7-2014

CenKNN: A scalable and effective text classifier

Guansong PANG
Singapore Management University, gspang@smu.edu.sg

Huidong JIN

Shengyi JIANG

Data Min Knowl Disc (2015) 29:593–625
DOI 10.1007/s10618-014-0358-x

CenKNN: a scalable and effective text classifier

Guansong Pang · Huidong Jin · Shengyi Jiang

Received: 3 March 2013 / Accepted: 27 May 2014 / Published online: 3 July 2014

Abstract A big challenge in text classification is to perform classification on a large-scale and high-dimensional text corpus in the presence of imbalanced class distributions and a large number of irrelevant or noisy term features. A number of techniques have been proposed to handle this challenge with varying degrees of success. In this paper, by combining the strengths of two widely used text classification techniques, K-Nearest-Neighbor (*KNN*) and centroid based (*Centroid*) classifiers, we propose a scalable and effective flat classifier, called *CenKNN*, to cope with this challenge. *CenKNN* projects high-dimensional (often hundreds of thousands) documents into a low-dimensional (normally a few dozen) space spanned by class centroids, and then uses the k-d tree structure to find K nearest neighbors efficiently. Due to the strong representation power of class centroids, *CenKNN* overcomes two issues related to existing *KNN* text classifiers, i.e., sensitivity to imbalanced class distributions and irrelevant or noisy term features. By working on projected low-dimensional data, *CenKNN* substantially reduces the expensive computation time in *KNN*. *CenKNN* also works

Responsible editor: Hendrik Blockeel, Kristian Kersting, Siegfried Nijssen, Filip Zelezny.

This work was mainly done when Guansong Pang was with Guangdong University of Foreign Studies, China.

G. Pang (✉)
Clayton School of Information Technology, Monash University, Melbourne, VIC 3800, Australia
e-mail: Guansong.Pang@monash.edu

H. Jin
CSIRO Computational Informatics, GPO Box 664, Canberra, ACT 2601, Australia
e-mail: Warren.Jin@csiro.au

S. Jiang
School of Informatics, Guangdong University of Foreign Studies, Guangzhou 510006, China
e-mail: jiangshengyi@163.com

better than *Centroid* since it uses all the class centroids to define similarity and works well on complex data, i.e., non-linearly separable data and data with local patterns within each class. A series of experiments on both English and Chinese, benchmark and synthetic corpora demonstrates that although *CenKNN* works on a significantly lower-dimensional space, it performs substantially better than *KNN* and its five variants, and existing scalable classifiers, including *Centroid* and *Rocchio*. *CenKNN* is also empirically preferable to another well-known classifier, support vector machines, on highly imbalanced corpora with a small number of classes.

Keywords Text classification · *KNN* · Centroid · Dimension reduction · Imbalanced classification

1 Introduction

Text Classification is the task of assigning predefined classes to text documents and has become an effective tool to organize the surging volume of text data in electronic format such as emails, web pages, news reports, scientific articles, blogs and microblogs. One of the big challenges in numerous real-world text classification applications such as spam detection, web page organization, news filtering and organization, news event tracking, social media user classification, blog classification and sentiment classification (Sebastiani 2002; Aggarwal and Zhai 2012; Yang et al. 2000; Pang et al. 2013), is that the large size and high dimensionality of a text dataset renders various classification techniques less effective. There is another common challenge to perform classification on imbalanced text corpora, where at least one class is underrepresented in the training data (He and Garcia 2009; Forman 2003). It is expected that the continuous exponential growth of online text documents will aggravate this challenge (Forman 2003). Also, in many text datasets, a large number of term features may be irrelevant or noisy, which restrain classifiers' performance in terms of both effectiveness and efficiency (Forman 2003; Han et al. 2001; Yang and Pedersen 1997). Therefore, there is a growing need to develop effective text classification techniques to deal with the problem of classification on large-scale and high-dimensional text data in the presence of imbalanced class distributions and irrelevant or noisy features.

The K-Nearest-Neighbor (*KNN*) text classification algorithm is a popular instance-based learning method, which uses all the training documents to search K nearest neighbors for test documents to make predictions (Sebastiani 2002; Aggarwal and Zhai 2012; Wu and Kumar 2008). A test document is normally assigned to the most abundant class of its K nearest neighbors. *KNN* is a straightforward yet remarkable classifier, which has been shown as one of the most effective methods for text classification (Joachims 1998; Lam and Han 2003; Yang and Liu 1999). However, it is computationally expensive due to its lazy learning pattern, especially when classifying large-scale and high-dimensional document collections. Moreover, *KNN* text classification is sensitive to irrelevant or noisy term features. The success of the *KNN* classifier is dependent on the availability of effective similarity measures, e.g., the *cosine* measure, which however will become less effective in the presence of irrelevant or noisy features because these features have the same influence as informative features in the

similarity measures (Han et al. 2001; Cunningham and Delany 2007). At the same time, *KNN* also suffers from the class imbalance problem due to its majority voting classification scheme (Batista et al. 2004; Sun et al. 2009; Mani and Zhang 2003; Tan 2005). Specifically, in the presence of imbalanced class distributions, training documents from small classes (i.e., minority classes) occur sparsely, in contrast to the dense distribution of documents from large classes (i.e., majority classes). Given a test document, its nearest neighbors bear higher probabilities of training documents from majority classes. As a result, *KNN* is likely to be biased towards majority classes. Another issue in *KNN* text classification is how to decide an appropriate value for K to enable *KNN* to obtain consistently favorable performance on corpora of different characteristics such as different class distributions (e.g., balanced and imbalanced corpora).

Fast search trees like k-d trees (Bentley 1975) are widely used neighborhood search structures to speed up *KNN* classification, but they perform poorly on documents of tens of thousands of different term features. A popular speedup method for*KNN* text classification is to use an inverted index but with the premise that documents are of short length and a large stop word list is used (Wang et al. 2011; Manning et al. 2008). The combination of *KNN* with latent semantic indexing (*LSI*) dimension reduction or its variants is another commonly used method (Sun et al. 2004; Liu et al. 2004; Yang and King 2009). However, these dimension reduction methods normally require parameter tuning and expensive computation.

Numerous methods have been proposed to improve *KNN* text classification accuracy[1], such as cluster-based methods, feature-based weighting methods and instance-based weighting methods (Han et al. 2001; Lam and Han 2003; Tan 2005; Guo et al. 2006; Tan 2006; Pang and Jiang 2013; Jiang et al. 2012). In general, these methods improve *KNN*'s accuracy to some degree while still suffering from issues like expensive computation time or sensitivity to imbalanced class distributions and redundant features.

In this paper, by combining *KNN* with efficient centroid-based text classification (denoted by *Centroid*) techniques, we propose a scalable and effective flat classifier[2], called *CenKNN*, to scale up *KNN* text classification as well as to improve its effectiveness on high-dimensional and large-scale corpora with imbalanced class distributions and irrelevant or noisy term features. The basic idea of *CenKNN* is to use an effective and efficient class-centroid-based dimension reduction method to substantially reduce dimensionality of documents, and then employ the k-d tree structure to conduct a rapid K nearest neighbors search for *KNN* classification.

Motivated by the remarkable performance of *Centroid* (Han and Karypis 2000), for *CenKNN*, we propose a class-centroid-based dimension reduction method, called *CentroidDR*. The underlying assumption of *CentroidDR* is that documents (with the exception of outlying and noisy documents) are normally closer to their inherent class

[1] In this paper, classification accuracy is referred to the performance in micro-averaging F_1 (denoted by $microF_1$) and macro-averaging F_1 (denoted by $macroF_1$) values, rather than the ratio between the number of correctly classified test documents and the total number of test documents.

[2] By "flat" classifiers, we focus on text classification tasks without considering a class hierarchy. Flat classifiers are the building blocks for successful hierarchical text classifiers.

centroid rather than other centroids. This rationale has been demonstrated by the effectiveness of *Centroid* on a wide range of both balanced and imbalanced corpora (Han and Karypis 2000). The success of centroid-based classification techniques is mainly due to the strong representation power of class centroids, which have good robustness to imbalanced class distributions and irrelevant or noisy term features (Lam and Han 2003; Guo et al. 2006; Han and Karypis 2000). Another reason for *Centroid*'s impressive performance is that the *cosine* similarity measure function accounts for term dependencies between class centroids and documents (Han and Karypis 2000). However, unlike *Centroid* (which finds the most similar class centroid for test documents in order to perform classification), *CentroidDR* aims to use class centroids and the *cosine* similarity measure to reduce the dimensionality of documents. Basically, *CentroidDR* projects high-dimensional documents into a low-dimensional space spanned by class centroids. On this class-centroid-based space, intuitively, *Centroid* is essentially *CentroidDR* plus a simple linear classifier. In *CenKNN*, the simple linear classifier is replaced by*KNN*, a sophisticated non-linear classifier, to find a better classification boundary in the class-centroid-based space.

By combining *CentroidDR* with *KNN*, *CenKNN* essentially utilizes all the class centroids and the *cosine* similarity measure to define similarities between projected documents and search for neighborhoods to make predictions in the low-dimensional class-centroid-based space. Therefore, *CenKNN* benefits from both the strengths of *Centroid* and *KNN*. A series of experiments on both English and Chinese, benchmark and synthetic corpora demonstrates its promising performance. Compared to *KNN* and its variants based on feature selection, dimension reduction or clustering techniques, *CenKNN* is scalable, and can run one to two orders of magnitude faster, and can classify documents with markedly greater accuracy. *CenKNN* can generate substantially better results than existing scalable classifiers, e.g., *Centroid* and *Rocchio*. *CenKNN* is also empirically preferable to another well-known classifier, Support Vector Machines (*SVM*), on highly imbalanced corpora with a small number of classes.

Our contribution in this paper is to propose a scalable and effective flat classifier for large-scale and high-dimensional text data with imbalanced class distributions and irrelevant or noisy term features. This is illustrated by a series of experimental results on a variety of corpora.

The rest of this paper is organized as follows. Related work is discussed in Sect. 2 and we introduce our proposed method in Sect. 3. Section 4 presents a series of experimental results and comparisons. The paper is concluded in Sect. 5.

2 Related work

2.1 Existing text classification algorithms

A wide range of text classification algorithms and their variants has been developed over the years, e.g., *KNN* (Han et al. 2001; Lam and Han 2003; Wang et al. 2011; Pang and Jiang 2013; Jiang et al. 2012; Yang 1994), *Centroid* (Han and Karypis 2000; Tan and Cheng 2007; Guan et al. 2009), *Rocchio* (Lam and Han 2003; Pang and Jiang 2013; Joachims 1996) and *SVM* (Joachims 1998; Joachims 2001; Kim et

al. 2005; Wan et al. 2012). Below we briefly introduce these algorithms and discuss their advantages and drawbacks.

In *KNN* classification, to determine the class of a test instance, there are a range of methods on how to use its K nearest neighbors. The most straightforward method is to assign the majority class among the neighbors to the instance, while it is often more intuitive to assign higher weight to the nearer neighbors in the class assignment, such as distance weighted voting methods (Cunningham and Delany 2007). The process of distance-weighted *KNN* text classification can be described briefly as follows: given a test document $\mathbf{d_t}$, find the nearest neighbors for $\mathbf{d_t}$ among the training document set D, and score class candidates for $\mathbf{d_t}$ based on the classes of the neighbors, as detailed in formula (1). *KNN* then assigns the class with the highest score to $\mathbf{d_t}$.

$$y_t = \arg\max_{C_j} \sum_{\mathbf{d_i} \in KNN^{\mathbf{d_t}}} \text{sim}(\mathbf{d_t}, \mathbf{d_i}) \text{I}(\mathbf{d_i}, C_j) \tag{1}$$

where $KNN^{\mathbf{d_t}}$ denotes the set of the K nearest neighbors of $\mathbf{d_t}$, $\text{sim}(\mathbf{d_t}, \mathbf{d_i})$ denotes the similarity between $\mathbf{d_t}$and $\mathbf{d_i}$, and $\text{I}(\mathbf{d_i}, C_j)$ is the indicator function, which is 1 when $\mathbf{d_i}$ belongs to the class C_j and otherwise 0. *KNN* has been reported as one of the most effective algorithms for text classification (Joachims 1998; Lam and Han 2003; Yang and Liu 1999). But it suffers from several drawbacks such as expensive computation time on test documents, of $O(N \cdot Tr_{avg} \cdot Te_{voc})$(where N is the number of training documents, $Tr_{avg}$is the average number of unique terms of training documents and Te_{voc} denotes the number of unique terms in a given test document) when using the linear search method to search for the neighbors (Sebastiani 2002; Cunningham and Delany 2007; Manning et al. 2008), sensitivity to skewed class distributions (Batista et al. 2004; Sun et al. 2009; Tan 2005, 2006) and irrelevant or noisy features (Han et al. 2001; Cunningham and Delany 2007), and parameter (i.e., the K value) tuning (Sebastiani 2002; Guo et al. 2006).

