Using *k*NN Model-based Approach for Automatic Text Categorization

Gongde Guo[1], Hui Wang[1], David Bell [2], Yaxin Bi [2], and Kieran Greer [1]

School of Computing and Mathematics, University of Ulster
Newtownabbey, BT37 0QB, Northern Ireland, UK [1]
{G.Guo, H.Wang, Krc.Greer@ulst.ac.uk}

School of Computer Science, Queen's University Belfast
Belfast, BT7 1NN, UK [2]
{DA.Bell, Y.Bi} @qub.ac.uk

Abstract An investigation has been conducted on two well known similarity-based learning approaches to text categorization: the *k*-nearest neighbor (*k*-NN) classifier and the Rocchio classifier. After identifying the weakness and strength of each technique, a new classifier called the *k*NN model-based classifier (*k*NNModel) has been proposed. It combines the strength of both *k*-NN and Rocchio.

A text categorization prototype system has been presented. It implements kNNModel along with kNN, Rocchio and Support Vector Machine (SVM). An evaluation has been carried out on two common document corpora, namely, the 20-newsgroup collection and the ModApte version of the Reuters-21578 collection of news stories. The experimental results show that the *k*NN model-based approach outperforms the *k*-NN and Rocchio classifiers, and is comparable to SVM, which is used as a benchmark in our experiments.

1 Introduction

Text categorization (TC) is the task of assigning a number of appropriate categories to a text document. This categorization process has many applications such as document routing, document management, or document dissemination [1]. Traditionally each incoming document is analyzed and categorized manually by domain experts based on the content of the document. Extensive human resources have to be spent on carrying out such a task. To facilitate the process of text categorization, automatic categorization schemes are required. Its goal is to build categorization models that can be used to classify text documents automatically.

Many classification methods have been applied to TC; for example, Naïve Bayes (NB) probabilistic classifiers [2], Decision Tree classifiers [3], Decision Rules [4], regression methods [5], Neural Network [6], *k*-NN classifiers [5, 7], Support Vector Machine (SVM) [8, 9], and Rocchio classifiers [10, 11] etc. In many applications, such as mining a large web repository, the efficiency of those schemes is often the key element to be considered. Sebastiani has pointed this out in his survey on text categorization [12].

k-NN and Rocchio are two classifiers frequently used for TC, and they are both similarity based. The *k*-NN algorithm uses the training examples as a basis for computing

similarity. For a data record *t* to be classified, its *k* nearest neighbors are retrieved, and this forms a neighborhood of *t*. Majority voting among the data records in the neighborhood is used to decide the classification for *t*. However, to apply *k*-NN we need to choose an appropriate value for *k*, and the success of classification is very dependent on this value. Moreover kNN is a lazy learning method as no model needs to be built (learned) and nearly all computation takes place at the classification stage. This prohibits it from being applied to areas where dynamic classification is needed for a large repository. However, *k*-NN has been applied to text categorization since the early days of its research [12] and is known to be one of the most effective methods on the Reuters corpus of newswire stories – a benchmark corpus in text categorization.

The Rocchio method [10, 11] however can deal with these problems to some extent. In its simplest form, it uses *generalized instances* (the averaged weights of instances belonging to one category is the generalized instance of this category) as models to replace the whole collection of training instances by summarizing the contribution of the instances belonging to each category. This method is efficient and easy to implement, since learning a classifier basically comes down to averaging weights, and classifying a new instance only requires computing the inner product between the new instance and the generalized instances. It is a similarity-based algorithm as it uses these generalized instances as a basis for computing the inner product based similarity. Moreover, the Rocchio method can deal with noise to some extent via summarizing the contribution of the instances belonging to each category. For example, if a feature mainly appears in many training instances of a category, its corresponding weight in the generalized instance will have a larger value for this category. Also, if a feature mainly appears in training instances of other categories, its weight in the generalized instance will tend to zero [1]. Therefore, the Rocchio method can filter out certain irrelevant features to some extent. On the other hand, one drawback of the Rocchio classifier is that it restricts the hypothesis space to the set of linear separable hyperplane regions, which has less expressive power than that of *k*-NN algorithms [1].

