A Close Look at Nearest Neighbor Text Classification

David D. Lewis[1,2] and Vladimir Menkov[2]

Ornarose, Inc.[1]; Consulting Computer Scientist[2]

Abstract

(**U**) 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, as well as how to partially fix that problem. Interestingly, our experiments show the classic algorithm, not widely used in text classification, has the highest effectiveness of the three studied.

1 Introduction

(**U**) A nearest neighbor text classification (NNTC) system looks up, for each new document to be classified, similar documents ("neighbors") in a pool of manually classified documents. It then classifies the new document based on the categories to which these neighbors have been assigned. This intuitively appealing approach has been widely used in both research systems and commercial products.

(**U**) Advantages of NNTC include: efficiency with large numbers of classes (since neighbor lookup need be done only once) [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).

(**U**) Much of the popularity of NNTC in information retrieval (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, and even the same software can be used, particularly in experimental studies.

(**U**) This has meant, however, that NNTC methods for IR 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].

(**U**) In this paper we take a closer look at two weighted nearest neighbor algorithms popular in NNTC (Yang's original 1994 algorithm [14], and her more recent *kNN.avg1* algorithm [18]), and compare them with the classic unweighted k-NN classifier that has been widely studied and used in pattern recognition. We know of no previous systematic comparison between weighted and unweighted nearest neighbor approaches for text classification.

(**U**) We begin by describing each of three algorithms and the problem of setting thresholds. We then provide an alternate mathematical characterization of *kNN.avg1*, suggesting that no single, globally selected, threshold is likely to work well with *kNN.avg1* classifiers, except in very limited circumstances. We hypothesize that *kNN.avg1* will have a certain unusual behavior as neighborhood size is changed, and that its effectiveness can be improved by the use of locally selected thresholds.

(**U**) We then present an experimental study of the three algorithms, confirming our hypotheses about *kNN.avg1* and characterizing the behavior of all the algorithms as neighborhood size varies. We end by speculating on how work from NNTC and pattern recognition could be combined to lead to more effective NNTC algorithms.

2 Nearest Neighbor Classification Algorithms

(**U**) Nearest neighbor classification (NNC) algorithms have been extensively studied in the learning sciences [2, 5, 8]. Like other supervised learning algorithms, they are given a set of examples of known class membership (training examples), and produce a classifier that assigns a new example (a test example) to one of several classes.

(**U**) As with other supervised learning algorithms, we can measure the quality of a NNC classifier by using it to classify a set of test examples disjoint from the training data. We then compare the NNC-produced classifications with known correct classi-

fications, and compute a measure of agreement or disagreement, i.e. an *effectiveness measure*. (See Section 5.3 for more discussion of effectiveness measures.)

(**U**) Most supervised learning algorithms do considerable computation to transform their training data into a classifier, typically one taking the form of a mathematical function. Very little computation is then required to apply the classifier to test examples.

(**U**) NNC algorithms, in contrast, do little advance computation on the training data, but substantial computation at the time each test example is classified. When the test data is presented, the NNC trains a new *local* classifier for each test example, and discards that local classifier once the test example has been classified. During the training of each local classifier, only a subset of the training examples (its *neighbors*), those most similar to that test example, are used.

(**U**) Nearest neighbor methods are sometimes called *lazy learning* methods, because learning is delayed until it is ready to be applied. They are also called *local learning* methods, as the training data is restricted to the neighbors of a test example.

2.1 Cosine Similarity Measure

(**U**) Applying NNC to text data requires a way to measure similarity between a test document and training documents. Much of our discussion does not rely on the specific choice of the similarity measure. It is worth mentioning, however, that the cosine correlation [12, p. 124] is the most commonly used similarity measure in NNTC. This is simply the dot product

$$\text{sim}(\vec{x}, \vec{z}) = (\frac{\vec{x}}{\|\vec{x}\|}, \frac{\vec{z}}{\|\vec{z}\|}) = \sum_j (\frac{x_j}{\|\vec{x}\|}, \frac{z_j}{\|\vec{z}\|}), \tag{1}$$

of test document $\vec{x}$ with training document $\vec{z}$, after normalizing both to have a Euclidean norm $\|\vec{w}\| = \sqrt{(\vec{w}, \vec{w})}$ of 1.0. (We assume that the documents have been converted to numeric vectors by some text representation strategy, e.g. as in Section 5.1.) With the above normalization, 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 $\|\vec{x} - \vec{z}\| = \sqrt{2(1 - (\vec{x}, \vec{z}))}$, assuming again that vectors have been normalized to have a Euclidean norm of 1.0.

2.2 The Traditional k-NN Classifier

(**U**) 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 the test example to the class that is most frequent among the neighbors.

(**U**) For binary classification, which is the focus of this paper, this corresponds to assigning the class to which at least half the neighbors belong. In the binary case (classifying documents as belonging or not belonging to some category C), we will refer to the two classes as *Yes* and *No*. Assignment to the *Yes* class is equivalent to deciding that the document belongs to C, and assignment to the *No* class is equivalent to deciding that the document does not belong to C.