Centroid is a class centroid based linear classifier. In *Centroid*, each class is represented by its centroid, and a test document is assigned to the class label of its closest centroid. Its classification boundary is the linear boundary between class centroids. Despite its simplicity, it substantially outperforms many other popular classifiers, such as naïve Bayes, *KNN* and C4.5, on a wide range of both balanced and imbalanced corpora (Han and Karypis 2000). *Centroid* combines prevalent term features within each class centroid, so that each class centroid is distinctive and separable from others. Although the class centroids of majority classes tend to contain some term features of minority classes, the average weights of these features in majority classes are much smaller than those in minority classes. So the centroid-based representation model is less likely to be biased towards majority classes. In terms of efficiency,*Centroid* can scale up well with data size as it has computational complexity of $O\left(l \cdot N \cdot Tr_{avg}\right) + O(l \cdot Cen_{avg} \cdot Te_{voc})$ (where l and Cen_{avg} are the number of classes and the average number of unique terms in a class centroid respectively). However, *Centroid* can be plagued by modeling misfit problems when documents are not linearly separable by the boundaries between class centroids (Tan and Cheng 2007).

Rocchio functions similarly to *Centroid* except that it uses prototype vectors to classify documents. The prototype vectors are generalized from class centroids, calculated as formula (2).

$$\mathbf{pv_j} = \alpha \frac{1}{|C_j|} \sum_{\mathbf{d_i} \in C_j} \mathbf{d_i} - \beta \frac{1}{|D - C_j|} \sum_{\mathbf{d_m} \in D - C_j} \mathbf{d_m} \tag{2}$$

where $\mathbf{pv_j}$denotes the prototype vector of the class C_j. Parameters α and β adjust the relative importance of positive and negative documents. *Rocchio* is equivalent to *Centroid* with $\alpha = 1$ and $\beta = 0$. *Rocchio* has similar merits to *Centroid*, though *Rocchio* has slightly higher computation time than *Centroid* due to *Rocchio*'s inclusion of prototype vector generation. Moreover, centroid-based representation models, such as class centroids or generalized class centroids (i.e., prototype vectors), are the summarizations of training documents, which have stronger representation power than single documents, because this kind of model can distill out certain prevalent or distinctive term features, and is robust to noisy or irrelevant features (Lam and Han 2003; Guo et al. 2006). Both *Rocchio* and *Centroid* have shown impressive performance on a range of corpora (Pang and Jiang 2013; Han and Karypis 2000), which also can be observed from our experimental results. Apart from the hypothesis of linear separability, another drawback of *Rocchio* and *Centroid* is that they are not adapted to handle classes with underlying sub-clusters (or say local patterns Vilalta et al. 2003) rather than one single cluster, i.e., they fail to fully represent a class within which there are fine-grained sub-classes (Pang and Jiang 2013; Han and Karypis 2000).

The main principle of *SVM* is to determine support vectors that maximize margins of separation between classes in a hyperplane. *SVM* performs well on linearly separable space, and it can use kernel methods such as polynomial and *RBF* kernels, to adapt to non-linearly separable data space. It has been reported in (Joachims 1998; Yang and Liu 1999; Kim et al. 2005; Lan et al. 2009) that, compared to polynomial and *RBF* kernels, the linear kernel can enable *SVM* to achieve better or very comparable text classification accuracy and to perform more efficiently. This is due to the fact that text documents distribute throughout a very high dimensional space in most text classification problems and they are often linearly separable there (Joachims 1998). However, *SVM* performs less effectively on skewed data, since its optimization goal is biased against minority classes in order to minimize total error (He and Garcia 2009; Forman 2003; Tang and Liu 2005). We will compare our proposed methods with these four classifiers in Sects. 4.3–4.6 extensively.

Other widely used algorithms include naïve Bayes and neural networks (Sebastiani 2002). Naïve Bayes is a simple and easy-to-implement text classifier. Despite its simplicity, it can achieve impressive classification performance in a lot of applications. However, it has poor classification performance when handling skewed corpora (Yang and Liu 1999; Jiang et al. 2012). Neural networks are also known as one of the most effective classification algorithms, but they require very expensive computation in high-dimensional text documents. Due to these methods' high sensitivity to the class imbalance problem or high computation time, we will not compare them with our techniques directly in this paper.

2.2 Previous work on improving KNN

A range of previous work has aimed at improving the *KNN* text classification algorithm. In order to enhance *KNN*'s efficiency, an inverted index is often used to preprocess training documents for *KNN* classification over test documents (Wang et al. 2011; Yang 1994), which hopefully can reduce the classification time complexity of *KNN* from $O(N \cdot Tr_{avg} \cdot Te_{voc})$ to $O(n \cdot Te_{voc} + p \cdot Tr_{avg} \cdot Te_{voc})$ (n is the size of vocabulary and p is the number of training documents that contain terms appearing in the test document) (Manning et al. 2008). But the efficiency of inverted index based *KNN* classification is closely related to classification tasks. The inverted index will work well in the context where a large number of stop words is used and the terms of the test document do not overlap with a large number of training documents (Manning et al. 2008). It should be noted that some informative term features would be removed if a large stop word list is employed.

Data compression methods like text feature selection or dimension reduction are also commonly integrated into *KNN* text classification to save classification time (Yang and Pedersen 1997; Sun et al. 2004). Many text feature selection methods have been proposed to reduce the number of term features (Sebastiani 2002; Forman 2003; Yang and Pedersen 1997), such as information gain, document frequency (DF) and mutual information. However, we might lose critical information for classification when preserving only a small proportion of term features (Han et al. 2001; Joachims 1998), resulting in the degradation of *KNN* classification accuracy. Dimension reduction methods, such as Latent Semantic Indexing (*LSI*), supervised *LSI*, local *LSI* and topic modeling techniques (Sun et al. 2004; Liu et al. 2004; Du et al. 2010, 2012), are commonly used in text classification, but they normally require parameter tuning and expensive computation. Random projection (*RP*) techniques are efficient tools for dimension reduction (Bingham and Mannila 2001; Achlioptas 2003; Lin and Gunopulos 2003). They are motivated by the Johnson–Lindenstrauss lemma that a set of samples in a high-dimensional space can be projected onto a subspace of small size of dimensionality such that distances between the samples are nearly preserved (Achlioptas 2003). Although *RP* is of linear time complexity and can perform more efficiently than *LSI* and its variants, it does not preserve as much information as *LSI*-based methods (Lin and Gunopulos 2003).

Fast search trees, such as k-d trees (Bentley 1975), B^+ trees (Jagadish et al. 2005), SR-trees (Katayama and Satoh 1997) and lower bound trees (Chen et al. 2007), can substantially speed up *KNN* classification without any loss in classification accuracy. However, fast search tree methods such as k-d trees and B^+ trees, cannot perform well on text data, since the neighborhood searching in text classification normally operates on a large number of documents distributed throughout a very high dimensional space; or they need to spend a lot of time on clustering training documents to construct the search tree, e.g., lower bound trees.

In terms of classification accuracy, various cluster-based methods have been proposed for *KNN* to deal with its sensitivity to noise (Lam and Han 2003; Guo

et al. 2006; Pang and Jiang 2013; Jiang et al. 2012). For example, *INNTC* is an improved cluster-based *KNN* text classification method (Jiang et al. 2012). It uses a constrained incremental clustering algorithm with a clustering radius to group training documents into clusters. *INNTC* then classifies test documents by searching the nearest neighbors upon clusters instead of original training documents. Similar work can be found in (Lam and Han 2003; Guo et al. 2006; Pang and Jiang 2013). The difference is that they utilize *Rocchio* to produce generalized instances or cluster centroids to replace the original training documents. As the clusters or generalized instances are compact summarizations of training documents, they can perform more efficiently and effectively than *KNN*. However, the process of clustering or generalized instance generation is time-consuming, and they still need to identify the neighbors on the high-dimensional term feature space.

Some works aim to examine *KNN*'s sensitivity to imbalanced class distributions (Batista et al. 2004; Mani and Zhang 2003; Tan 2005, 2006) and irrelevant or noisy features (Han et al. 2001; Wettschereck et al. 1997). Instance-based or feature-based weighting methods have been explored to improve the sensitivity of *KNN* text classification, but they do not try to reduce *KNN*'s high computation time. *KNN* can also be improved to conduct hierarchical text classification (Wang et al. 2011) and multi-label text classification (Zhang and Zhou 2007), but we focus on non-hierarchical single-label text classification.

3 Proposed scalable and effective text classifier

We use the Vector Space Model (*VSM*) (Salton et al. 1975), a widely used model in text classification tasks, to represent documents. In *VSM*, given a set of N training documents $D = \{(\mathbf{d}_1, y_1), (\mathbf{d}_2, y_2), \cdots, (\mathbf{d_N}, y_N)\}$and a set of predefined classes $y_i \in \{C_1, C_2, \cdots, C_l\}$, each document is represented by a term-based vector $\mathbf{d_i} = \{t_1, t_2, \cdots, t_n\} \in \mathbb{R}^n$ and weighted by $TF \times IDF$ (*TF* and *IDF* are short for Term Frequency and Inverse DF respectively). n is the size of the vocabulary derived from D. The vocabulary consists of different terms in D.

3.1 Proposed dimension reduction method: CentroidDR

CentroidDR first computes the centroids of all classes, and then maps documents into the class-centroid-based space via the *cosine* similarity measure function. A class centroid is the mean representation vector of documents within the class, as detailed in formula (3), which can be generated very efficiently.

$$\mathbf{centroid_j} = \frac{1}{|C_j|} \sum\nolimits_{\mathbf{d_i} \in C_j} \mathbf{d_i} \tag{3}$$

So $\mathbf{centroid_j}$denotes the centroid of the class C_j, $j = 1, 2, \cdots, l$. We project documents onto the new space via formula (4).

$$x_i^{(j)} = \text{sim}(\mathbf{d_i}, \mathbf{centroid_j}) = \frac{\langle \mathbf{d_i}, \mathbf{centroid_j} \rangle}{\|\mathbf{d_i}\| \times \|\mathbf{centroid_j}\|} = \frac{\sum_{s=1}^{n} d_i^{(s)} \times centroid_j^{(s)}}{\sqrt{\sum_{s=1}^{n} \left(d_i^{(s)}\right)^2} \times \sqrt{\sum_{s=1}^{n} \left(centroid_j^{(s)}\right)^2}} \tag{4}$$

where $s = 1, 2, \cdots, n$; the superscripts s and j denote the indices of the dimensions for original documents $\mathbf{d_i}$ and projected data $\mathbf{x_i}$ respectively. Details of *CentroidDR* are given as follows:

Algorithm 1 (*CentroidDR*)

Input: Given a set of training documents D.

Output: The projected data $D^* = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \cdots, (\mathbf{x_N}, y_N)\}$, where $\mathbf{x_i} \in \mathbb{R}^l$, $y_i \in \{C_1, C_2, \cdots, C_l\}$, for $i = 1, 2, \cdots, N$.

(1) Compute the centroid of each class via formula (3).
(2) For each document, compute the similarities between the document and all of the centroids, and assign these similarity values to the dimension values of the projected data via formula (4).
(3) Obtain the projected data $D^* = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \cdots, (\mathbf{x_N}, y_N)\}$. Each $\mathbf{x_i}$ is projected from $\mathbf{d_i}$. The coordinates of the original documents in the projected space are the similarities obtained in step (2), and their class labels are retained from their original documents.

3.2 Proposed text classifier: CenKNN

Our proposed *CenKNN* is described in Algorithm 2. We first use *CentroidDR* to project high-dimensional documents onto a low dimensional class-centroid-based space, and then employ a k-d tree to store the projected data and search the K nearest neighbors to classify documents.

The k-d tree search method is a binary search tree for associate searching (Bentley 1975). A k-d tree is a data structure used to store data in a k-dimensional space. Tree's leaf nodes store instances, and its inner nodes correspond to the axis-oriented splits of the space. For constructing a k-d tree, the construction algorithm simply chooses one coordinate of the k dimensions to iteratively divide the remaining instances. Given a target instance, to search for its nearest neighbors, the k-d tree search method first finds a leaf node that contains the target instance, and then recursively rolls back from that leaf node and scans nearby leaf nodes. When the distance from the next leaf node cannot improve, the method stops the search. The k-d tree search method is one of the most efficient methods for searching for the nearest neighbor. Therefore, it is a commonly used search method to replace the linear search method to scale up *KNN* classification. However, it only performs well on data of small dimensionality as its computational complexity increases exponentially with the dimensionality of the dataset, and is therefore not directly applicable for high dimensional text data.

On the class-centroid-based space, the number of dimensions is equivalent to the number of classes contained in a corpus, so it is typically very small in most text

classification tasks, compared to the size of vocabulary in a typical text corpus (e.g., two classes in spam detection, legitimate and spam mails). Therefore, we can construct the k-d tree upon the projected data and search the nearest neighbors very efficiently. This enables *CenKNN* to become a scalable text classifier.

It should be noted that the k-d tree search method is only efficient for *Minkowski* metrics (Bentley 1975). One widely used distance measure among *Minkowski* metrics for the k-d tree search is the *Euclidean* distance. However, the *Euclidean* distance has less effective performance than the *cosine* similarity measure in *KNN* text classification (Wu and Kumar 2008). To tackle this problem, *CenKNN* uses the k-d tree search method via *Euclidean* distance on the normalized space of the projected data. By doing this, *CenKNN* can work on the k-d tree to locate the neighbors in terms of the cosine similarity. This is supported by the following lemma.