The generalized instance set (GIS) algorithm proposed by Lam et al [1] is an attempt to overcome the weakness of the *k*-NN algorithms and linear classifiers. The main idea for the GIS algorithm is to construct more than one generalized instance (GI) for a category, in contrast to only one generalized instance for a category in linear classifiers like the Rocchio method. Though better performance was been obtained in experiments, some drawbacks still exist. One drawback is that the performance of GIS depends on the order in which positive instances are chosen, and on the value of *k*. The other drawback is that the removal of the top-*k* instances from the training collection after obtaining a generalized instance will directly affect the calculation of further generalized instances.

Having identified the weakness and strength of each technique, we propose a new method called the *k*NN *model-based algorithm* (we call it *k*NNModel for simplicity) by combining the strengths of the *k*-NN and Rocchio classifiers (details to be presented later). The proposed method has been implemented and integrated into our text categorization prototype system along with the standard *k*-NN algorithm, the Rocchio algorithm, and the SVM algorithm. Extensive experiments have been conducted on two

common document corpora so that we can compare the categorization performance of different approaches. The results show that the proposed new method outperforms the *k*-NN algorithm and the Rocchio classifier in all experiments, and it is comparable to the SVM algorithm.

The rest of the paper is organized as follows. Section 2 describes the architecture of text categorization, as implemented in our prototype, and the functionality of each component. Section 3 introduces the basic idea of the proposed *k*NN model-based algorithm. The experimental results are described and discussed in Section 4. Section 5 ends the paper with a discussion on existing problems and addresses further research directions.

2 The Architecture of Text Categorization

In this section, we give an overview of the architecture of text categorization and describe the functionality of each component in text categorization. Text categorization usually comprises three key components: data pre-processing, classifier construction, and document categorization. Data pre-processing implements the function of transferring initial documents into a compact representation and will be uniformly applied to training, validation, and classification phases. Classifier construction implements the function of inductive learning from a training dataset, and document categorization implements the function of document classification. All three components together make text categorization practicable.

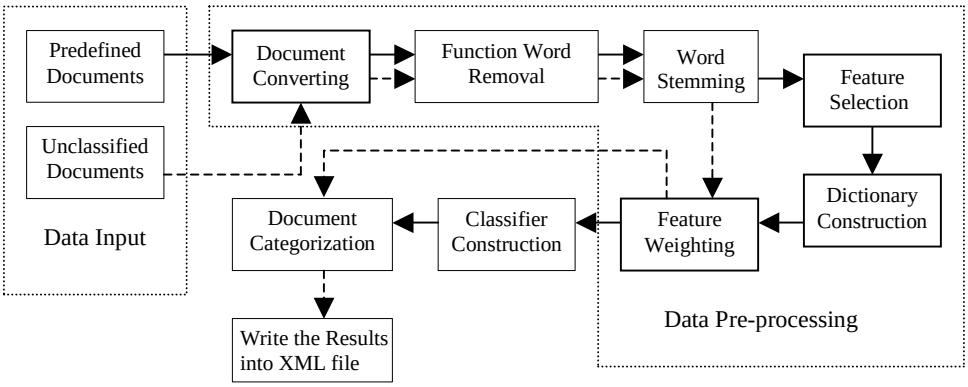

Fig. 1. The architecture of text categorization

In Fig. 1, the arrow with dashed line represents the data flow in the categorization process and the arrow with the solid line represents the data flow in the classifier construction process.

2.1 Data Pre-processing

Data pre-processing comprises six sub-components including document conversion function word removal, word stemming, feature selection, dictionary construction, and feature weighting. The functionality of each component is described as follows:

(1) Document converting –converts different types of documents such as XML, PDF, HTML, DOC format to plain text format.
(2) Function word removal –removes topic-neutral words such as articles (a, an, the), prepositions (in, of, at), conjunctions (and, or, nor), etc. from the documents.
(3) Word stemming –standardizes word's suffixes (e.g., labeling -- label, introduction -- introduct).
(4) Feature selection – reduces the dimensionality of the data space by removing irrelevant or less relevant features. In our prototype, we choose information gain as a feature selection criterion.
(5) Dictionary construction –constructs a uniform dictionary, which is used as a reference for converting the text document to a vector of features. Each feature in the vector corresponds to a word in the dictionary.
(6) Feature weighting –assigns different weights to words in the dictionary. We use standard normalized *tfidf* as the weighting function in our TC prototype system.