(**U**) We can view the traditional k-NN classifier as computing this score of the test document $\vec{x}$ with respect to the *Yes* class as follows:

$$s(\vec{x}) = \sum_{\vec{z} \in P_k(\vec{x})} 1.0, \tag{2}$$

where $\vec{z}$ is a training document and $P_k(\vec{x})$ is the intersection of $N_k(\vec{x})$ with class *Yes*. Assigning the majority class corresponds to assigning *Yes* if $s(\vec{x})$ is greater than $k/2$ and assigning *No* otherwise.[1]

(**U**) 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 cause the learning algorithm to produce *less* flexible, less complex classifiers. Indeed, the effective number of model parameters for a k-NN classifier is n/k where n is the training set size [5, pp. 15-16].

(**U**) To get an intuition as to why this is so, consider a peculiar variant of the standard k-NN classifier. The variant classifier breaks the training data up in advance into nonoverlapping size k groups of examples. For each test example it finds the group to which the test example is most central, and assigns the test example to the class

[1] In the case of a tie, i.e. $s(\vec{x}) = k/2$, theoretical analyses usually assume that the class is chosen randomly with probability 1/2. For an operational classifier one will usually want a deterministic tie-breaking rule.

(*Yes* or *No*) which is most frequent in that group. There would be (roughly) n/k such groups, and we would need to estimate the proportion of *Yeses* in each one, for a total of n/k numeric parameters. Larger values of k would mean fewer parameters to estimate, i.e. less complex models. The actual k-NN classifier corresponds to a version of this where we always have the good luck that the test example falls exactly in the middle of one of the groups, but the complexity analysis is similar.

(**U**) Thus the 1-NN classifier is the most flexible of traditional k-NN classifiers, and also the most susceptible to overfitting. A $1,000,000$-NN classifier would be very robust against overfitting, but would have very limited classification ability unless a monstrously large training set was available. Indeed, for $k <= 1,000,000$, it simply assigns the majority class to all test examples.

2.3 The Yang94 Algorithm

(**U**) 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 by her as just k-NN [16] . For clarity, we will refer to this algorithm as *Yang94*.

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

$$s(\vec{x}) = \sum_{\vec{z} \in P_k(\vec{x})} \text{sim}(\vec{x}, \vec{z}), \tag{3}$$

where $\text{sim}(\cdot, \cdot)$ is some similarity measure between documents.

(**U**) The *Yang94* method has typically been used with a similarity measure that takes values between 0 and 1, most commonly the cosine correlation described in Section 2.1. With similarity measures of this form, *Yang94* treats a training example that is identical (similarity 1.0) to the test example in the same way that traditional k-NN would, but gives lower weights to training examples that differ in any way from the test example.

(**U**) The original presentation of *Yang94* used this method for ranking controlled vocabulary categories with respect to a test document. In later work [16, 17, 18], Yang explored thresholding (Section 3) and other approaches for converting class scores to binary classification decisions.

2.4 Yang's Later NNTC Algorithms

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

$$s(\vec{x}) = \sum_{\vec{z} \in P_k(\vec{x})} \text{sim}(\vec{x}, \vec{z}) - \sum_{\vec{z} \in Q_k(\vec{x})} \text{sim}(\vec{x}, \vec{z}) \tag{4}$$

where $Q_k(\vec{x}) = N_k(\vec{x}) \backslash P_k(\vec{x})$ is the set of neighbors of $\vec{x}$ not belonging to class *Yes*.

(**U**) The $kNN.sum$ algorithm is potentially problematic for low frequency classes. When the neighborhood is at all large, a document's score might be dominated by similarities to many negative examples rather than similarities to the few, but more important, positive examples. This motivated Yang's second algorithm, *kNN.avg1*, which computes $s(\vec{x})$ as

$$s(\vec{x}) = \frac{1}{|P_k(\vec{x})|} \sum_{\vec{z} \in P_k(\vec{x})} \text{sim}(\vec{x}, \vec{z}) - \frac{1}{|Q_k(\vec{x})|} \sum_{\vec{z} \in Q_k(\vec{x})} \text{sim}(\vec{x}, \vec{z}). \tag{5}$$

Here $|S|$ is the number of documents in set S. If either $|P_k(\vec{x})|$ or $|Q_k(\vec{x})|$ is 0, the corresponding term is ignored. The intent is that by working with average rather than total similarities, the problem of large number of negative examples in the neighborhood will be reduced.

(**U**) Yang's third algorithm, $knn.avg2$, goes further by using, instead of $P_k(\vec{x})$ and $Q_k(\vec{x})$, two independently computed neighborhoods, one containing the k positive examples closest to the test document, and the other the k closest negative examples. This is not a nearest neighbor classifier in the usual sense. The least similar training example might be included in a "neighborhood" (particularly when few positive examples are available), while much more similar negative examples might be omitted. We therefore will not discuss $knn.avg2$ further here.

3 Thresholding

(**U**) Each algorithm described above can be viewed as computing a score for the class *Yes* with respect to a test example. The question then is how to go from a score for *Yes* to a decision of whether to assign *Yes* or *No* to the document. The most common approach is compare the test document's score with a threshold value, and assign *Yes* if the score is greater than or equal to the threshold. We discuss two approaches to choosing thresholds below.