Lemma 1 *The K nearest neighbors returned from a k-d tree on a normalized space are identical with the K nearest neighbors found from the original data space via cosine similarity measure.*

Proof In the original data space, the cosine similarity measure between two document vectors $\mathbf{d_i}$ and $\mathbf{d_j}$ can be denoted as $\text{cosine}\left(\mathbf{d_i}, \mathbf{d_j}\right) = \frac{\langle \mathbf{d_i}, \mathbf{d_j} \rangle}{\|\mathbf{d_i}\| \times \|\mathbf{d_j}\|} = \langle \frac{\mathbf{d_i}}{\|\mathbf{d_i}\|}, \frac{\mathbf{d_j}}{\|\mathbf{d_j}\|} \rangle = \langle \hat{\mathbf{d}}_\mathbf{i}, \hat{\mathbf{d}}_\mathbf{j} \rangle$, where $\hat{\mathbf{d}}_\mathbf{i}$ is the normalized vector of $\mathbf{d_i}$. The Euclidean distance of two normalized vectors is $\|\hat{\mathbf{d}}_\mathbf{i} - \hat{\mathbf{d}}_\mathbf{j}\|^2 = \|\hat{\mathbf{d}}_\mathbf{i}\|^2 + \|\hat{\mathbf{d}}_\mathbf{j}\|^2 - 2 \times \langle \hat{\mathbf{d}}_\mathbf{i}, \hat{\mathbf{d}}_\mathbf{j} \rangle = 2 - 2 \times \langle \hat{\mathbf{d}}_\mathbf{i}, \hat{\mathbf{d}}_\mathbf{j} \rangle$. Therefore, the K nearest neighbors with the smallest Euclidean distance w.r.t. $\hat{\mathbf{d}}_\mathbf{i}$ are identical with KNN with K nearest neighbors with the highest cosine similarity w.r.t. $\mathbf{d_i}$.

Algorithm 2 (*CenKNN*)
Input: Given a set of training documents D and a test document $\mathbf{d_t}$.
Output: The class label of document $\mathbf{d_t}$.

(1) Project the high dimensional documents D onto a class-centroid-based space via *CentroidDR*, and obtain the projected data $D^* = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \cdots, (\mathbf{x_N}, y_N)\}$.
(2) Normalize the projected document vectors in D^*.
(3) Build a k-d tree on the normalized data.
(4) For the test document $\mathbf{d_t} \in \mathbb{R}^n$, project it onto the class-centroid-based space, and normalize it, we then obtain $\hat{\mathbf{x}}_\mathbf{t} \in \mathbb{R}^l$.
(5) Search for the K nearest neighbors of $\hat{\mathbf{x}}_\mathbf{t}$ over the k-d tree.
(6) Classify $\hat{\mathbf{x}}_\mathbf{t}$ based on the *KNN* decision rule, as detailed in formula (5).

$$y_t = \arg\max_{C_j} \sum_{\hat{\mathbf{x}}_\mathbf{i} \in KNN^{\hat{\mathbf{x}}_\mathbf{t}}} \left(1 - \text{dist}\left(\hat{\mathbf{x}}_\mathbf{t}, \hat{\mathbf{x}}_\mathbf{i}\right)\right) \text{I}(\hat{\mathbf{x}}_\mathbf{i}, C_j) \tag{5}$$

where $KNN^{\hat{\mathbf{x}}_\mathbf{t}}$ denotes the set of K nearest neighbors of $\hat{\mathbf{x}}_\mathbf{t}$, $\text{dist}\left(\hat{\mathbf{x}}_\mathbf{t}, \hat{\mathbf{x}}_\mathbf{i}\right)$ denotes the *Euclidean* distance between $\hat{\mathbf{x}}_\mathbf{t}$ and $\hat{\mathbf{x}}_\mathbf{i}$, and $\text{I}(\hat{\mathbf{x}}_\mathbf{i}, C_j)$ is the indicator function, which is 1 when $\hat{\mathbf{x}}_\mathbf{i}$ belongs to C_j otherwise 0.

We illustrate the working scheme of *CenKNN* on two classes of documents from the Fudan text classification corpus (see Sect. 4.2 for more details about this corpus),

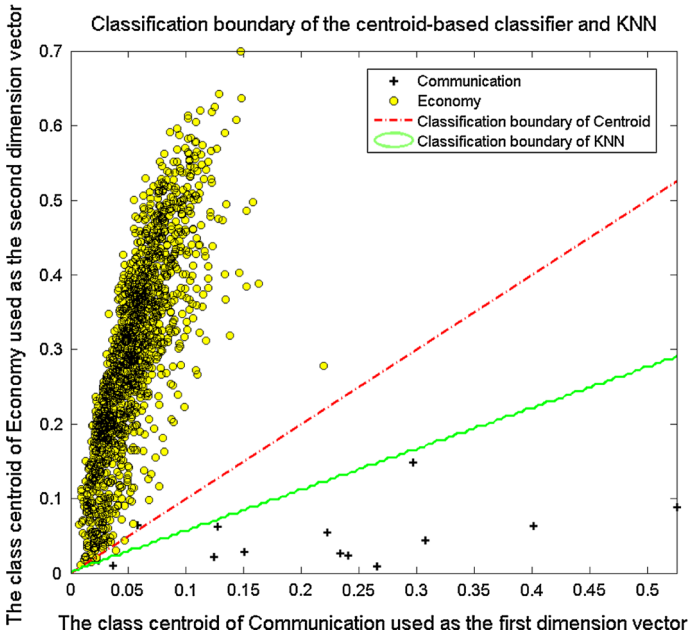

Fig. 1 Visualization of classification results on projected test data derived from classes *Communication* and *Economy*

Communication and *Economy*, which contain 1,613 training documents (of which 1,600 documents belong to *Economy*) and 64,439 different terms (a large number of terms are irrelevant or noisy features) in total. These documents therefore originally span a 64,439-dimensional space. Recall that *CenKNN* consists of *CentroidDR* and *KNN* classification with a k-d tree. In *CentroidDR*, we first compute the centroids of classes *Communication* and *Economy*, denoted as $\mathbf{centroid}_1$ and $\mathbf{centroid}_2$; and then for each document $\mathbf{d_i}$, we calculate its similarities with $\mathbf{centroid}_1$ and $\mathbf{centroid}_2$; these similarity values are then assigned to $x_i^{(1)}$ and $x_i^{(2)}$ respectively, and the class label y_i of $\mathbf{x_i}$ is retained from document $\mathbf{d_i}$. In this way, we obtain a set of projected data in $\mathbb{R}^l$ space, with $l = 2$ in this case. The projected data in the *Communication* class would be distributed near $\mathbf{centroid}_1$ (as they are normally closer to their inherent class centroid than other class centroids), and most of these are separable from that of *Economy*, and vice versa.

There are 14 and 1,601 test documents in the classes *Communication* and *Economy* respectively. To show the effectiveness of *CenKNN* clearly, we plot the *Centroid* and *KNN* classification results on the projected test data in Fig. 1, where the*red dashed line* indicates the classification boundary of *Centroid*, which misclassifies 5 documents that are not linearly separable via the central linear boundary between the class centroids of *Communication* and *Economy*. In the original term feature space, *KNN* ($K = 10$) performs worse than *Centroid*, misclassifying 6 documents. We use the*green contour* to represent the *KNN* ($K = 10$) classification boundary on the projected data. The figure shows that *KNN* with *cosine* similarity measure works on the 2-D data better than *Centroid*, and only misclassifies 2 documents. Since the class distribution of this

Table 1 Time complexity comparison between *KNN*, *Centroid* and *CenKNN*

	Training stage	Classification stage
KNN	$O(N \cdot Tr_{avg})$	$O(N \cdot Tr_{avg} \cdot Te_{voc})$ or $O(n \cdot Te_{voc} + p \cdot Tr_{avg} \cdot Te_{voc})$
Centroid	$O(l \cdot N \cdot Tr_{avg})$	$O(l \cdot Cen_{avg} \cdot Te_{voc})$
CenKNN	$O(l \cdot N \cdot Cen_{avg} + l \cdot N \cdot \log N)$	$O(l \cdot Cen_{avg} \cdot Te_{voc} + K \cdot \log N)$

dataset is highly skewed, very few misclassified documents can lead to a significant difference in the classification accuracy. In particular, *CenKNN* (i.e., *KNN* working on the class-centroid-based space), with $microF_1 = 0.9988$ and $macroF_1 = 0.9627$, consistently outperforms *Centroid* ($microF_1 = 0.9969$ and $macroF_1 = 0.9208$) and *KNN* ($microF_1 = 0.9963$ and $macroF_1 = 0.8793$) in both $microF_1$ and $macroF_1$, especially in $macroF_1$. In terms of execution time, *CenKNN* takes 9 seconds on this classification task, and runs much faster than *KNN* (150 seconds), and is slightly slower than *Centroid* (5 seconds). Similar results and comparisons can be found in the experiment section.

Computation complexity analysis The CenKNN method consists of three main stages: document projection via CentroidDR, k-d tree construction and k-nearest-neighbor search over the k-d tree. In the training stage, we need to generate class centroids and project training documents onto the class-centroid-based space via CentroidDR, which has time complexity $O(l \cdot N \cdot Cen_{avg})$. The k-d tree construction is built on the projected data and has time complexity $O(l \cdot N \cdot \log N)$ (Bentley 1975). In terms of classification, CenKNN first maps a test document onto the projected space. The time complexity of this mapping is $O(l \cdot Cen_{avg} \cdot Te_{voc})$. The k-d tree is then applied to search for the neighbors, and its time complexity is $O(K \cdot \log N)$, which is expected to be constant when N is sufficiently large with respect to k (Bentley 1975; Moore and Hall 1990).

A time complexity comparison between *KNN*, *Centroid* and *CenKNN* is shown in Table 1. In the training stage, *KNN* only needs to scan the document collection once to index training documents (inverted index or general index), and has $O(N \cdot Tr_{avg})$. *Centroid* generates l class centroids and has time complexity of $O(l \cdot N \cdot Tr_{avg})$. Since $Cen_{avg}$is often far larger than log$N$, the time complexity of *CenKNN* is $O(l \cdot N \cdot Cen_{avg})$, which is slightly larger than *Centroid* and *KNN*. In terms of classification stage, the complexity of *KNN* is $O(N \cdot Tr_{avg} \cdot Te_{voc})$ when using the linear search method to search for neighborhoods. When an inverted index is employed, it will become $O(n \cdot Te_{voc} + p \cdot Tr_{avg} \cdot Te_{voc})$ (where the former part refers to identifying training documents that have term overlap with the test document; the latter part denotes the similarity computation complexity). *Centroid* needs to compute the similarities between the test document and all the class centroids to perform classification, and it has time complexity $O(l \cdot Cen_{avg} \cdot Te_{voc})$. In *CenKNN*, likewise, $Cen_{avg}$is normally far larger than log$N$, and so *CenKNN*'s classification time is dominated by the test document projection, and has $O(l \cdot Cen_{avg} \cdot Te_{voc})$.

From the above complexity analysis, it is clear that the main difference between *KNN*, *Centroid* and *CenKNN* is decided by the classification stage. In general, *KNN*

classification complexity is linear to the size of training documents. It would be independent of the size of training documents and linear to the size of vocabulary, provided that an inverted index is used and the test document has no term overlap with a large number of training documents (i.e., p is very small compared to N) (Manning et al. 2008). In this case, *KNN*'s classification time can be reduced by a factor of 10 or more (Manning et al. 2008). The classification time complexity of *CenKNN* and*Centroid* is dependent on the average size of class centroids and the number of classes, and is independent of the size of training documents. Therefore, when datasets only contain a small number of classes and a large number of training and test documents, *CenKNN* and *Centroid* can run orders of magnitude faster than *KNN* on these datasets.

4 Experimental results and comparisons

4.1 Experimental settings

We first carried out experiments on real world corpora (see details in Sect. 4.2) to analyze the performance of *CenKNN* in terms of classification accuracy at the beginning of Sect. 4.3. *KNN* and *Centroid* were used as baseline classifiers. *INNTC* [3] and *Rocchio* were used as comparison classifiers. The results reported in Jiang et al. (2012) demonstrated that *INNTC* could obtain relatively stable performance when its clustering radius r and the K value were set in the interval $[3.0 \times ex, 10.0 \times ex]$ and [5, 45] respectively (ex is the average of the similarities of sampling document pairs). After running *INNTC* with different r and K values on the corpora used, we found that *INNTC* could obtain stable performance with $r = 9.0 \times ex$ and $K = 10$ respectively. We used this parameter setting for *INNTC* in our experiments. This parameter tuning method is also applied to *Rocchio*, and its adjustment parameters, i.e., α and β, are finally set as $\alpha = 2\beta = 1$. The default K value of *CenKNN* and *KNN* was 10, as our sensitivity examination experiments show that *CenKNN* and *KNN* can achieve their typical classification performance when $K = 10$.

Later in Sect. 4.3, *CenKNN* was also compared with other *KNN* variants using different data compression methods, including *DF* and Information Gain (*IG*) feature selection methods, and *LSI* and *RP* dimension reduction methods. *DF*, *IG* and *LSI* are commonly used data compression methods and have been reported as promising methods for text classification (Yang and Pedersen 1997; Sun et al. 2004). *RP* has emerged as another powerful tool for dimension reduction in recent years (Bingham and Mannila 2001; Achlioptas 2003; Lin and Gunopulos 2003; Papadimitriou et al. 1998). Sparse *RPs* (Bingham and Mannila 2001) were used in our experiments, since they can achieve a threefold speedup compared to conventional RPs. For both *LSI +KNN* and *RP+KNN*, we used the k-d tree to search for the K nearest neighbors because *LSI* and *RP* can project documents into a lower dimension space.