The information gain and the *tfidf* function are defined respectively as follows:

$$IG(t_k, c_i) = \sum_{c \in \{c_i, \bar{c}_i\}} \sum_{t \in \{t_k, \bar{t}_k\}} P(t,c).\log \frac{P(t,c)}{P(t).P(c)} \tag{1}$$

$$tfidf(t_k, d_i) = \#(t_k, d_i).\log \frac{|T_r|}{\#T_r(t_k)}, \; w_{ki} = \frac{tfidf(t_k, d_i)}{\sqrt{\sum_{i=1}^{|T_r|} (tfidf(t_k, d_i))^2}} \tag{2}$$

In formula 1, $P(t, c)$ is the probability of a term t occurs in a document belonging to category c, $P(t)$ is the probability of a term t occurs in a document, and $P(c)$ is the probability of a document belonging to a category c. In formula 2, $\#(t_k, d_i)$ is the number of times t_k occurs in d_i, and $\#T_r(t_k)$ is the document frequency of term t_k, that is the number of documents in T_r in which t_k occurs, $|T_r|$ is the number of documents in the training dataset, and w_{ki} is the normalized term weight.

Each document will be converted into a compact representation and will be applied to training, validation, and classification phases.

2.2 Classifier Construction

Classifier construction is the key component of automatic text categorization. The role of this component is to build a classifier by learning from predefined documents, which will be used to classify unknown documents. In our TC prototype system, we have implemented four classification algorithms: *k*-NN, SVM, Rocchio, and *k*NNModel. We give a brief description for each algorithm except for the *k*NN model-based algorithm, which will be described in detail in Section 3.

(1) SVM Based Categorization Algorithm

SVM has been introduced into TC by Joachims [8] and subsequently used by many TC researchers. Let $D_n = \{(\vec{d}_1, c_1), \cdots, (\vec{d}_n, c_n)\}$ be a set of *n* instances for training, where $\vec{d}_i \in \Re^N$, and category $c_i \in \{-1,+1\}$. SVM learns linear decision rules $f(\vec{d}) = sign\{\vec{w}.\vec{d} + \delta\}$ described by a weight vector *w* and a threshold δ. If D_n is linearly separable, SVM finds the hyperplane with maximum Euclidean distance to the closest training instances. If D_n is non-separable, the amount of training error is measured using slack variables ξ_i. Computing the hyperplane is equivalent to solving the following optimization problem [9].

$$minimize: \quad V(\vec{w}, \delta, \vec{\xi}) = \frac{1}{2}\vec{w} \cdot \vec{w} + C\sum_{i=1}^{n} \xi_i \tag{3}$$

$$subject\ to: \quad \forall_{i=1}^{n} : c_i[\vec{w} \cdot \vec{w} + \delta] \geq 1 - \xi_i \tag{4}$$

$$\forall_{i=1}^{n} : \xi_i > 0 \tag{5}$$

The factor *C* in (3) is a parameter used for trading off training error vs. model complexity. The constraints (4) require that all training instances be classified correctly up to some slack ξ_i.

(2) *k*-NN Algorithm

The *k*-NN algorithm is a similarity-based learning algorithm that has been shown to be very effective for a variety of problem domains including text categorization [5, 7]. Given a test document, the *k*-NN algorithm finds the *k* nearest neighbors among the training documents, and uses the categories of the *k* neighbors to weight the category candidates. The similarity score of each neighbor document to the test document is used as the weight of the categories of the neighbor document. If several of the *k* nearest neighbors share a category, then the per-neighbor weights of that category are added together, and the resulting weighted sum is used as the likelihood score of candidate categories. A ranked

list is obtained for the test document. By thresholding on these scores, binary category assignments are obtained [5].

