A Closer Look at Nearest Neighbor Text Classification (DRAFT: Do not distribute or cite. Comments welcome.)

David D. Lewis
Independent Consultant
Chicago, IL, USA
dave@daviddlewis.com

Vladimir Menkov
Independent Consultant
Penticton, BC, Canada
vmenkov@cs.indiana.edu

ABSTRACT

We present a mathematical and experimental study of three nearest neighbor classification algorithms: two that are popular in text classification and a classic one from pattern recognition. Our mathematical analysis suggests a systematic problem with threshold setting in one of the text classification algorithms, and our experiments show the classic algorithm has the highest effectiveness of the three studied.

Categories and Subject Descriptors

I.2.6 [**Artificial Intelligence**]: Learning; H.3.1 [**Information Storage and Retrieval**]: Content Analysis and Indexing—*Indexing methods*

General Terms

Algorithms, Experimentation, Theory

Keywords

text categorization, supervised learning, nearest neighbor, local methods, lazy learning, calibration

1. INTRODUCTION

A nearest neighbor text classification (NNTC) system classifies documents by finding similar manually classified documents and classifying the new document in a similar way. This intuitively appealing approach has been widely used in both research systems and commercial products.

Advantages of NNTC include efficiency with large numbers of classes [15], simplicity of implementation, ease of integrating new training data, straightforward handling of both binary and 1-of-k classification problems, and ability to fit arbitrary decision boundaries given sufficient training data.

Much of the popularity of NNTC in IR comes from the analogy between finding neighbors of a test document in a NNTC system, and finding high scoring documents for a query in a statistical text retrieval system. Similar text representation and term weighting techniques can be used, and even the same software, particularly in experimental studies.

This has meant, however, that NNTC methods have tended to be designed by analogy with ranked retrieval methods, rather than following from the properties of text classification itself. In contrast, substantial theoretical analysis and development of nearest neighbor methods specifically for classification has occurred in pattern recognition [2, 8] and statistics [5], suggesting that there may be insights there to be applied to text classification.

In this paper we take a closer look at two weighted nearest neighbor algorithms popular in NNTC (Yang's original 1994 algorithm, and her more recent *kNN.avg1* algorithm), and compare them with the classic unweighted k-NN classifier that is widely used in pattern recognition. We know of no previous systematic comparison between weighted and unweighted nearest neighbor approaches for text classification.

We begin by describing each of three algorithms, and in particular the issue of how thresholds can be set. We then provide an alternate mathematical characterization of *kNN.avg1*, showing that no threshold setting mechanism is likely to succeed with it except in very limited circumstances, and hypothesizing that it will have a certain unusual behavior as neighborhood size is changed.

We then present an experimental study of the three algorithms, confirming our hypothesis about *kNN.avg1* and characterizing the behavior of the algorithms as neighborhood size varies. We end by speculating on some synthesizes of work from NNTC and pattern recognition that could lead to more effective NNTC algorithms.

2. NEAREST NEIGHBOR CLASSIFICATION ALGORITHMS

Nearest neighbor classification (NNC) algorithms have been extensively studied in the learning sciences [2, 5, 8]. Just like other supervised learning algorithms, they are given a set of examples of known class membership (training examples), they produce a classifier that can decide which of the possible classes a new example (a test example) belongs to.

However, NNC algorithms differ from other supervised learning algorithms in doing little computation in advance of seeing test data, but substantial computation at the time test data is classified. When the test data is presented, the NNC trains a different *local* classifier for each test example, then discards that local classifier once the test example has been classified. During the training of each local classifier, examples that are most similar to that test example, i.e. its *neighbors*, are used solely or primarily.

Nearest neighbor methods are sometimes called *lazy learning* methods, to indicate that learning is delayed until it is ready to be applied. They are also called *local learning* methods, particularly when a nontrivial supervised learning algorithm is used to produce the local classifier from the neighbors.