[3] Our experiments showed that another variant we proposed in Pang and Jiang (2013) had quite similar performance as *INNTC* (Jiang et al. 2012). So we only compared *CenKNN* with *INNTC* rather than both of them.

We then examined the scalability of *CenKNN* on a very large-scale and high-dimensional corpus in Sect. 4.4. In Sect. 4.5, to examine the sensitivity of *CenKNN*, we applied *CenKNN* to perform classification with different K values, as the K value is the only parameter in *CenKNN*. We also investigated the sensitivity of *CenKNN* with respect to varying numbers of classes.

We finally compared *CenKNN* with *SVM* and its variant in Sect. 4.6. We used *LibSVM* (Chang and Lin 2011) for *SVM* implementation. Following Yang and Liu (1999); Guan et al. (2009); Lan et al. (2009), the linear kernel and the default settings were used (Joachims 1998; Yang and Liu 1999; Guan et al. 2009).

A series of large scale hierarchical text classification (*LSHTC*) Pascal challenges has drawn researchers' attention to classification on a corpus with large-scale text documents and hierarchical classes (Kosmopoulos et al. 2010). The majority of hierarchical text classifiers applied into the challenges were modified from flat text classifiers with hierarchical classification strategies (Kosmopoulos et al. 2010; Han et al. 2012; Miao and Qiu 2009). Moreover, these modified flat text classifiers (e.g., *KNN*, *SVM*, *Centroid*) have achieved promising performance in the challenges, such as the arthur systems of the past *LSHTC* Pascal challenges (Wang et al. 2011, 2013). Obviously, the effectiveness and efficiency of flat text classifiers plays an important role in the success of hierarchical text classifiers. We therefore focus on examining the non-hierarchical classification performance of *CenKNN* in this paper, though it can be recursively used for hierarchical classification.

4.2 Datasets and performance metrics

A series of experiments has been conducted on both English and Chinese corpora to evaluate our proposed method. To have a comprehensive examination of our proposed method, both balanced and imbalanced corpora were selected as our experimental data, i.e., the ratio of the number of training documents in the smallest and the largest class (referred to as skew ratio) ranges from 1:1 up to 1:284. We first briefly describe the benchmark corpora used.

Reuters-21578[4] is a standard benchmark corpus for text classification. We used the "ModApte" split version of Reuters-21578, which is one of the most popular splits. To ensure a sufficient number of documents to train the classifiers and perform classification, classes with no less than 10 training documents and one test document were selected. Under this constraint, 35 out of 115 classes are left in our subset (denoted $R35$), which contains 6,454 and 2,513 training and test documents respectively. After filtering out stop words, the subset has 21,476 different terms in total.

20Newsgroup[5] is a collection of nearly 20,000 newsgroup documents, organized evenly into 20 different newsgroups. Different preprocessing leads to three versions of this corpus. We used the "by-date" version, denoted by *News20*, which had 11,268 and 7,503 training and test documents respectively. The size of vocabulary for this corpus is 53,239 after removing stop words.

[4] Reuters-21578 is available at http://archive.ics.uci.edu/ml/databases/reuters21578/.

[5] 20Newsgroup is available at http://qwone.com/~jason/20Newsgroups/.

TanCorp[6] is a widely used Chinese text classification corpus. This corpus contains 14,150 documents in total, which are organized into two hierarchies. The first hierarchy has 12 classes, which are divided into 60 classes in the second hierarchy. We focus on checking the flat text classification performance of *CenKNN*, so this corpus is used with the first level classes, denoted by *Tan12*. Two thirds of the documents are split into the training set and the rest is regarded as test documents. There are 53,907 terms contained in this corpus.

The Fudan University text classification corpus[7] is from the Chinese natural language processing group in the Department of Computer Information and Technology at Fudan University. It is another widely used corpus for Chinese text classification. The collection contains 20 classes with 9,804 Chinese training documents and 9,833 Chinese test documents. The entire Fudan Univ. text classification corpus was used (referred to as *Fudan20*). During preprocessing, some types of stop words were filtered out, i.e., prepositions, conjunctions, pronouns, interjections, auxiliary particles and particles of speech. We obtained 335,664 different terms at this stage. Since the documents are collected from academic journals, they contained many numbers and mathematical symbols. To save experiment time, we removed these terms to obtain a document collection with 101,351 terms.

The *DMOZ* data used is a large-scale dataset based on the *ODP* (Open Directory Project) web directory data, which was constructed and distributed by the first edition of the Large Scale Hierarchical Text classification (*LSHTC*) challenge[8] in 2009. Two datasets, a large one and a small one, were used in this challenge. We used the large one (referred to as *DMOZ10*). The number of classes in this dataset, with hierarchy, amounts to 12,294, and there are 10 classes in the first level of the hierarchy. We focus on examining *CenKNN*'s non-hierarchical classification performance on these 10 classes. *DMOZ10* consists of training data, validation data and test data. However, the test data cannot be used for evaluating our classifiers, because the class labels of the test data are all set to 0, in order to prevent the challenge participants from knowing the true class of the data. We therefore used the training data to train classifiers and tested the classifiers on the validation data. Overall, *DMOZ10* consists of 10 classes and contains 93,805 training documents and 34,905 test documents, and the size of the vocabulary is 368,113.

Table 2 is a summary of these five corpora. In Forman (2003), the ratio 1:67 was suggested to be the threshold for highly skewed corpora. Following this threshold, *R35* is a highly skewed corpus; *Fudan20* is quite close to highly skewed; and *DMOZ10*, *Tan12* and *News20* belong to low skewed or balanced corpora. In terms of the split of documents into training and test document sets, for *News20*, *DMOZ10*, *Fudan20* and *R35*, we used the original splits, designed by the corpus creators or developers, to facilitate other researchers to compare their techniques with our method on these

[6] TanCorp is available at http://www.searchforum.org.cn/tansongbo/corpus.htm.

[7] Fudan University text classification corpus is available at http://www.nlp.org.cn/docs/download.php?doc_id=294.

[8] DMOZ datasets are available at http://lshtc.iit.demokritos.gr/node/3.

Table 2 Basic information of the corpora *News20*, *Tan12*, *DMOZ10*, *Fudan20* and *R35*

Corpora	# of Classes	# of Terms	# of Train Docs	# of Test Docs	Skew ratio
News20	20	53, 239	11, 268	7, 503	1:1
Tan12	12	53, 907	9, 431	4, 719	1:20
DMOZ10	10	368, 114	93, 805	34, 905	1:28
Fudan20	20	101, 351	9, 804	9, 833	1:64
R35	35	21, 476	6, 454	2, 513	1:284

corpora. Since *Tan12* is available without a specific corpus split, we used the ratio 2:1 to split this corpus into training and test document sets[9].

Two commonly used performance measures, $microF_1$ and $macroF_1$, were used as classification accuracy metrics. F_1 is a combination measure of precision and recall, i.e., $F_1 = \frac{2 \times precision \times recall}{precision + recall}$. Both $microF_1$ and $macroF_1$ were used for the overall performance evaluation of different classifiers. $MicroF_1$ is the global calculation of F_1 regardless of classes. $MacroF_1$ is the average over the F_1 values of all the classes. Therefore, the $microF_1$ value is normally dominated by the performance on majority classes while $macroF_1$ treats every class equally, and so $macroF_1$ is a more important measure for classification performance on skewed corpora than $microF_1$.

4.3 Classification performance

Extensive experiments were conducted to compare *CenKNN* with *Centroid*, *Rocchio*, *KNN* and its five variants on the five corpora except *DMOZ10*, which is a very large-scale corpus and is quite time-consuming for *KNN* and most of its variants. Comparison results on *DMOZ10* will be given in the coming subsections.

Comparison with Centroid, Rocchio, KNN and its variant using clustering techniques We first compare *CenKNN* with Centroid, Rocchio (parameterized version of Centroid), *KNN* and its variant using clustering techniques (i.e., *INNTC*). Table 3 shows the $microF_1$ and $macroF_1$ values of *CenKNN, KNN, INNTC, Centroid* and *Rocchio* on *News20, Tan12, Fudan20* and *R35* with different skew ratios (underlined numbers indicate the best performance per measure). The p-values of paired Student's t-tests of *CenKNN* against the other four classifiers over F_1 crossing classes are listed in Table 4 (bold numbers indicate statistical significance) [10]. It can be observed that *CenKNN* consistently outperforms *KNN, INNTC, Centroid* and *Rocchio* on both balanced and imbalanced corpora in terms of both $microF_1$ and $macroF_1$ values. Compared to *KNN*, on average, *CenKNN* has about 6.18 and 7.42 percent improvements in $microF_1$ and $macroF_1$ values respectively. Student's t test result shows these improvements are

[9] This version of Tan12 is available at http://www.scholat.com/vpost.html?pid=3047.

[10] In our significance test, for each classifier per data set, the F_1 values for each class of all the classes were used as sample data. The paired-sample t-test was used to test the null hypothesis that the pairwise difference between the F_1 values of two classifiers has a mean equal to zero. 0.05 is used as the statistical significance level throughout this paper.

Table 3 Performance of five classifiers on both English and Chinese corpora

		CenKNN	*KNN*	*INNTC*	*Centroid*	*Rocchio*
News20	$microF_1$	0.7710	0.7345	0.7294	0.7608	0.7549
	$macroF_1$	0.7640	0.7306	0.7335	0.7550	0.7505
Tan12	$microF_1$	0.9243	0.8979	0.8887	0.9099	0.9074
	$macroF_1$	0.8915	0.8489	0.8456	0.8732	0.8720
Fudan20	$microF_1$	0.9023	0.8715	0.8723	0.8266	0.8556
	$macroF_1$	0.7756	0.6975	0.7324	0.7154	0.7327
R35	$microF_1$	0.9491	0.8365	0.8822	0.9256	0.9113
	$macroF_1$	0.8905	0.8152	0.8480	0.8531	0.8343

Table 4 p values from paired one-side Student's t tests of *CenKNN* against other classifiers over F_1 crossing classes

	KNN	*INNTC*	*Centroid*	*Rocchio*	
CenKNN	**0.0125**	**0.0069**	**0.0194**	**0.0019**	*News20*
	0.0427	**0.0064**	**0.0135**	**0.0338**	*Tan12*
	0.0002	**0.0058**	**0.0000**	**0.0001**	*Fudan20*
	0.0020	**0.0134**	0.1526	**0.0391**	*R35*

statistical significantly over all the four corpora, as the corresponding p values range 0.0002 to 0.0427. Similar improvements can be found over the other three methods.

Comparison with KNN variants using feature selection methods We compared our method to the combinations of *KNN* with two widely used text feature selection methods, i.e., *IG* and *DF*. The proportions of the most informative term features we used range from 0.1 to 1.0. *CenKNN* was performed on the full feature space by default. To have a fair performance comparison, we applied *CenKNN* on features selected by *IG* or *DF* in this subsection. For *KNN*, the optimal K value is used with respect to different corpora based on the results of our sensitivity examination over different K values (i.e., $K = 50$ on *News20*, $K = 40$ on *Tan12*, $K = 5$ on *Fudan20*, $K = 20$ on *R35*). The results are illustrated in Figs. 2 and 3. The results show that given different proportions of term features and different skew ratios, *CenKNN* performs better than *KNN* with *IG* or *DF* over all the four corpora. This is especially clear for the skewed corpora.

It can also be observed from Fig. 3 that, after removing substantial term features (e.g., 50–90 %), *CenKNN* works stably on all the four corpora. This indicates that large proportions of term features in these four corpora are irrelevant or noisy features. *KNN* is less stable. After removing the redundant features, it had some improvements (say, after removing 80 % terms in *Fudan20* or *Tan12*) or became clearly worse in terms of $macroF_1$ in *News20* and *R35*. *CenKNN* performs consistently better than *KNN* with or without irrelevant or noisy features in both of $microF_1$ and $macroF_1$ values. Additionally, the results also suggest that using feature selection methods (e.g., *IG*) might substantially downgrade the *KNN* text classification accuracy.