[HW Comment: the above paragraph is hard to understand]

(3) Rocchio Algorithm

The Rocchio method is a linear classifier. Given a training dataset T_r, it directly computes a classifier $\vec{c}_i = < w_{1i}, w_{2i}, \ldots, w_{ri} >$ for category c_i by means of the formula:

$$w_{ki} = \beta \cdot \sum_{\{d_j \in POS_i\}} \frac{w_{kj}}{|POS_i|} - \gamma \cdot \sum_{\{d_j \in NEG_i\}} \frac{w_{kj}}{|NEG_i|} \quad (6)$$

where w_{kj} is the weight of the term t_k in document d_j, $POS_i = \{d_j \in T_r \mid \hat{\Phi}(d_j, c_i) = T\}$, and $NEG_i = \{d_j \in T_r \mid \hat{\Phi}(d_j, c_i) = F\}$. $\hat{\Phi}(d_j, c_i) = T$ (or *F*) means document d_j belonging to (or not belonging to) category c_i. In formula 6, β and γ are two control parameters used for setting the relative importance of positive and negative instances. The profile of c_i is the centroid of its positive training examples. A classifier built by means of the Rocchio method rewards the closeness of a test document to the centroid of the positive training instances, and its distance from the centroid of the negative training instances [12].

2.3 Document Categorization

The document categorization component directly uses the model created in the classifier construction phase to classify new instances. All documents to be classified must be pre-processed as in the classifier construction phase.

3 *k*NN Model-Based Algorithm

3.1 The Basic Idea of *k*NNModel

*k*NN is a case-based learning method, which keeps all the training data for classification. Being a lazy learning method prohibits it in many applications such as dynamic web mining for large repositories. One way to improve its efficiency is to find some representatives of the whole training data for classification, viz. building an inductive learning model from the training dataset and using this model (set of representatives) for classification. *k*NN is a simple but effective method for classification and has been shown to be one of the most effective methods on the Reuters corpus of newswire stories in text categorization. This motivates us to build a model for *k*NN to improve its efficiency

whilst preserving its classification accuracy. Rocchio, on the other hand, is quite efficient, as learning a classifier comes down to building a generalized instance for each category, which are then used as a basis for classification. A well-known disadvantage for Rocchio is its linear characteristic of dividing the space of data points linearly. As an example, consider Fig. 2. The small crosses and circles are two classes of instances with different categories. The big circle denotes the inference area of the classifier for the cross category. As the data points for the cross category occur in disjoint clusters, it causes the classifier built by the Rocchio method to miss most of them, as the centroid of these data points may fall outside all of these clusters. One way to overcome this drawback is to build several local centroids for one category. The number of centroids for a category depends on the dataset given for training. Fig. 3 shows the modeling result of *k*NNModel for category '+'.

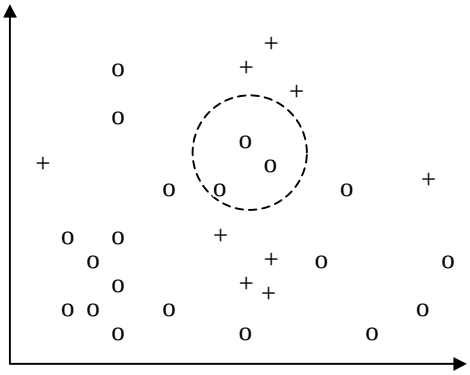

Fig. 2. The generalized instance of category "+" (Rocchio classifier)

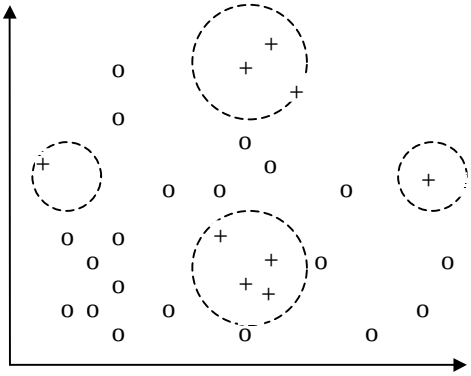

Fig. 3. The representatives of category "+" (*k*NNModel algorithm)

Note that, for ease of illustration, document similarities are viewed here (also in the following graphical illustration) in terms of Euclidean distance rather than, as is more common, in terms of dot product or cosine.

If we use Euclidean distance as our similarity measure for illustration, it is clear that many data points with the same class label are close to each other in many local areas. In each local region, the central data point d_i, called *centroid*, looking at Fig. 4 for example, with some extra information such as $Num(d_i)$ - the number of data points inside the local region and $Sim(d_i)$ - the similarity of the most distant data point inside the local region to d_i, might be a good representative of this local region. If we take these representatives plus the generalized instances of each category as a model to represent the whole training dataset, one merit is that it better reflects the data distributions of each category via both local and global generalization, thus probably improving its classification accuracy; the other merit is that it significantly reduces the number of data points used for classification, thereby improving efficiency. The disjoint clusters problem is readily solved by creating

more than one representative for a category. For a new data point to be classified, we take into account its distance to each generalized instance and whether it falls into each representative's inference area, and then decide the category it belongs to. See section 3.2 for more detail.

