Intelligent Automation And Soft Computing, 2019

Vol.25,no.1,35–48

An improved k-nearest neighbor algorithm using tree structure and pruning technology

Juan Li

School of Distance Education, Shaanxi Normal University, 710062 Xi'an, China

ABSTRACT

K-Nearest Neighbor algorithm (*KNN*) is a simple and mature classification method. However there are susceptible factors influencing the classification performance, such as *k* value determination, the overlarge search space, unbalanced and multi-class patterns, etc. To deal with the above problems, a new classification algorithm that absorbs tree structure, tree pruning and adaptive *k* value method was proposed. The proposed algorithm can overcome the shortcoming of *KNN*, improve the performance of multi-class and unbalanced classification, reduce the scale of dataset maintaining the comparable classification accuracy. The simulations are conducted and the proposed algorithm is compared with several existing algorithms. The results indicate that the proposed algorithm can obtain higher classification efficiency and smaller search reference set on UCI datasets. Furthermore, the proposed algorithm can overcome the shortcoming of *KNN* and improve the performance of multi-class and unbalanced classification. This illustrates that the proposed algorithm is an expedient method in design nearest neighbour classifiers.

KEY WORDS: KNN; TPKNN; unbalanced class patterns; tree structure; tree pruning; penalty parameter

1 INTRODUCTION

CLASSIFICATION algorithms are often divided into supervised and non-supervised algorithm. Supervised classification of patterns into predefined categories is a very common task. Some popular algorithms that have successfully been applied to text classification, including probabilistic classification such as Naive Bayes (Mccallum & Nigam, 1998), decision tree classifiers, rule-based classifiers, and maximum margin classifiers like *SVM* (Platt, 2013). All these algorithms above need to build classifier models in advance. However, *K*-Nearest Neighbor (*KNN*) that was proposed by Aha, Kibler and Albert (1991), a type of instance-based learning, does not build a classifier in advance. *KNN* is the simplest of all machine learning algorithms: when a new pattern arrives, *KNN* finds the k nearest neighbors to the new pattern from the training space based on some suitable similarity or distance metric. The highest or plurality class label among the nearest neighbors is the class label for the new pattern.

However, *KNN* also has some inherent disadvantages (Han, Karypis, & Kumar, 2001). *KNN* does not settle with the multi-class classification especially unbalanced classification. When deal with each new unlabeled pattern, the entire training space must be scanned and the overlarge computing consumption is been obtained. In addition, *KNN* is susceptible to the impact of the boundary data or outlier. What's more, the determination of *K* value has not yet been solved satisfactorily.

There are several advanced *KNN* methods proposed in the recent literature to settle class unbalance problems. *CGW* (Liu & Chawla, 2011) uses the probability of attribute values given class labels to weight prototypes in *KNN*. Zeng, Yang and Zhao (2009) have utilized the distance weighted local learning in each class to obtain the pseudo nearest neighbor of the new unlabeled pattern and assigns the unlabeled pattern with these obtained pseudo nearest neighbors, reduces the pattern distribution sensitivity. In order to avoid the parameter influence of *KNN* and realize the non-parameter input running, Zeng, Yang, & Zhao (2009) have proposed the *LMS* algorithm by

CONTACT Juan Li ally_2004@126.com

utilizing the local mean vector and class statistics. In a study conducted by Milli & Bulut (2017), two novel setting approaches of *K* value were presented to increase the prediction accuracy of recommender systems based on *KNN*.

Moreover, in order to reduce the computational burden and the sensitivity of *KNN*, there are many effective search methods and many reduction technologies have been proposed.

On the one hand, several effective search methods can be obtained in some literatures. As a good search and construction algorithm, the *k-d* tree (Friedman, Bentley, & Finkel, 1977; Sample, Matthew, Mark, & Purcell, 2001) is a search structure based on nearest neighbor technique. *K*-d tree can increase the search efficiency, especially in *NN* searching application. The Voronoi networks was proposed by Krishna, Thathachar, & Ramakrishnan(2000), can reduce the memory requirements and the computation cost, then and has the comparable the generalization error with the Bayes classifier. In addition, ensemble idea has been widely adopted in classification problem. Franti, Virmajoki, & Hautamaki (2006) proposed a fast agglomerative clustering method using an approximate nearest neighbor graph for reducing the number of distance calculations and the method can maintain the classification accuracy close to that of the full search. Zhang & Srihari (2004) proposed a cluster-based tree algorithm to accelerate *KNN* classification without any presuppositions about the metric form and properties of a dissimilarity measure. Zou, Wang, Wei, Li, & Yang (2014) presented a generative probabilistic prediction ensemble framework including the missing date predict process and an enhanced similarity method.

On the other hand, reducing the scale of training set and removing those little contribution patterns are good at improving *KNN*'s runnability. As we known, *KNN* is sensitive to the distribution of training patterns. patterns that locate at different region have different classification contribution. patterns nearby class boundaries have higher contribution for pattern classification. On the contrary, most of patterns that contain the vast number of inner patterns and all of outlier patterns have little contribution for pattern classification. So if the little loss of classification accuracy can be guarantee, the training set can be removed these little classification contribution patterns. And then the smaller scale of higher contribution patterns can lessen the computation and storage consumption due to the smaller search space. To effectively larger scale of training set, the partition technologies must be adopted. So how to obtain the boundary area and design the partitioning strategies have been studied in some literatures (Barandela, Ferri, & Sánchez, 2005; Khazaee & Ebrahimzadeh, 2013; Lumini & Nanni, 2006; Mollineda, Ferri, & Vidal, 2002; Olvera-López, Carrasco-Ochoa, & Martínez-Trinidad, 2010). Khazaee & Ebrahimzadeh (2013) adopted *SVM* in the feature partitioning strategies to achieve the classification performance of the special problem. Cluster-based learning (*CBL*) algorithms are proposed in above literatures, in which patterns are not only patterns per se, but also the weighted averages of patterns. In particular, following the same scheme, Mollineda, Ferri, & Vidal (2002) presented the Generalized-Modified Chang algorithm, which merges the same-class nearest clusters and selects the centers from the new merged clusters as prototypes. In a study conducted by Lumini & Nanni (2006), after splitting the training set into c clusters, the selected prototype patterns are the centers of the c clusters of *CBL*. Based on the clustering idea, a new improved clustering algorithm that names as Prototype Selection by Clustering (*PSC*) (Olvera-López, Carrasco-Ochoa, & Martínez-Trinidad, 2010) selects border patterns and some interior patterns. In recently, the k-means clustering algorithm, the fuzzy c-means algorithm, and some hierarchical clustering algorithms are widely used in some pattern reduction application. Divide-and-conquer approach has attracted many attentions in saving the runtime consumption. Raicharoen, Lursinsap, & Lin (2005) proposed an algorithm which can realize the prototypes' correct selection insensitive to the scan sequence of patterns by building some separating hyper-planes located among the POC-NN essential patterns. Haro-García & García-Pedrajas (2009) divided the original training set into some smaller subsets where the pattern selection algorithm is applied. Fayed & Atiya (2009) proposed a pattern reduction algorithm, namely, the template reduction for *KNN*(TR*KNN*). Rico-Juan & Iñesta (2012) proposed new rank methods to select the best prototypes from a training set based on the relevance factor that is from 0 to 1 and is used to select the best candidates for each class.