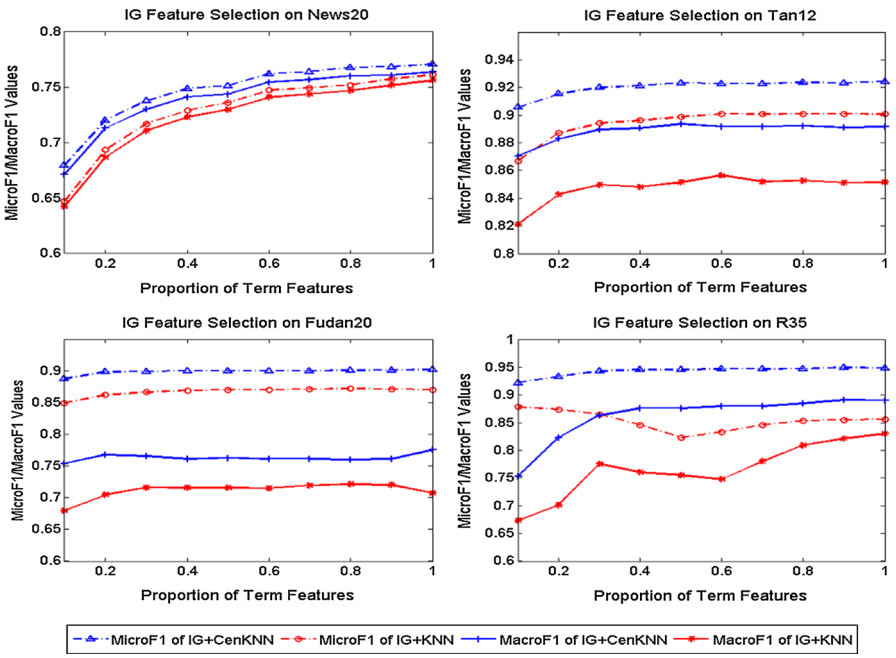

Fig. 2 Comparison of *CenKNN* and *KNN* with *IG* feature selection

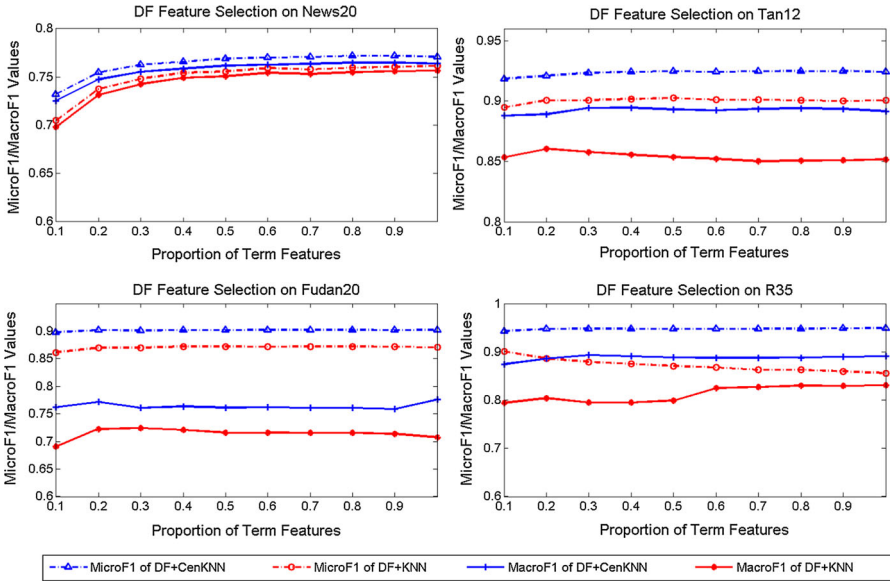

Fig. 3 Comparison of *CenKNN* and *KNN* with *DF* feature selection

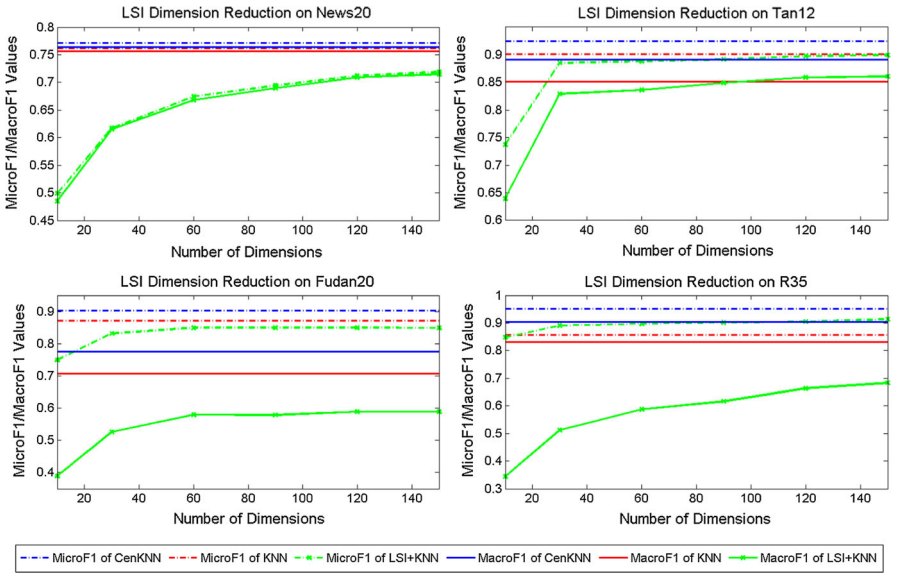

Fig. 4 Performance of *KNN* with *LSI* dimension reduction compared to *CenKNN* and *KNN*

Comparison with KNN variants using dimension reduction methods We also compared our method to the combinations of *KNN* with *LSI* or *RP* dimension reduction. Both *LSI* and *RP* dimension reduction performed on the original full term feature space. Instead of the default K value, *KNN* also used the optimal K value with respect to different corpora in this subsection. To have a broad comparison to *KNN* and *CenKNN*, we tested *LSI* on the corpora *News20, Tan12, Fudan20* and *R35* using various dimensions (i.e., 10, 30, 60, 90, 120, 150). To conduct a comparison of *CenKNN* to *RP+KNN* and *LSI+KNN*, we only report their detailed results on two typical corpora, one balanced corpus *News20*, and one highly skewed corpus *R35*. Similar comparison results could be found on the other two corpora. The experiments of *RP+KNN* were also conducted with a set of dimensions, i.e., 100, 200, 300, 400, 500, 600. Both *LSI* and *RP* dimension reduction were conducted in Matlab. We used the svds function to perform Singular Value Decomposition *(SVD)* for *LSI*. The svds function is an efficient *SVD* tool for very large sparse matrices, which is likely to reduce *LSI*'s time complexity from $O(N \cdot n^2)$ to $O(n \cdot N \cdot Tr_{avg})$ (Lin and Gunopulos 2003).

The results of *LSI+KNN* and *RP+KNN* are shown in Figs. 4 and 5 respectively. The results of *RP+KNN* derive from 10 independent runs to avoid randomness. In these two figures, *KNN* and *CenKNN* were used as comparison classifiers, and they both worked on the full term feature space rather than the *LSI* or *RP* based projected space, which is why their $microF_1$ and $macroF_1$ values are flat regardless of the changes of dimensionality.

It can be seen from Fig. 4 that although *LSI+KNN* sometimes obtains some improvements over *KNN*, e.g., the $macroF_1$ values on *Tan12* and the $microF_1$ values on *R35*, it performs substantially worse than *CenKNN* on these four corpora. Figure 5 shows *RP+KNN*'s mean curve with a 95 percent confidence interval over 10 runs. In Fig.

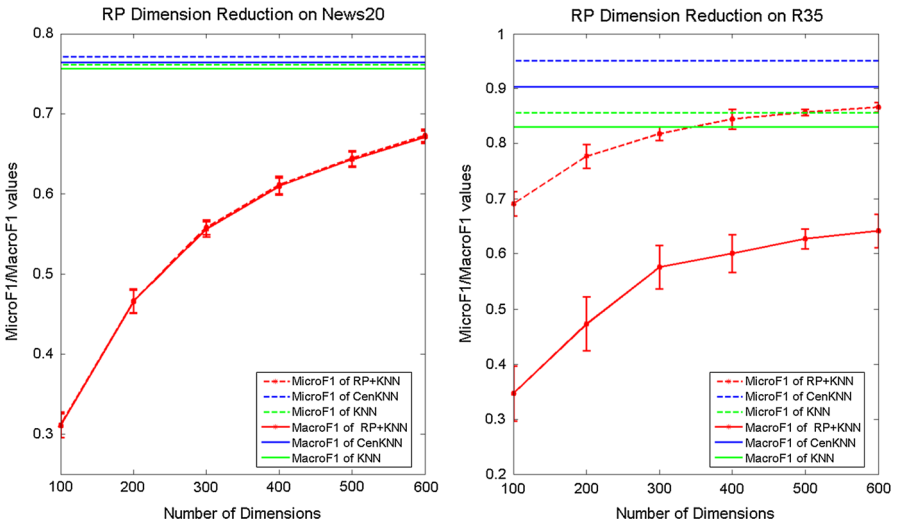

Fig. 5 Performance of *KNN* with *RP* dimension reduction compared to *CenKNN* and *KNN*

Table 5 Comparison of *RP+KNN*,*LSI+KNN* and *CenKNN*

Classifier	*News20*				*R35*			
	Dimension	Time	$microF_1$	$macroF_1$	Dimension	Time	$microF_1$	$macroF_1$
RP+KNN	100D	156+160	0.3117	0.3105	100D	35+32	0.6906	0.3465
	600D	390+869	0.6725	0.6708	600D	97+169	0.8661	0.6417
LSI+KNN	10D	23+9	0.4985	0.4850	10D	9+1	0.8484	0.3435
	150D	200+227	0.7188	0.7145	150D	87+42	0.9134	0.6830
CenKNN	20D	29+22	0.7710	0.7640	35D	10+11	0.9491	0.8905

Underlined numbers indicate the best performance per measure

5, in the performance on *News20*, *RP+KNN* obtains nearly equivalent performance in $microF_1$ and $macroF_1$ values, which are all statistically significantly lower than those of *KNN* and *CenKNN* as its upper bound of the 95 % confidence intervals lies far below its counterparts. Similarly, *RP+KNN* works poorly on *R35*, with the exception that *RP+KNN* achieves higher $microF_1$ values than *KNN*, when the dimension size of the projected space reaches 500. The superiority of *CenKNN* over *RP+KNN* is still statistically significant on *R35*. Increasing dimensionality size of the projected space is expected to further improve the classification accuracy of *RP+KNN*, at the expense of longer classification time because *KNN* will run much slower with a larger size of dimensionality.

We conducted an overall comparison between *RP+KNN*,*LSI+KNN* and *CenKNN* in terms of time efficiency (i.e., dimension reduction time + *KNN* classification time in seconds) and classification accuracy. For *RP+KNN* and *LSI+KNN*, we directly derived the most efficient results and the most accurate results above. The only parameter of *CenKNN* was still set by default, i.e., $K = 10$, though it was not the optimal

setting. The comparisons are shown in Table 5. Although *CenKNN* performs on a significantly lower-dimensional space than *RP+KNN*(600D)[11] and *LSI+KNN*(150D), it obtains substantially better classification accuracy. Since *RP* does not preserve as much information as *LSI*, *KNN* needs to work on a *RP*-based projected space with a larger size of dimensionality in order to obtain a comparable accuracy to *LSI+KNN*, which is consistent with the results reported in Lin and Gunopulos (2003). In terms of execution time, *LSI+KNN*(10D) achieves the best efficiency with *CenKNN* second, while it performs poorly on both of *News20* and *R35*. *CenKNN* runs about 8 times and 6 times faster than *LSI+KNN*(150D), and is about 25 times and 13 times faster than *RP+KNN*(600D), on *News20* and *R35* respectively. The speedup of *CenKNN* over *RP+KNN*(600D) and *LSI+KNN*(150D) will increase substantially when dealing with data with large numbers of documents, since *CenKNN* can project documents efficiently and works on a smaller dimensionality size.

4.4 Scalability examination

In real-world text classification applications, it is of great importance that classifiers can scale up well with the number of text documents. We used *DMOZ10* to examine the scalability of *CenKNN*, with Centroid and *KNN* as baselines. To demonstrate the classifiers' scalability, we first sampled (via stratified random sampling where each class is a stratum) multiple subsets of training documents of *DMOZ10* using a range of sampling ratios (i.e., 1/20, 1/10, 1/8, 1/5, 1/4, 1/3 and 1/2). The three classifiers were then trained on these subsets and the entire training document set, and finally classified the whole test documents. We repeated this process 10 times. The number of documents in the training sets range from 4,691 documents up to 93,805 documents. The mean curves of $microF_1$ and $macroF_1$ of these three classifiers, with a 95 percent confidence interval, are presented in Fig. 6. The figure shows that *CenKNN* statistically significantly outperforms *Centroid* and *KNN* in the $microF_1$ values regardless of the changes of training data size as its lower bound lies higher than the upper bounds of its two counterparts on each sampling proportion. In $macroF_1$, the mean curve of *CenKNN* consistently outperforms *Centroid* and *KNN* though the difference is not statistically significant. It is worthwhile to note that, compared to the fluctuating performance of *Centroid*, the accuracy of *CenKNN* and *KNN* has small variations at a specific sampling ratio and goes up steadily with increasing size of training documents.

The total execution time and online execution time, i.e., computation time on test data, including dimension reduction time on test data (if applicable) and class assignment time, on *DMOZ10* are shown in Fig. 7. To show the scalability clearly, time cost is plotted on a logarithmic scale. As shown in the left panel, *CenKNN* takes 57 seconds in the classification task with 4,691 training documents and 34,905 test documents (i.e., when sampling ratio is 1/20), and takes 239 seconds in the classification task with 93,805 training documents and 34,905 test documents (i.e., when the entire training data set is used). *CenKNN* runs about 25 times to 107 times faster than *KNN*, and is comparably fast as *Centroid*. In fact, the execution time of *CenKNN* and *Centroid*

[11] Hereafter this format refers to the classifier working on a space with a specific dimension.