2.1 The Classic k-NN Classifier

The oldest [4] and most widely used NNC algorithm is the *k-NN classifier* (which, for clarity, we will sometimes call here the *traditional* k-NN classifier). A k-NN classifier finds the set, $N_k(\vec{x})$, of k training examples most similar to a test example, $\vec{x}$. It then assigns to the test example to the class that is most frequent among the neighbors. In the binary case (where we will refer to the two classes as *Yes* and *No*, and also talk of *Yes* as a category being "assigned"), this corresponds to assigning the class to which the majority of the neighbors belong.[1]

The hypothesis space of traditional k-NN classifiers is characterized by a single complexity parameter, k, the neighborhood size. In contrast to most learning algorithms, larger values of the complexity parameter instruct the learning algorithm to fit a *less* flexible classifier. Indeed, the effective number of parameters for a k-NN classifier is n/k where n is the training set size [5, pp. 15-16]. Thus the 1-NN classifier is both the most flexible of traditional k-NN classifiers, and the most susceptible to overfitting. A $1,000,000$-NN classifier would be very robust against overfitting, but have very limited classification ability.

2.2 The Yang94 Algorithm

The traditional k-NN method, while common in pattern recognition, has only rarely been used in text classification. More common are NNTC methods that give more weight to neighbors which are closer to the test example. The most widely used such algorithm was introduced by Yang [14], originally under the name *ExpNet*, but more recently referred to as just k-NN [16] . For clarity, we will refer to this algorithm as *Yang94*.

When applied to a binary classification problem, the *Yang94* algorithm computes the score $s(l, \vec{x})$ of class l for test example $\vec{x}$ as:

$$s(l, \vec{x}) = \sum_{\vec{z} \in P_k(l, \vec{x})} sim(\vec{x}, \vec{z}) \qquad (1)$$

where $\vec{z}$ is a training document, *sim()* is some similarity measure between documents, and $P_k(\mathrm{l}, \vec{x})$ is the subset of $N_k(\vec{x})$ belonging to class l.

The *Yang94* method has typically been used with a similarity measure that takes on values between 0 and 1, most commonly the cosine correlation [12, p. 124]. This is simply the dot product of the test vector and the training vector, under the assumption that both have been scaled to have a Euclidean norm of 1. With similarity measures of this form, *Yang94* treats a training example identical to the test example in the same way that traditional k-NN would, but gives training examples that differ in any way from the test example proportionally lower weights.

The original presentation of *Yang94* used the method only to rank classes for a test document, not to make a discrete decision of which to assign. In later work [16, 18, 17], Yang explored several approaches for converting test document scores to binary classifications.

2.3 Yang's Later NNTC Algorithms

Yang [18] recently proposed several new NNTC algorithms, each computing a somewhat different score for a test document. The first, $kNN.sum$, sets $s(l, \vec{x})$ to:

$$\sum_{\vec{z} \in P_k(l, \vec{x})} sim(\vec{x}, \vec{z}) - \sum_{\vec{z} \in Q_k(l, \vec{x})} sim(\vec{x}, \vec{z}) \qquad (2)$$

where $Q_k(\mathrm{l}, \vec{x}) = N_k(\vec{x}) - P_k(\mathrm{l}, \vec{x})$ is the set of neighbors of $\vec{x}$ not belonging to class l.

The $kNN.sum$ algorithm is clearly problematic for low frequency classes, since a test document's score is likely to be dominated by distances to the many negative examples rather than distances to the few, but important, positive examples. Yang's second algorithm, *kNN.avg1*, therefore computes $s(l, \vec{x})$ as:

$$\frac{1}{|P_k(l, \vec{x})|} \sum_{\vec{z} \in P_k(l, \vec{x})} sim(\vec{x}, \vec{z}) - \frac{1}{|Q_k(l, \vec{x})|} \sum_{\vec{z} \in Q_k(l, \vec{x})} sim(\vec{x}, \vec{z}). \qquad (3)$$

The intent is that by working with the average rather than total similarity to positive and negative examples, the problem of large number of negative examples in the neighborhood will be reduced.

