Multinomial Naive Bayes for Text Categorization Revisited

Ashraf M. Kibriya, Eibe Frank, Bernhard Pfahringer, and Geoffrey Holmes

Department of Computer Science
University of Waikato
Hamilton, New Zealand
{amk14, eibe, bernhard, geoff}@cs.waikato.ac.nz

Abstract. This paper presents empirical results for several versions of the multinomial naive Bayes classifier on four text categorization problems, and a way of improving it using locally weighted learning. More specifically, it compares standard multinomial naive Bayes to the recently proposed transformed weight-normalized complement naive Bayes classifier (TWCNB) [1], and shows that some of the modifications included in TWCNB may not be necessary to achieve optimum performance on some datasets. However, it does show that TFIDF conversion and document length normalization are important. It also shows that support vector machines can, in fact, sometimes very significantly outperform both methods. Finally, it shows how the performance of multinomial naive Bayes can be improved using locally weighted learning. However, the overall conclusion of our paper is that support vector machines are still the method of choice if the aim is to maximize accuracy.

1 Introduction

Automatic text classification or text categorization, a subtopic in machine learning, is becoming increasingly important with the ever-growing amount of textual information stored in electronic form. It is a supervised learning technique, in which every new document is classified by assigning one or more class labels from a fixed set of pre-defined classes. For this purpose a learning algorithm is employed that is trained with correctly labeled training documents. The documents are generally represented using a "bag-of-words" approach, where the order of the words is ignored and the individual words present in the document constitute its features. The features present in all the documents make up the feature space. Since the number of words can be very large, the resulting learning problems are generally characterized by the very high dimensionality of the feature space, with thousands of features. Hence the learning algorithm must be able to cope with such high-dimensional problems, both in terms of classification performance and computational speed.

Naive Bayes is a learning algorithm that is frequently employed to tackle text classification problems. It is computationally very efficient and easy to implement. There are two event models that are commonly used: the multivariate

Bernoulli event model and the multinomial event model. The multinomial event model—frequently referred to as multinomial naive Bayes or MNB for short—generally outperforms the multivariate one [2], and has also been found to compare favorably with more specialized event models [3]. However, it is still inferior to the state-of-the-art support vector machine classifiers in terms of classification accuracy when applied to text categorization problems [4–7, 1]. However, recently a new algorithm has been proposed, called "transformed weight-normalized complement naive Bayes" (TWCNB), that is easy to implement, has good running time and is claimed to be nearly as accurate as support vector machines [1]. TWCNB is a modified version of MNB that is derived by applying a series of transformations relating to data and MNB itself.

In this paper we revisit the transformation steps leading from MNB to TWCNB. We show that using TFIDF scores instead of raw word frequencies indeed improves the performance of MNB, and that the same holds for document length normalization. However, our results also show that, depending on the particular text categorization dataset, it may not be necessary to perform the other transformation steps implemented in TWCNB in order to achieve optimum performance. Finally, we show how multinomial naive Bayes can be improved using locally weighted learning.

The paper is structured as follows. In Section 2 we describe our experimental setup. This includes the datasets we have used, how and what kind of features we extracted from them, and the transformations we apply to those features. We also describe the MNB and TWCNB classifiers. In Section 3 we present empirical results comparing standard MNB to TWCNB and support vector machines. Then, in Section 4, we show how MNB can be improved by transforming the input, and compare it again to the other learning algorithms. We also present results for locally weighted learning applied in conjunction with MNB. We summarize our findings in Section 5.

2 Experimental Setup

In this section we describe the datasets we used in our experiments and how we generated features from them. We also discuss the MNB and TWCNB learning algorithms.

2.1 Datasets

For our experiments we have used the 20 newsgroups, industry sector, WebKB, and Reuters-21578 datasets, which are frequently used in the text classification literature. The first three of these are single-label datasets whereas the Reuters-21578 is a multi-label dataset (i.e. with multiple class labels per document).