As we known, there are three main independent methods to decide the value of *K*, but any of them exist some drawbacks (Gong & Liu, 2011). So the fixed *K* value ignores the influence of the category and the pattern number of training set and dynamic obtaining the value of *K* (Li, Qin, & Yu, 2004; Wang, Neskovic, & Cooper, 2006) is presented.

Although a lot of algorithms that have been introduced above are adopted to improve the classification performance and can cope with some classification problem well, but these algorithms only focus on one specific of the *KNN*'s disadvantage and do not consider all of *KNN*'s disadvantages as a whole.

In this paper, a new classification algorithm that absorbs tree structure, tree pruning and adaptive k value method was proposed, taking account of the above challenges. The authors integrate the idea of tree technology into *KNN* and construct a new Tree-Pruning-*KNN* model (*TPKNN*) that differs with B+-tree (Jagadish, Ooi, Tan, Yu, & Zhang, 2005), P-tree (Li, Shi, Charastrakul, & Zhou, 2009) and R-

tree(Zheng, Lee, & Lee, 2003) which is a simple classification algorithm that uses tree structure and search region partitioning to decrease the search space, improves search efficiency, keeps the dominance and obtains a superiority of *KNN* over the unbalanced pattern number category of *KNN*. In order to reduce the bad influence of boundary and noise patterns, the average distance among classes is obtained by computing the distances among tree nodes. Based on the average distance, the class trees are split and the nodes with little classification contribution are discard, and the boundary patterns are remained. *TPKNN*, an incremental learning algorithm, avoids selecting the uncertain *K* value through calculating the maximum classification efficiency. *TPKNN* does well in paralleling because it builds the tree classifier in advance for each class.

To be specific, the main contributions in this paper are as follows:

1) The dynamic acquired method of K can adapt the specific problem and cope with the disadvantages of the traditional parameter settings of *KNN*;

2) The fast tree building strategy is introduced. Using the strategy, the building operation can quickly build a tree structure for each class meanwhile maintaining the distribution of training patterns without the pattern scan sequence influence;

3) Several tree treatment operations can reduce the search space and obtain the effective reference patterns meanwhile maintaining the distribution of original training patterns;

4) The proposed algorithm can cope with the unbalanced classification problem well by adopting the adjusted parameters;

5) The proposed algorithm can reduce the influence of noise and can be used as a preprocessor of some hybrid algorithms.

This paper is organized as follows. Section 2 and 3 describe and analyze the proposed algorithm respectively in detail. Experimental parameters and experimental results are presented in Section 4. Finally, Section 5 concludes with future work.

2 TPKNN ALGORITHM

THE main contribution of *TPKNN* is as follows: first, replace a small core set for all training search region; second, keep the distribution and the numbers of the training patterns; third, solve the indivisible patterns by taking account of the relations between unbalance and other specific states.

In this section, the proposed algorithm will be presented intuitively in order to understand how to build an efficient tree structure to reduce the search space and quickly find many nearest distance patterns that can be used to compute the score of each class and select the class label with the highest score. The proposed algorithm consists of the following steps that just contains some key functional blocks:

1. Building tree structure: build tree structure by single scan of the training set. *TPKNN* uses space partitioning to part the whole training space into several small core sets of data in high data density region. The search space is divided into many core sets. A core set can be seen as a tree node, so the search space can be represented as a tree structure. A tree node contains the pattern number, max distance (equal to radius), within distance (abbreviated as *WD*), between distance (abbreviated as *BD*), a representative pattern (abbreviated as *Rep*), a stability and a pattern-index at least. The mean pattern of a core set is the *Rep* of a tree node.

2. Adjusting and incremental updating tree structure: adjust a tree node when every training pattern is absorbed to a tree node and update tree structure when a new core set occurs and a new node is inserted into the tree. The main adjustment and updating benchmark is the Euclidean distance. In addition, the adjusting process considers the stability as a significant and useful judgment condition. The adjusting and updating strategy will be introduced in the follow section.

3. Pruning tree: as is well known, the more different location among different class label patterns, the more different distance and the more different contribution with them. Then these patterns far from the class boundaries are useless than those near them. Therefore, how to find the border and interior core sets is the concerned objective. To solve this problem, these class distances between the different class patterns are calculated by computing the distances among the different class labels trees and the distances within the same label patterns. The two ways are adopted to reduce the size of training patterns. First, for each class tree, the core sets whose distances to the tree root are smaller than the within average distance are removed. Second, for each different class tree pair, the core sets whose distances to the tree root are larger than the average distance between two classes are removed. Then, by merging the remaining core sets, the new patterns are obtained.

4. Classifying: compute each pre-class scores of k closest patterns of training patterns and determine the class label whose training pattern has the highest class score. In this step, the penalty parameters are used to improve the low classification accuracy caused by unbalanced patterns and boundary patterns and the k dynamic value is set in order to reduce k's influence.

Now the main steps can be shown in detail.

2.1 Generation tree structure

The tree structure of *TPKNN* is an approximately balanced tree center on the geometric center of the training patterns. There are three kinds of tree nodes: the root, non-leaf nodes and leaf nodes. In this paper, three kinds of nodes have the same function and property characteristics. Nodes are representative

patterns for any high density regions that is called as core set whether noise or boundary patterns.

The goal of *TPKNN* is to categorize patterns based on the maximum similarity by which the training patterns are parted into different core set and the size of search space is reduced. Thus the nodes of the tree are the mean patterns in every core set and those patterns in the same core set should be closed to each other by the Euclidean distance. When a new training pattern arrives, *TPKNN* decides which core set it belongs to or whether it is a new tree node by comparing the distances between the new pattern and the tree nodes.

The trees of *TPKNN* are obtained by one pass scan over labeled training patterns. Different scan sequences or different patterns distribution turn out to be different tree structures on the same testing patterns. Building a compact and rational tree structure is our goal. The accuracy of *TPKNN* depends on the initialized distance radius which determines the number of core nodes sets. The tree nodes are these core sets that belongs to different regions with the closest Euclidean distance. So the different region range will decide the tree structure and search efficiency.

In order to weaken the above influences, the appropriate distance radius through calculating the distance relations with some random patterns are worth estimating. In this paper, there are two basic premises. One is that a pattern belongs to only one class. The other is that one tree is built for each class. Now the following is how to build an effective tree for all patterns with the same label.

Given a set of labeled training patterns TR and C classes, let $|TR|$ be the number of training patterns, $|C|$ be the number of classes, b_j be the numbers of patterns in class c_j, T_j be the tree structure of class c_j, $|T_j|$ be the node number of T_j, r_j be the root of T_j, c_{jk} be a core set k of a class c_j, $|c_j|$ be the number of a core set k of a class c_j, and n_{jk} be the node of core set k of a class c_j. Let sq be the queue for storage core sets that include a new test pattern ts. The number of working patterns is obtained in c_j by $b_j = \sum_{i=1}^{k} |c_{jk}|$ and the number of working training patterns by $|TR| = \sum_{j=1}^{|C|} b_j$.