3.1 Analytic Thresholding

(**U**) 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. When there are two classes, this is equivalent (Equation 2) to computing a score equal to the number of *Yes* neighbors of the test instance, and deciding *Yes* for the test instance if that score is equal or greater than a threshold value, $k/2$.

(**U**) Most past research on the traditional k-NN classifier uses error rate (proportion of test examples classified incorrectly) as the effectiveness measure. If a different effectiveness measure is used, is a threshold of $k/2$, or any threshold at all, still appropriate?

(**U**) We can examine this question by looking at yet another formulation of the traditional k-NN classifier. If we divide the score in Equation 2 by the neighborhood size k:

$$s(\vec{x}) = \frac{1}{k} \sum_{\vec{z} \in P_k(\vec{x})} 1.0, \tag{6}$$

and divide the threshold $k/2$ by k as well, the behavior of the classifier will of course be unchanged. The threshold would then be $k/2k = 0.5$. The score is now equal to the maximum likelihood estimate of $p(\mathit{Yes}|\vec{x})$, the posterior probability that test instance $\vec{x}$ belongs to class *Yes*, estimated using only the data in the neighborhood [8, p. 191].

(**U**) Deciding *Yes* when an estimate $\hat{p}(\mathit{Yes}|\vec{x})$ of the true probability $p(\mathit{Yes}|\vec{x})$ is greater

than 0.5 is in fact the Bayes optimal decision rule, under the assumption that the estimated probability is equal to the true probability [3].

(**U**) Viewing a traditional k-NN classifier as implicitly computing a local estimate of $p(\mathit{Yes}|\vec{x})$ provides a natural way to generalize the k-NN classifier to any effectiveness measure. We simply make classification choices such as to maximize or minimize (as appropriate) the expected value of the test set effectiveness under the assumption that the estimated probabilities are correct [3, 6]. When optimizing expected effectiveness leads to strategy based on comparing a score to a threshold, we call this approach *analytic* thresholding. We give one example of computing an analytic threshold in Section 5.4.

3.1.1 Anomalies of the Analytic Approach

(**U**) The fact that the value $\hat{p}(\mathit{Yes}|\vec{x})$ implicitly computed by a k-NN classifier is only an estimate of $p(\mathit{Yes}|\vec{x})$ means that an analytic threshold is not necessarily optimal. To take an extreme case of misestimation, suppose k is larger than n_y, the number of positive examples of the class in the training data. Then the traditional k-NN classifier's implicit estimate $\hat{p}(\mathit{Yes}|\vec{x})$ can never be higher that n_y/k, no matter how high the unknown true value of $p(\mathit{Yes}|\vec{x})$ is for a particular test document. For very small values of n_y, it may be impossible for $\hat{p}(\mathit{Yes}|\vec{x})$ to ever exceed the analytic threshold for a given value of k. (See Section 6 and Figure 2 for an example.) In this circumstance, a classifier using some lower threshold value might well have better effectiveness.

3.2 Thresholding using Validation Data

(**U**) Another approach to thresholding is to treat an NNTC's score as a number roughly monotonic with, but not an estimate of, $p(\mathit{Yes}|\vec{x})$. In this case it is reasonable to simply search directly for a threshold that optimizes whatever effectiveness measure is of interest. This can be done even for NNTCs where we do not have a probabilistic interpretation for the scores, or where we do not trust the quality of the probability estimates.

(**U**) Yang [7, 17] has suggested using cross-validation or a held out validation set to choose the threshold for an NNTC in this case. We consider here a particular validation approach (not used by Yang) called leave-one-out cross-validation (LOOCV) [8, pp. 69-71]. LOOCV can be used to set the threshold for a binary NNTC as follows:

1. For each document in the training set of n documents
   (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. For each unique score observed on a training document, compute the effectiveness that would result if the entire training set were classified using that score as the threshold. Choose the threshold that gives the highest effectiveness.

(**U**) The effect of the above is to treat each training document as if it were a test document. We compute the score that would be given this hypothetical test document by an NNTC trained on the remaining $n-1$ training examples and using a neighborhood size of k. The scores produced by the n different NNTCs (each based on $n-1$ examples) for test examples would be similar to those produced by a single NNTC based on all n examples. Thus the threshold chosen by LOOCV is reasonable for the NNTC based on all n examples.

(**U**) LOOCV is sometimes viewed as too computationally expensive to be practical, but for nearest neighbor classifiers it is quite reasonable. Step 2 can be made very efficient by sorting the training documents by score, and preceding through the sorted list from highest to lowest score, incrementally updating effectiveness for each threshold tried. This process can be terminated early if, as is common, a point is reached where too few positive documents remain at lower scores for a lower threshold to produce better effectiveness than already seen.

(**U**) The major computational cost then is finding the neighbors for each training document, but this takes no more time than classifying an equivalent number of test

documents would. Sorting scores requires $O(n \log n)$ time, after which the threshold can be found in $O(n)$ time. Further speedups in average time can be gotten by special treatment of documents with similarities of 0. An estimate of where the threshold is likely to fall can sometimes be used to avoid sorting all nonzero scores.