In the 20 newsgroups data the task is to classify newsgroup messages into one of 20 different categories. The version of the 20 newsgroups data that we have used in our experiments is the one that is referred to as 20news-18828 (available from `http://people.csail.mit.edu/people/jrennie/20Newsgroups/`).

It has all the fields removed from the news messages' header apart from the "from:" and "subject:" fields. All the cross-posted duplicate documents have also been removed, resulting in only 18,828 documents compared with 19,997 in the original 20 newsgroups data.

The industry sector data contains a collection of corporate WWW pages, divided into categories based on the type of company. There are 105 classes and 9,637 documents.

The WebKB data also has a collection of WWW pages and is available from `http://www-2.cs.cmu.edu/afs/cs.cmu.edu/project/theo-20/www/data/`. The WWW pages are from four computer science departments and split into several categories. Like [2] we used only four of the seven classes in the original data: "student", "faculty", "course", and "project", resulting in 4199 documents.

The Reuters-21578 data is a collection of newswire articles available from `http://kdd.ics.uci.edu/`. We followed the same approach as [1], using the "ModApte" split into training and test data and removing all classes with no test or training document. This left us with 7770 training and 3019 test documents in 90 classes. Note that a single newswire article may pertain to several categories (i.e. this is a multi-label problem). The standard approach to tackling this problem is to build a binary classifier for each category and that is the method we employed.

2.2 Feature Generation

In the bag-of-words approach each document is represented as a set of words and the number of times each word occurs in the document. In other words, each document has the words as its attributes or features and each attribute can take on an integer value counting the number of times the particular word occurs in the document. The set of words (also called "dictionary") is generated from all the documents present in a dataset. For a particular dataset, we first determine its dictionary by reading all the documents present in it. Then, for each document, we record the number of times each of the words in the dictionary occurs in it including those that did not occur by giving them a value zero. Note that we treated the Reuters-21578 dataset differently than the other datasets in that we determined the dictionary only from the training documents. We formed words by considering only contiguous alphabetic sequences. We also ignored words that were in our list of stopwords for all the datasets apart from WebKB.

For many of the experimental results presented in this paper we converted the word counts of a document using the TFIDF transformation before applying the learning algorithms. The TFIDF transformation takes the original word frequency f and transforms it [1]. Assuming that df is the number of documents containing the word under consideration, and D the total number documents, then the transformed attribute value becomes:

$$TFIDF(word) = log(f + 1) \times log(\frac{D}{df}).$$

We also considered normalizing the resulting word vectors to have the same length [1]. We evaluated two options: normalizing to length one and normalizing to the average vector length observed in the dataset. We found that performance can sometimes improve substantially using the latter approach. For conciseness, we shall refer to these conversions as TFIDF and together with normalization as TFIDFN.

2.3 Multinomial Naive Bayes

Let us now discuss how multinomial naive Bayes computes class probabilities for a given document. Let the set of classes be denoted by C. Let N be the size of our vocabulary. Then MNB assigns a test document t_i to the class that has the highest probability $\Pr(c|t_i)$, which, using Bayes' rule, is given by:

$$\Pr(c|t_i) = \frac{\Pr(c)\Pr(t_i|c)}{\Pr(t_i)}, \qquad c \in C \tag{1}$$

The class prior $\Pr(c)$ can be estimated by dividing the number of documents belonging to class c by the total number of documents. $\Pr(t_i|c)$ is the probability of obtaining a document like t_i in class c and is calculated as:

$$\Pr(t_i|c) = (\sum_n f_{ni})! \prod_n \frac{\Pr(w_n|c)^{f_{ni}}}{f_{ni}!}, \tag{2}$$

where f_{ni} is the count of word n in our test document t_i and $\Pr(w_n|c)$ the probability of word n given class c. The latter probability is estimated from the training documents as:

$$\widehat{\Pr}(w_n|c) = \frac{1 + F_{nc}}{N + \sum_{x=1}^{N} F_{xc}}, \tag{3}$$

where F_{xc} is the count of word x in all the training documents belonging to class c, and the Laplace estimator is used to prime each word's count with one to avoid the zero-frequency problem [2]. The normalization factor $\Pr(t_i)$ in Equation 1 can be computed using

$$\Pr(t_i) = \sum_{k=1}^{|C|} \Pr(k)\Pr(t_i|k). \tag{4}$$

Note that that the computationally expensive terms $(\sum_n f_{ni})!$ and $\prod_n f_{ni}!$ in Equation 2 can be deleted without any change in the results, because neither depends on the class c, and Equation 2 can be written as:

$$\Pr(t_i|c) = \alpha \prod_n \Pr(w_n|c)^{f_{ni}}, \tag{5}$$

where α is a constant that drops out because of the normalization step.

2.4 Transformed Weight-Normalized Complement Naive Bayes

As mentioned in the introduction, TWCNB [1] has been built upon MNB and is very similar to it. One difference is that the TFIDFN transformation is part of the definition of the algorithm. But the key difference is that TWCNB estimates the parameters of class c by using data from all classes apart from c (i.e. it uses the "complement"). To this end Equation 3 is called "word weight" rather than probability and redefined in the following way:

$$w_{nc} = log(\frac{1 + \sum_{k=1}^{|C|} F_{nk}}{N + \sum_{k=1}^{|C|} \sum_{x=1}^{N} F_{xk}}), \qquad k \neq c \wedge k \in C \tag{6}$$

The word weights are then normalized for each of the classes so that their absolute values sum to one and the classification for test document t_i is based on

$$\text{class}(t_i) = \text{argmax}_c[\log(\Pr(c)) - \sum_n (f_{ni} w_{nc})], \tag{7}$$

which, because the value of $log(Pr(c))$ is usually negligible in the total, can be simplified to

$$\text{class}(t_i) = \text{argmin}_c[\sum_n (f_{ni} w_{nc})]. \tag{8}$$

The parallels between MNB and TWCNB can easily be observed if we look at the classification rule for MNB given in [1]:

$$\text{class}(t_i) = \text{argmax}_c[\log(\Pr(c)) + \sum_n f_{ni} \log(\frac{1 + F_{nc}}{N + \sum_{x=1}^{N} F_{xc}})] \tag{9}$$

This rule is essentially the same as Equation 1 if we drop the denominator, take the log and use Equation 5 instead of Equation 2.

Note that we found TWCNB without normalization of the word weights (which is referred to as "TCNB" in the rest of this paper) to be very similar in performance compared to TWCNB. This will be discussed in more detail in Section 3.

As mentioned earlier, multi-label datasets like Reuters-21578 are usually handled differently than single-label datasets. For multi-label datasets, a classifier's performance is often measured using the precision-recall break-even point, for which we need some kind of document score representative of how likely a document is to belong to a class. Normally, a different classifier is trained for every class: it learns to predict whether a document is in the class (positive class) or not (negative class). This approach is also known as "one-vs-rest". Although this method can be used with MNB, it does not work when used with TWCNB. Unlike MNB, where Equation 2 can be used directly in conjunction with the one-vs-rest method, TWCNB's scores (Equation 7 and Equation 8) cannot be used directly because they are not comparable across different test documents

due to the missing normalization step. Hence a different method is used in [1], as described in the appendix of that paper. This method can be called "all-vs-rest". Based on the all-vs-rest approach, TWCNB's score is calculated as follows [8]:

$$docScore(t_i) = \sum_n (f_{ni} w_{nA} - f_{ni} w_{nR}). \tag{10}$$

In the above, w_{nA} is the word weight with data from all the classes, and w_{nR} is the word weight obtained from the "rest" (i.e. all documents not pertaining to the class that we are computing the score for).