Let n_{jk} denote the mean pattern of c_{jk}:

$$n_{jk} = \frac{1}{|c_{jk}|} \sum_{i=1}^{|c_{jk}|} x_{ji} \tag{1}$$

The Euclidean distance function $D(\bullet,\bullet)$ is used to compute the distance between two patterns within search process and decide the detail search core set. The nearest neighbor function $NN(x,c)$ indicates the nearest neighbor from the class c for a pattern x.

Since the cosine similarity is utilized to evaluate the similarity of patterns in the paper, the notation $S(\bullet,\bullet)$ indicates the similarity function of two patterns. $S(j)$ is the average similarity between and the nodes of T_j. The function S_{avg} denotes the average similarity of the whole training patterns.

$$S(x, y) = \cos(x, y) \tag{2}$$

$$S(j) = \frac{\sum_{k,i=1}^{|T_j|} S(n_{ji}, n_{jk})}{|T_j|}, i \neq k \tag{3}$$

$$S_{avg} = \frac{\sum_{j=1}^{|C|} (1 - \frac{b_j}{|TR|}) S(j)}{\sum_{j=1}^{|C|} |T_j|} \tag{4}$$

Taking different classes with different number of texts into account, this paper uses the number of various types of training sets and the average similarity to amend the selection of K value. In this way, the negative interference of unbalanced classes can be reduced. Using $(1 - \frac{b_j}{|TR|})$ to fix the j type's similarity values, the inaccurate classification results with too large or too small class patterns are eliminated. Formula (4) can be seen as the average similarity of the training patterns because the nodes layout of tree is approximated to the pattern distribution.

Taking S_{avg} as a standard, the similarities between ts and n_{jk} are divided into two parts. One part is greater than S_{avg}, the other is not. The large one is reserved, its number is calculated and assigned to K. Therefore, the dynamic value of K is obtained.

Algorithm1: Building tree structure

1. Randomly scan several labeled patterns and estimate the distribution of c_j. Get the mean pattern r_j of those random patterns and the standard

deviation ε_{j0} of all distances between any of the two patterns. Let ε_{j0} be the initial stability parameter for all core sets of c_j.

2. Compute the distances between r_j and the random patterns. Get D_{max} and D_{min}. Let γ_j be the initial radius for all care sets of c_j.

$$\gamma_j = \begin{cases} D_{min} & D_{max} \geq 2 * D_{min} \\ \dfrac{D_{min}}{2} & D_{max} < 2 * D_{min} \end{cases}$$

3. Set $k = 0$, $n_{jk} = r_j$, and $\gamma_{jk} = \gamma_j$.

4. If the training set is not Φ, read a new labeled pattern s.

5. For each n_{jk}, calculate $D(s, n_{jk})$. Then there are two stations:

1) $s \in c_{jm}$, if $\exists k$ and $(D(s, n_{jk}) - \gamma_{jk}) < 0$ and $m = arg\min_k (D(s, n_{jk}) - \gamma_{jk})$. Go to Step 6.

2) For each n_{jk}, S is a new core set where $(D(s, n_{jk}) - \gamma_{jk}) > 0$. Then $c_{j(k+1)} = s$ and $c_{j(k+1)}$ is inserted into the tree as the child of n_{jm} where $m = arg\min_k D(s, n_{jk})$. Go to Step 7.

6. Adjust n_{jk} when a new pattern is absorbed. Then go to Step 8.

7. Let $k = k + 1$. Update T_j by the updating decision rule.

8. If the training set is Φ, then stop the algorithm. Otherwise, go to Step 4.

The tree takes into account any noise and isolated patterns. It does not discard any patterns and can maintain the original state.

2.2 Adjusting core set

When s is absorbed into c_{jk}, the new mean pattern n'_{jk}, the new density center γ'_{jk}, and the new stability parameter ε'_{jk} of c_{jk} are obtained. Considering the above changes, the core set will be replaced by the new c_{jk} and ε_{jk} if and only if it gets smaller ε'_{jk} in n'_{jk}.

The adjusting process is elaborated as follows:

1. Obtain the new n'_{jk} of c_{jk}.

2. Calculate the new ε'_{jk}, and get the new $\gamma'_{jk} = maxD(n'_{jk}, y), y \in c_{jk}$. Calculate the new pattern density of c_{jk} with γ'_{jk}.

3. New n_{jk} will be adjusted if the new density is larger than before and $\varepsilon'_{jk} < \varepsilon_{jk}$. Set $\varepsilon_{jk} = \varepsilon'_{jk}$, $n_{jk} = n'_{jk}$, and $\gamma_{jk} = \gamma'_{jk}$.

In this procedure, the stability parameter and the pattern density are obtained and used as two discriminate principles that are used to determine whether to adjust the tree node and some related information.

2.3 Inserting a new core set into a tree

When $s \notin c_{jm}, \forall c_{jm} \in T_j$, s is inserted into T_j as a new core set $c_{j(|T_j|+1)}$. Before the inserting process, $c_{j(|T_j|+1)}$ must be initialized. Set the stability parameter as $\varepsilon_{j(|T_j|+1)} = \dfrac{\sum_{m=1}^{|T_j|} \varepsilon_{jm}}{|T_j|}$, the radius $\gamma_{j(|T_j|+1)} = min\gamma_{jm}, m = 1,...,|T_j|$, the new mean pattern $n_{j(|T_j|+1)} = s$, etc..

The inserting process is elaborated as follows:

1. Initialize the above two parameters for the new core set $c_{j(|T_j|+1)}$.

2. Find $m = arg\min_k (D(s, n_{jk}) - \gamma_{jk})$, $k = 1,...,|T_j|$.

3. Set c_{jm} as the parent of $c_{j(|T_j|+1)}$.

4. Set $|T_j| = |T_j| + 1$.

5. Update T_j by the incremental updating decision rule.

In this procedure, first, these parameters are initialized for a new core set formed from s by the above mentioned dynamic adaptive strategy, which can reduce the influence caused by the initial class tree established based on the selection of random patterns and avoid the pre-selected parameter that helps to obtain the static tree structure. The predefined parameter and the static structure do not reflect the pattern distribution and can affect the classification results easily.

2.4 Incremental updating tree structure

Retaining the original pattern distribution is the goal of the paper. The shorter search path and higher search speed can be obtained when the tree layout is similar to the pattern distribution. In this paper, the patterns distribution is reflected by the layout of nodes.

The patterns center drifts when a new core set is inserted into the tree. The tree structure is updated dynamically in order to gradually obtain the geometrical center. When the center of a tree is drift,

the updating process must be invoked. In this paper, the updating process is shown as follows.

1. Get the mean pattern s_m of T_j's nodes n_{jk}.
2. Find $m = arg\min_k(D(s_m, n_{jk}) - \gamma_{jk})$.
3. Set n_{jm} as new r_j of T_j.
4. Insert other nodes into T_j by the minimum distance principle. A node n will be inserted into T_j as a child of one node whose distance is the minimum distance between n and the current nodes of T_j.