3.2.1 Anomalies of Validation-Based Thresholding

(**U**) LOOCV and other validation approaches to thresholding are themselves supervised learning algorithms, and can produce peculiar results when little training data is available. To take an extreme but easy to analyze case, suppose there are $n \geq 3$ training examples, $n_y = 1$ positive example, and that we are finding a threshold for a traditional 2-NN classifier with error rate as the effectiveness measure.

(**U**) When the single positive example (call it $\vec{x}_p$) gets its turn to be treated as a test example during LOOCV, it is guaranteed to get a score (i.e. number of positive neighbors) of 0. (Since $\vec{x}_p$ is the only positive example, and LOOCV does not allow an example to participate in classifying itself, the two neighbors of $\vec{x}_p$ must be negative.) When any negative example is treated as a test example, it will get a score of either 0 or 1, depending on whether $\vec{x}_p$ is a neighbor.

(**U**) If any negative example gets a score of 1, a threshold of 2 will be chosen for the 2-NN classifier. A threshold of 2 applied to the LOOCV-generated scores results in only one error ($\vec{x}_p$ with its score of 0). But a threshold of 2 also means that the classifier never says *Yes*, since no test document will have two positive neighbors ($\vec{x}_p$ being the only potential positive neighbor).

(**U**) Only if all negative examples get a score of 0 during the LOOCV process will a threshold of 1 be chosen. This requires a peculiar data configuration where no negative example has the positive example in its neighborhood.

(**U**) By comparison, the analytic threshold for a k-NN classifier is simply $k/2$ on the scale of number of examples (0.5 in terms of estimated probability), i.e. 1 for a 2-NN classifier. This is arguably more reasonable, though of course no choice of threshold will give a terribly effective classifier when so few positive examples are available. The point is that cross-validation is itself a supervised learning process, and gives unstable results when little training data is available.

4 An Alternative View of kNN.avg1

(**U**) In this section, we focus on Yang's *kNN.avg1* algorithm, finding another interesting anomaly. When the dot product (Section 2.1) is used as the similarity measure in *kNN.avg1*, the score of a document, $s(\vec{x})$, can be rewritten as

$$s(\vec{x}) = \frac{\sum_{\vec{z} \in P_k(\vec{x})} \sum_j x_j z_j}{|P_k(\vec{x})|} - \frac{\sum_{\vec{z} \in Q_k(\vec{x})} \sum_j x_j z_j}{|Q_k(\vec{x})|}. \tag{7}$$

This is in turn equal to the dot product

$$s(\vec{x}) = (\vec{x}, \vec{q}(\vec{x})) \tag{8}$$

of the document vector $\vec{x}$ with the vector $\vec{q}(\vec{x})$ whose components are given by

$$q_j = \frac{\sum_{\vec{z} \in P_k(\vec{x})} z_j}{|P_k(\vec{x})|} - \frac{\sum_{\vec{z} \in Q_k(\vec{x})} z_j}{|Q_k(\vec{x})|}. \tag{9}$$

In both cases j is an index to term weights in the document vectors. The vector $\vec{q}(\vec{x})$ is exactly the vector that would result from applying the Rocchio learning algorithm [10] to the neighbors of $\vec{x}$, using parameters $\alpha = 0$ (no query), $\beta = \gamma = 1$, and no feature selection.

(**U**) 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 two roles: a similarity measure defining a set of unweighted neighbors, and the function used to apply the locally trained Rocchio model to the test instance.

(**U**) Recasting *kNN.avg1* as a local version of Rocchio points out a problem. Yang assumes that a single global threshold will be used with *kNN.avg1*, with the threshold found by a validation procedure. However, as the above shows, each score produced by a *kNN.avg1* model results from a different local Rocchio model, and these models need not necessarily produce comparable scores. A true local learning version of Rocchio would compute not only the Rocchio linear model using the local data, it would compute the Rocchio threshold at least taking into account the local data. We would expect *kNN.avg1* to work well only in those circumstances where a global thresholding procedure will produce roughly the same result as a local thresholding procedure.

(**U**) One such circumstance is when k is close to n, the training set size. In this case, all test documents will generate roughly the same local Rocchio model, because all neighbors will contain many of the same training documents. Either a local or global thresholding method will choose roughly the same threshold.

(**U**) The other circumstance is when $k = 1$. First, we note that a local threshold setting procedure 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. We then argue that the global threshold setting procedure produces a classifier that acts similarly to a 1-NN classifier, though here we have only an asymptotic argument.

(**U**) Consider the behavior of any validation-based global threshold setting procedure as the training set size approaches infinity. Given an infinite training set, the single nearest neighbor of any validation example will be identical to the test example, so all *kNN.avg1* scores will be either -1.0 or 1.0. Validation then has only three choices for thresholds: -1.0, 1.0, or some value greater than 1.0.

(**U**) A threshold of -1.0 corresponds to saying *Yes* to all examples. For the class frequencies and effectiveness measures typically seen in text classification this would give poor effectiveness and so would not be chosen.