3 Evaluating Standard Multinomial Naive Bayes

In this section we present experimental results comparing MNB with TCNB, TWCNB and linear support vector machines. For learning the support vector machines we used the sequential minimal optimization (SMO) algorithm [9] as implemented in the Weka machine learning workbench [10], using pairwise classification to tackle multi-class problems. Moreover, in the case of the Reuters' data, we fit logistic models to SMO's output based on maximum likelihood [11] because this improved performance significantly. The complexity parameter C was set to 10.

For each of the three single-label datasets mentioned above we present results with a full vocabulary and with a reduced vocabulary of 10,000 words. We pruned the vocabulary by selecting words with the highest information gain. The WebKB and industry sector datasets are collections of web pages, so we also present results obtained by removing HTML tags for these collections. For WebKB we used the top 5,000 words with the highest information gain after removing the tags, and did not remove stopwords. We did so to achieve high accuracy as MNB is reported to have the highest accuracy at 5,000 words [2]. As for TCNB and TWCNB, we also converted the raw word counts for SMO using the TFIDFN conversion, normalizing the document vectors to length one.

To estimate accuracy, we performed 5 runs of hold-out estimation for the 20 newsgroups and industry sector datasets, randomly selecting 80% training and 20% test documents for the 20 newsgroups data, and 50% training and 50% test documents for the industry sector data. For WebKB we performed 10 runs with 70% training and 30% test documents. The results reported are average classification accuracy over all runs. The Reuters-21578 results, however, are reported as precision-recall break-even points. The macro result is the average of the break-even points for all 90 individual Reuters' categories whereas the micro average is the weighted average of the break-even points, with the weight for each class being equal to the number of positive class documents in the test set.

The way we calculated the break-even point for the various classifiers in our experiments is similar to the way it is calculated in the "Bow" toolkit (available from `http://www-2.cs.cmu.edu/~mccallum/bow/`). First, we obtain the document score for every test document from our classifier and sort these scores in

Table 1. Comparison of MNB with TCNB, TWCNB and SMO

Dataset	MNB	TCNB	TWCNB	SMO with TFIDFN
20news18828-113232words	88.36	91.03	90.91	93.52
20news18828-10000words	86.10	87.98	88.47	92.13
WebKB-NoStoplist-54948words	80.05	79.68	78.46	91.31
WebKB-NoHTMLTagsOrStoplist-5000words	85.98	87.62	85.46	93.24
WebKB-10000words	81.30	85.23	82.12	92.76
IndustrySector-95790words	54.22	92.37	92.36	91.65
IndustrySector-NoHTMLTags-64986words	64.00	88.32	88.28	88.60
IndustrySector-10000words	63.37	87.25	87.33	89.74
Reuters-21578 (Macro)	34.40	69.62	69.46	70.16
Reuters-21578 (Micro)	78.49	86.31	85.78	88.47

descending order. Then, starting from the top of our sorted list, we calculate the precision (i.e. $TP/(TP + FP)$) and recall (i.e. $TP/(TP + FN)$) for each possible threshold in this list (where the threshold determines when something is classified as positive). The break-even point is defined as the point where precision and recall are equal. However, quite often there is no threshold where they are exactly equal. Hence we look for the threshold where the difference between precision and recall is minimum and take their average as the break-even point. If there are several candidates with minimum difference then we use the one which gives the greatest average.

We can see from the results in Table 1 that MNB almost always performs worse than any of the other learning algorithms. This is consistent with previously published results [1]. However, as we shall show in the next section, its performance can be improved considerably by transforming the input.

Our results for the various classifiers are comparable to those that have been published before on these datasets. However, we cannot compare our results for TCNB because it was not evaluated separately from TWCNB in [1]. It is quite evident from Table 1 that the results for TCNB are mostly better than for TWCNB. Hence it appears that word-weight normalization is not necessary to obtain good performance using complement naive Bayes.

4 Improving Multinomial Naive Bayes

In this section we investigate a few ways to increase the accuracy of MNB. Note that these methods have been suggested before in [1] but not evaluated for simple MNB (just for T(W)CNB). As we mentioned earlier, we found that the TFIDF conversion to the data greatly improves the results for MNB. We first present the effect of this conversion on MNB.

