., Middletown, NJ 07748, USA*

Received 28 January 2001; accepted 11 May 2001

Abstract

We compare support vector machines (SVMs) to Rocchio, Ide regular and Ide dec-hi algorithms in information retrieval (IR) of text documents using relevancy feedback. It is assumed a preliminary search finds a set of documents that the user marks as relevant or not and then feedback iterations commence. Particular attention is paid to IR searches where the number of relevant documents in the database is low and the preliminary set of documents used to start the search has few relevant documents. Experiments show that if inverse document frequency (IDF) weighting is not used because one is unwilling to pay the time penalty needed to obtain these features, then SVMs are better whether using term-frequency (TF) or binary weighting. SVM performance is marginally better than Ide dec-hi if TF-IDF weighting is used and there is a reasonable number of relevant documents found in the preliminary search. If the preliminary search is so poor that one has to search through many documents to find at least one relevant document, then SVM is preferred. © 2002 Elsevier Science Ltd. All rights reserved.

Keywords: Information retrieval; Support vector machines; Relevancy feedback; Rocchio; Ide dec-hi

1. Introduction

1.1. Statement of the problem

Our problem is that of relevancy feedback within the context of information retrieval (IR). There is a set of documents that a user wants to retrieve within a database. Some of the articles are

☆ A much shorter version of this paper was presented at the 2001 International Conference on Machine Learning.

* Corresponding author. Tel.: +1-732-571-3698; fax: +1-732-571-5253.

E-mail addresses: drucker@monmouth.edu (H. Drucker), behzad@research.att.com (B. Shahrary), dcg@research.att.com (D.C. Gibbon).

relevant, some not. It is important to understand that relevancy is relative to the perception of the user, that is document D_j may be relevant to user U_k but not user U_p.

The user is assumed to present a preliminary (or initial) query to the system, in which case the system returns a set of ranked documents that the user examines. Although many documents may be retrieved by the system, the system only presents one screen of documents at a time. In our case, we assume that ten documents are returned on the initial screen with enough information for the user to gauge whether a document is relevant or not. An optimal preliminary query would only return relevant documents and all the user has to do is scroll through all the screens to find all the relevant documents. In actuality, depending on the quality of the initial query, many documents may be returned but few may be relevant. The initial query may be a Boolean query such as conjunction or disjunctions of key words or the inquiry could be a sophisticated inquiry in the form of a question.

In our case, we ignore the exact nature of the preliminary query and assume that the return of the documents from this initial query is poor (three or less relevant documents from the full screen of ten documents). However, our technique will work no matter how many documents are returned from the initial query. We believe that a hard test of an IR system with relevancy feedback occurs when the number of relevant documents returned in the initial query is low. We assume that if the documents returned in the initial screen are all relevant, then the user will just scroll to the next screen while if no relevant documents are returned, the user continues to scroll through the screens until at least one relevant document is returned on a screen and then the first feedback iteration begins. Thus if there are between one and nine relevant documents returned on the initial screen, the user marks the relevant documents (unmarked documents are taken as non-relevant) and the system goes though a first feedback iteration and another set of the top ten ranked documents are returned. These feedback iterations continue until the user terminates the procedure.

We first concentrate on the state when between one and nine (inclusive) relevant documents are returned in the initial screen. Our method will be based on the use of support vector machines (SVMs) (Drucker, Wu, & Vapnik, 1999; Joachims, 1998; Vapnik, 1998) with comparisons to other IR relevancy feedback techniques: Rocchio (1971), Ide regular and Ide dec-hi (Salton & Buckley, 1990; Harman, 1992). These algorithms will be examined in detail later but suffice to say now that all except Ide dec-hi use all the relevant and non-relevant documents on the first screen while Ide dec-hi uses all the relevant documents and the top ranked non-relevant document.

Recall that we are paying particular attention to the case where the initial retrieval is poor. As anyone who has done IR or web searches will attest, it is rather discouraging to get a return of a search stating that the search has found thousands of documents when in fact most of the documents on the first screen (the highest ranked documents) are not relevant to the user. Our typical user is hypothesized as preferring to mark the top ten returned documents as relevant or not and going through a series of feedback iterations rather that examining many screens to mark all the relevant documents.

Summarizing: In the initial preliminary search we obtain either (1) no relevant documents, (2) one to nine relevant documents or (3) all relevant documents. In case (1), we will be forced to go to succeeding screens until we get one screen with at least one relevant document. All the documents on that last screen and previous screens will be used in the first feedback iteration. In case (2) we mark the relevant documents on the first screen (unmarked on the first screen are non-relevant) and go through feedback iterations. In case (3) we go to the next screen. We will concentrate on the situation when the number of relevant documents returned on the first screen is low (three or less) and could be zero. Please distinguish between the preliminary query that returns the first set of documents and the first feedback iteration which starts with that initial set of documents marked by the user.

After the first feedback iteration and on all subsequent iterations we will examine only the first screen no matter how many of the returned documents are relevant (even if none). We then track performance as a function of feedback iteration.

In Section 1.2 we discuss SVMs, the Rocchio algorithm and Ide algorithms. Section 2 discusses the various options for the terms in the documents vectors, namely binary term weights, term frequency and inverse term frequency. We discuss the concepts of stemming and the use of stop lists in Section 3. In Section 4 we describe the performance metrics of precision, recall and coverage ratio and argue that coverage ratio is the best metric to compare performance. In Section 5 we describe the test set and in Section 6 we describe our experiments using a random set of relevant and non-relevant documents. However, in Section 7, we describe a set of experiments where the preliminary documents are determined from a keyword search. Finally, we reach our conclusions in Section 8.

1.2. Techniques investigated

One difference between our study and others is the simultaneous tracking of performance as a function of feedback iteration and the use of SVMs. Although there have been many studies of the use of SVMs in text retrieval (Drucker et al., 1999; Joachims, 1998; Vapnik, 1982, 1998), most studies emphasize finding the method that optimizes performance after one feedback iteration.