Yang's third algorithm, $knn.avg2$, goes further by computing separate neighborhoods from positive and negative examples. This does not fit cleanly into the nearest neighbor framework since, in the worst case, all training documents must be used. We therefore will not treat $knn.avg2$ further here.

3. THRESHOLDING

In pattern recognition, the k-NN classifier is usually used with a threshold derived analytically from the neighborhood size. This may have advantages over thresholds tuned to training data when the amount of training data is small.

3.1 Analytic Thresholding

Most experimental work, and almost all theoretical analyses of the traditional k-NN classifier, assume a test instance is assigned to the class which is most frequent among the neighbors. But this is appropriate only when all mistaken class assignments have the same cost.

This can be seen by observing that a k-NN classifier implicit estimates $p(C = l|\vec{x})$, the posterior probability that test instance $\vec{x}$ belongs to class l, by the proportion of neighbors, $k_l(\vec{x})/k$, belonging to class l [8, p. 191]. If this estimate was exact, then the optimal classification rule would be to assign the test document to a class l such that the expected utility of the classification was optimal [3, 6]. In the binary case with equal costs, this gives the traditional majority wins rule, but with unequal costs would choose different values.

For instance, if true positives have a utility of $+2$, and false positives a utility of -1, as we will examine in our

[1] In the case of a tie, theoretical analyses usually assume that the class is assigned randomly, but for an operational classifier one will usually want a deterministic tiebreaking rule.

experiments, we should assign a document to the *Yes* class when $p(C = \text{Yes}|\vec{x}) >= 1/3$, i.e. at least 1/3rd of neighbors have the *Yes* label.

3.2 Thresholding by Cross-Validation

More generally, we can recognize that the estimates of class proportions are imperfect. For instance, if the neighborhood size is larger than the number of positive examples of some class, $p(C = l|\vec{x})$ is guaranteed to be underestimated in some cases, perhaps grossly so.

In the binary case at least, this suggests the alternative of simply treating the number of neighbors belonging to the *Yes* class simply as a score (by happenstance an integer) that is computed for a test document and compared against a threshold. In the case of algorithms such as *Yang94* and *kNN.avg1*, we do in fact get real-valued scores and comparison against a threshold is the natural approach.

Yang has suggested using cross-validation [7] or a held out validation set [17] as approaches to setting the per-class thresholds with her methods. The cross-validation approach that is maximally efficient in using training data is leave-one-out cross-validation (LOOCV) [8, pp. 69-71]. LOOCV can be used to set the threshold for a binary NNTC system as follows:

1. For each training document
   (a) Find its $k+1$ nearest neighbors among the training documents.
   (b) Discard the document itself from the neighbors (thus the name "leave-one-out"), leaving k neighbors.
   (c) Compute and record the score these k neighbors give this training document.
2. Sort the scores of the training documents and compute the effectiveness that would result from using each of the document scores as the threshold. Choose the threshold that gives the highest effectiveness. (This can be done in linear time in the number of training examples.)

The major cost here is finding the neighbors for each training document, but this takes no more time than classifying an equivalent number of test documents would.

It is important to recognize that cross-validation is not magic. It itself a supervised learning process, and its accuracy at finding a threshold will depend on how much training data is available. To take an extreme case, if there is only a single positive example of the class, LOOCV will never see a positive example in the validation set. It will therefore choose an extremely conservative threshold. By comparison, the analytic threshold for a 3-NN classifier with our (Section 5.3) effectiveness measure would be the comparatively sensible value, 1 (i.e. assign a test document to the class if at least 1 of the 3 neighbors is a class member).

3.3 Multiple Simultaneous Classifications

We have so far assumed that only a single classification decision (either binary or 1-of-k) needs to be made for each test document. An advantage of NNTC is its efficiency when multiple classification decisions need be made. The expensive part of NNTC (looking up the neighbors) need be done only once, while the inexpensive part (running a typically very simple learning algorithm on the neighbors) is done for each class.