In the process, it is ensured that the tree nodes layout is close to the pattern distribution through updating the tree center step by step. Finally, the tree center that is close to the geometrical center of training patterns is obtained. The built tree structure can retain the distribution of original training patterns well. Due to the minimum distance cost tree, the search path is shorter when the class label of the new test pattern is found.

2.5 Pruning tree structure

In the above introduction, our goal is getting the subset of the training set through these within and between distances. How to obtain these distances and use them?

First, for each class, the within distance can be calculated by computing all distances among these non-root nodes $n_{jk}, k = 1,...,|T_j|$ with r_j for T_j.

$$D(c_j) = e^{-\beta}\frac{\sum_{j=1}^{|T_j|} D(c_{jk}, r_j)}{|T_j| - 1} \tag{5}$$

Taking $D(c_j)$ as a standard, the nodes which are computed between $n_{jk}, k \neq 0$ and r_j are divided into two parts. One part is greater than $D(c_j)$, the other is not. The smaller part is near by the pattern center and is useless than the other one. Thus the greater part is reserved and the smaller one is removed. But when the special ring distribution of the training set occurs, the pruning step slow down or stop at pruning speed according to the relation of the maximum and minimum distances. To solve this problem, the adjusting parameter $\beta(\beta > 0)$ is used to speed up or slow down pruning ratio. When $1 > \beta > 0$, a high within pruning ratio is got and the pruning operation is accelerated. When $\beta \geq 1$, a low within pruning ratio is got and the pruning operation is slow down.

Second, the distances between each different label class-pairs are obtained by Formula (6).

$$D_{ij} = e^{1-\frac{b_i}{b_j}}\frac{\sum_{k=1}^{|T_i|} D(n_{ik}, nn(n_{ik}, c_j))}{|T_i|}, i \neq j, i, j = 1,...,|C| \tag{6}$$

These core sets whose distance is larger than D_{ij} from T_i are removed and the others are retained. It ensures to remove these core sets that belong to the same class. It would remove a lot of core sets that are useless for two current classes and useful for other categories. However, they can be retained by other remove operations. The parameter $e^{1-\frac{b_i}{b_j}}$ has the same rule as β. For major class c_i and minor class c_j, $\frac{b_i}{b_j} > 1$ and $e^{1-\frac{b_i}{b_j}} < 1$, the between pruning operation is speeded up. When minor class c_i and major class c_j, $\frac{b_i}{b_j} < 1$ and $e^{1-\frac{b_i}{b_j}} > 1$, the between pruning operation is slow down and more patterns that belong to the minor class are kept. Therefore, in the next step, the results for every class are merged and the new tree structure is got.

Finally, the retained core sets are merged for each class tree. Then the patterns size is reduced and the boundary character is maintained and strengthened. The new tree structure is used as a *TPKNN* classifier. In this paper, the pruning process is shown as follows.

1. Get the within distances of every $T_j, j = 1,...,|C|$.
2. For each class, Find these nearest neighbors from other different classes.
3. Obtain the between distance matrix $D_{|C|\times|C|}$ by Formula (6).
4. Remove these core sets with larger distance by the above strategy.
5. Merge the core sets for each class.
6. Obtain new working b_j, T_j.

In the process, the running result can reduce the size of working reference patterns to decrease the search time complexity and strength the boundary factor and keep the approximate accuracy as used for the original patterns. Of course, the pruning step can reduce each class size whether major or minor classes. To reduce the excessive pruning about the minor class, these classes ratios are fully taken into account. When the minor class tree was pruned, *TPKNN* uses $e^{1-\frac{b_i}{b_j}}$ to expand or reduce the pruning magnitude. But *TPKNN* only obtains the size reduction using a simple intuitive method and approximate class boundaries.

2.6 Using TPKNN classifier

Test patterns are classified by computing a similarity score for each class and selecting the class with the highest score. Pre-class similarity scores are computed by a similarity function with the information of the K nearest neighbors of the test patterns in a class.

Algorithm2: Using the TPKNN model

Input: test pattern ts, T_j of C_j

1. Calculate and store $D(n_{jk}, r_j)$ for each n_{jk} in an increasing order.
2. Compute S_{avg}.
3. For $j = 1,2,...,|C|$ do
4. For $k = 1,2,...,|T_j|$ do
5. Find the right core set c_{jk} with $\min_k D(n_{jk}, ts)$.
6. Calculate $S(ts, n_{jk})$.
7. Get the number of those core sets whose $S(ts, n_{jk}) > S_{avg}$.
8. End for
9. Insert c_{jk} into sq.
10. End for
11. Set K=the total number of those core sets whose $S(ts, n_{jk}) > S_{avg}$, $j = 1,2,...,|C|$, $k = 1,2,...,|T_j|$.
12. Find the set nnk of the K nearest patterns in sq.
13. For $j = 1,2,...,|C|$ do
14. $s_j = \sum_{x \in nnk} S(ts, x)$
15. End for
16. Get k_j of the patterns belonging to c_j in nnk.
17. Return c_m, where $m = arg\min_j (\frac{k_j}{|b_j|})^{\alpha} S_j$.

In step 1-2, these distances will be used to determine the appropriate core set of a test pattern. S_{avg} will be used to set the dynamic value K. In step 17, by using $(\frac{k_j}{|b_j|})^{\alpha}$ ($0 \le \alpha \le 1$) to fix s_j similarity values, *TPKNN* can eliminate the classification results which are not accurate because the number of class patterns is too large or too small. The contribution to the class score limited to a very small positive value, rather to a lager negative score.

3 ALGORITHM ANALYSES

In this section, the computational complexity of *TPKNN* and the time complexities of *TPKNN* and k-d strategy are discussed.

3.1 Time-memory complexity analysis

Classical *KNN* requires no training at all. *KNN* takes $\theta(1)$ training time consumption if K value is predefined. The test complexity of *KNN* is very large because the whole training patterns are searched a time when each unlabeled pattern is classified. In practice, some preprocessing steps, such as the step of predefined K value, preprocessing documents, and partition strategy, etc., have to be performed. It makes more sense to preprocess training documents once as part of the training phase rather than repeatedly classifying a new test pattern.

In this paper, the partition strategy and dynamic K value are adopted to reduce the test complexity of *KNN*. Due to the tree structure building in advance, the K nearest patterns can be easy obtained by searching the high-density region with higher similarity, rather than searching the whole training space. So one of the main consumptions is the class tree structure building process. Building tree consists of single scan, core adjusting, and tree updating. The single scan takes $\theta(N)$ complexity. During this pass, values b_j, ε_{jk}, n_{jk}, $|c_{jk}|$ are stored and T_j is computed several times, requiring $\theta(N)$ storage.

For each training pattern, the adjusting process will be invoked only once. When a new training pattern is absorbed, the process needs to compare with other patterns except for itself in a core set. Therefore, a class c_j takes $\theta(d_j^2)$ time and $\theta(d_j)$ storage space, so this process takes $\theta(N)$ time and $\theta(N)$ space.