(**U**) A threshold greater than 1.0 means saying *No* to all examples. This threshold would be chosen only if rejecting all examples has higher expected effectiveness than classifying an example based on a training example identical to it. For the common effectiveness measures this can be shown to imply a very unlikely distribution of class labels, but we will omit the detailed analysis here.

(**U**) Therefore, we argue that asymptotically a threshold of 1.0 will be chosen. That gives behavior identical to that of a 1-NN classifier. Though this argument holds only for infinite training sets, it should become increasingly true as training set size increases.

(**U**) For neighborhood sizes that are neither tiny nor a substantial fraction of the training set, *kNN.avg1* scores are produced by local Rocchio models based on moderate sized, mostly non-overlapping training sets. There is no reason to believe these scores will be on a common scale, and thus a single global threshold should work poorly in this case, giving low effectiveness.

(**U**) The above analysis suggests that *kNN.avg1* should exhibit an unusual pattern of effectiveness as neighborhood size is varied. Effectiveness at very small neighborhood sizes should be reasonable (similar to that of a traditional 1-NN classifier), as should effectiveness at very large neighborhood sizes (similar to that of a traditional Rocchio classifier). In between, effectiveness should degrade.

(**U**) This behavior, with good effectiveness at extreme values of the complexity parameter and poor effectiveness at intermediate values, is exactly the opposite of normal behavior of a learning algorithm as the complexity parameter is varied. It was therefore of interest to see whether experiment bears out this theoretical prediction. More to the point, our analysis suggests ways to improve *kNN.avg1*, as discussed in the following section.

4.1 Local Thresholding with *kNN.avg1*

(**U**) Our analysis suggests that the *kNN.avg1* might be improved by training the classifier threshold locally rather than globally. To classify document $\vec{x}$, we would compare its *kNN.avg1* score $s(\vec{x}) = (\vec{x}, \vec{q}(\vec{x}))$ not with a pre-set global threshold, but with some threshold $\tau(\vec{x})$ computed based on the neighbors of $\vec{x}$.

(**U**) A straightforward approach would be to apply the usual LOOCV procedure to the set of "$\vec{x}$-based Rocchio scores" computed as dot products $(\vec{z}, \vec{q}(\vec{x}))$ of the local Rocchio vector $\vec{q}(\vec{x})$ for document $\vec{x}$ with a number of training-set documents $\vec{z}$ from some vicinity $N_m(\vec{x})$ of document $\vec{x}$. This size m of that vicinity might be the same as k, or it might be chosen differently. This true local Rocchio method appears entirely logical; however, implementing it would be rather costly and will take us far out of the k-NN framework. To obtain the m scores $(\vec{z}, \vec{q}(\vec{x}))$ needed for finding the local threshold $\tau(\vec{x})$, we will need either to explicitly compute the vector $\vec{q}(\vec{x})$ (which may be quite dense, if k is high) and then compute the m dot products, or to compute the mk dot products between each vector in $N_m(\vec{x})$ and each vector in $N_k(\vec{x})$.

(**U**) Another choice of local threshold, more in line with the k-NN data organization, is a threshold directly based on the actual *kNN.avg1* scores (8). That is, to obtain the individualized threshold" $\tau(\vec{x})$, we apply LOOCV to the set of scores $s(\vec{z}) = (\vec{z}, \vec{q}(\vec{z}))$ of all training-set documents $\vec{z}$ from some vicinity $N_m(\vec{x})$. This thresholding

procedure, which we will refer to as LOOCV-local(m), is fairly inexpensive, since we only need to score each document $\vec{z}$ in the training set once, and can then reuse its score for each local threshold $\tau(\vec{x})$ where $\vec{z} \in N_m(\vec{x})$. While one may hope that the thresholds so obtained may be better than those produced by the global LOOCV, the fact remains that the document scores used in computing $\tau(\vec{x})$ are not the same as the Rocchio scores that would be produced using the Rocchio vector $\vec{q}(\vec{x})$; thus the resulting threshold may not be as good as the true local Rocchio threshold.

(**U**) The choice of the thresholding neighborhood size m for LOOCV-local(m) is of some importance as well. Although $m = k$ is the simplest choice, it is probably not optimal for small values of k, since it does not provide sufficient amount of information to the thresholding algorithm.

5 Methods

(**U**) 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).

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

5.1 Text Representation

(**U**) 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.

(**U**) 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

$$w_d(t) = \begin{cases} (1 + \ln\, n(t,d)) \ln\left(\frac{|\mathcal{D}|+1}{n(t)+1}\right), & \text{if } \; n(t,d) > 0 \\ 0 & \text{if } \; n(t,d) = 0, \end{cases} \tag{10}$$

where $n(t)$ is the number of training 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.

(**U**) 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. Thus, finally, the component x_j of the feature vector $\vec{x}$, corresponding to term t_j, will be

$$x_j = \frac{w_d(t_j)}{\sqrt{\sum_s w_d(s)^2}}. \tag{11}$$

(**U**) 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

(**U**) We use the dot product between two vectors as our measure of their similarity, as described in Section 2.1.

5.3 Effectiveness Measure

(**U**) 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 pre-set threshold can be optimal for the F-measure [6].