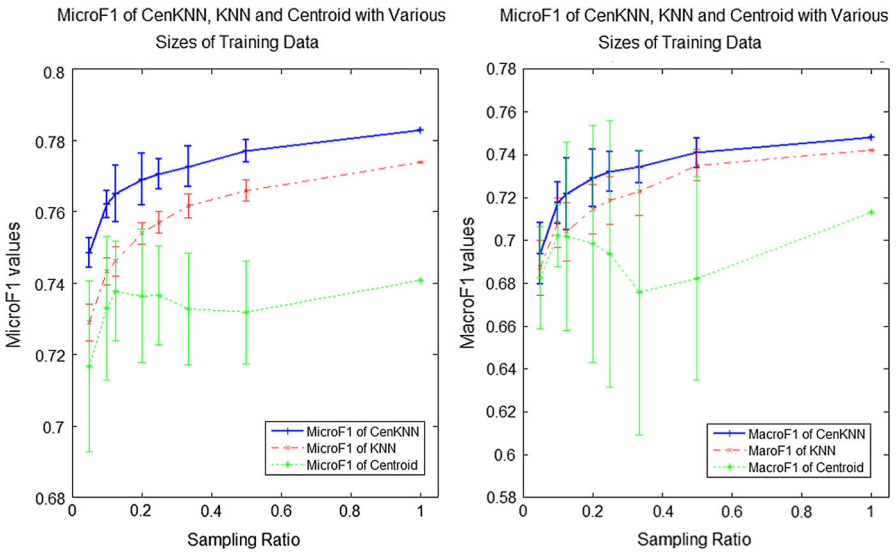

Fig. 6 Mean curves along with a 95 percent confidence interval of *CenKNN*, *KNN* and *Centroid* on *DMOZ10*

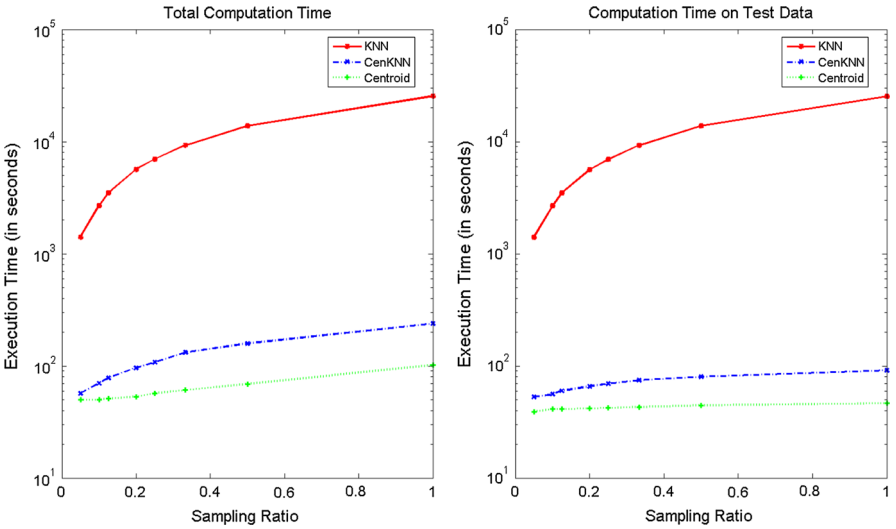

Fig. 7 Computation time of *CenKNN*, *KNN* and *Centroid* on *DMOZ10* with varying size of training documents

increases very slowly with the number of text documents, as opposed to the rapid increase of *KNN*. Similar to *Centroid*, *CenKNN* can scale up well with data size.

The online execution time of these classifiers is detailed in the right panel of Fig. 7. Since *KNN* is a lazy learning method, its online execution time is nearly equivalent to the total execution time. *CenKNN* is about 27–278 times faster than *KNN*. Apart from the test documents' projection time, the speedup of *CenKNN* over *KNN* ranges

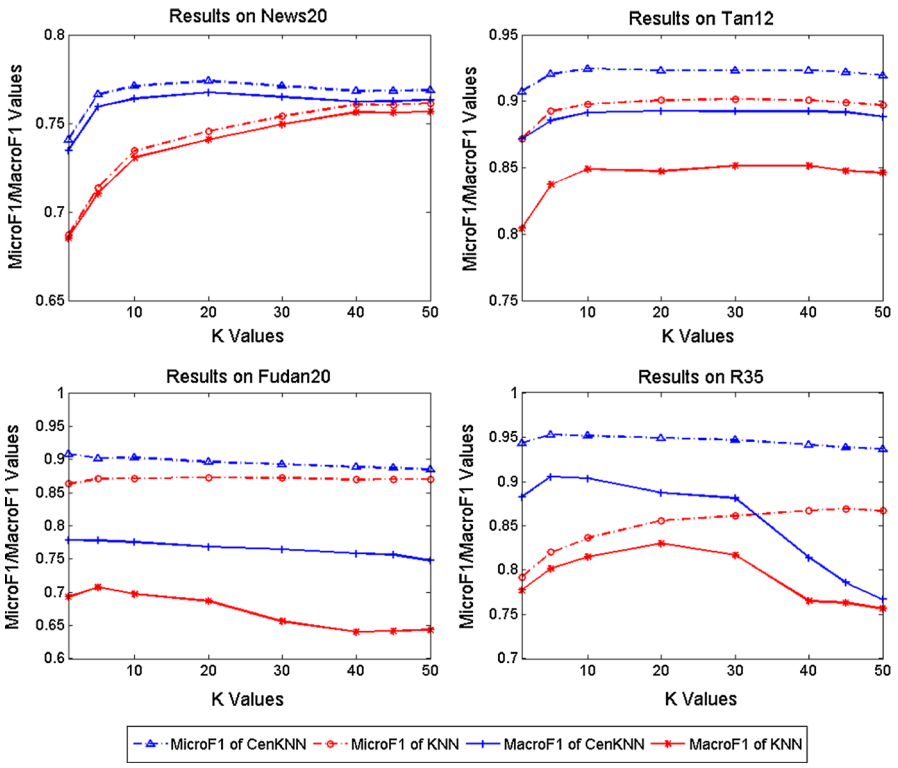

Fig. 8 Performance of *CenKNN* and *KNN* over various K values

from 37 to 550 times. *CenKNN* runs slightly slower than *Centroid*. It is worth noting that *CenKNN* requires significantly less main memory than *KNN* as only projected data for each document is stored in *CenKNN*. Therefore, considering the accuracy degradation of *KNN* on the sampled training documents (as shown in Fig. 6), the speedup of *CenKNN* can be very high when the entire training document set cannot fit into the main memory.

4.5 Sensitivity examination over different *K values* and number of classes

We employed various K values (i.e., 1, 5, 10, 20, 30, 40, 45, 50) to perform classification to analyze the sensitivity of our method, using *KNN* as a baseline. The experimental results are shown in Fig. 8. *CenKNN* has more effective and stable performance than *KNN* over all the different K values on these four corpora. The superiority of *CenKNN* becomes clearer on skewed corpora. It should be noted that the classification performance will be substantially degraded when using large K values on highly skewed corpora. As a specific example, many minority classes in *R35* have only 10–20 training documents while majority classes have hundreds or thousands of documents. Thus, if we use K values that are too large (e.g., larger than 20), we will obtain high F_1 values for majority classes but low F_1 values for minority classes since minority

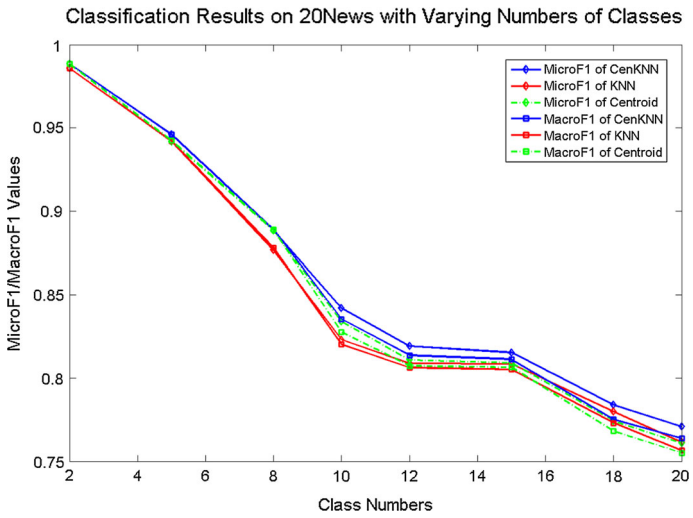

Fig. 9 Performance of *CenKNN*, *Centroid* and *KNN* with varying numbers of classes on *News20*

classes lack sufficient training documents to make predictions. As a result, $macroF_1$ values decrease dramatically with the increase of K values in *R35*, while $microF_1$ values remain relatively steady.

With the increase of class number, the discrepancy between different classes would reduce. So it would become more challenging to perform dimension reduction and classification effectively. For *CenKNN*, the change of class number also means the change of the dimensionality in the class-centroid-based space built by *CentroidDR*. We used the *News20* to examine the effect of varying class numbers on the performance of *CenKNN*, with *KNN* and *Centroid* as baselines. The main reason for choosing *News20* is that classification results on *News20* can better reflect the relationship between the number of classes and classification performance, as there are no skewed class distribution issues and fewer irrelevant term features in *News20* (This can be observed from Fig. 3). The experiments start by randomly selecting two classes, ending up with twenty classes[12]. The results are shown in Fig. 9, which demonstrates that although the classification performance of all the classifiers decreases rapidly with increasing class number, *CenKNN* is able to outperform *KNN* and *Centroid* consistently over different numbers of classes. This suggests that *CenKNN* can achieve good performance regardless of the variation in class numbers.

4.6 Comparison with SVM and its variant

We now turn to examine *CenKNN*'s performance against another state of the art text classifier, *SVM*, as well as *CenSVM*, in which *SVM* performs on the class-centroid-based space.

[12] These synthetic corpora are available at http://www.scholat.com/vpost.html?pid=2395.

Table 6 *CenKNN*, *SVM* and *CenSVM* classification results on the five corpora

	CenKNN		*SVM*		*CenSVM*	
	$microF_1$	$macroF_1$	$microF_1$	$macroF_1$	$microF_1$	$macroF_1$
News20	0.7710	0.7640	0.8040	0.8010	0.7728	0.7660
Tan12	0.9243	0.8915	0.9479	0.9200	0.9243	0.8892
DMOZ10	0.7829	0.7481	0.8513	0.8279	0.7869	0.7535
Fudan20	0.9023	0.7756	0.9191	0.7640	0.9064	0.7679
R35	0.9491	0.8905	0.9387	0.8665	0.9033	0.6992

Underlined numbers indicate the best performance per measure

Table 6 shows the classification results of *CenKNN* against *SVM* and *CenSVM* on *News20*, *Tan12*, *DMOZ10*, *Fudan20* and *R35*. *CenSVM* is *SVM* with *RBF* kernel performing on the class-centroid-based space. For the γ value in the *RBF* kernel, a set of values (i.e., 0.03, 0.1, 1, 3, 5, 10, 20, 30, 40 and 50) along with other default settings has been used to check its classification performance on all the five corpora[13]. Our results showed that *CenSVM* obtained the best performance on most of the five corpora when the γ value was 30. *CenSVM* performed less effectively and stably when changing to the other γ values. Here we present the experimental results of *CenSVM* with the γ value setting as 30 in order to facilitate a fair comparison. It can be seen from Table 6 that *SVM* obtains better performance than *CenKNN* on balanced and low skewed corpora, e.g., *News20*, *Tan12* and *DMOZ10*. However, on corpora with greater levels of class skew, *SVM* becomes less effective. When there is a quite highly skewed corpus (i.e., *Fudan20*), *CenKNN* achieves higher $macroF_1$ than *SVM*. In the highly skewed corpus *R35*, *CenKNN* outperforms *SVM* in both of $microF_1$ and $macroF_1$. This is because *SVM* performs poorly on minority classes on which *CenKNN* performs well, and dragged down the $macroF_1$ values. *CenSVM* performs similarly as *CenKNN* except *R35*, on which *CenSVM* performs substantially worse than *CenKNN*. Its $macroF_1$ value on *R35* is as low as 0.6992, which indicates relatively less stability for a fixed γ value.

The computation time of these three classifiers are summarized in Table 7. *CenKNN* runs much faster than *SVM*, especially for large corpora. For instance, for *DMOZ10*, *SVM* takes 29,410 seconds, while *CenKNN* takes only 239 seconds, about 123 times faster. In terms of online execution time, *CenKNN* consistently outperforms *SVM* except on *R35*. Although *CenSVM* can save a bit on classification time, it runs slower than *CenKNN* in terms of total computation time.

To further examine the influence of class skew on the performance of *CenKNN*, *SVM* and *CenSVM*, we utilized *Fudan20*to create several subsets of highly skewed corpora. There are 9 minority classes in *Fudan20* that have 25–75 training documents, against 11 majority classes with documents between 466 and 1,600. We randomly sampled half

[13] The results of *CenSVM* using different kernels show that non-linear kernels perform better than the linear kernel on the five corpora. This suggests that documents are often not linearly separable in the low-dimensional class-centroid-based space. Our results also show that the *RBF* kernel outperforms the polynomial kernel. These results have been made available at http://www.scholat.com/portalPaperInfo_Eng.html?paperID=19993\&Entry=pines.