The results are shown in Table 2. TFIDF refers to the case where we applied the TFIDF transformation and did not normalize the length of the resulting

Table 2. Effect of the transformed input on MNB

Dataset	MNB	MNB with TFIDF	MNB with TFIDFN$_a$	MNB with TFIDFN
20news18828-113232words	88.36	91.40	92.56	89.69
20news18828-10000words	86.10	89.96	90.93	89.00
WebKB-NoStoplist-54948words	80.05	79.89	80.16	75.14
WebKB-NoHTMLTagsOrStoplist-5000words	85.98	88.05	88.30	84.98
WebKB-10000words	81.30	87.47	87.71	79.86
IndustrySector-95790words	54.22	85.69	88.43	84.09
IndustrySector-NoHTMLTags-64986words	64.00	75.82	81.40	79.69
IndustrySector-10000words	63.37	83.22	85.57	81.77
Reuters-21578 (Macro)	34.40	45.17	42.12	20.08
Reuters-21578 (Micro)	78.49	78.82	76.52	70.94

feature vectors. TFIDFN$_a$ refers to the case where we have normalized the feature vector for each document to the average vector length observed in the data, rather than one. TFIDFN refers to the case where we normalize to length one (i.e. the normalization used in [1]).

The results show that the TFIDF transformation dramatically improves the performance of MNB in almost all cases. TFIDFN$_a$ leads to a further improvement, which is especially significant on the industry sector data (only on the Reuters data there is a small drop compared to TFIDF). TFIDFN, on the other hand, is not very beneficial compared to simple TFIDF. Hence it appears that it is very important to normalize to an appropriate vector length when using normalization in conjunction with MNB. A potential explanation for this is that the Laplace correction used in MNB may start to dominate the probability calculation if the transformed word counts become too small (as they do when the normalized vector length is set to one). A similar effect may be achieved by changing the constant used in the Laplace correction from one to a much smaller value. However, we have not experimented with this option.

Table 3 below shows how the improved MNB with TFIDFN$_a$ compares with TCNB and SMO. Looking at the results we can see that the improved MNB outperforms TCNB in all cases on the 20 newsgroups and WebKB datasets, whereas TCNB outperform MNB on the industry sector and Reuters-21578 datasets. SMO is still superior to all other learning schemes.

The results show that it is not always beneficial to apply T(W)CNB instead of standard MNB. Applying the TFIDF transformation with an appropriate vector length normalization to MNB can lead to better results. Based on our results it is not clear when T(W)CNB produces better results for a given collection of documents. The performance may be related to the skewness of the class distribution because the industry sector data has a skewed class distribution. However, the class distribution of WebKB is also skewed, so there is no clear evidence for this.

Table 3. Comparison of improved MNB with TCNB, and SMO

Dataset	MNB with TFIDFN$_a$	TCNB	SMO with TFIDFN
20news18828-113232words	92.56	91.03	93.52
20news18828-10000words	90.93	87.98	92.13
WebKB-NoStoplist-54948words	80.16	79.68	91.31
WebKB-NoHTMLTagsOrStoplist-5000words	88.30	87.62	93.24
WebKB-10000words	87.71	85.23	92.76
IndustrySector-95790words	88.43	92.37	91.65
IndustrySector-NoHTMLTags-64986words	81.40	88.32	88.60
IndustrySector-10000words	85.57	87.25	89.74
Reuters-21578 (Macro)	42.12	69.62	70.16
Reuters-21578 (Micro)	76.52	86.31	88.47

Note that the good performance of T(W)CNB on the Reuters data can be attributed to the all-vs-rest method discussed in Section 2.4, which is used to obtain the confidence scores for computing the break-even points. In fact, applying the all-vs-rest method to standard MNB results in a classifier that is equivalent to TCNB+all-vs-rest, and produces identical results on the Reuters data. Hence the industry sector data is really the only dataset where T(W)CNB improves on standard MNB.