SVMs have been studied in the context of the IR filtering problem (Dumais, Platt, Heckerman, & Sahami, 1998; Joachims, 1998). It is understood that both relevancy feedback and filtering problems are both classification problems in that documents (in our case) are assigned to one of two classes (relevant or not). However, in the filtering situation, we usually have a marked set of documents termed the training set and use that set to train a classifier. Performance is then judged on a separate test set. In a sense, filtering could be considered relevancy feedback with only one feedback iteration. The problem with using filtering rather than many iterations of relevancy feedback is that (1) one has to mark "many" documents in the training set to obtain reasonable classification rates on the test set (2) how many is "many" depends on the problem and is not known in advance (3) since what are to be considered relevant documents is user dependent, this would mean that every user must construct a different training set. Multiple iterations of feedback could be considered to be an attempt to maximize performance on a test set that includes all documents except the ones already marked. In that sense, relevancy feedback is similar to what is termed active learning (Schohn & Cohen, 2000; Tong & Koller, 2000) in that we try to maximize test performance using the smallest number of documents in the training set. However the important difference between relevancy feedback and active learning is that active learning may force the user to mark many more non-relevant documents than in IR feedback and our supposition is that the user wants to see the maximum number of relevant documents at each feedback iteration. Additionally, in active learning we are interested in maximizing performance on the entire test set. In our case we are interested in maximizing performance on the next ten documents retrieved.

IR and relevancy feedback has a long history. In the Rocchio (1971) algorithm formulation we have a set of documents, each document represented by a vector $\mathbf{D}_j$ whose size is the size of the vocabulary of words in all the documents after pruning some words. An element of this vector indicates some property of a particular word that occurs in the article – zero if it does not appear, "1" if it appears and we are using binary features, and the number of occurrences of that word in the article if we are using term-frequency (TF) weighting. The preliminary query (not necessary returned by a Rocchio feedback iteration) contains N total documents. If the preliminary search realizes between one and nine relevant documents (and the resultant number of non-relevant documents), then N is ten. If there are no relevant documents on the first screen then we search subsequent screens until there is at least one relevant document – in this case N is a multiple of ten. We will ignore the case of ten relevant documents returned on the preliminary search because then the preliminary query is very good and one just goes to subsequent screens to retrieve more relevant documents.

The first feedback iteration using Rocchio after the initial (non-Rocchio) search forms the following query:

$$\mathbf{Q}_1 = \frac{\beta}{R} \sum_{\text{Relevant}} \mathbf{D}_i - \frac{\gamma}{N-R} \sum_{\text{Non-relevant}} \mathbf{D}_i.$$

β and γ are constants used to assign the relative importance of the relevant and non-relevant documents to the query. R is the number of relevant documents retrieved in the preliminary query and N is the total number of documents retrieved in the preliminary query. Negative elements of the vector are clipped to zero (Rocchio, 1971). To implement the first iteration we form the dot product of this first query against all the documents $(\mathbf{Q}_1 \cdot \mathbf{D}_j)$ where the documents are those not yet marked as relevant and non-relevant and then we rank the dot products from high to low, present the ten largest dot products for the user to mark as relevant and non-relevant and then continue to the next iteration.

After the first feedback iteration, we form subsequent iterations:

$$\mathbf{Q}_j = \alpha \mathbf{Q}_{j-1} + \frac{\beta}{R} \sum_{\text{Relevant}} \mathbf{D}_i - \frac{\gamma}{N-R} \sum_{\text{Non-relevant}} \mathbf{D}_i,$$

where α represents the relative importance of the prior query. We have a number of concerns with the Rocchio algorithm that we feel will make it problematic for use; mainly that it depends on three constants (α, β, γ). Most studies on the Rocchio algorithm vary the three constants to determine the optimum choice of these constants. However, we feel that is unfair – a separate validation set should have been used. Furthermore, even if one has a validation set, one does not have time to search for that optimum set of constants. Thus, we will use the following choices of α, β, γ as 8, 16, and 4, respectively, a choice that seemed to work well elsewhere (Buckley, Salton, & Auen, 1994). Since all the negative elements of the resultant query are set to zero and since we are assuming that most of the documents returned in the first iteration will be non-relevant, there may be many elements of the query set to zero making for very poor performance on the next iteration.

Schapire, Singer, and Singhal (1998) investigate a modified Rocchio algorithm and boosting as applied to text filtering. Although their investigation was not in the relevancy feedback domain, they did show that a modified Rocchio algorithm could do much better than the original Rocchio algorithm. However, their algorithm requires multiple passes over documents and is problematic for large databases. Joachims (1997) compared Rocchio and naïve Bayes in text categorization while Salton and Buckley (1990) examine Rocchio and probabilistic feedback but only for one feedback iteration.

The Ide regular algorithm (Salton & Buckley, 1990; Harman, 1992) is of the following format:

$$\mathbf{Q}_j = \mathbf{Q}_{j-1} + \sum_{\text{Relevant}} \mathbf{D}_i - \sum_{\text{Non-relevant}} \mathbf{D}_i,$$

where for the first feedback iteration, the $\mathbf{Q}$ on the right-hand side is zero and as usual, all negative elements of the resultant query are set to zero.

The Ide dec-hi has basically the same form as Ide regular except the last summation has only one term, namely the highest ranked non-relevant document.

The final technique will be based on the use of SVMs. SVMs can best be understood in the context of Fig. 1 where the black diamonds represents the relevant vectors $\mathbf{D}$ in high dimensional space and the empty diamonds represent the non-relevant documents and it is assumed in this figure that the document vectors are linearly separable.

When SVMs are constructed, two sets of hyperplanes are formed (the solid lines), one hyperplane going through one or more examples of the non-relevant vectors and one hyperplane going through one or more examples of the relevant vectors. Vectors lying on the hyperplanes are termed support vectors and in fact define the two hyperplanes. If we define the margin as the orthogonal distance between the two hyperplanes, then a SVM maximizes this margin. Equivalently, the optimal hyperplane (the dashed line half-way between the support hyperplanes) is such that the distance to the nearest vector is maximum.

Before we introduce the key concepts it should be noted that if we followed typical convention we should use lower case bold characters as vectors and upper case bold characters as matrices. However, the common convention in IR seems to be to use upper case bold $\mathbf{D}$ as the document vector. The key concepts we want to use are the following: There are two classes: $y_i \in \{-1, 1\}$ where +1 is assigned to a document if it is relevant and -1 if the class is non-relevant and there are N labeled training examples:

$$(\mathbf{D}_1, y_1), \ldots, (\mathbf{D}_N, y_N) \quad \mathbf{D} \in R^d,$$

where d is the dimensionality of the vector.