The updating process only refers to resetting these tree nodes relationship when a new core set is obtained. Then, it takes $\theta((\sum_{i=1}^{|C|}|T_j|)^2)$. Therefore, the process takes computing complexity $\theta(N)$.

The Pruning process obtains *WD* and *BD*. For *WD*, it takes $\theta(|T_j|)$ for one class. So it takes $\theta(\sum_{j=1}^{|C|}|T_j|)$ for all classes. For *BD*, it takes $\theta(|T_j| \times \sum_{i=1, i\neq j}^{|C|}|T_i|)$ for one class, so it takes $\theta(\sum_{j=1}^{|C|}\theta(|T_j| \times \sum_{i=1, i\neq j}^{|C|}|T_i|)) = \theta((\sum_{j=1}^{|C|}|T_j|)^2)$ for all classes.

To sum up, the training process takes computing complexity $\theta((\sum_{j=1}^{|C|}|T_j|)^2)$.

The test time complexity of a new unlabeled pattern is linear complexity with the average number of core sets. Although test time complexity

takes $\theta(\frac{N}{\sum_{i=1}^{|C|}|T_j|}+\sum_{i=1}^{|C|}|T_j|)$, T_j is a new tree that removes some central and noise core sets after pruning process and N is the number of new working search space.

Although *TPKNN* is complicated, which involves multiple passes on building the tree and takes more time than traditional training *KNN*, it gets the faster search speed than *KNN*. Meanwhile, it is clear that *TPKNN* is a approximate linear time algorithm based on above analysis. As we known, many incremental classification algorithms, such as *CNN* and *ILVQ* (Xu, Shen, & Zhao, 2012), etc., all have approximate linear complexity. However, those nonincremental classification algorithms all need scan the whole training dataset several times and have larger nonlinear complexities than that of these incremental algorithms include *TPKNN*, *ILVQ* and *CNN*. In the following section, the experimental results show that *TPKNN* effectively reduces the size of training set while maintaining the same level of classification accuracy as *KNN*.

3.2 Discussion in TPKNN and k-d strategy

As mentioned above, *TPKNN* can partition the search space and quickly find the *K*-nearest neighbors. Although *TPKNN* can reduce the search space, *TPKNN* causes the additional preprocessing memory-time complexity, especially to the large size training sets, due to building these class trees for each class. A *k-d* tree is a space-partitioning data structure for organizing points in a k-dimensional space and fast obtain the appropriate search region that is close to the special pattern.

A *k-d* tree is adopted in *KNN* to get the *K*-nearest neighbors quickly through space dimension partition. The *k-d* tree strategy does not need to build class tree in advance or reduce the pattern size. A *k-d* strategy does not need extra space cost to build tree and does not need extra time cost to adjust tree structure. Although *TPKNN* has the same building step with the *k-d* tree strategy, it has its particular pruning step and searching step.

Two strategies can be discussed according to the size of training patterns. For small or middle training sets, *TPKNN* can achieve building and pruning step fast and has the approximate the same execute speed. For large training sets, *TPKNN* needs more execution time to build tree structure because it needs to adjust tree structure incrementally. But *TPKNN* is faster over k-d tree strategy due to *TPKNN* can reduce the size of reference training set by it pruning step. So it is worthless and unfair to compare the whole execute time between *TPKNN* and *k-d* strategy. In the future, how to improve the performance of the proposed algorithm, in especial the building step of the proposed algorithm is our next work.

4 EXPERIMENTAL EVALUATION

In Section 3, *TPKNN* has a lower complexity than *KNN*, its testing time complexity is superior to *KNN* meanwhile. To verify the effectiveness of the proposed *TPKNN* approach, experiments are performed on several datasets from the UCI repository of machine learning databases (http://archive.ics.uci.edu/ml/). From the repository, the following balanced benchmark data sets are selected in Table 1 and the other unbalanced data sets are shown in Table 5. 20% patterns are selected randomly as independent and additional testing patterns in advance and take 5-fold cross-validation for other patterns in order to obtain the average performances of *KNN* and *TPKNN*.

Table 1. Characteristics of selected balanced datasets

DataSet	Number	Classes	Features
ris	150	3	4
Wine	178	3	6
Mobile	205	3	26
WaveForm	5000	3	21
Spam	4601	2	57
Letter	20000	26	16
Statlog	6435	7	33

Table 2. Characteristics of selected unbalanced datasets

DataSet	Number	Classes	Features	MinClass
Balance	625	3	4	7.84%
Cmc	1473	3	9	22.61%
WallFollow	5456	4	23	6.01%
Heart	270	2	13	44.44%
Hepatitis	155	2	19	20.65%
Ionosphere	351	2	34	35.9%
Pima	768	2	8	34.9%

For unbalanced data sets, when the minor class is important class, the performance metrics that is from the information retrieval community is adopted. They are based on a confusion matrix (see Table 3) that *TP* is the number of true positives and *TN* is the number of true negatives and *FP* is the number of false positives and *FN* is the number of false negatives and $F_\beta - measure$ (Köknar-Tezel & Latecki, 2011). These numbers are used to define the matrix that evaluates the generalized performance whether balanced or unbalanced, such as recall, precision, and $F_\beta - measure$. The formulas for these metrics are given below in Table 3.

Table 3. Confusion matrix

	Predicted positive	Predicted negative
Actual positive	*TP*	*FN*
Actual negative	*FP*	*TN*

$$Recall = \frac{TP}{TP + FN}$$

$$Precision = \frac{TP}{TP + FP}$$

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN}$$

$$F_\beta = (1 + \beta^2)\frac{Precision \times Recall}{\beta^2 \times Precision + Recall}$$

There are much useful information that can be obtained from these above formulas. The precision is the ability to identify for positive patterns and it is a non-negative real number that is less than 1. In general, a classification algorithm with the higher precision has the higher recognition ability. The recall is the number of *TP*s divided by the number of examples that are actually positive. There is always a trade-off between precision and recall, but for datasets where the cost of false negatives is high, a high recall value is preferable. The $F_\beta - measure$ reflects the important factor among $Precision$ and $Recall$.

In our experiments, these performances of *TPKNN* with *KNN* are compared. In order to present the experimental results, $Precision$, $Recall$, $Accuracy$ and $F_\beta - measure$ to are utilized to evaluate the performance measures. In this paper, $Recall$ has similar cost to $Precision$ and set $\beta = 1$. In this paper, the value of K is predefined to the classical *KNN*. In Tables 4, 5 and 6, *KNN*'s experimental results are the average values when *K*=3, 5 and 7. *TPKNN* has a dynamic *K* value. Tables 5 and 6 show the comparison of $Precision$, $Recall$, $Accuracy$ and F_1 between *TPKNN* and *KNN*.

In addition, in order to show the reduction ability of *TPKNN*, some prototype selection algorithms, such as *CNN*, *ENN*, *PSC* and *ILVQ*, are selected to compare against with it. To *ENN*, the parameter *K* is set by 3, 5, and 7. To *PSC*, the parameter *c* is set by $6m$ and $8m$ (m is the number of class labels in the training dataset). To *ILVQ*, the parameters λ and *AgeOld* are set by $\lfloor \sqrt{n} \rfloor$ (n is the number of patterns in the training dataset).