4.1 Using locally weighted learning

In this section we discuss how MNB can be improved further using locally weighted learning [12]. Our method is essentially the same as what has been applied earlier to the multivariate version of naive Bayes [13], and found to perform very well on other classification problems. The idea is very simple. For each test document we train an MNB classifier only on a subset of the training documents, namely those ones that are in the test document's neighborhood, and weight those documents according to their distance to the test instance. Then, instead of using the feature values of a training instance directly in the MNB formulae (i.e. raw word counts or TFIDF values), we multiply them by the weight of the corresponding training instance. The number of documents in the subset (also called the "neighborhood size") is determined through a user-specified parameter k. Each training document in the subset is assigned a weight which is inversely proportional to its distance from the test document.

In our setting we calculate the Euclidean distance of all the training documents from the test document, and divide all the distances with the distance of the kth nearest neighbor. Then the weight of each training document is computed based on the following linear weighting function:

$$f(d_i) = \begin{cases} 1 - d_i \text{ if } d_i <= 1 \\ 0 \quad\quad \text{if } d_i > 1 \end{cases} \tag{11}$$

Table 4. Applying LWL to MNB with TFIDFN$_a$

Dataset	MNB	LWL+ MNB with k=50	LWL+ MNB with k=500	LWL+ MNB with k=5000
20news18828-113232words	92.56	93.15	93.65	90.87
20news18828-10000words	90.93	93.29	92.96	89.93
WebKB-NoStoplist-54948words	80.16	77.77	78.10	75.23
WebKB-NoHTMLTagsOrStoplist-5000words	88.30	87.28	88.91	88.63
WebKB-10000words	87.71	83.96	86.37	87.01
IndustrySector-95790words	88.43	89.50	89.53	89.65
IndustrySector-NoHTMLTags-64986words	81.40	85.32	84.03	83.25
IndustrySector-10000words	85.57	86.85	86.29	86.58
Reuters-21578 (Macro)	42.12	56.29	48.18	50.29
Reuters-21578 (Micro)	76.52	84.31	80.30	75.14

where d_i is the normalized distance of training document i.

This gives a weight zero to all the documents that are further away than the kth nearest one from the test document. Hence those documents are effectively discarded. Once the weights are computed based on this formula, we normalize them so that their sum is equal to the number of training documents in the neighborhood, as in [13]. Note that, unlike [13] we did not normalize the feature values to lie in $[0, 1]$ as we found it to degrade performance.

Table 4 above gives a comparison of MNB to MNB used in conjunction with locally weighted learning (LWL). We report results for three different subset sizes (50, 500, and 5000). The input data to locally weighted MNB had all the transformations applied to it (i.e. the TFIDF transformation and vector length normalization described above) before the distance calculation was performed to weight the documents. The same transformations were applied in the case of MNB. We can see that in most cases LWL can improve the performance of MNB if the appropriate subset size is chosen, only on the WebKB data there is no improvement. Moreover, optimum (or close-to-optimum) performance is achieved with the smaller subset sizes ($k = 50$ or $k = 500$), and in some cases there appears to be a trend towards better performance as the size becomes smaller (more specifically, on *20news18828-10000words*, *IndustrySector-NoHTMLTags-64986words*, and the Reuters data), indicating that size 50 may not be small enough in those cases. However, we have not experimented with values of k smaller than 50.

Table 5 gives a comparison of SMO with the best results we have been able to achieve with locally weighted MNB. Note that the results for the latter method are optimistically biased because they involve a parameter choice (the neighborhood size) based on the test data. However, even with this optimistic bias for the MNB-based results, SMO performs better in almost all cases, in particular on the WebKB data.