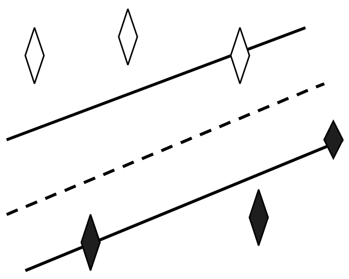
Fig. 1. Support vectors and separating hyperplanes.

If the two classes are linearly separable, then one can find an optimal weight vector $\mathbf{Q}^*$ that describes an optimal separating hyperplane such that the distance from the separating hyperplane to the closest vector of any class is maximum. These conditions are as follows:

$$\begin{aligned} &\mathbf{Q}^* \cdot \mathbf{D}_i - b \geqslant 1 && \text{if } y_i = 1, \\ &\mathbf{Q}^* \cdot \mathbf{D}_i - b \leqslant -1 && \text{if } y_i = -1 \end{aligned}$$

or equivalently:

$$y_i(\mathbf{Q}^* \cdot \mathbf{D}_i - b) \geqslant 1,$$

where b is the bias.

Training examples that satisfy the equality are termed support vectors. The support vectors define two hyperplanes, one that goes through the support vectors of one class and one goes through the support vectors of the other class. The orthogonal distance between the two hyperplanes defines the margin and this margin is maximized when the norm of the weight vector $\|\mathbf{Q}^*\|$ is minimum. Vapnik (1998) has shown we may perform this minimization by maximizing the following function with respect to the variables α_j:

$$W(\alpha) = \sum_{i=1}^{N} \alpha_i - 0.5 \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j (\mathbf{D}_i \cdot \mathbf{D}_j) y_i y_j$$

subject to the constraints that $\sum_{i=1}^{N} \alpha_i y_i = 0$ and $\alpha_i \geqslant 0$ where it is assumed there are N training examples, $\mathbf{D}_i$ is one of the training vectors and $\cdot$ represents the dot product. If $\alpha_i > 0$ then it can be shown that $y_i(\mathbf{Q}^* \cdot \mathbf{D}_i - b) = 1$ and the D_i that corresponds to the nonzero α_i is a support vector. For an unknown vector $\mathbf{D}_j$ classification then corresponds to finding:

$$F(\mathbf{D}_j) = \operatorname{sgn}\{\mathbf{Q}^* \cdot \mathbf{D}_j - b\},$$

where

$$\mathbf{Q}^* = \sum_{i=1}^{r} \alpha_i y_i \mathbf{D}_i$$

and the sum is over the r support vectors taken from the training set. The advantage of the linear representation is that $\mathbf{Q}^*$ can be calculated after training and classification amounts to computing the dot product of this optimum weight vector with the input vector.

For the non-separable case, training errors are allowed and we now must minimize: $\|\mathbf{Q}^*\|^2 + C\sum_{i=1}^{N} \xi_i$ subject to the constraint:

$$y_i(\mathbf{Q}^* \cdot \mathbf{D}_i - b) \geqslant 1 - \xi_i, \quad \xi_i \geqslant 0,$$

where ξ_i is a slack variable and allows training examples to exist in the region (margin) between the two hyperplanes that go through the support points of the two classes. We can equivalently maximize $W(\alpha)$ but the constraint is now $0 \leqslant \alpha_i \leqslant C$ instead of $0 \leqslant \alpha_j$. Maximizing $W(\alpha)$ is quadratic in α subject to constraints and may be solved using quadratic programming techniques, some of which are particular to SVMs (Joachims, 1998). Details for solving this problem also may be obtained from Vapnik (1982, 1998), Cristianini and Shawe-Taylor (2000) and Cherkassky and Mulier (1998).

Linear SVMs execution speeds are very fast and there is only one parameter to tune (C), which is a restriction on the largest value of α. In most learning algorithms, if there are many more examples of one class than another, the algorithm will tend to correctly classify the class with the larger number of examples, thereby driving down the error rate. Since SVMs minimize the error rate by trying to separate the patterns in high dimensional space, the result is that SVMs are relatively insensitive to the relative numbers of each class. For example, new training examples that are "behind" the support vectors of the same class will not change the optimum hyperplane since their values of α are zero.

In our model of relevancy feedback, after construction of $\mathbf{Q}^*$ and b, we calculate $\mathbf{Q}^* \cdot \mathbf{D}_i - b$ for all documents *not* seen so far and rank them from high to low (assuming the relevant documents are of class +1) and return the top ten to the user for marking. (Strictly speaking, the bias term b is not needed since the rankings will remain the same whether the bias is subtracted from the dot product or not). $\mathbf{Q}^* \cdot \mathbf{D}_j - b$ represents the proportional distance from the optimal separating hyperplane to the vector $\mathbf{D}_j$. The documents not used in the training set may be inside or outside the margin (since they were not used to generate the present support vectors). Those documents on the class +1 side of the optimal hyperplane and furthest from the optimal hyperplane are the top ranked documents. Some of these top-ten ranked documents may in fact be non-relevant and in the next feedback iteration these newly marked vectors (in addition to those marked in previous feedback iterations) are used to construct a new set of support vectors. We contrast this with active learning (Schohn & Cohen, 2000; Tong & Koller, 2000) where the emphasis will be to take vectors in the next feedback iteration from within the margin. We don't want to do this in relevancy feedback, as many of the points within the margin will be non-relevant and not useful to the user. If the top-ten ranked documents are outside the margin and are all relevant, then in the next feedback iteration the support vectors will not change. If any of the top ten documents are within the margin, the next set of support vectors will be different from the present set.

Solving the previous set of equations is done using SVM$^{\text{light}}$ (see Acknowledgement section). There are not many candidate vectors to consider as support points. In general the number of potential support points is ten times the iteration number and the training time is usually under three seconds although in some cases, it takes longer for the algorithm to converge (up to 30 s).

Joachims (1998) looked at SVMs in text categorization and compared this to naïve Bayes, C4.5, k-nearest neighbor, and Rocchio. Although not a relevancy feedback study, it discusses the issue of whether all or just some of the features should be used (features are the elements of the vectors). Although reducing the number of features does improve performance on some algorithms (k-nearest neighbor, C4.5 and Rocchio), it does not for naïve Bayes and SVM. It is our contention that we cannot waste time searching for the best set of features and so we use the full set of features in our study.