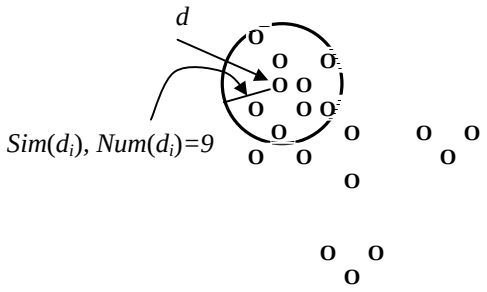

Fig. 4. The first obtained representative

In the model construction process, each data point has its largest neighborhood, which covers the maximal number of data points with the same class label. This largest neighborhood is called *local neighborhood*. Based on these local neighborhoods, the largest local neighborhood, called *global neighborhood*, can be obtained and seen as a representative of all the data points covered by it. For data points not covered by any representative, we repeat the above operation until all the data points have been covered by some representatives. The model comprises both the created representatives and the generalized instances of each category. Obviously, we don't need to choose a specific k for our method in the model construction process, the number of data points covered by a representative can be seen as an optimal k, but it is different for different representatives. The k is generated automatically in the model construction process. Further, using a list of chosen representatives plus the generalized instances of each category as a model not only reduces the amount of data used for classification, but also significantly improves its efficiency. Moreover, as the model reflects the true distribution of each category from both local and global aspects, it probably results in improvement of classification accuracy. Finally, multi-representatives for a category resolve the disjoint clusters problem.

3.2 Modeling and Classification Algorithm

Let $\boldsymbol{D}$ be a collection of n pre-labeled documents $\{d_1, d_2, \ldots, d_n\}$ with m categories. Document $d_i \in \boldsymbol{D}$ is represented by a feature vector of the form $<w_{i1}, w_{i2}, \ldots, w_{il}>$, where w_{ij} is the numeric weight for the j-th feature and l is the total number of features. Typically, each feature corresponds to a word or phrase appearing in the training corpus after the removal of function words, word stemming, and feature selection. In the kNN

model-based algorithm without any extra explanation, we will use the information gain (IG) as the default feature selection criterion, the term frequency combined with inverse document frequency (TFIDF) as the default weighting measure, and the cosine similarity Δ as the default similarity metric.

$$\Delta(d_i, d_j) = \frac{\sum_{k=1}^{l} w_{ik} \cdot w_{jk}}{\sqrt{\sum_{k=1}^{l} w_{ik}^2} \sqrt{\sum_{k=1}^{l} w_{jk}^2}} \quad (7)$$

In the following algorithms, we will use the term *data tuple* to represent a document with or without compact representation after data preprocessing as a convention. Additionally, a data tuple d_i is said to be *covered* by a representative $<Cls(d_j), Sim(d_j), Num(d_j), Rep(d_j)>$ if $\Delta(d_i, d_j) >= Sim(d_j)$.

The detailed model construction algorithm is described as follows:

(1) Create a similarity matrix from the given training dataset.
(2) Label all data tuples as "ungrouped".
(3) For each 'ungrouped' data tuple, find its local neighborhood.
(4) Among all the local neighborhoods obtained in step 3, find its global neighborhood N_i. Create a representative $<Cls(d_i), Sim(d_i), Num(d_i), Rep(d_i)>$ into $\boldsymbol{M}$ to represent all the data tuples covered by N_i, and then label as "grouped" all the data tuples covered by N_i.
(5) Repeat step 3 and step 4 until all the data tuples in the training dataset have been set to 'grouped'.
(6) Calculate the generalized instance $\vec{c_i}$ for each category c_i and add $<c_i, 0, n, Rep(\vec{c_i})>$ to $\boldsymbol{M}$.
(7) Model $\boldsymbol{M}$ consists of all the representatives collected from the above learning process.