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

$$T11U = 2 \cdot A - 1 \cdot B + 0 \cdot C + 0 \cdot D = 2A - B \tag{12}$$

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.

(**U**) When we compute 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:

$$\begin{aligned} \text{MinNU} &= -0.5 && (13)\\ \text{MaxU} &= 2\,R && (14)\\ \text{T11NU} &= \text{T11U/MaxU} && (15)\\ \text{T11SU} &= \frac{\max(\text{T11NU}, \text{MinNU}) - \text{MinNU}}{1 - \text{MinNU}}. && (16) \end{aligned}$$

Here R is the number of test documents that are category members, and is equal to $A + C$ for any set of classification decisions. As a point of comparison, note T11U is always in the interval $[-N,$ MaxU] (where N is the number of test documents that are not category members), T11NU is always in [-N/MaxU, 1], and T11SU is always in [0,1]. The range [0, MaxU] of T11U maps to [0,1] of T11NU, and [1/3,1] of T11SU. Scores below this range correspond to classifiers that do worse than simply saying *No* to all test documents.

5.4 Algorithms, Parameters, and Implementation

(**U**) We implemented three algorithms described in Section 2: traditional k-NN, *Yang94*, and kNN.avg1, using a modified version of the Lemur toolkit (*www.cs.cmu.edu /~lemur*). Ties (for instance in comparing similarities to find the nearest neighbors)

Method name and paper section	Score function $s(\vec{x})$	Threshold		
		Analytic ($k/3$)	LOOCV	LOOCV-local(k)
Traditional (Section 2.2)	$\sum_{\vec{z}\in P_k(\vec{x})} 1.0$	$\surd$	$\surd$	+
Yang94 (Section 2.3)	$\sum_{\vec{z}\in P_k(\vec{x})} \mathrm{sim}(\vec{x},\vec{z})$	–	$\surd$	+
kNN.sum (Section 2.4)	$\sum_{\vec{z}\in P_k(\vec{x})} \mathrm{sim}(\vec{x},\vec{z}) -$ $- \sum_{\vec{z}\in Q_k(\vec{x})} \mathrm{sim}(\vec{x},\vec{z})$	–	+	+
kNN.avg1 (Section 2.4)	$\frac{1}{\lvert P_k(\vec{x})\rvert} \sum_{\vec{z}\in P_k(\vec{x})} \mathrm{sim}(\vec{x},\vec{z}) -$ $- \frac{1}{\lvert Q_k(\vec{x})\rvert} \sum_{\vec{z}\in Q_k(\vec{x})} \mathrm{sim}(\vec{x},\vec{z})$	–	$\surd$	$\surd$

Table 1: Summary of k-NN methods described in the paper. A check mark ($\surd$) in the table means that we have carried out experiments with this method; a + means that a method makes sense, but we have not implemented it. A '–' means that the combination of the similarity function and the thresholding method is not sensible.

were broken deterministically in favor of higher document IDs. (We also tested breaking ties using a pseudorandom hash on document ID, but found only trivial differences in results.)

(**U**) For all three algorithms, we carried out experiments with global thresholds trained by LOOCV. In addition, we tested alternative thresholding approaches for two of the algorithms.

(**U**) First, for kNN.avg1, we tested LOOCV-local(k) thresholding (i.e., choosing $m = k$), as described in Section 4.1.

(**U**) Second, for traditional k-NN we tested analytic thresholding (Section 3.1). This treats the traditional k-NN classifier as implicitly estimating $p(\mathit{Yes}|\vec{x})$, and chooses a threshold that optimize the expected value of effectiveness under the assumption that $\hat{p}(\mathit{Yes}|\vec{x}) = p(\mathit{Yes}|\vec{x})$.

(**U**) For a linear utility measure, expected utility is maximized if we assign *Yes* when

$$p(\mathit{Yes}|\vec{x}) \geq \frac{(\lambda_{yn} - \lambda_{nn})}{(\lambda_{ny} - \lambda_{yy} + \lambda_{yn} - \lambda_{nn})}, \tag{17}$$

where λ_{rs} is the utility of deciding class r (y for *Yes* or n for *No*) when class s is correct. For our utility function, $\lambda_{yy} = 2, \lambda_{yn} = -1, \lambda_{nn} = 0$, and $\lambda_{ny} = 0$, the resulting threshold is $(-1-0)/(0-2+(-1)-0) = 1/3$. That is, we had the traditional k-NN classifier assign *Yes* when $k/3$ or more of the neighbors of a test instance had the *Yes* label.

(**U**) Table 1 summarizes the methods described in this paper, and marks those which have been used in the experiments.

(**U**) The main parameter varied in experiments was the neighborhood size k. Except for $k = 1$, $k = 2$, and $k = 23149$ (the full training set size), all neighborhood sizes were multiples of 3 to allow an exact analytic threshold ($k/3$) for the T11U effectiveness measure. For $k = 1$ and $k = 2$ we used 1 as the analytic threshold, and for $k = 23149$ we used 7716.

6 Results

(**U**) Table 2 shows macroaveraged T11SU for a range of neighborhood sizes for each of the five algorithms. For convenience, the same data are presented graphically in Figure 1.