Others studies in IR that are relevant are that of Buckley, Salton, and Allen (1993) who examined IR within the context of a routing problem. They use the Rocchio algorithm modified so that the last term in the equation defining the new query includes not only the non-relevant documents marked on the present screen but assumes that all unseen documents are non-relevant and included in the last term. The same three authors (1994) also examined the use of locality information to improve performance. We mention the study of incremental feedback (Aalbersberg, 1992) where only the top ranked document is retuned to the user and marked as relevant or not and is another example of text categorization. Finally, it should be pointed out that all the techniques discussed here are vector techniques. Harman (1992) compares many models including probabilistic models (as opposed to vector space models) and is the only paper we could find that tracks performance as a function of iteration.

2. Term weighting issues

We discuss the issue of the term t_i in the document vector $\mathbf{D}_j$. In the IR field, this term is called the term weighting while in the machine learning field, this term is called the feature and in linear algebra it is the ith element of the vector $\mathbf{D}_j$. t_i states something about word i in document $\mathbf{D}_j$. If this word is absent, t_i is zero. If the word is present, then there are several options. One option is that the term weight is a count of the number of times this word occurs in this document (called the TF). The next option is that this term just indicates whether this word is present or not (binary term weighting). In the original Rocchio algorithm, each term TF is multiplied by a term $\log(N/n_i)$, where N is the total number of documents in the collection and n_i is the number of documents in which this term occurs. This last term is called the inverse document frequency (IDF). Usually $\mathbf{D}_j$ is normalized to unit length.

A popular combination for feature i is the multiplication of the TF by the IDF (TF-IDF). Salton and Buckley (1988) discuss in detail various term weighting options. Schapire et al. (1998) also look at different term weighting options. However, using TF-IDF requires two passes over the data because IDF cannot be determined until all the words and the number of articles in which that word is present is calculated. A compensating fact is that the IDF determination need only be done once for any static collection of documents. However, if we add (or eliminate) articles from a database, then IDF must be updated periodically (as must be the dictionary). Although we will evaluate algorithms using TF-IDF we would prefer not to use it in practice. The question will be whether using TF-IDF is much better than using just TF. Buckley et al. (1994) use TF-IDF for the preliminary query but not for the documents themselves since the determination of IDF is quickly done for the small numbers of queries as opposed to the large number of documents.

3. Stemming and stop lists

Full stemming is the removal of all suffixes of a word. For example, “builder”, “building”, and “builds” will all be reduced to their common root “build”. There could also be partial stemming such as changing plural forms to their singular. Stemming reduces the size of the document vectors. One performance issue will be the effect of stemming on retrieval accuracy. But there are other performance issues such as retrieval speed and size of the inverted index. Buckley et al. (1993, p. 68), discuss these options in detail.

Another technique to reduce the dimensionality is the removal of “stop” words, that is, a list of words that will not be used in constructing the document vector. The use of a stop list may or may not improve performance. Words like “a”, “an”, and “the” probably do not help in determining relevancy. If the stop list consists of an a priori list of words, then only one pass over the documents is needed. However, if the stop list is based on the proportion of documents that has certain words, then two passes are required over the documents – once to count the number of all words and the second pass to eliminate either very common words or very rare words. One version of this is to remove words that do not occur in at least three documents (Drucker et al., 1999). This eliminates rare or misspelled words.

We tried the following combinations of pre-processing techniques and algorithms

1. Stemming using the Porter (1980) stemmer or not.
2. Eliminating words that do not occur in at least three documents or not.
3. TF, TF-IDF, or binary features.
4. Four algorithms: Rocchio, Ide regular, Ide deci-hi and SVMs.

Number 1 can be done "on the fly" but number 2 and TF-IDF of number 3 requires two passes over the document. In all cases, all one or two letter words were eliminated, words reduced to lower case, and the words "and" and "the" were eliminated.

Although Drucker et al. (1999) had shown that TF-IDF is not necessary for SVMs in a classification task, we investigate that option for relevancy feedback. Salton and Buckley (1988) showed that using binary features gives the worst performance in comparison to TF or TF-IDF and we do not consider that option. Recall that we would prefer not to use TF-IDF because it requires two passes to calculate IDF.

4. Performance metrics

There are too many ways to assess the effectiveness of the feedback process to discuss here in detail. References are Lewis (1995), Tauge-Sutcliffe (1992), Saracevic (1975), Mizzaro (1997), and Korfhage (1997). However, traditionally recall and precision are used. Let R be the number of relevant documents in the collection, n_{Rel} be the number of relevant documents actually retrieved in a feedback iteration and N be the total number of documents returned in the feedback iteration (typically, N is 10 here). Precision (p) and recall (r) are then defined as

$$p = n_{\mathrm{Rel}}/N, \quad r = n_{\mathrm{Rel}}/R.$$

Although we assume N is ten if one through nine relevant documents are returned on the initial preliminary search, if we did change N, both the recall and precision will change (because n_{Rel} does) and at some point both the recall and precision will be approximately equal and this is termed the recall–precision break-even point and is a popular measure of performance. Schapire et al. (1998) give very reasonable arguments why conventional metrics such as the precision–recall break-even point are not very informative to the user. In particular, we concur with their contention that since recall cannot be calculated by the user until all relevant documents are seen by the user (if this is even possible except in an exhaustive search), the user cannot know (in terms of recall) how well the IR search is going.

For these reasons, we will emphasize the coverage ratio as the performance metric. Coverage ratio is a cumulative metric and is the ratio of the cumulative total number of relevant documents retrieved so far to the cumulative number of relevant documents that would have been retrieved in an ideal search. The coverage ratio is ideally unity at each iteration.

To take into account of the fact that at some point an ideal IR relevancy feedback system will run out of relevant documents to retrieve, we define the coverage ratio as

$$\text{coverage ratio} = \begin{cases} \sum n_{R_i}/10i & \text{when } 10i \leqslant R, \\ \sum n_{R_i}/R & \text{otherwise}, \end{cases}$$

where R is the total number of relevant documents in the entire collection excluding those found in the preliminary search and n_{R_i} is the number of relevant documents at iteration i. Do not confuse this definition of R with that used in the Rocchio algorithm where R there is the number of relevant documents returned at the present iteration. The user can only calculate the top ratio because the user has no way of knowing if a decreasing coverage ratio is due to poor performance of the algorithm or if the system is running out of relevant documents. Basically the user probably does not care – to the user, a declining precision and coverage ratio means that no more relevant topics are being retrieved. Precision and coverage ratio are a measure of user satisfaction because the user would like to see all relevant documents returned per feedback iteration (precision) and cumulatively (coverage ratio). Until the ideal search would run out of documents to retrieve, coverage is the precision averaged over prior iterations and thus makes the coverage ratio a much smoother function than the (instantaneous) precision that we have observed to be quite erratic. From a system perspective an alternative definition of coverage ratio could be:

$$\text{coverage ratio} = \sum n_{R_i}/R.$$

However once again the user has no way of knowing the number of relevant documents R in the entire collection.