In the above algorithm, $\boldsymbol{M}$ represents the created model. The elements of representative $<Cls(d_i), Sim(d_i), Num(d_i), Rep(d_i)>$ represent, respectively, the class label of d_i; the lowest similarity to d_i among the data tuples covered by N_i; the number of data tuples covered by N_i; and a representation of d_i itself. In step (4), if more than one neighborhood has the same maximal number of neighbors, we choose the one with the maximal value of $Sim(d_i)$, viz. the one with the highest density, as representative. In step 6, n is the number of data tuples for training. As the minimal value of cosine similarity is 0, we assign $Sim(\vec{c_i}) = 0$. This setting allows the global distribution of data tuples belonging to each category to have the certain inference on classifying new documents.

The classification algorithm is described as follows:

(1) For a new data tuple d_t to be classified, calculate its similarity to all representatives in model $\boldsymbol{M}$ as follows:

For each representative in the model, if d_t is covered by a representative <$Cls(d_j)$, $Sim(d_j)$, $Num(d_j)$, $Rep(d_j)$>, that is, $\Delta(d_t, d_j)$ is larger than $Sim(d_j)$, then add the contribution $\Delta(d_t, d_j)$ of d_t to $Cont$ ($Cls(d_j)$), viz. $Cont$ ($Cls(d_j)$) = $Cont$ ($Cls(d_j)$) + $\Delta(d_t, d_j)$.

(2) Classify d_t by $Cls(d_x)$ if $Cont(Cls(d_x)) = \max\{Cont(Cls(d_i)) \mid i=1,2,\ldots,m\}$.

In text categorization, it is common for instances in the training collection to contain some level of noise due to, for example, missing appropriate features, typographical errors in texts, or wrong category assigned by a human [1]. In an attempt to improve the classification accuracy for *k*NNModel, we integrated the pruning work into the process of model construction by modifying step 3 in the model construction algorithm to allow each local neighborhood to cover ε data tuples (called error tolerant degree) with different categories to the majority category in this neighborhood. Experimental results are reported in the next section.

4 Experiment and Evaluation

We have conducted experiments on two commonly used corpora in text categorization research: 20-newsgroups, and ModApte version of the Reuters-21578 collection of news stories. All documents for training and testing involve a pre-processing step, which includes tasks of function word removal, word stemming, feature selection, and feature weighting. We use information gain as the feature selection criterion and the normalized *tfidf* as the weighting function in our text categorization prototype system.

Experimental results reported in this section are based on the so-called "F_1 - measure," viz. the harmonic mean of precision and recall.

$$F_1(recall, precision) = \frac{2 \times recall \times precision}{recall + precision} \tag{8}$$

In the above formula, *precision* and *recall* are two standard measures widely used in text categorization literature to evaluate an algorithm's effectiveness [12] on a given category, where

$$precision = \frac{true \quad positive}{(true \quad positive) + (false \quad positive)} \times 100 \tag{9}$$

$$recall = \frac{true \quad positive}{(true \quad positive) + (false \quad negative)} \times 100 \tag{10}$$

In the formulas above, for a category *c*, the true positive is the number of documents belonging to category *c* that are correctly classified as category *c*; the false positive is the number of documents not belonging to category *c* that are incorrectly classified as category *c*; the false negative is the number of documents belonging to category *c that are* incorrectly classified as non-category *c* by a classifier.

We also use the *macroaveraged* F_1 to evaluate the overall performance of the algorithms on given datasets. The *macroaveraged* F_1 computes the F_1 values for each category and then takes the average over the per-category F_1 scores. Given a training dataset with *m* categories, assuming the F_1 value for the *i*-th category is $F_1(i)$, the *macroaveraged* F_1 is defined as:

$$macroaveraged\ F_1 = \frac{\sum_{i=1}^{m} F_1(i)}{m} \tag{11}$$

4.1 Datasets for Experiment

(1) Reuters-21578 Corpus

The Reuters dataset has been used in many text categorization experiments. The data was originally collected by the Carnegie group from the Reuters newswire in 1987. There are now at least five versions of the Reuters dataset widely used in the TC community. We choose the ModApte version of the Reuters-21578 collection of news stories, downloaded from http://www.daviddlewis.com/resources/testcollections/reuters21578. In our experiments, we used the seven most frequent categories from this corpus as our dataset for training and testing. Each category in this subset contains 200 documents. The seven most frequent categories are: {Acq, Corn, Crude, Earn, Interest, Ship, Trade}.