Further, it will usually be the case that most classes will have no positive examples in the neighborhood, in which case no work will need to be done for those classes. (All the locally applied algorithms reduce to trivial reject-all classifiers in that case.)

The time required to train thresholds does scale with the number of classes, but again the major expense (lookup of neighbors in the LOOCV procedure) need to be done only once, accumulating scores for all classes, before training a threshold for each class.

4. AN ALTERNATE VIEW OF KNN.AVG1

When dot product is used as the similarity measure in *kNN.avg1*, the score of a document, $s(l, \vec{x})$, can be rewritten as:

$$\frac{\sum_{\vec{z}\in P_k(l,\vec{x})} \sum_j x_j z_j}{|P_k(l,\vec{x})|} - \frac{\sum_{\vec{z}\in Q_k(l,\vec{x})} \sum_j x_j z_j}{|Q_k(l,\vec{x})|} \tag{4}$$

which is in turn equal to:

$$\sum_j x_j \left(\frac{\sum_{\vec{z}\in P_k(l,\vec{x})} z_j}{|P_k(l,\vec{x})|} - \frac{\sum_{\vec{z}\in Q_k(l,\vec{x})} z_j}{|Q_k(l,\vec{x})|} \right) \tag{5}$$

where in both cases j is an index to term weights in the document vectors. The second expression is exactly the dot product of $\vec{x}$ with a linear model trained from the neighbors using the Rocchio algorithm [10], with parameters $\alpha = 0$ (no query) and $\beta = \gamma = 1$.

Thus, in addition to being a weighted NNC algorithm, *kNN.avg1* is mathematically equivalent to an unweighted, local learning version of the Rocchio algorithm. Under this second interpretation, the dot product plays the role both of a similarity measure defining a set of unweighted neighbors, and a second role as the function used to apply the locally trained Rocchio model to the test instance.

This view suggests a problem with the *kNN.avg1* method. A true local learning version of Rocchio would also compute the threshold for the local Rocchio model using the local data at classification time. The *kNN.avg1* method instead is based on using a single threshold, found using a validation method, for every local Rocchio model produced.

There are two neighborhood sizes where Yang's global threshold setting procedure will produce roughly the same result as if thresholds were set locally. The first is when $k = 1$. In this case, all positive validation examples would be assigned a positive score during training, while all negative validation examples would be assigned a negative score. The global threshold setting procedure will pick a small positive value near 0.0 as the global threshold. The local threshold setting procedure, on the other hand, degenerates to traditional k-NN when $k = 1$, i.e. assign the test document to the class when the single neighbor belongs to the class, i.e. if the score of the test document is > 0.0.

The other case is when $k = n$, the training set size. In this case, all the local Rocchio models are identical to each other. If the entire training set were used both for Rocchio training and as the validation set in Yang's approach, the same threshold would be found as in a local procedure.

If LOOCV were used, the validation documents would be scored by multiple Rocchio models trained from $n-1$ documents instead of a single Rocchio model trained on n documents, but the threshold will likely be extremely similar unless n or the class frequency are very small.

This analysis suggests that *kNN.avg1* should exhibit an unusual pattern of effectiveness as neighborhood size is varied. Effectiveness should be reasonable with very small and very large neighborhood sizes, where a globally trained threshold gives similar results to a locally trained one, but poor effectiveness at intermediate neighborhood sizes where the variability of the local Rocchio models is maximal. This is exactly the opposite of normal behavior of a learning algorithm as the complexity parameter is varied.

5. METHODS

The data set we used to test our hypotheses is the RCV1-v2 corpus [7], a corrected version of Reuters Corpus Version 1 [11]. We used the same training/test split as in the RCV1-v2 paper, i.e. 23,149 training documents (documents from August 1996) and 781,265 test documents (documents from September 1996 through August 1997).