(**U**) The most notable observation is that, despite the preference for weighted methods in NNTC work, the traditional k-NN algorithm (*Trad*) actually has a slightly higher peak effectiveness than the weighted ones, when thresholds for all algorithms are chosen using LOOCV.

(**U**) Looking in more detail, we see that *Trad* and *Yang94* with LOOCV thresholds show the classic 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. $k = 1$).

(**U**) For the most part *Trad* with an analytic threshold also follows the classic complexity vs. effectiveness pattern. The pattern is violated, however, for neighborhood sizes $k = 1$ and $k = 2$, where we use a threshold of 1, even though that is larger than

	Analytic	LOOCV			LOOCV-local(k)
k	Trad	Trad	Yang94	kNN.avg1	kNN.avg1
1	0.452	0.464	0.480	0.481	–
2	0.391	0.485	0.491	0.487	–
3	0.330	0.515	0.500	0.471	–
6	0.491	0.521	0.517	0.447	0.354
12	0.539	0.536	0.528	0.412	0.357
24	0.538	0.552	0.542	0.380	0.381
48	0.527	0.561	0.552	0.371	0.426
96	0.509	0.571	0.562	0.363	0.472
192	0.491	0.581	0.562	0.358	0.503
384	0.460	0.582	0.568	0.363	0.520
768	0.421	0.572	0.565	0.373	–
1536	0.396	0.553	0.551	0.400	–
3072	0.367	0.525	0.534	0.460	–
6144	0.352	0.495	0.513	0.524	–
12288	0.345	0.449	0.488	0.552	–
23149	0.336	0.336	0.474	0.549	–

Table 2: Test set T11SU, macroaveraged over 103 categories, for five nearest neighbor text classification methods at a range of sizes (k) of neighborhood.

$k/3$. Curiously, effectiveness actually improves with this violation, suggesting that the analytic threshold of 1 for $k = 3$ is on average too low. In fact, LOOCV with $k = 3$ picked a threshold of 2 for almost all categories. *Trad* with analytic thresholds is worse than *Trad* with LOOCV-thresholds at almost all neighborhood sizes.

(**U**) The poor effectiveness of analytic thresholding compared with LOOCV-based thresholding is less surprising for large neighborhood sizes, since the limited number of examples forces the implicit estimate of $p(\mathit{Yes}|\vec{x})$ to be too low (Section 3.1.1).

(**U**) We can see this in Figure 2, which plots category-level T11SU values for a 192-NN classifier against the number of positive training examples n_y for the category. Note that values based on analytic thresholds ($k/3 = 64$) are especially poor on categories with few positive training examples.

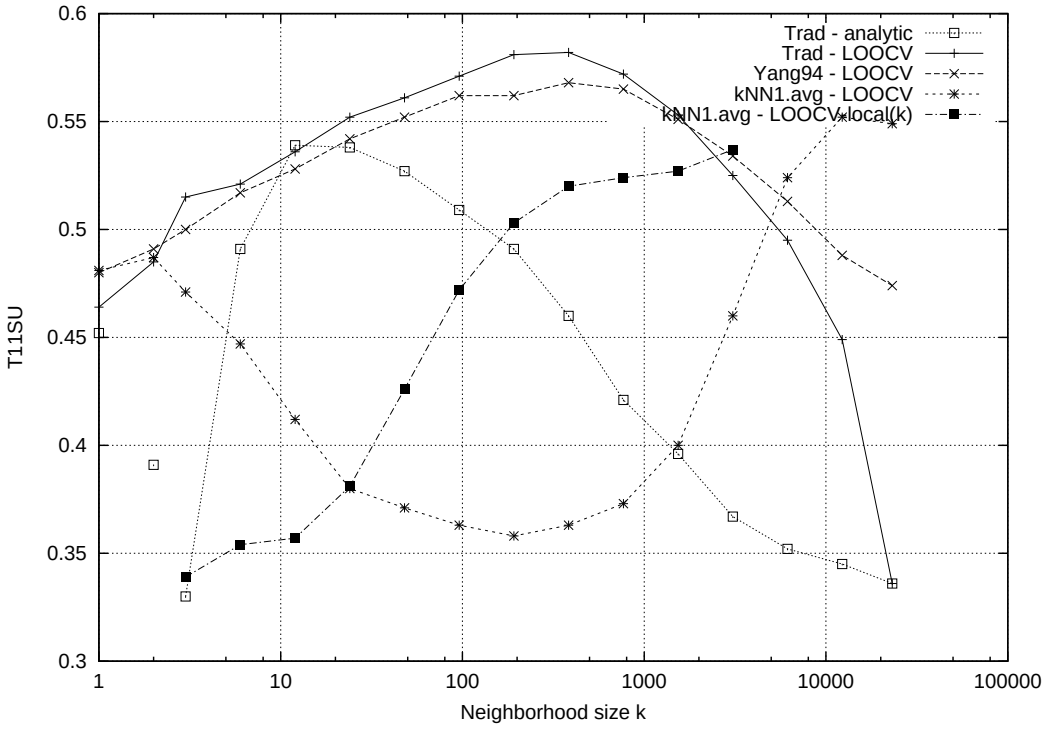

Figure 1: Test set T11SU, macroaveraged over 103 categories, for five nearest neighbor text classification methods at a range of sizes (k) of neighborhood. (Same data as in Table 2.)