Table 7 Computation time (in seconds) of *CenKNN*, *SVM* and *CenSVM*

	Total execution time			Online execution time		
	CenKNN	*SVM*	*CenSVM*	*CenKNN*	*SVM*	*CenSVM*
News20	51	238	52	35	62	25
Tan12	25	241	30	9	59	9
DMOZ10	239	29, 410	942	91	5, 049	271
Fudan20	131	918	142	73	309	68
R35	21	60	34	14	6	7

Underlined numbers indicate the best performance per measure

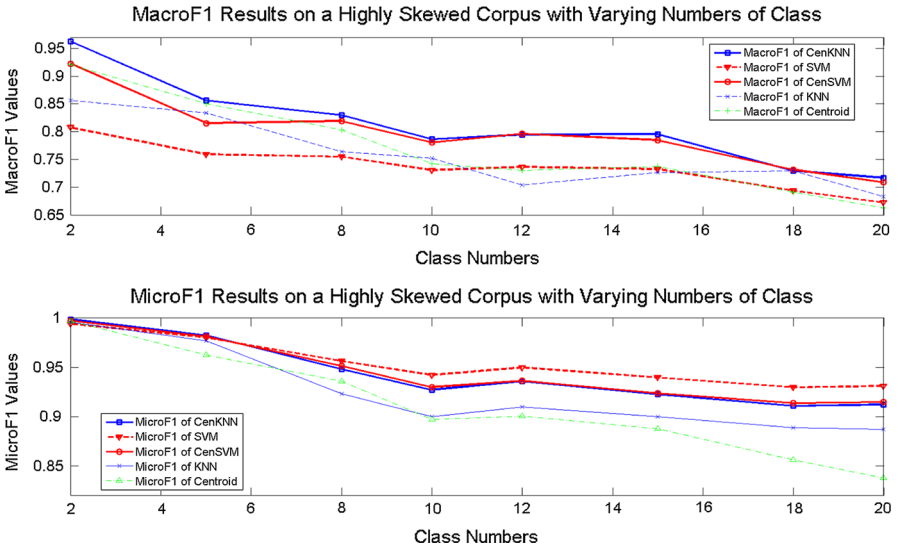

Fig. 10 *CenKNN*, *SVM* and *CenSVM* comparisons on a highly skewed corpus with increasing class numbers

of the documents from all the minority classes. Together with the 11 majority classes, we had a new highly skewed corpus with the skew ratio of approximately 1:123. We used this corpus to further create 8 highly skewed subsets with the same skew ratio but with different class numbers that range from 2 to 20. The experiments start by classification on the largest and the smallest classes, followed by randomly adding minority and majority classes, ending up with twenty classes. Each subset contains nearly the same number of minority classes and majority classes[14]. We checked the performance of *CenKNN*, *SVM* and *CenSVM* given data with a fixed highly skewed ratio and varying class numbers. The $microF_1$ and $macroF_1$ results of *CenKNN*, *SVM* and *CenSVM* on these 8 subsets, with *KNN* and *Centroid* as baselines, are shown in Fig. 10.

The figure shows that, with the increase of the class number, the performance of *CenKNN*, *SVM* and *CenSVM* is degraded, while *CenKNN* and *CenSVM* substantially

[14] These synthetic corpora are available at http://www.scholat.com/vpost.html?pid=2396.

outperform *SVM* over all the subsets in terms of $macroF_1$ values. *CenKNN* performs often better than *CenSVM* in terms of $macroF_1$ while quite similarly in terms of $microF_1$. It is worth noting that the gap between *SVM* and *CenKNN* (or *CenSVM*) in terms of the $macroF_1$ value decreases with class numbers, while the gap on the $microF_1$ value increases. This could be because the skew ratio between any two different classes (except the smallest and the largest classes) is lower than 1:123 with the increase of the class number. That is, for these 8 data sets with the skew ratio 1:123, *CenKNN* outperforms *SVM* consistently in terms of $macroF_1$. *CenKNN* has more competitive $macroF_1$ values when the class size ratios become extreme. Recall that$microF_1$ is of less importance for measuring classification performance on skewed corpora than $macroF_1$, especially for highly skewed corpora. Therefore, although *SVM* outperforms *CenKNN* (or *CenSVM*) in the average $microF_1$ value, *CenKNN*'s performance on these highly skewed corpora is more desirable than that of *SVM* and *CenSVM*.

We also conducted experiments to check the performance of *CenKNN*, *SVM* and *CenSVM* given data with a fixed class number and varying imbalance ratios. Two subsets were selected from *DMOZ10*, consisting of two classes for each subset, denoted by *DMOZ10-S1* and *DMOZ10-S2* respectively[15]. *DMOZ10-S1* contains one large class and one small class with 6,921 training documents and 2,555 test documents in total, and its vocabulary size is 62,334. *DMOZ10-S2* consists of two of the largest classes in *DMOZ10*, with 24,335 training documents and 9,281 test documents in total. There are 176,828 terms contained in this subset. For these two subsets, we kept documents of the larger class unchanged and randomly sampled training and test documents of another class according to 13 different class imbalance ratios, ranging from 1:10 to 1:130. *CenKNN*, *SVM* and *CenSVM* then performed on these subsets sampled. The mean curves of $microF_1$ and $macroF_1$ of these three classifiers on *DMOZ10-S1* and *DMOZ10-S2*, with a 95 percent confidence interval over 10 runs, are shown in Figs. 11 and 12 respectively. *KNN* and *Centroid* were used as baselines. The results demonstrate that the performance of these five classifiers decrease with imbalance ratios in terms of $macroF_1$. For both subsets of data, when the skew ratio is small, *CenKNN* is not as good as *SVM*. However, *CenKNN* clearly outperforms *SVM* in terms of both $microF_1$ and $macroF_1$ when the skew ratio is large (for example, >30 for *DMOZ10-S1* and >100 for *DMOZ10-S2*). Thus, *CenKNN* is more resilient to imbalanced text corpora than *SVM*. *CenSVM* has similar classification performance to *CenKNN* for these two subsets of *DMOZ10*.

4.7 Discussion

The promising performance of *CenKNN* has been illustrated by our extensive experiments on a variety of corpora. We now discuss when to use *CenKNN*.

The effectiveness and efficiency of *CenKNN* is closely related to the performance of the class-centroid-based dimension reduction method *CentroidDR* and the k-d tree search. The strong representation power of the class-centroid-based space enables

[15] These two subsets are available at http://www.scholat.com/vpost.html?pid=4038.

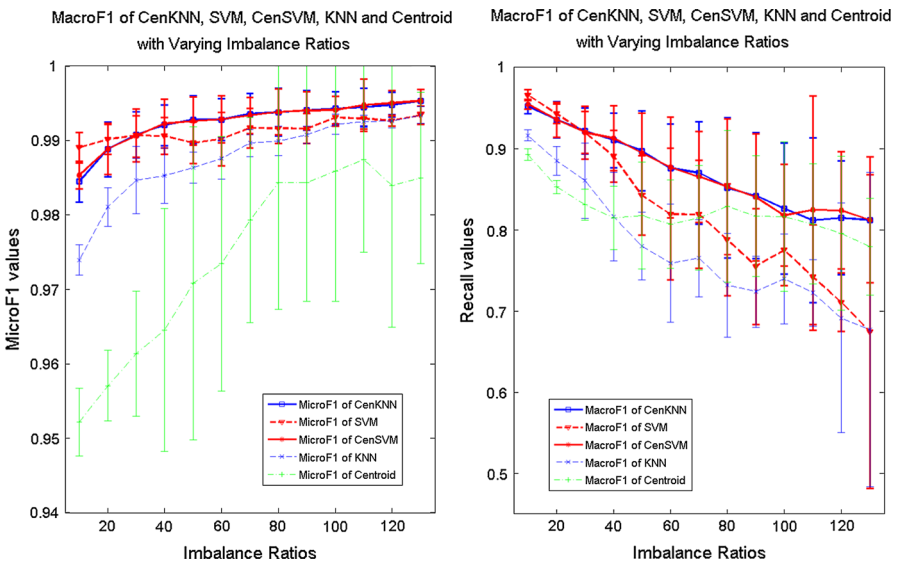

Fig. 11 *CenKNN*,*SVM* and *CenSVM* comparisons on *DMOZ10-S1* with varying imbalance ratios

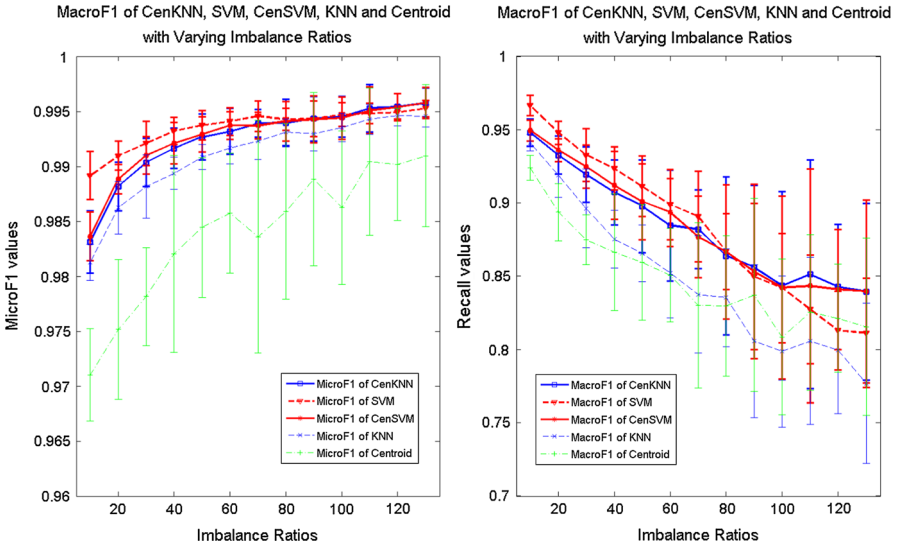

Fig. 12 *CenKNN*, *SVM* and *CenSVM* comparisons on *DMOZ10-S2* with varying imbalance ratios

CenKNN to obtain substantially better classification accuracy than *KNN* and its variants, though *CenKNN* normally works on a significantly lower-dimensional space. Compared to the performance gap between *CenKNN* and *KNN* on balanced corpora (e.g., *News20*), the superiority of *CenKNN* becomes clearer when dealing with more complex corpora, such as imbalanced corpora with large proportions of irrelevant or noisy term features (e.g., *Tan12*, *Fudan20* and *R*35), where *KNN* itself cannot perform

well. The performance of *CentroidDR* is dependent on the *Centroid* classifier, because *CentroidDR* uses class centroids to define similarity. Therefore, *CenKNN* can usually obtain good performance in corpora where *Centroid* works well, such as the classification results on *News20*, *Tan12*, *Fudan20* and *R*35. But this does not necessarily mean that *CenKNN* would perform less effectively when *Centroid* performs poorly, since *CenKNN* makes prediction via the *KNN* classification rule, which has better classification ability than *Centroid*. A case in point is the results on *DMOZ10*. The main reason why *Centroid* obtains the worse classification accuracy on this corpus, we conjecture, is that the classification assumption of *Centroid* is strongly inconsistent with the characteristic of *DMOZ10*, within which each class contains a number of sub-clusters, i.e., child classes in the original data format. This may result in degradation of the performance of *CentroidDR* to some extent. However, due to the better classification ability of *KNN*, *CenKNN* could also obtain favorable classification accuracy on *DMOZ10*. It is worth noting that although classes within the corpus *Tan12* bear underlying sub-clusters, *CenKNN* and *Centroid* can perform pretty well on this corpus, and outperform *KNN* and its variants. This may be because, compared to the complex class hierarchy in *DMOZ10*, *Tan12* only contains few sub-clusters within each class, and this has less effect on the representation power of class centroids. Overall, *CenKNN* is also likely to achieve desirable classification accuracy on corpora with underlying sub-clusters, where *Centroid* may perform less effectively.

Searching for neighborhoods in a k-d tree is important to *CenKNN*'s classification efficiency. Finding K nearest neighbors in a k-d tree has time complexity $O(K \cdot \log N)$ providing that the size of the training dataset N is sufficiently large compared to data dimensionality, i.e., the number of classes l or the k value in the k-d tree. Otherwise, the computational complexity of the k-d tree search is expected to increase exponentially with data dimensionality (Bentley 1975). $N \gg 2^k$ was suggested to be a general rule to ensure k-d tree search efficiency (Bentley 1975). In such cases, the classification time of *CenKNN* is definitely dominated by the test document projection as $O\left(l \cdot Cen_{avg} \cdot Te_{voc}\right)$ (i.e., test document projection's time complexity) is far larger than $O(K \cdot \log N)$. Also, $O\left(l \cdot Cen_{avg} \cdot Te_{voc}\right)$ is far smaller than *KNN*'s classification time complexity $O(N \cdot Tr_{avg} \cdot Te_{voc})$. Our experiments show that *CenKNN* is at least two orders of magnitude faster than *KNN* when performing classification on large-scale corpora (e.g., 100,000 training documents) with no more than 10 classes. Our experiments also show that even when there are cases slightly breaking the general rule for the k-d tree search efficiency,*CenKNN* can still obtain very preferable classification efficiency. For example, our experiments also show that *CenKNN* can still run more than 50 times faster than *KNN* on corpora with 100,000 training documents and 20 classes. This is because although the k-d tree neighborhood search takes a bit of time in these contexts, it is still relatively small comparing to the test document projection time in *CenKNN* or document similarity calculation time in *KNN*. Therefore, *CenKNN* will obtain its typical classification efficiency when it is applied to corpora with large numbers (e.g., hundreds of thousands) of documents and a small number of classes (say no more than 20 classes). Such application scenarios are not uncommon in a wide range of domains such as large-scale spam detection, sentiment classification, news organization, social media user classification, web page organization and blog classification.