We used the 103 Topic categories from RCV1-v2 to define 103 separate binary classification problems. The set of documents belonging to each categories was as defined by the corrected qrels files distributed by the RCV1-v2 authors.

5.1 Text Representation

We obtained the tokenized form of the official RCV1-v2 distribution. No further text processing was done on these tokens. Document vectors based on the tokenized data were then created, with each coordinate of the vectors corresponding to a unique unstemmed term.

The weight of a term in a vector was computed using *log TF* $\times$ *idf* term weighting with cosine normalization [1]. We gave term t in document d a initial weight of:

$$w_d(t) = (1 + \log_e n(t,d)) \times \log_e \frac{|\mathcal{D}|+1}{n(t)+1} \tag{6}$$

where $n(t)$ is the number of documents that contain t, $n(t,d)$ is the number of occurrences of term t in document d, and $|\mathcal{D}|$ is the number of training documents (23,149). The form of IDF used, where we add 1 to the numerator and denominator, guarantees the weight is defined for words that did not occur in the training data. It can also be justified, at least for test documents, as a prequential estimate of IDF, since we care about the IDF of a word for a test document exactly in those cases where we have a document in hand containing the word.

Document length normalization is particularly important for NNTC, since otherwise long documents will tend to disproportionately show up as neighbors. We used cosine normalization [1] (i.e. normalizing vectors to have a Euclidean norm of 1.0), the most common normalization approach in past NNTC work. This makes the final weights in the the feature vectors be:

$$w'_d(t) = \frac{w_d(t)}{\sqrt{\sum_t w_d(t) \times w_d(t)}}. \tag{7}$$

Despite the similarity of weighting schemes, our vectors are not numerically identical to those distributed with the RCV1-v2 paper, due to their use of feature selection, additional unlabeled documents in computing IDF weights, and the above-mentioned difference in the IDF formula.

5.2 Similarity Measure

We use the dot product between two vectors:

$$sim(\vec{x}, \vec{z}) = \sum_j x_j z_j \tag{8}$$

as our measure of their similarity. Since we normalize document vectors to have a Euclidean norm of 1.0, the dot product corresponds to the cosine of the angle between the vectors. It also gives the same ordering of neighbors as would Euclidean distance, again because vectors have a Euclidean norm of 1.0.

5.3 Effectiveness Measure

The most common effectiveness measures used in text classification studies are the F-measure and various forms of linear utility measures. We use a linear utility measure since we are interested in close examination of thresholding methods, and no threshold can be optimal for the F-measure across all test sets [6].

We chose the linear utility measure from the TREC-2002 (TREC-11) evaluation [9]:

$$T11U = 2 \times A - 1 \times B + 0 \times C + 0 \times D = 2A - B \tag{9}$$

where A is the number of test documents the system correctly assigns to the category, B is the number of test documents incorrectly assigned to the category, C is the number of category members the system neglects to assign to the category, and D is the number of category nonmembers which correctly are not assigned to the category.

When presenting an average effectiveness value over a set of categories we use macroaveraging [13], i.e. first computing effectiveness for each category separately, and taking the average of those values. However, because the range of utility values is very strongly affected by the number of positive examples of the category in the test set, we take the macroaverage of scaled utility values rather than raw ones. The scaled utility measure, T11SU, is that used in TREC-2002, where:

$$\begin{aligned}
\text{MinNU} &= -0.5 && (10)\\
\text{MaxU} &= 2 * (A + C) && (11)\\
\text{T11NU} &= T11U/\text{MaxU} && (12)\\
\text{T11SU} &= \frac{\max(\text{T11NUMinNU}) - \text{MinNU}}{1 - \text{MinNU}} && (13)
\end{aligned}$$

5.4 Algorithms, Parameters, and Implementation

We implemented the three algorithms described in Section ?, traditional k-NN, *Yang94*, and kNN.avg1 using a modified version of the Lemur toolkit (*www.cs.cmu.edu/˜lemur*). Ties (for instance in scores of neighbors) were broken deterministically by document ID. (We also tested breaking ties using a pseudorandom hash on document ID, but found only trivial differences in results.)