4.1 Experiment in the pattern reduce

Through the above pruning criteria, the pattern size is reduced, which could be useful for reducing the runtimes of the classification process, little significantly reducing classification accuracy when using origin patterns. In order to verify the efficiency about the reduce criteria, the reducing validity and the real working for classify in the reduced set are shown through several special experiments. The results are shown in Table 4.

Table 4. The reduced results for *TPKNN*

DataSet	original scale	node's number	reduced scale
ris	150	35	92
Wine	178	43	118
Mobile	205	56	140
WaveForm	5000	825	3725
Spam	4601	714	3509
Letter	20000	2764	15527
Statlog	6435	1023	4368
Balance	625	87	457
Cmc	1473	257	931
WallFollow	5456	845	4019
Heart	270	59	187
Hepatitis	155	41	96
Ionosphere	351	65	211
Pima	768	151	435

In Table 4, it can be obtained that the average number of the tree nodes in building the class tree before the pruning step and the average reduced results after the pruning step. The performances of the proposed building and pruning strategies are obtained. *TPKNN* increases the additional space requirements for building these class trees. For small DataSets, like Iris, Wine, Mobile, .etc., the tree needs more tree nodes than the middle and large datasets that include WaveFrom, Spam, Letter, WallFollow, .etc. *TPKNN* yields about 50% reducing ratio for the above original training sets.

From Figure 1, the following conclusions can be obtained. To *CNN*, its inherent disadvantage is the sensitivity of the random pattern sequence. So the disadvantage influent its results that are sensitive and unstable. To *PSC*, although the different cluster number can get the different reduction ratio, its results are relative stable and better than *CNN* in 10 over 12. To *ILVQ*, although it has the better fast incremental processing performance and the superior average compression ratio than the other compared algorithms, its reduction results are relative unstable and worse than *PSC* and *TPKNN*. To the small scale of datasets, *TPKNN* needs more tree nodes to maintain the distribution of dataset, so it has the unremarkable reduction ratio than *CNN* and *PSC*. To the middle and large scale of datasets, the pruning technology can delete the majority patterns and reduce the scale of dataset remarkably. It can obtain the better reduction radios than *PSC* in Letter, WaveForm and Spam. So *TPKNN* can be deemed to have the comparable reduction ability than *CNN* and *PSC* and is an effective compression algorithm.

4.2 Experiment in balanced sets

TPKNN is a classification algorithm for balanced and unbalanced pattern set. Although many strategies that are effective in unbalanced pattern set are adopted by *TPKNN*, they lead to little influence on balanced pattern set. In our paper, Formula (4) and Formula (6) is less influence for balanced class classification due

to the approximate same size for every class patterns whether original or pruning scale. Due to the fact that each class has the approximate same size, $\frac{b_j}{|S|}$ and $e^{1-\frac{b_j}{b_j}}$ can obtain the approximate same influence on class similarity measure and the pruning threshold to each class set.

From Table 5, *TPKNN* can get better classification results than *KNN* except Iris and Spam. Therefore, it is an obvious conclusion that *TPKNN* has better classification efficiency and can be well applied in these selected balanced datasets. Furthermore, we compare the classification accuracy results between *TPKNN* and three compared algorithms in these selected balanced datasets. *TPKNN* performs the best on five of the seven datasets, and on the remaining two datasets, has no significant advantage over other compared algorithms.

4.3 Experiment in unbalanced sets

TPKNN can improve the accuracy in unbalance class classification with small number of features. Therefore, the experiment results indicate that *TPKNN* can effectively avoid the impact of the size of the training pattern set and the testing pattern set and has high classification efficiency in these above datasets (see Table 2).

In this paper, two amendatory penalty parameter are adopted in order to improve the classification recognition ability to the unbalanced situation. First, $e^{1-\frac{b_i}{b_j}}$ can prevent more large-scale reduce (worse-case) that cause the less ratio in the working patterns for minor class. Second, $\frac{b_j}{|S|}$ is adopted to strength the minor class's influence on S_{avg} and $(\frac{k_j}{b_j})^{\alpha}$ is used to get each class's similarity that takes into account not only the k nearest neighbors, but also the global and local pattern distribution whether major or minor class.

In Table **6**, the results indicate that *TPKNN* can effectively avoid the impact of the size of unbalanced class training set and is effective to use the working global and local class pattern size. It has high classification efficiency in applying to the above datasets. Based on Figure **3**, the analysis conclusion is similar to that of Figure **2**.

On the whole, based on Table 4-6 and Figure 1-3, It is obvious that *TPKNN* is a good algorithm and compared *PSC* with *ILVQ* and on classification accuracy and pattern reduction.

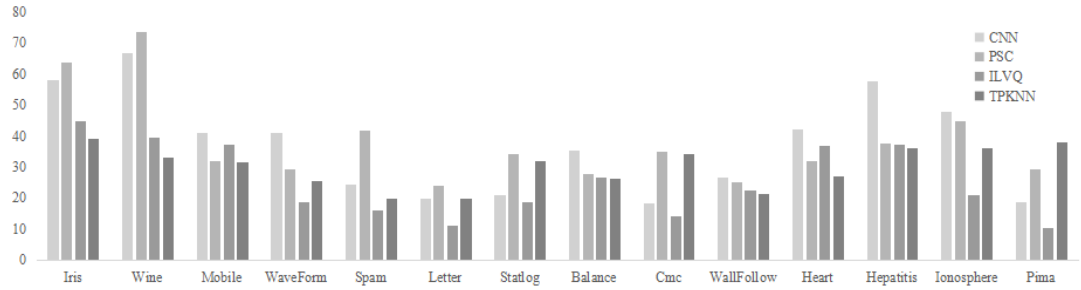

Figure 1. **Reduction ratios obtained by compared methods.**

Table 5. **The tree node's numbers and the reduced results for *TPKNN***

DataSet	*Precision*		*Recall*		*Accuracy*		*F_1-measure*	
	KNN	*TPKNN*	*KNN*	*TPKNN*	*KNN*	*TPKNN*	*KNN*	*TPKNN*
Iris	0.942	0.939	0.961	0.942	0.955	0.937	0.867	0.925
Wine	0.969	0.901	0.967	0.915	0.947	0.863	0.878	0.886
Mobile	0.893	0.836	0.858	0.843	0.891	0.831	0.832	0.827
WaveForm	0.873	0.881	0.903	0.896	0.912	0.901	0.887	0.862
Spam	0.925	0.838	0.945	0.821	0.959	0.812	0.873	0.819
Letter	0.901	0.833	0.921	0.822	0.905	0.809	0.845	0.817
Statlog	0.887	0.867	0.903	0.823	0.915	0.846	0.902	0.838