(2) 20-Newsgroup Corpus

The 20-newsgroup contains approximately 20,000 newsgroup documents being partitioned (nearly) evenly across 20 different newsgroups. We used the 20-news-18828 version downloaded from http://www.ai.mit.edu/~jrennie/20Newsgroups/. In this dataset duplicates have been removed and we only keep the "From" and "Subject" headers in each document. In experiments, we choose a subset from this corpus for training and testing. The subset includes 20 categories, and each category contains 200 documents.

4.2 Evaluation

In our experiments, we use the ten-fold cross validation method to evaluate the *macroaveraged* F_1 of different algorithms on the above two datasets. The basic settings of parameters for each algorithm as well as the F_1 value and the *macroaveraged* F_1 value are shown in Table 1 to Table 3 respectively. Different values of parameters have been tried on each algorithm to ensure that the experimental results faithfully reflect the performance of the algorithms. The value of k for k-NN algorithm includes 5, 15, 25, 35, 45, 55, 65; the pair value of (β, γ) used for the Rocchio algorithm includes (1, 0.1), (1, 0.2),…, (1, 1); the value of ε for the kNNModel method includes 3, 4, 5, 6, 7. The experimental results are listed in Table 2 to Table 3.

Table 1. The basic settings of parameters for each algorithm

Dataset	**IG**	**SVM**	***k*-NN**	**Rocchio**	***K*NNModel**
Reuters-21578	0.003	C=1, *C-SVC*	k =35	β=1, γ=0.2	ε = 5
20-newsgroup	0.006	C=1, *C-SVC*	k =45	β=1, γ=0.2	ε = 5

In Table 1, heading IG is short for Information Gain. The values in this column are the thresholds for feature selection on different datasets. The *C-SVC* means C-Support Vector Classification (binary case) [13]. Other headings in Table 1 have been introduced in section 2.2.

Table 2. The comparison of the performances (F_1) on Reuters-21578 subset

Category	**SVM**	***k*-NN**	**Rocchio**	***k*NNModel**
Acq	90.74	78.81	84.03	86.08
Corn	94.74	87.08	87.74	91.85
Crude	87.30	84.12	84.51	84.72
Earn	95.14	88.02	90.39	89.45
Interest	92.78	79.67	80.84	83.26
Ship	88.02	84.02	86.22	86.73
Trade	91.81	80.69	81.64	80.00
macroaveraged F_1	91.50	83.20	85.05	86.01

Table 3. The comparison of the performances (F_1) on 20-newsgroup subset

Category	**SVM**	***k*-NN**	**Rocchio**	***k*NNModel**
alt.atheism	94.85	88.67	83.85	91.38
comp.graphics	70.26	65.47	70.00	68.06
comp.os.ms-windows.misc	72.99	65.27	66.67	67.40
comp.sys.ibm.pc.hardware	60.59	55.78	55.89	58.70
comp.sys.mac.hardware	63.87	56.92	57.91	61.04

comp.windows.x	82.74	80.00	80.21	80.10
misc.forsale	75.06	67.92	72.68	73.13
rec.autos	78.67	76.17	75.81	77.75
rec.motorcycles	89.74	88.50	88.42	89.41
rec.sport.baseball	88.18	85.86	88.61	90.00
rec.sport.hockey	91.73	90.12	93.27	92.73
sci.crypt	94.33	89.16	89.43	88.61
sci.electronics	73.15	70.17	69.09	73.60
sci.med	93.91	88.40	88.22	90.54
sci.space	89.80	83.45	86.76	86.83
soc.religion.christian	86.70	83.17	81.00	83.33
talk.politics.guns	93.64	89.05	85.51	88.94
talk.politics.mideast	96.12	91.39	93.09	91.54
talk.politics.misc	87.38	85.11	75.37	83.29
talk.religion.misc	88.38	80.83	74.18	79.47
macroaveraged F_1	83.60	79.07	78.80	80.79

Table 4. The comparison of the efficiencies of 4 algorithms

Dataset	NumOfDoc	SVM	*k*-NN	Rocchio	*k*NNModel
Reuters	1400	26.7	66.7	19.2	26.7
20-newsgroup	4000	322.6	1136.2	206.9	213.0
Total	5400	349.3	1202.9	226.1	239.7

In Table 4, heading NumOfDoc is the number of the documents for tests. The values in other columns represent times in seconds spent on testing by different algorithms. We have done tests on two subsets. One subset contains 1400 documents randomly chosen from Reuters collection, and the other one contains 4000 documents randomly selected from 20-newsgroup collection.