5. A test set

A set of documents labeled by topic can be used to simulate the relevance feedback environment. For a test set we use the Reuters corpus of news articles (www.reseach.att.com/~lewis). This is a database of over 11,000 articles. Each article is delimited by SGML tags to indicate (among other items) the beginning and end of the article and the topic(s) assigned to that article. Some articles have multiple topics. Processing of the database proceeded as follows:

1. Eliminate articles that have no topics to give us a total of 11,367 articles and 120 unique topics.
2. Eliminate the words "the" and "and", all punctuation and all all-number words.
3. Eliminate all one and two letter words.
4. Change all words to lower case.

We then generated a vocabulary for the database in four different versions

1. Full stemming.
2. Full stemming and removal of words that did not occur in at least three articles. This tends to eliminate misspelled words from the vocabulary. It could also remove rare words that are useful in locating certain topics. This tradeoff will be evaluated. Elimination of these words requires two passes over the database and would not be used unless the performance improves significantly over the no-removal case.
3. No stemming.
4. No stemming and removal of words that did not occur in at least three articles.

The sizes of the vocabularies for these four cases are indicated in Table 1.

Table 1
Number of unique words after processing

Feature processing type	Number of unique words
Stemming	27,478
No stemming	31,487
Stemming and elimination of words that do not occur in at least three articles	10,410
No stemming and elimination of words that do not occur in at least three articles	12,764

For each of the four cases above, vectors were formed for each article whose size is that of the vocabulary size in Table 1. The element i for a vector corresponding to a particular article indicates whether the word i is present or not (binary features), or how many times that word occurs (TF features) or IDF normalized (TF-IDF). In all cases, the magnitudes of the vectors were unity.

We define the visibility of a topic as the percent of total documents that have that topic. We will track both precision and coverage ratio as a function of iteration parameterized in the following way: (1) visibility of the topic and (2) the number of documents returned in the preliminary search. The number of returned documents in the preliminary search will be restricted to one or three. Later, we will discuss the issue of no returned documents in the preliminary search. Although we decline to report averages because we believe averages conceal the poor performance of algorithms on low-visibility documents, we do not have the space to report the metrics for all topics (nor would that be especially illuminating) and therefore we report the metrics for a sample of topics ranging from common to rare (Table 2). In other words, we will first assume that topic number one is the relevant topic and all others are non-relevant. Then we will assume topic number five is the most relevant topic and all others non-relevant, etc.

Examining Table 2, since all “corn” and “soybean” topics are also “grain” topics, all soybean and corn articles will have at least two topics. Thus, when we consider “corn” as the relevant topic, we consider all “grain” articles that are also “corn” articles as relevant while “grain” articles that are not “corn” articles are non-relevant. gnp, earn, lei topics are short for “gross national product”, “earnings” and “leading economic indicators”.

Table 2
Some of the topics in the database

Rank	Topic name	Number	Visibility (%)
1	Earn	3987	33
5	Grain	628	5.5
10	Corn	254	2.2
15	Gnp	120	1.4
20	Soybean	78	0.68
30	Iron–steel	67	0.58
40	Palm oil	43	0.43
50	Fuel	28	0.24
60	Lei	17	0.15
70	Rapeoil	8	0.07

Metrics reported are the results of averaging ten experiments (and twenty experiments for the rarer cases) with the same initial number of preliminary documents, the same visibility, the same preprocessing and the same algorithm. Preprocessing includes the choice of vocabulary (Table 1) and whether using binary, TF, or TF-IDF weighting. The only difference between each experiment is that the specific preliminary documents for the preliminary search are different and picked at random. Picking a set of preliminary documents may not quite mimic what happens in a realistic setting. For instance, if a Boolean query is run to get the initial set of documents, these documents are more similar than just picking documents at random with the same topic assignment.

In comparing two different algorithms, we use the same random seed for the same experiment number. For example, in comparing experiment number 1 using Rocchio vs. experiment number 1 using Ide regular, we start with the identical relevant and non-relevant preliminary documents. Thus if there is a difference between the averages using the two algorithms, it can only be attributed to the algorithm since we start with the same initial set of documents, the same assignment of term weighting (whether TF, TF-IDF, or binary), same choice of stemming or non-stemming and the same visibility. For visibility corresponding to the 1st, 5th, 10th, and 15th most visible documents, we will go through ten feedback iterations. Starting at the 15th most visible document, we will go through twenty feedback iterations because by this point, in an ideal algorithm, all the documents with that topic should have been retrieved (Table 2). Whether one algorithm is better than another will be based on the coverage ratio after the last feedback iteration.

6. Experimental results

In Fig. 2, we assume one document retrieved on the initial search for two cases: one case where the topic has a high visibility (33%) and the other case with low visibility (1.4%). For each case, we show the results of two algorithms: SVM using binary features and Rocchio using TF-IDF averaged over ten experiments.

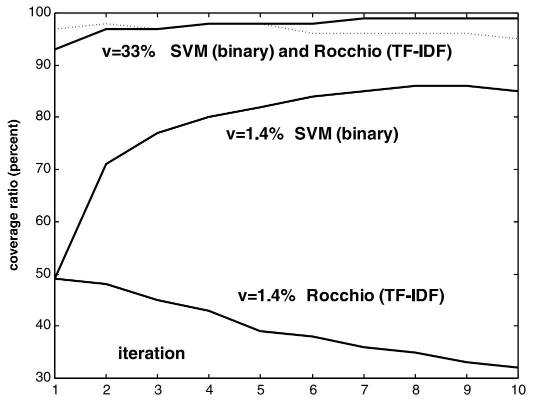

Fig. 2. Coverage ratio versus number of iterations for two different visibilities. The dashed line is for Rocchio using TF-IDF features. One document retrieved in preliminary search.