The main parameter varied in experiments was the neighborhood size. Except for $k = 1$, $k = 2$, and $k = 23149$ (the full training set size), all neighborhood sizes were multiples

k	Analytic Trad	LOOCV Trad	LOOCV Yang94	LOOCV kNN.avg1
1	0.45	0.46	0.48	0.48
2	0.39	0.49	0.49	0.49
3	0.32	0.51	0.50	0.47
6	0.49	0.52	0.52	0.45
12	0.54	0.54	0.53	0.41
24	0.54	0.55	0.54	0.38
48	0.53	0.56	0.55	0.37
96	0.51	0.57	0.56	0.36
192	0.49	0.58	0.56	0.36
384	0.46	0.58	0.57	0.36
768	0.42	0.57	0.56	0.37
1536	0.40	0.55	0.55	0.40
3072	0.37	0.52	0.53	0.46
6144	0.35	0.49	0.51	0.52
12288	0.35	0.45	0.49	0.55
23149	0.34	0.34	0.47	0.55

Table 1: Test set T11SU, macroaveraged over 103 categories, for four nearest neighbor text classification methods at a range of sizes (k) of neighborhood.

of 3 to allow an exact analytic threshold (1/3rd of neighborhood size) for the T11U effectiveness measure.

6. RESULTS

Table 1 shows macroaveraged T11SU for a range of neighborhood sizes for each of the three algorithms with thresholds selected by LOOCV, as well as traditional k-NN (*Trad*) with analytic thresholds (1/3 of neighborhood size).

The most notable observation is that, despite the preference for weighted methods in NNTC work, the traditional k-NN algorithm (*Trad*) actually has a higher peak effectiveness.

Looking in more detail, we see that *Trad* and *Yang94* show the usual variation of test set effectiveness with model complexity. Effectiveness starts out low when very simple models are fit (e.g. $k = 23149$), increases as more complex models are fit, and eventually declines when models become so complex that overfitting occurs (e.g. $k = 1$).

There is a deviation from the pattern for *Trad* with neighborhood sizes ($k = 1$ and $k = 2$) that are too small to use a threshold of 1/3'rd the neighborhood size. The deviation is that effectiveness improves when going from $k = 3$ with a threshold of 1, to ($k = 1$ and $k = 2$) suggesting that the analytic thresholds are too low. Indeed, *Trad* with analytic thresholds is worse than *Trad* with LOOCV-thresholds at almost all neighborhood sizes.

The *kNN.avg*1 algorithm shows a completely different pattern from the other algorithms, as we hypothesized in Section 4. For $k = 1$ it is similar to traditional k-NN and *Yang94*. At intermediate neighborhood sizes, effectiveness is low because of using a common threshold with highly variable Rocchio models. Then at large neighborhood sizes it acts essentially as a classic Rocchio model, with reasonable effectiveness.

7. SUMMARY

Nearest neighbor classification algorithms are widely used in text classification, but tend to be based on analogies with text retrieval, rather than a foundation of statistical classification theory. We have shown that the classic k-NN algorithm from pattern recognition actually has superior text classification effectiveness to two popular weighted NNTC algorithms.

We also have provided an alternate mathematical analysis that explains the anomalous behavior of one of the weighted algorithms, *kNN.avg1*, showing that it fits Rocchio models locally but a threshold globally.

The comparatively good effectiveness of *kNN.avg1* at extreme values of neighborhood size suggests, however, that using more sophisticated local methods might provide a method that would consistently exceed the effectiveness of traditional k-NN. Using the Rocchio method with a local threshold would be one possibility, but in fact any supervised learning method can be applied locally, providing a rich space of possibilities that remain to be explored.

8. ACKNOWLEDGMENTS

(Removed for anonymity.)