(**U**) As noted in Section 3.2, k-NN with analytic thresholds is guaranteed to never assign a test example to the category if the number of positive training examples for the category is less than $k/3 = 64$. In practice, this bad behavior persists up to n_y's of 250 or so. Thus the T11U for these categories is therefore 0, and the T11SU is 1/3. In contrast, LOOCV can often find reasonable (i.e. lower) thresholds for these low frequency categories (down to $n_y = 20$ or so), achieving nontrivial test set utility scores.

(**U**) Finally, as hypothesized, the *kNN.avg1* algorithm shows a completely different pattern (Figure 1) than model complexity would predict. For $k = 1$ it has similar

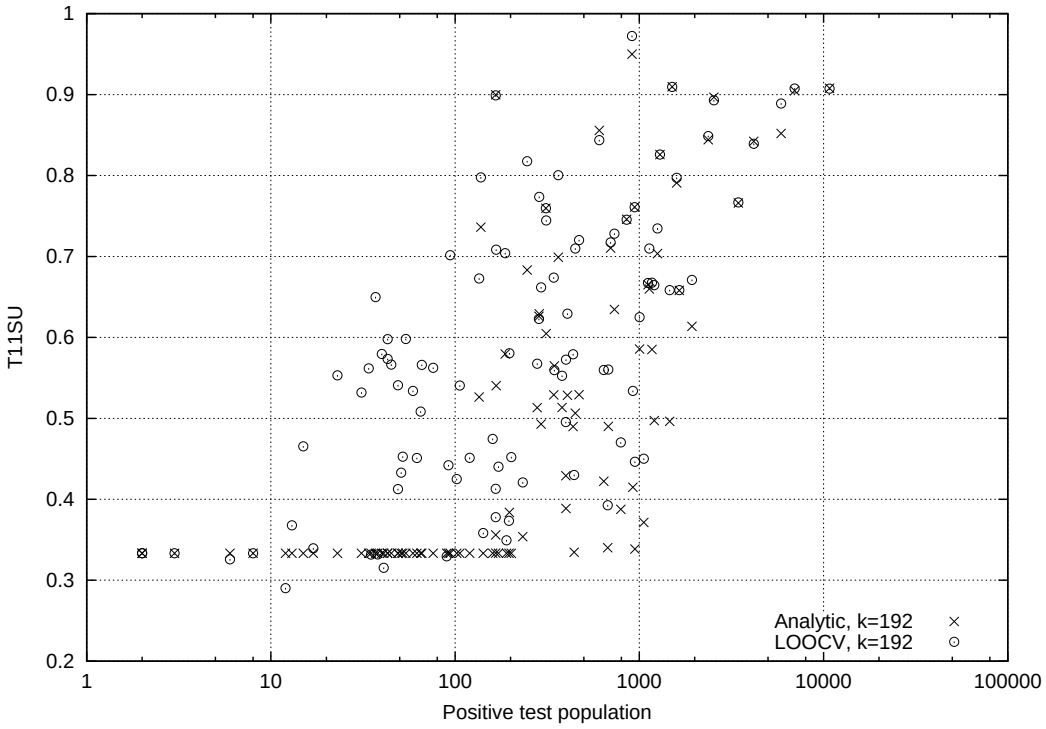

Figure 2: Test set T11SU against the number of the category's positive examples in the training set, for each of the 103 categories. Traditional k-NN with $k = 192$, with analytic thresholds ($k/3 = 64$), and with LOOCV threshold.

effectiveness 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.

(**U**) Comparison with the LOOCV-local(k) results shows that using even a simplistic local thresholding mechanism may noticeably improve performance of *kNN.avg1* at the intermediate neighborhood sizes. The comparatively poorer effectiveness at smaller neighborhood sizes suggests it might be desirable to use a larger size for the neighborhood used to compute the threshold than for the neighborhood used to

compute the linear model.

7 Summary

(**U**) Nearest neighbor classification algorithms are widely used in text classification, but tend to be based on analogies with text retrieval, rather than on a foundation of statistical classification theory. We have shown that the classic k-NN algorithm from pattern recognition, when used with the LOOCV thresholding procedure and the k value optimal for it, has text classification effectiveness at least as good as the two more popular, and more complex, weighted NNTC algorithms.

(**U**) 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, and demonstrated that using even a crude locally selected threshold can significantly improve its effectiveness.

(**U**) 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

(**U**) We thank Sabine Buchholz, Paul Kantor, David Madigan, Fred Roberts, and Yiming Yang for suggestions, comments, and/or feedback. Special thanks to Alex Genkin for his help with the evaluation code. We also express gratitude to Reuters, Ltd. for making Reuters Corpus Volume 1 available to the IR community. The authors thank the KD-D Group for its support through National Science Foundation grant number EIA-0087022 to Rutgers University. Any opinions or conclusions in this paper are the authors' and do not necessarily reflect those of the sponsors.

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.