Let us discuss the high visibility case first ($v = 33\%$). Since the two algorithms have almost identical performance, we have used one label to identify the top two graphs. Recall that precision is identical to the coverage ratio in the first feedback iteration. Both algorithms do very well on the first iteration returning on the average approximately 9.5 relevant documents on a screen of ten documents. At the final iteration, the coverage ratio is approximately 95% for both algorithms indicating that by the tenth iteration an ideal case would return one hundred documents and both these algorithms return cumulatively about 95 documents by the tenth iteration. Therefore both these algorithms do approximately equally well.

Now let us examine the case where the visibility is low. At the first iteration the precision is about 50% indicating, that on the average, five of ten relevant documents would be returned for both these algorithms when one has only one relevant document returned on the preliminary search. However, by the tenth iteration, SVM will have returned approximately eighty-five of the hundred that possibly could be returned in the ideal case while Rocchio will have returned only thirty of hundred relevant documents. Since, for both the high and low visibility cases, there are more than a hundred relevant documents in the database, the coverage at the last iteration is the precision averaged over the ten experiments. That is, on the average, for the high visibility case using either of the two techniques, the average number of relevant documents returned on each of the ten iterations is 9.5. However, for the low visibility case, on the average only three relevant documents out of ten documents will be returned at every iteration using Rocchio while for SVM, on the average 8.5 relevant documents will be returned.

Tables 3 and 4 indicate the coverage ratio at the end of the tenth iteration using stemmed and non-stemmed vocabularies, respectively, and assuming one relevant document returned in the initial search. Ide regular is not even reported since its performance is abysmal (zero at the end of ten iterations) probably due to the fact that there are many non-relevant documents used in the initial iteration which forces negative term weights in the query that are then clipped to zero. In Rocchio this effect is reduced because the non-relevant summation term is divided by the number of non-relevant documents while in Ide dec-hi only one non-relevant document is used.

There is not enough evidence here to recommend any particular weighting scheme for the SVM algorithm, at least on a performance issue. Ide dec-hi (with TF-IDF) does slightly better than the other two Rocchio-type algorithms except for the high visibility case. Based on these tables, SVM is much better and it does not matter whether one stems or not. In using non-TF-IDF features (because of the one-pass requirement to obtain them), since there is no difference in SVMs whether one uses binary or TF features, one might as well use binary features as they are easier to obtain and use stemming as this reduces the vocabulary size.

Table 3
Coverage ratio at the tenth iteration using stemmed data. One relevant and nine non-relevant documents assumed returned at the initial search. Values are in percent

Visibility	SVM binary	SVM TF	SVM TF-IDF	Ide hi TF-IDF	Ide hi TF	Rocchio TF-IDF
33	99	100	100	95	95	95
5.5	86	87	95	57	44	51
2.2	75	74	81	37	27	29
1.4	85	85	86	34	32	32

Table 4
Coverage ratio at the tenth iteration using non-stemmed data. One relevant and nine non-relevant documents assumed returned at the initial search. Values are in percent

Visibility	SVM binary	SVM TF	SVM TF-IDF	Ide hi TF-IDF	Ide hi TF	Rocchio TF-IDF
33	99	100	99	95	95	96
5.5	88	88	94	57	45	50
2.2	74	75	76	38	28	29
1.4	83	86	87	36	33	33

We also did experiments that created tables similar to those of Tables 3 and 4 where we assumed three relevant articles returned in the initial search. The relative performance among algorithms remained the same although absolute performance increased slightly except for the highest visibility case.

Although we would prefer not to remove words from a dictionary that do not occur at least three times (because it requires two passes over the data), it may be worthwhile if performance is enhanced or the performance is about the same since this reduces the size of the vocabulary (Table 1). Table 5 gives the performance for this type of preprocessing and should be compared to Table 3. SVM (binary and TF-IDF weighting) and the two best non-SVM algorithms were used. This table shows that the SVM performance is degraded so that one should not remove words that do not occur in at least three articles. Rocchio gives better performance except for the highest visibility case when words that do not occur in at least three documents are removed.

Precision can be calculated directly by the user. However, because the precision may vary dramatically from one iteration to the next, it is not a good measure of system performance. Coverage ratio is a smoother function. In addition, if the precision starts to decline, the user still does not know whether the decline is due to the algorithm or absence of relevant documents. As an example (Table 6) of this, we show the precision at the tenth iteration assuming one initial document returned in the preliminary search with stemming. The SVMs are superior to all the other algorithms. For SVM-TF and a visibility of 2.2%, we get a precision of 55% that seems out of place compared to the cell immediately to the left. However, examining the data one iteration before, the precision is there 75%. The precision may become erratic and therefore the coverage ratio is preferred as a metric as it is smoother.

Table 5
Coverage ratio at the tenth iteration using stemmed data and removing all words that did not occur in at least three articles. One relevant document assumed returned at the initial search. Values are in percent

Visibility	SVM binary	SVM TF-IDF	Ide dec-hi TF-IDF	Rocchio TF-IDF
33	99	95	95	95
5.5	86	95	57	55
2.2	71	78	37	42
1.4	84	85	33	34

Table 6
Precision at the tenth iteration using stemmed data. One relevant document assumed returned at the initial search. Values are in percent

Visibility	SVM binary	SVM TF	SVM TF-IDF	Ide hi TF-IDF	Ide hi TF	Rocchio TF-IDF
33	100	100	100	96	93	91
5.5	94	97	100	34	33	35
2.2	73	55	79	26	15	23
1.4	75	79	86	33	23	26

Table 7
Coverage ratio after twenty feedback iterations using stemmed data. The inverse of the visibility documents are retrieved before one obtains one relevant document. Values are in percent

Visibility	SVM binary	SVM TF	SVM TF-IDF
1.4	86	86	87
0.68	72	71	73
0.58	85	84	87
0.37	100	95	100
0.24	41	40	43
0.15	100	100	100
0.07	100	94	100

The procedures discussed up to now assumed that there was at least one relevant document returned on the first screen. None of these procedures would work if there were no relevant documents returned in the preliminary search – in that case one has to go to subsequent screens to find a relevant document. How many screens one has to search will depend on the sophistication of the preliminary query. Our assumption is that one has to examine a number of documents equal to the inverse of the visibility to find one relevant document. In Table 7, we show the coverage ratio for documents that have very low visibility. We show the results after twenty feedback iterations. By that time (Table 1), one should have retrieved all documents. The non-SVM algorithms are not shown because they performed badly, approximately zero at the end of twenty iterations. The reason for that poor performance of the non-SVM algorithms can be directly attributed to the assumption that one has to search through many non-relevant documents in the preliminary search to find one relevant document. In the case of Rocchio, the presence of so many non-relevant vectors causes many terms of the resultant query to be zero. In the case of Ide dec-hi, use is not made of all the non-relevant documents (since only the top-ranked one is maintained) while the SVM algorithms use all the non-relevant documents to achieve a good optimum hyperplane.