Table 5. Comparison of best results for locally weighted MNB with SMO

Dataset	MNB with TFIDFN$_a$ & LWL	SMO with TFIDFN
20news18828-113232words (k=500)	93.65	93.52
20news18828-10000words (k=50)	93.29	92.13
WebKB-NoStoplist-54948words (k=500)	78.10	91.31
WebKB-NoHTMLTagsOrStoplist-5000words(k=500)	88.91	93.24
WebKB-10000words (k=5000)	87.01	92.76
IndustrySector-95790words (k=5000)	89.65	91.65
IndustrySector-NoHTMLTags-64986words (k=50)	85.32	88.60
IndustrySector-10000words (k=50)	86.85	89.74
Reuters-21578 (Macro) (k=50)	56.29	70.16
Reuters-21578 (Micro) (k=50)	84.31	88.47

5 Conclusions

This paper has presented an empirical comparison of several variants of multinomial naive Bayes on text categorization problems, comparing them to linear support vector machines. The main contribution of this paper is the finding that standard multinomial naive Bayes can be improved substantially by applying a TFIDF transformation to the word features and normalizing the resulting feature vectors to the average vector length observed in the data. If this is done, it can, depending on the dataset, outperform the recently proposed transformed weight-normalized complement naive Bayes algorithm, which also includes the TFIDF transformation and normalization to (unit) vector length, but exhibits two additional modifications—weight normalization and complement-based classification—that appear to represent a departure from standard Bayesian classification. Additionally, we found that the effect of weight-normalization on complement naive Bayes was negligible.

We have also shown how the performance of multinomial naive Bayes can be further improved by applying locally weighted learning. However, even if the best neighborhood size is chosen based on the test data, this improved classifier is still not competitive with linear support vector machines. Hence the overall conclusion is to use support vector machines if their (significantly larger) training time is acceptable, and, if not, to consider standard multinomial naive Bayes with appropriate transformations of the input as an alternative to complement naive Bayes.

References

1. Rennie, J.D.M., Shih, L., Teevan, J., Karger, D.R.: Tackling the poor assumptions of naive Bayes text classifiers. In: Proceedings of the Twentieth International Conference on Machine Learning, AAAI Press (2003) 616–623

2. McCallum, A., Nigam, K.: A comparison of event models for naive Bayes text classification. Technical report, American Association for Artificial Intelligence Workshop on Learning for Text Categorization (1998)
3. Eyheramendy, S., Lewis, D.D., Madigan, D.: On the naive Bayes model for text categorization. In: Ninth International Workshop on Artificial Intelligence and Statistics. (2003) 3–6
4. Joachims, T.: Text categorization with support vector machines: Learning with many relevant features. In: Proceedings of the Tenth European Conference on Machine Learning, Springer-Verlag (1998) 137–142
5. Dumais, S., Platt, J., Heckerman, D., Sahami, M.: Inductive learning algorithms and representations for text categorization. In: Proceedings of the Seventh International Conference on Information and Knowledge Management, ACM Press (1998) 148–155
6. Yang, Y., Liu, X.: A re-examination of text categorization methods. In: Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, ACM Press (1999) 42–49
7. Zhang, T., Oles, F.J.: Text categorization based on regularized linear classification methods. Information Retrieval **4** (2001) 5–31
8. Rennie, J.: Personal communication regarding WCNB (2004)
9. Platt, J.: Fast training of support vector machines using sequential minimal optimization. In Schölkopf, B., Burges, C., Smola, A., eds.: Advances in Kernel Methods—Support Vector Learning. MIT Press (1998)
10. Witten, I., Frank, E.: Data Mining: Practical machine learning tools and techniques with Java implementations. Morgan Kaufmann (1999)
11. Platt, J.: Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. In Smola, A., Bartlett, P., Schölkopf, B., Schuurmans, D., eds.: Advances in Large Margin Classifiers. MIT Press (1999)
12. Atkeson, C.G., Moore, A.W., Schaal, S.: Locally weighted learning. Artificial Intelligence Review **11** (1997) 11–73
13. Frank, E., Hall, M., Pfahringer, B.: Locally weighted naive Bayes. In: Proceedings of the Conference on Uncertainty in Artificial Intelligence, Morgan Kaufmann (2003)