In comparison to *SVM* and its variant, *CenKNN* is more resilient to imbalanced corpora. *CenKNN* is preferable to *SVM* when dealing with text data with highly skewed class distributions, say, the skew ratio is larger than 100, and a small number of classes, as is the case on spam detection, news filtering and news event tracking. Also, the computational complexity of *CenKNN* is far smaller than *SVM*. *CenKNN* can run up to orders of magnitude faster than *SVM*. Note that *SVM* obtains the best classification accuracy on large balanced corpora (e.g., *DMOZ10*), though at cost in execution time. *CenSVM* can substantially reduce *SVM*'s execution time, but it still normally performs less efficiently than *CenKNN*, and sometimes is about four times slower. *CenSVM* normally has quite similar classification accuracy as *CenKNN*, but occasionally it is really poor. Comparing to *CenKNN*, *CenSVM* requires much more attention on stability or parameter tuning.

5 Conclusions and future work

We proposed the method *CenKNN*, which combined a class-centroid-based dimension reduction method (i.e., *CentroidDR*) with the k-d tree search method. In particular, by integrating the strengths of the classifiers *Centroid* and *KNN*,*CenKNN* becomes an efficient non-linear text classifier that possesses good robustness to imbalanced class distributions and noisy or irrelevant term features. In terms of efficiency, compared to the expensive computation time of *KNN* and *SVM*, *CenKNN* has linear computation complexity in terms of the number of training and test documents, which is comparable to *Centroid* and *Rocchio*, and so it can scale up well with data size. Although *CenKNN* is simple and works on a significantly lower-dimensional space, it is able to obtain substantially greater accuracy than *KNN* and its variants on a range of corpora, and perform substantially better than two scalable classification algorithms (i.e., *Centroid*, *Rocchio*). *SVM* is a better choice for large balanced corpora in terms of classification accuracy (recall that it is at the expense of longer execution time), while *CenKNN* is preferable to *SVM* when dealing with highly imbalanced corpora (e.g., the skew ratio is larger than 100) with a small number of classes in terms of both accuracy and efficiency. The situation changes when *SVM* works on centroid-based data space, but its stability or parameter setting requires further investigation. These results have been illustrated by a series of experiments conducted on a diverse range of corpora.

Throughout all our experiments, the only parameter K in *CenKNN* was 10 by default, yet *CenKNN* could obtain consistent superior performance on a variety of corpora. This indicates that *CenKNN* is able to eliminate the issue of choosing an appropriate value for K in *KNN* text classification.

We are interested in improving *CenKNN* to handle sub-clusters and/or a large number of classes, e.g., through the use of hierarchical classification and class hierarchy. Since *KNN* performs well in multi-label text classification tasks, we also plan to extend our work to multi-label text classification.

Acknowledgments We thank the anonymous reviewers whose constructive comments helped improve the paper substantially. We also wish to thank Dr. Alexander B. Zwart for his helpful comments on refining this paper. This paper's revision was partly conducted when Guansong Pang was a visiting student in the Web Sciences Center at the University of Electronic Science and Technology of China. He would like to

thank his supervisor Prof. Mingsheng Shang in the Web Sciences Center for the support on this work. This work was supported in part by the National Natural Science Foundation of China under Grant No. 61070061 and No.61202271, and by the National Social Science Foundation of China under Grant No. 13CGL130.

References

Achlioptas D (2003) Database-friendly random projections: Johnson–Lindenstrauss with binary coins. J Comput Syst Sci 66(4):671–687

Aggarwal CC, Zhai C (2012) A survey of text classification algorithms. Mining text data. Springer, New York

Batista GE, Prati RC, Monard MC (2004) A study of the behavior of several methods for balancing machine learning training data. ACM SIGKDD Explor Newslett 6(1):20–29

Bentley JL (1975) Multidimensional binary search trees used for associative searching. Commun ACM 18(9):509–517

Bingham E, Mannila H (2001) Random projection in dimensionality reduction: applications to image and text data. In: Proceedings of the seventh ACM SIGKDD international conference on knowledge discovery and data mining, pp. 245–250 (2001)

Chang C, Lin C (2011) LIBSVM: a library for support vector machines. ACM Trans Intell Syst Technol (TIST) 2(3):27

Chen Y, Hung Y, Yen T, Fuh C (2007) Fast and versatile algorithm for nearest neighbor search based on a lower bound tree. Pattern Recognit 40(2):360–375

Cunningham P, Delany SJ (2007) k-Nearest neighbour classifiers. Dublin: Technical Report UCD-CSI-2007-4

Du L, Buntine W, Jin H (2010) A segmented topic model based on the two-parameter Poisson–Dirichlet process. Mach Learn 81(1):5–19

Du L, Buntine W, Jin H, Chen C (2012) Sequential latent Dirichlet allocation. Knowl Inf Syst 31(3):475–503

Forman G (2003) An extensive empirical study of feature selection metrics for text classification. J Mach Learn Res 3:1289–1305

Guan H, Zhou J, Guo M (2009) A class-feature-centroid classifier for text categorization. In: Proceedings of the 18th international conference on World Wide Web, pp. 201–210 (2009)

Guo G, Wang H, Bell D, Bi Y, Greer K (2006) Using kNN model for automatic text categorization. Soft Comput 10(5):423–430

Han EH, Karypis G (2000) Centroid-based document classification: analysis and experimental results. In: Proceedings of the 4th European Conference on Principles of Data Mining and Knowledge Discovery, pp. 116–123 (2000)

Han E, Karypis G, Kumar V (2001) Text categorization using weight adjusted k-nearest neighbor classification. In: Proceedings of the 5th Pacific-Asia Conference on Knowledge Discovery and Data Mining, pp. 53–65

Han X, Li S, Shen Z (2012) A k-NN method for large scale hierarchical text classification at LSHTC3. In: Third Pascal Challenge on Large Scale Hierarchical Text classification (2012)

He H, Garcia EA (2009) Learning from imbalanced data. IEEE Trans Knowl Data Eng 21(9):1263–1284

Jagadish HV, Ooi BC, Tan K, Yu C, Zhang R (2005) iDistance: an adaptive B+-tree based indexing method for nearest neighbor search. ACM Trans Database Syst (TODS) 30(2):364–397

Jiang S, Pang G, Wu M, Kuang L (2012) An improved K-nearest-neighbor algorithm for text categorization. Expert Syst Appl 39(1):1503–1509

Joachims T (1996) A Probabilistic Analysis of the Rocchio Algorithm with TFIDF for Text Categorization. In: Proceedings of the 14th International Conference on Machine Learning, pp. 143–151 (1996)

Joachims T (1998) Text categorization with support vector machines: Learning with many relevant features. In: Proceedings of the 10th European Conference on Machine Learning, pp. 137–142

Joachims T (2001) A statistical learning learning model of text classification for support vector machines. In: Proceedings of the 24th annual international ACM SIGIR conference on research and development in information retrieval, pp. 128–136 (2001)

Katayama N, Satoh S (1997) The SR-tree: an index structure for high-dimensional nearest neighbor queries. ACM SIGMOD Rec 26:369–380

Kim H, Howland P, Park H (2005) Dimension reduction in text classification with support vector machines. J Mach Learn Res 6:37–53

Kosmopoulos A, Gaussier E, Paliouras G, Aseervatham S (2010) The ECIR 2010 large scale hierarchical classification workshop. ACM SIGIR Forum 44(1):23–32
Lam W, Han Y (2003) Automatic textual document categorization based on generalized instance sets and a metamodel. IEEE Trans Pattern Anal Mach Intell 25(5):628–633
Lan M, Tan CL, Su J, Lu Y (2009) Supervised and traditional term weighting methods for automatic text categorization. IEEE Trans Pattern Anal Mach Intell 31(4):721–735
Lin J, Gunopulos D (2003) Dimensionality reduction by random projection and latent semantic indexing. In: Proceedings of SDM'2003 Workshop on Text Mining Workshop (2003)
Liu T, Chen Z, Zhang B, Ma W, Wu G (2004) Improving text classification using local latent semantic indexing. In: Proceedings of the 4th IEEE International Conference on Data Mining, pp. 162–169 (2004)
Mani I, Zhang I (2003) kNN approach to unbalanced data distributions: a case study involving information extraction. In: Proceedings of ICML'2003 Workshop on Learning from Imbalanced Datasets (2003)
Manning CD, Raghavan P, Schütze H (2008) Introduction to information retrieval. Cambridge University Press, Cambridge
Miao Y., Qiu X. Hierarchical centroid-based classifier for large scale text classification. In: First Pascal Challenge on Large Scale Hierarchical Text classification (2009).
Moore AW, Hall T (1990) Efficient memory-based learning for robot control. Doctoral dissertation, University of Cambridge (1990)
Pang G, Jiang S (2013) A generalized cluster centroid based classifier for text categorization. Inf Process Manag 49(2):576–586
Pang G, Jiang S, Chen D (2013) A simple integration of social relationship and text data for identifying potential customers in microblogging. Advanced data mining and applications. Springer, Berlin
Papadimitriou CH, Tamaki H, Raghavan P, Vempala S (1998) Latent semantic indexing: a probabilistic analysis. In: Proceedings of the 17th ACM SIGACT-SIGMOD-SIGART symposium on principles of database systems, pp. 159–168 (1998)
Salton G, Wong A, Yang C (1975) A vector space model for automatic indexing. Commun ACM 18(11):613–620
Sebastiani F (2002) Machine learning in automated text categorization. ACM Comput Surv (CSUR) 34(1):1–47
Sun JT, Chen Z, Zeng HJ, Lu YC, Shi CY, Ma WY (2004) Supervised latent semantic indexing for document categorization. In: Proceedings of the 4th IEEE International Conference on Data Mining, pp. 535–538 (2004)
Sun Y, Wong AK, Kamel MS (2009) Classification of imbalanced data: a review. Int J Pattern Recog Artif Intell 23(4):687–719
Tan S (2005) Neighbor-weighted k-nearest neighbor for unbalanced text corpus. Expert Syst Appl 28(4):667–671
Tan S (2006) An effective refinement strategy for KNN text classifier. Expert Syst Appl 30(2):290–298
Tan S, Cheng X (2007) An effective approach to enhance centroid classifier for text categorization. In: Proceedings of the 11th European conference on Principles and Practice of Knowledge Discovery in Databases, pp. 581–588 (2007)
Tang L, Liu H (2005) Bias analysis in text classification for highly skewed data. In: Proceedings of the 5th IEEE International Conference on Data Mining, pp. 781–784 (2005)
Vilalta R, Achari M, Eick CF (2003) Class decomposition via clustering: a new framework for low-variance classifiers. In: Proceedings of the 3rd IEEE International Conference on Data Mining, pp. 673–676 (2003)
Wan CH, Lee LH, Rajkumar R, Isa D (2012) A hybrid text classification approach with low dependency on parameter by integrating K-nearest neighbor and support vector machine. Expert Syst Appl 39(15):11880–11888
Wang X, Zhao H, Lu B (2011) Enhance k-nearest neighbour algorithm for large-scale multi-labeled hierarchical classification. In: Second Pascal Challenge on Large Scale Hierarchical Text classification (2011)
Wang X, Zhao H, Lu B (2013) A Meta-Top-down Method for Large-scale Hierarchical Classification. IEEE Trans Knowl Data Eng, 99 (2013). doi:10.1109/TKDE.2013.30
Wettschereck D, Aha DW, Mohri T (1997) A review and empirical evaluation of feature weighting methods for a class of lazy learning algorithms. Artif Intell Rev 11(1–5):273–314
Wu X, Kumar V, Ross Quinlan J, Ghosh J, Yang Q, Motoda H, Mclachlan GJ, Ng A, Liu B, Yu PS (2008) Top 10 algorithms in data mining. Knowl Inf Syst 14(1):1–37

Yang Y (1994) Expert network: effective and efficient learning from human decisions in text categorization and retrieval. In: Proceedings of the 17th annual international ACM SIGIR conference on research and development in information retrieval, pp. 13–22 (1994)
Yang Y, Ault T, Pierce T, Lattimer CW (2000) Improving text categorization methods for event tracking. In: Proceedings of the 23rd annual international ACM SIGIR conference on research and development in information retrieval, pp. 65–72
Yang H, King I (2009) Sprinkled latent semantic indexing for text classification with background knowledge. Lect Notes Comput Sci 5507:53–60
Yang Y, Liu X (1999) A re-examination of text categorization methods. In: Proceedings of the 22nd annual international ACM SIGIR conference on research and development in information retrieval, pp. 42–49
Yang Y, Pedersen JO (1997) A comparative study on feature selection in text categorization. In: Proceedings of the 14th International Conference on Machine Learning, pp. 412–420
Zhang M, Zhou Z (2007) ML-KNN: a lazy learning approach to multi-label learning. Pattern Recognit 40(7):2038–2048