All can be seen, binary features are as good as TF or TF-IDF features. By the twentieth iteration one should have retrieved the 120 documents with a visibility of 1.4% and the 78 documents with a visibility of 0.68% but one has only found approximately 86% and 72% of those. However, these are very rare documents in a database with over 11,000 documents. On the other hand, sometimes we were able to retrieve all the documents in the lowest visibility topics.

7. Preliminary search based on keywords

The results reported above used a random set of relevant and non-relevant documents in the preliminary search. This has the advantage of then being able to average over multiple experiments but has the disadvantage in that the preliminary set of documents are not retrieved in a realistic manner, as perhaps in a keyword search. Thus, if a good keyword search finds a relevant document in the first screen, even if the topic has low visibility, the resultant performance may be better than using the randomized set of preliminary documents assumed in the prior section. The rule in finding the preliminary set of documents here is to order articles by the number of occurrences of a keyword in the document. This is a sort on the TF of a keyword. In a keyword search of a high-visibility subject, many (if not all) of the documents retrieved on the initial screen will be relevant. Thus, if all the documents in the preliminary search are relevant, our procedure is to continue to successive screens until there was at least one non-relevant document and then start the first iteration. We maintain the rule that if the first screen has no relevant documents then we continue until we find a relevant document.

In Table 8, we show the coverage ratio after ten iterations. In the prior tables, coverage ratio was the number of relevant documents left in the database after the preliminary search divided the number obtained by an ideal search. In comparing the performance of two algorithms on the same topic, this was reasonable since the preliminary searches started out with the same number of relevant documents. However, we are about to use different sets of initial conditions and

Table 8
Coverage ratio (in percent) after ten iterations using stemmed data

Topic	Keyword	Preliminary search	Coverage ratio (percent)				
			SVM Binary	SVM TF	Ide dec-hi TF	SVM TF-IDF	Ide dec-hi TF-IDF
Earn	Earn	7 relevant, 3 non-relevant	100	100	97	97	97
	Earnings	9 relevant, 1 non-relevant	100	100	100	100	82
Grain	Grain	19 relevant, 1 non-relevant	95	93	59	99	95
Corn	Corn	38 relevant, 2 non-relevant	85	85	51	88	86
Gnp	Gnp	8 relevant, 2 non-relevant	82	90	59	90	90
Soybean	Soy	7 relevant, 3 non-relevant	49	64	39	69	67
Iron–steel	Iron	7 relevant, 3 non-relevant	67	70	49	79	81
	Steel	8 relevant, 2 non-relevant	64	59	53	76	76
Palm oil	Palm	18 relevant, 2 non-relevant	94	87	68	98	95
	Oil	2 relevant, 8 non-relevant	93	86	78	95	95
Fuel	Fuel	4 relevant, 6 non-relevant	50	39	39	63	75
Lei	Indicator	5 relevant, 5 non-relevant	100	100	75	100	100
	Leading	6 relevant, 4 non-relevant	100	100	77	100	100
Rapeoil	Rapeseed	1 relevant, 9 non-relevant	100	100	89	100	63

therefore coverage here will be redefined as the total number of documents retrieved, including those in the keyword search divided by the number returned in an ideal performance. The second column of Table 8 indicates the keyword used to order the preliminary search and the third column indicates the number of relevant and non-relevant documents in the preliminary set of documents. Note that there is always at least one relevant document in the first screen, but in some cases we went to multiple screens to find at least one non-relevant document. We group the algorithms together as those requiring TF-IDF (and hence requiring two passes over the database) and those that do not.

In this table, there is little difference between using SVM using TF or binary weighting, binary being better in five searches, worse in three searches and the same in the remaining six (of the fourteen total) searches. However the use of TF-IDF weighting usually improves the performance of SVM over the other two weighting techniques. If one decides not to use TF-IDF weighting then SVMs are invariably better than Ide dec-hi. However, if one proposes to use TF-IDF weighting, then the performance of SVM is better than Ide dec-hi in six searches, worse in two searches, and the same in the remaining six searches. TF-IDF features dramatically increase the performance of Ide dec-hi over its TF counterpart.

We tried another set of experiments where we thought the problem would be harder. In the preliminary search we used the top-ranked (using TF of the keyword) relevant document and the nine top-ranked (again using TF) non-relevant documents. Conceptually, this should give us a lower performance than that of Table 8 because there are less relevant documents in the first screen. The values are shown in Table 9.

Table 9
Coverage ratio in ten iterations using stemmed data. One top-ranked relevant document and nine top-ranked non-relevant document in the preliminary search

Topic	Keyword	Coverage ratio (percent)				
		SVM Binary	SVM TF	Ide dec-hi TF	SVM TF-IDF	Ide dec-hi TF-IDF
Earn	Earn	100	98	97	98	96
	Earnings	100	100	97	96	82
Grain	Grain	82	95	59	98	92
Corn	Corn	77	89	58	90	93
Gnp	Gnp	86	87	61	67	88
Soybean	Soy	46	61	44	74	65
Iron–steel	Iron	52	65	52	73	76
	Steel	61	69	55	95	73
Palm oil	Palm	88	86	86	95	98
	Oil	84	86	86	95	95
Fuel	Fuel	42	39	36	43	59
Lei	Indicator	100	100	95	100	100
	Leading	100	100	100	100	100
Rapeoil	Rapeseed	100	100	89	100	63

A cell-by-cell comparison of this table with Table 8 shows that the performance is sometimes better, sometimes worse. Thus, placing many non-relevant documents in the preliminary search does not seem to guarantee a harder problem. However, we can draw the same general observations: using TF-IDF weighing improves the performance of Ide dec-hi; that with non-TF-IDF weighting schemes, SVM is to be preferred and that with TF-IDF features, SVM is better than Ide dec-hi in six cases, worse in five cases, and tied in the remaining three searches. It should be remembered that the results of Tables 8 and 9 are those of one experiment – we did not have the liberty of averaging over multiple experiments.