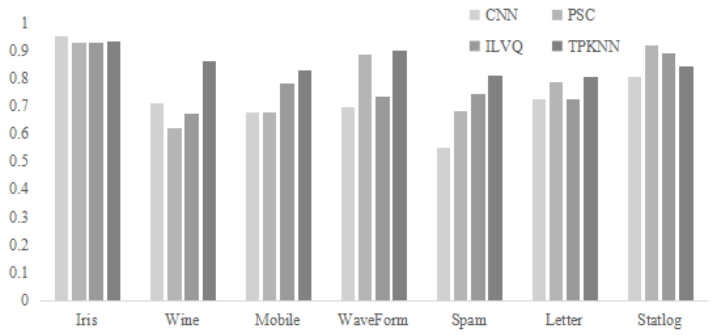

Figure 2. **Classification accuracies obtained by *CNN*, *PSC*, *ILVQ* and *TPKNN* on balanced sets**

Table 6. **The results obtained by *TPKNN* and *KNN* on unbalanced sets**

DataSet	*Precision*		*Recall*		*Accuracy*		*F_1-measure*	
	KNN	*TPKNN*	*KNN*	*TPKNN*	*KNN*	*TPKNN*	*KNN*	*TPKNN*
Balance	0.852	0.843	0.871	0.847	0.863	0.841	0.859	0.838
Cmc	0.694	0.735	0.807	0.812	0.846	0.793	0.751	0.796
WallFollow	0.877	0.855	0.882	0.867	0.879	0.857	0.897	0.869
Heart	0.816	0.802	0.835	0.818	0.827	0.809	0.823	0.814
Hepatitis	0.748	0.746	0.831	0.816	0.774	0.778	0.827	0.815
Ionosphere	0.851	0.867	0.864	0.873	0.857	0.885	0.859	0.867
Pima	0.757	0.732	0.793	0.754	0.818	0.725	0.792	0.749

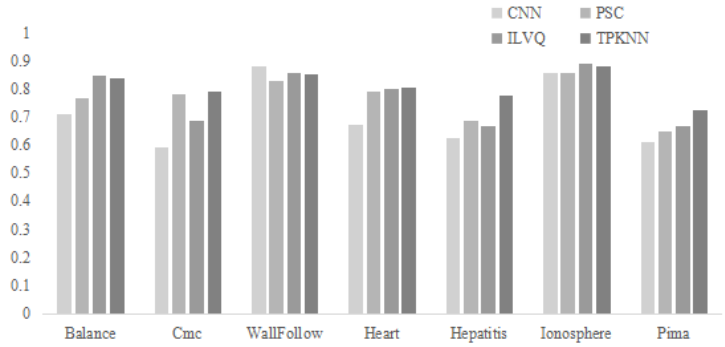

Figure 3. **Classification accuracies obtained by *CNN*, *PSC*, *ILVQ* and *TPKNN* on unbalanced sets**

4.4 Experiment in TPKNN and SVM

In the section, the one-to-one *SVM* algorithm is adopted and the multi-class *SVM* program is achieved based on two *SVM* library functions from MATLAB. *TPKNN* and SVM are repeated five times to achieve the average results based on 5-fold cross-validation and K=5. Although *SVM* can perform the best on ten of the fourteen datasets, it need to adjust a *SVM* MaxIter parameter from 15000 to more than 100000 and cause a heavy time cost. Based on Table 5-7, the results indicate that *TPKNN* has the comparable classification precision with *KNN* and *SVM*.

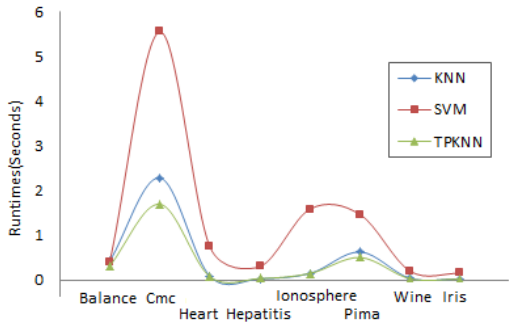

Figure 4. **Runtimes spent by three methods using the middle and small datasets**

Due to *SVM* spends much runtime that is more than two hours in large-scale dataset. So Figure 4 only contain the results in small-scale and middle-scale. Based on Figure 4, it is obviously noticed that *TPKNN* are the fastest methods than *KNN* and *SVM*. And therefore, *TPKNN* can shorten the whole runtime that includes preprocess step(*SVM*'s training step or *TPKNN*'s partitioning and pruning step), classification step, etc, while keeping the small precision loss.

4.5 Experiment in Semeion Handwritten Digit Dataset

Although the above experiments have shown the effectiveness of *TPKNN*, the Semeion Handwritten Digit dataset, which has been widely adopted in many research literatures, is chosen as the limited purpose dataset to verify the practical problem solving ability of the proposed algorithm. The dataset consists of 1593 semeion handwritten digits from around 80 persons with 256 features.

In order to simplify the process, the paper adopts the same experiment environment for all of the experiments. Table 8 gives the results for the handwritten digits dataset. Although the reduction ratio obtained by *TPKNN* is larger than that obtained by *CNN* and *ENN*, the results in Table 8 still indicate *TPKNN* can better deal with the special application, have an edge in terms of the less running time, the high classification accuracy and the more stable reduction ratio.

In order to verify the effectiveness of the proposed algorithm, the influence of high dimension, lager feature number and the feature relationship are ignored in this paper. So the datasets consisting of patterns with high dimension or large feature number are not listed here and two algorithms' performance cannot be evaluated in the field. As well all know, to classify patterns with high dimension or larger features, dimensionality reduction and effective feature extraction must be adopted. *TPKNN* will be verified on these datasets with high dimension or other special statuses in the next step.

Table 7. The precision of *TPKNN* and *SVM* in 14 UCI datasets

DataSet	Iris	Wine	Mobile	WaveFo	Spam	Letter	Statlog	Balance	Cmc	WallFollo	Heart	Hepatiti	Ionospher	Pima
SVM	0.948	0.937	0.864	0.907	0.821	0.924	0.897	0.916	0.969	0.893	0.867	0.811	0.869	0.743
TPKNN	0.937	0.913	0.846	0.901	0.832	0.836	0.869	0.851	0.815	0.866	0.832	0.826	0.887	0.753

Table 8. Operational efficiency results obtained by compared algorithms on Semeion Handwritten Digit dataset

Algorithm	*KNN*	*CNN*	*ENN*	*PSC*	*ILVQ*	*TPKNN*
Accuracy	0.909	0.591	0.736	0.672	0.933	0.941
Reduction ratio	100	16.21	12.47	33.94	31.58	22.47
Runtime	756.39	214.34	595.28	456.92	372.35	274.74

5 CONCLUSION

THIS paper is focused on providing the enhancer on fast search speed, a superiority of unbalanced datasets, an incremental growth that is a simple incremental learning algorithm than traditional *KNN*. For each class, *TPKNN* constructs a tree and can reduce a certain percentage of patterns roughly. Some measures are taken to eliminate the impact of the value of *K* through comparing the change in classification accuracy and increase the search speed by find the high similarly search set. The time complexity is linear in the number of the training patterns, and the same space complexity as *KNN*. Experiments performed on unbalanced datasets that *TPKNN*'s performance is at least comparable to, and often better than that of these compared algorithms.