9. REFERENCES

[1] C. Buckley, G. Salton, and J. Allan. The effect of adding relevance information in a relevance feedback environment. In W. B. Croft and C. J. van Rijsbergen, editors, *SIGIR 94: Proceedings of the Seventeenth Annual International ACM-SIGIR Conference on Research and Development in Information Retrieval*, pages 292–300, London, 1994. Springer-Verlag.

[2] B. V. Dasarathy, editor. *Nearest Neighbor (NN) Norms: NN Pattern Classification Techniques.* IEEE Computer Society Press, Los Alamitos, CA, 1991.

[3] R. O. Duda and P. E. Hart. *Pattern Classification and Scene Analysis.* Wiley-Interscience, New York, 1973.

[4] E. Fix and J. L. Hodges. Discriminatory analysis. nonparametric discrimination: Small sample performance. Report 11, USAF School of Aviation Medicine, Randolph AFB, Texas, Aug. 1952. Reprinted in Agrawala, *Machine Recognition of Patterns*, IEEE Press, New York, 1977.

[5] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction.* Springer, New York, 2001.

[6] D. D. Lewis. Evaluating and optimizing autonomous text classification systems. In E. A. Fox, P. Ingwersen, and R. Fidel, editors, *SIGIR '95: Proceedings of the 18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 246–254, New York, 1995. Association for Computing Machinery.

[7] D. D. Lewis, Y. Yang, T. Rose, and F. Li. RCV1: A new benchmark collection for text categorization research. *Journal of Machine Learning Research*, 2003. To appear.

[8] B. D. Ripley. *Pattern Recognition and Neural Networks.* Cambridge University Press, Cambridge, 1996.

[9] S. Robertson and I. Soboroff. The TREC 2002 filtering track report. In E. M. Voorhees and D. K. Harman, editors, *Text REtrieval Conference (TREC 2002)*, Gaithersburg, MD, 2003. Department of

Commerce, National Institute of Standards and Technology. To appear.

[10] J. J. Rocchio, Jr. Relevance feedback in information retrieval. In G. Salton, editor, *The SMART Retrieval System: Experiments in Automatic Document Processing*, pages 313–323. Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1971.

[11] T. Rose, M. Stevenson, and M. Whitehead. The reuters corpus volume 1 - from yesterday's news to tomorrow's language resources. In *Proceedings of the Third International Conference on Language Resources and Evaluation*, Las Palmas de Gran Canaria, 29-31 May 2002.

[12] G. Salton and M. J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill Book Company, New York, 1983.

[13] J. M. Tague. The pragmatics of information retrieval experimentation. In K. Sparck Jones, editor, *Information Retrieval Experiment*, chapter 5. Butterworths, London, 1981.

[14] Y. Yang. Expert network: Effective and efficient learning from human decisions in text categorization and retrieval. In W. B. Croft and C. J. van Rijsbergen, editors, *SIGIR 94: Proceedings of the Seventeenth Annual International ACM-SIGIR Conference on Research and Development in Information Retrieval*, pages 13–22, London, 1994. Springer-Verlag.

[15] Y. Yang. An evaluation of statistical approaches to text categorization. Technical Report CMU-CS-97-127, Carnegie Mellon University, Pittsburgh, PA, April 10 1997.

[16] Y. Yang. An evaluation of statistical approaches to text categorization. *Information Retrieval*, 1(1/2):69–90, 1999.

[17] Y. Yang. A study on thresholding strategies for text categorization. In W. B. Croft, D. J. Harper, D. H. Kraft, and J. Zobel, editors, *SIGIR 2001: Proceedings of The 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 137–145, New York, NY, 2001. Association for Computing Machinery.

[18] Y. Yang, T. Ault, T. Pierce, and C. W. Lattimer. Improving text categorization methods for event tracking. In N. J. Belkin, P. Ingwersen, and M.-K. Leong, editors, *SIGIR 2000: Proceedings of The 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 65–72, New York, NY, 2000. Association for Computing Machinery.