8. Conclusions

We have analyzed the performance of SVM-based algorithms and compared them to Rocchio, Ide regular, and Ide dec-hi. In the first set of experiments we picked a random set of preliminary documents with a small, sometimes zero, number of relevant documents presented in the first screen. Based on these experiments, we can generally state that if the initial search is very poor and the visibility of the topic is low, then SVMs are superior to the other techniques investigated. In the second set of experiments, we did a keyword search to return the first set of preliminary documents. These keyword searches were invariably successful in that they would return a non-zero number of relevant documents in the preliminary search. In those cases, if one is allowed to use TF-IDF weighting (and suffer the time-penalty of two passes over the database), then SVM is marginally better than Ide dec-hi. However, if one decides not to use TF-IDF, then SVM using either binary or TF weighting is invariably better. There is not enough difference in performance between using binary or TF features in SVMs to recommend either weighting technique based on performance alone. However, constructing and manipulating binary vectors takes less time than TF-based vectors.

In a comparison of using stemmed data or not, using stemmed data is to be preferred because it reduces the size of the dictionary and does not affect performance. The elimination of words that do not occur in at least three documents does not improve SVM performance but does improve performance for the non-SVM techniques if one is willing to pay time penalty of two passes over the database.

We therefore have the following recommendations:

1. If one is using TF-IDF weighting and can be confident that the preliminary search returns many relevant documents, then SVM is marginally better than Ide dec-hi. However, the Ide dec-hi algorithm is simpler, runs faster, and one does not have worry about convergence of the SVM algorithm. We never had any problems with convergence but it can happen.
2. If one prefers not to use TF-IDF features, use SVM with binary feature weighting.
3. Use SVM if one is unsure of how successful the preliminary search will be since if the preliminary search is poor, the non-SVM algorithms will have poor performance.

Acknowledgements

Thanks go to Vladimir Vapnik for his insights and Thorsten Joachims who supplied the code for the support vector machine optimization problem. The code may be retrieved from: www-ai.informatik.uni-dortmund.de/thorsten/svm_light.html.

References

Aalbersberg, J. I. (1992). Incremental relevance feedback. In *Proceedings of the fifteenth annual international ACM SIGIR conference on research and development in information retrieval* (pp. 11–22).

Buckley, C., Salton, G., & Allen, J. (1993). Automatic retrieval with locality information using SMART. In *Proceedings of the first text retrieval conference (TREC-1)* (pp. 59–72).

Buckley, C., Salton, G., & Allen, J. (1994). The effect of adding relevance information in a relevance feedback environment. In *Proceedings of the seventeenth annual international ACM SIGIR conference on research and development in information retrieval* (pp. 292–300).

Cherkassky, V., & Mulier, F. (1998). *Learning from data*. New York: Wiley/Interscience.

Cristianini, N., & Shawe-Taylor, J. (2000). *Support vector machines*. Cambridge: Cambridge University Press.

Drucker, H., Wu, D., & Vapnik, V. N. (1999). Support vector machines for spam categorization. *IEEE Transactions on Neural Networks*, *10*, 1048–1054.

Dumais, S., Platt, J., Heckerman, D., & Sahami, M. (1998). Inductive learning algorithms and representations for text categorization. In *Proceedings of the seventh international conference on information and knowledge management* (pp. 148–155).

Harman, D. (1992). Relevance feedback revisited. In *Proceedings of the fifth international SIGIR conference on research and development in information retrieval* (pp. 1–10).

Joachims, T. (1997). A probabilistic analysis of the Rocchio algorithm with TFIDF for text categorization. *Proceedings of the fourteenth international conference on machine learning* (pp. 143–151). San-Francisco: Morgan Kaufmann.

Joachims, T. (1998). Text categorization with support vector machines: learning with features. In *European conference on machine learning* (pp. 137–142).

Korfhage, R. R. (1997). *Information storage and retrieval*. New York: Wiley (Chapter 8).

Lewis, D. (1995). Evaluating and optimizing autonomous text classification systems. In *Proceedings of the eighteenth annual international ACM–SIGIR conference on research and development in information retrieval* (pp. 246–254).

Mizzaro, S. (1997). Relevance: The whole history. *Journal of the American Society for Information Science*, *48*(9), 810–832.

Porter, M. F. (1980). An algorithm for suffix stripping. *Program*, *14*(3), 130–137.

Rocchio, J. J. (1971). Relevance feedback in information retrieval. In G. Salton (Ed.), *The SMART Retrieval System: Experiments in Automatic document processing* (pp. 313–323). Englewood Cliffs, NJ: Prentice–Hall.

Salton, G., & Buckley, C. (1988). Term-weighting approaches in automatic text retrieval. *Information Processing and Management*, *24*, 513–523.

Salton, G., & Buckley, C. (1990). Improving retrieval performance by relevance feedback. *Journal of the American Society for Information Science*, *41*, 288–297.

Saracevic, T. (1975). Relevance: A review of and a framework for thinking on the notion in Information Science. *Journal of the American Society for Information Science*, 321–343.

Schohn, G., & Cohen, D. (2000). Less is more: active learning with support vector machines. In *Proceedings of the seventeenth international conference on machine learning* (pp. 839–846).

Schapire, R. E., Singer, Y., & Singhal, A. (1998). Boosting and Rocchio applied to text filtering. In *Proceedings of the twenty-first annual international ACM SIGIR conference on information retrieval, SIGIR* (pp. 215–223).

Tauge-Sutcliffe, J. (1992). Measuring the informativeness of a retrieval process. In *Proceedings of the fifteenth annual international ACM SIGIR conference on research and development in information retrieval* (pp. 23–36).

Tong, S., & Koller, D. (2000). Support vector machine active learning with applications to text classification. In *Proceedings of the seventeenth international conference on machine learning* (pp. 999–1006). San-Francisco: Morgan Kaufmann.

Vapnik, V. (1982). *Estimation of dependencies based on empirical data*. Berlin: Springer.

Vapnik, V. (1998). *Statistical learning theory*. New York: Wiley.