Although *TPKNN* needs more time consumption to build tree structure and realize the pruning operation, it can obtain a small-scale effective reference set based on the original training set. The reference set can reduce the storage and running consumption in the classification process. In addition, *TPKNN* is very easy to implement and can be taken as the preprocessing of some classification algorithms. So *TPKNN* is an efficient multi-class classifier and be recommend as one of choices to evaluate on any classification problem.

6 ACKNOWLEDGMENT

THIS work was supported by the Fundamental Research Funds for the Central Universities (No. GK201703086).

7 REFERENCES

Aha, D. W., Kibler, D., and Albert, M. K. (1991). Instance-based learning algorithms, *Machine Learning*, 6(1), 37-66.

Barandela, Ricardo, Ferri, F. J., and Sánchez, J. S..(2005). Decision boundary preserving prototype selection for nearest neighbor classification, I*nternational Journal of Pattern Recognition and Artificial Intelligence*, 19(6): 787-806.

Fayed, H. A. & Atiya, A. F. (2009). A novel template reduction approach for the k-nearest neighbor

method, *IEEE Transactions on Neural Networks*, 20(5), 890-896.

Franti, P., Virmajoki, O., and Hautamaki, V..(2006). Fast agglomerative clustering using a k-nearest neighbor graph. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(11): 1875-1881.

Friedman, J. H., Bentley, J. H., and Finkel, R. A. (1977). An algorithm for finding best matches in logarithmic expected time, *ACM Transactions on Mathematical Software*, 3(3): 209-226.

Gong, A. & Liu, Y. (2011). Improved KNN classification algorithm by dynamic obtaining K, *ECWAC 2011*, Part I, CCIS 143: 320-324.

Han, E. H., Karypis, G., and Kumar, V. (2001). Text Categorization Using Weight Adjusted k-Nearest Neighbor Classification, *Pacific-Asia Conference on Knowledge Discovery and Data Mining in Springer, Berlin, Heidelberg*, 2035, 53-65.

Haro-García, A. D. & García-Pedrajas, N. (2009). A divide-and-conquer recursive approach for scaling up instance selection algorithms, *Data Mining & Knowledge Discovery*, 18(3), 392-418.

Jagadish, H. V., Ooi, B. C., Tan, K. L., Yu, C., and Zhang, R. (2005). Idistance:an adaptive b+-tree based indexing method for nearest neighbor search, *ACM Transactions on Database Systems*, 30(2), 364-397.

Khazaee, Ali & Ebrahimzadeh, Ataollah. (2013). Heart arrhythmia detection using support vector machines, *Intelligent Automation and Soft Computing,* 19(1), 1-9.

Köknar-Tezel, S. & Latecki, L. J. (2011). Improving svm classification on imbalanced time series data sets with ghost points, *Knowledge & Information Systems*, 28(1), 1-23.

Krishna, K., Thathachar, M. A. L. and Ramakrishnan, K. R.. (2000). Voronoi networks and their probability of misclassification, *IEEE Transactions on Neural Networks*, 11(6): 1361-1372.

Li, B., Qin, L., and Yu, S. (2004). An adaptive k-nearest neighbor text categorization strategy, *ACM Transactions on Asian Language Information Processing*, 3(4), 215-226.

Li, X., Shi, D., Charastrakul, V., and Zhou, J. (2009). Advanced p-tree based k-nearest neighbors for customer preference reasoning analysis, *Journal of Intelligent Manufacturing*, 20(5), 569-579.

Liu, Wei & Chawla, Sanjay. (2011). Class Condence Weighted kNN Algorithms for Imbalanced Data Sets. *PAKDD'11 Proceedings of the 15th PAKDD*, V(II): 345-356.

Lumini, A. & Nanni, L. (2006). A clustering method for automatic biometric template selection, *Pattern Recognition*, 39(3), 495-497.

Mccallum, A., & Nigam, K. (1998). A comparison of event models for naive bayes text classification, *IN AAAI-98 WORKSHOP ON LEARNING FOR TEXT CATEGORIZATION*, 62(2), 41-48.

Milli, M. & Bulut, H. (2017). The effect of neighborhood selection on collaborative filtering and a novel hybrid algorithm, Intelligent Automation and Soft Computing, 23(2): 261-269.

Mollineda, R. A., Ferri, F. J., and Vidal, E. (2002). An efficient prototype merging strategy for the condensed 1-nn rule through class-conditional hierarchical clustering, *Pattern Recognition,* 35(12), 2771-2782.

Olvera-López, J. A., Carrasco-Ochoa, J. A., and Martínez-Trinidad, J. F. (2010). A new fast prototype selection method based on clustering, *Pattern Analysis and Applications*, 13:131–141.

Platt, John C. (2013). Using analytic QP and sparseness to speed training of support vector machines, *Advances in neural information processing systems*, 557-563.

Raicharoen, T., Lursinsap, C., and Lin, F. (2005). A divide-and-conquer approach to the pairwise opposite class-nearest neighbor (POC-NN) algorithm for regression problem, *Pattern Recognition Letters*, 26(10), 1554-1567.

Rico-Juan, J. R., & Iñesta, J. M. (2012). New rank methods for reducing the size of the training set using the nearest neighbor rule, *Pattern Recognition Letters*, 33(10), 1434-1434.

Sample, Neal, Matthew, Haines, Mark, Arnold, and Purcell, Timothy. (2001). Optimizing Search Strategies in k-d Trees, *5th WSES/IEEE World Multiconference on (CSCC 2001)*.

Wang, J., Neskovic, P., and Cooper, L. N. (2006). Neighborhood size selection in the k-nearest-neighbor rule using statistical confidence, *Pattern Recognition*, 39(3), 417-423.

Xu, Y., Shen, F., and Zhao, J. (2012). An incremental learning vector quantization algorithm for pattern classification, Neural Computing & Applications, 21(6), 1205-1215.

Zeng, Yong, Yang, Yupu, and Zhao, Liang. (2009). Nonparametric classification based on local mean and class statistics, *Expert Systems with Applications*, 36(4): 8443-8448.

Zeng, Yong, Yang, Yupu, and Zhao, Liang. (2009). Pseudo nearest neighbor rule for pattern classification, *Expert Systems with Applications*, 36(2): 3587-3595.

Zhang, B. & Srihari, S. N..(2004). Fast k-nearest neighbor classification using cluster-based trees, I*EEE Transactions on Pattern Analysis and Machine Intelligence,* 26(4): 525-528.

Zheng, B., Lee, W. C., and Lee, D. L. (2003). Search K Nearest Neighbors on Air, *Mobile Data Management*, Springer Berlin Heidelberg, 2003:181-195.

Zou, T., Wang, Y., Wei, X., Li, Z., and Yang, G. (2014). An effective collaborative filtering via enhanced similarity and probability interval

prediction, *Intelligent Automation and Soft Computing*, 20(4), 555-566.

8 NOTES OF CONTRIBUTORS

Juan Li received a Ph.D. degree from School of Computer Science and Technology, Xidian University, Xi'an, China, in 2015. She is a lecturer at School of Distance Education, Shaanxi Normal University, Xi'an, China. Her research interest covers data mining and pattern recognition.