From the experimental results, it is clear that the *macroaveraged* F_1 of our proposed *k*NNModel method outperforms the *k*-NN algorithm and the Rocchio classifier on both datasets in ten-fold cross validation, and is comparable to the SVM algorithm. But the *k*NNModel method is more efficient than *k*-NN and SVM by keeping only a few representatives for classification, and is comparable to the Rocchio algorithm. So, the *k*NNModel is a possible replacement for *k*-NN and Rocchio classifiers in many applications such as dynamic web mining for large repositories where they are deemed not suitable .

5 Conclusions

In this paper, we present an investigation of two widely used approaches for text categorization under the framework of similarity-based learning -- the *k*-NN algorithm and the Rocchio classifier. We analyze both algorithms and identify some shortcomings of both. Based on the analysis, we developed a new approach called the *k*NN model-based algorithm, which combines the strengths of *k*-NN and the Rocchio classifier. We implemented our *k*NN model-based algorithm, along with a standard *k*-NN algorithm, Rocchio algorithm, and SVM algorithm in our TC prototype system. Extensive experiments were conducted on two common document corpora, namely the 20-newsgroup collection and the Reuters-21578 collection. All experimental results show that our proposed *k*NN model-based approach outperforms the *k*-NN and Rocchio classifier and is comparable to SVM. Further research is required into how to improve the classification accuracy of marginal data that falls outside the regions of representatives.

Acknowledgements

This work was partly supported by the European Commission project ICONS, project no. IST-2001-32429.

References

1. Lam, W. and Ho, C.Y. (1998). Using a Generalized Instance Set for Automatic Text Categorization. SIGIR'98, pages 81-89.
2. Lewis, D. D. (1998). Naïve (Bayes) at forty: The independent assumption in information retrieval. In Proceedings of ECML-98, 10th European Conference on Machine Learning, pages 4 -15.
3. Cohen, W. W. and Singer, Y. (1999). Context–Sensitive Learning Methods for Text Categorization. ACM Trans. Inform. Syst. 17, 2, pages 141-173.
4. Li, H. and Yamanishi, K. (1999). Text Classification Using ESC-based Stochastic Decision Lists. In Proceedings of CIKM-99, 8th ACM International Conference on Information and Knowledge Management, pages 122-130.
5. Yang, Y. and Liu, X. (1999). A Re-examination of Text Categorization Methods. In Proceedings of SIGIR-99, 22nd ACM International Conference on Research and Development in Information Retrieval, pages 42-49.
6. Ruiz, M. E. and Srinivasan, P. (1999). Hierarchical Neural Networks for Text Categorization. In Proceedings of SIGIR-99, 22nd ACM International Information Retrieval, pages 281-282.
7. Mitchell, T.M. (1996). Machine Learning. McGraw Hill, New York, NY.
8. Joachims, T. (1998). Text Categorization with Support Vector Machines: Learning with Many Relevant Features, In Proceedings of 10th European Conference on Machine Learning, Chemnitz, Germany, pages 137-142.

9. Joachims, T. (2001). A Statistical Learning Model of Text Classification for Support Vector Machines. In Proceedings of SIGIR-01, 24th ACM International Conference on Research and Development in Information Retrieval, pages 128-136.
10. Rocchio, Jr. J. J. (1971). Relevance Feedback in Information Retrieval. The SMART Retrieval System: Experiments in Automatic Document Processing, editor: Gerard Salton, Prentice-Hall, Inc., Englewood Cliffs, News Jersey, 1971.
11. Joachims, T. (1997). A Probabilistic Analysis of the Rocchio Algorithm with TFIDF for Test Categorization. In Proceedings of ICML-97, 14th International Conference on Machine Learning, pages 143-151.
12. Sebastiani, F. (2002). Machine Learning in Automated Text Categorization. ACM Computing Surveys, Vol.34, No.1, March 2002, pages 1-47.
13. Cortes, C. and V. Vapnik (1995). Support-Vector Network. Machine Learning 20, pages 273-297.