Class Based Weighted K-Nearest Neighbor over Imbalance Dataset

Harshit Dubey and Vikram Pudi

International Institute of Information Technology, Hyderabad,
Andhra Pradesh, India 500032
harshit.dubeyug08@students.iiit.ac.in, vikram@iiit.ac.in

Abstract. K-Nearest Neighbor based classifier classifies a query instance based on the class labels of its neighbor instances. Although kNN has proved to be a ubiquitous classification tool with good scalability, but it suffers from some drawbacks. The existing kNN algorithm is equivalent to using only local prior probabilities to predict instance labels, and hence it does not take into account the class distribution around neighborhood of the query instance, which results into undesirable performance on imbalanced data. In this paper, a modified version of kNN algorithm is proposed so that it takes into account the class distribution in a wider region around the query instance. Our empirical experiments with several real world datasets show that our algorithm outperforms current state-of-the-art approaches.

Keywords: Classification, K-Nearest Neighbor, Probability, Imbalance, Distribution.

1 Introduction

A data set is imbalanced, if its dependent variable is categorical and the number of instances in one class is different from those in the other class. In many real world applications such as Web page search, scam sites detection, fraudulent calls detection etc, there is a highly skewed distribution of classes. Various classification techniques such as kNN [6], SVM [5], and Neural Networks[10] etc have been designed and used, but it has been observed that the algorithms do not perform as good on imbalanced datasets as on balanced datasets. Learning from imbalanced data sets has been identified as one of the 10 most challenging problems in data mining research [17]. In the literature of solving class imbalance problems, various solutions have been proposed. Such techniques broadly include two different approaches: (1) modifying existing methods or (2) application of a pre-processing stage.

In the recent past, a lot of research centered at nearest neighbor methodology has been done. Although kNN is computationally expensive, it is very simple to understand, accurate, requires only a few parameters to be tuned and is robust with regard to the search space. Also kNN classifier can be updated at a very little cost as new training instances with known classes are presented. A

J. Pei et al. (Eds.): PAKDD 2013, Part II, LNAI 7819, pp. 305–316, 2013.

strong point of kNN is that, for all data distributions, its probability of error is bounded above by twice the Bayes probability of error [16]. However one of the major drawbacks of kNN is that, it uses only local prior probabilities to predict instance labels, and hence does not take into account, class distribution around the neighborhood of query instance. This results in undesirable performance on imbalanced data sets. The performance of kNN algorithm over imbalanced datasets can be improved, if it uses information about local class distribution while classifying instances.

Fig. 1 shows an artificial two-class imbalance problem, where the majority class "A" is represented by circles and the minority class "B" by triangles. The query instance is represented by cross. As can be seen from the figure, the query instance would have been classified as the majority class "A" by a regular kNN algorithm with k value equal to 7. But if the algorithm had taken into account the imbalance class distribution around the neighborhood of the query instances (say in the region represented by dotted square), it would have classified the query instance as belonging to minority class "B", which is the desired class.

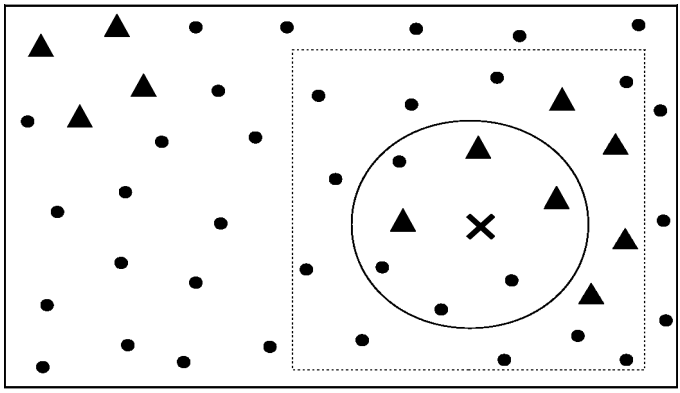

Fig. 1. A sample scenario where regular kNN algorithm will fail

In this paper, we propose a modified K-Nearest Neighbor algorithm to solve the imbalanced dataset classification problem. More specifically the contributions of this paper are as follows:

1. First we present a mathematical model of K-Nearest Neighbor algorithm and show that, it does not take into account nature of the data around the query instance.
2. To solve the above problem, we propose a Weighted K-Nearest Neighbor algorithm in which a weight is assigned to each of the class based on how its instances are classified in the neighborhood of query instance by the regular K-Nearest Neighbor classifier. The modified algorithm takes into account class distribution around the neighborhood of query instance. We ensure that the weights assigned do not give undue advantage to outliers.

3. A thorough experimental study of the proposed approach over several real world dataset was performed. The study confirms that our approach performs better than the current state-of-the-art approaches.

The organization of rest of the paper is as follows. In section 2, we throw light on related, and recent, work in the literature. Section 3 deals with problem formulation and mathematical model of kNN. We explain the modified algorithm in Section 4. In Section 5, experimental results are presented together with a thorough comparison with the state-of-the-art algorithms. Finally, in Section 6, conclusions are drawn.

2 Related Work

In the literature of solving class imbalance problems, various solutions have been proposed; such techniques broadly include two different approaches, modifying methods or the application of a pre-processing stage. The pre-processing approach focuses on balancing the data, which may be done either by reducing the set of examples (undersampling) or replicate minority class examples (oversampling) [8]. One of such earliest and classic work is the SMOTE method [3] which increases the number of minor class instances by creating synthetic samples. This work is also based on the nearest neighbor analogy. The minority class is over sampled by taking each minority class sample and introducing synthetic examples along the line segments joining any/all of the k minority class nearest neighbors. A recent modification to SMOTE proposes that, using different weight degrees on the synthetic samples (so-called safe-level-SMOTE [2]) produces better accuracy than SMOTE. Alternative approaches that modify existing methods focus on extending or modifying the existing classification algorithms so that they can be more effective in dealing with imbalanced data. HDDT [4] and CCPDT [15] are examples of such methods, which are modification of decision tree algorithms.

One of the oldest, accurate and simplest method for pattern classification and regression is K-Nearest-Neighbor (kNN) [6]. kNN algorithms have been identified as one of the top ten most influential data mining algorithms [19] for their ability of producing simple but powerful classifiers. It has been studied at length over the past few decades and is widely applied in many fields. The kNN rule classifies each unlabeled example by the majority label of its k-nearest neighbors in the training dataset. Despite its simplicity, the kNN rule often yields competitive results. A recent work on prototype reduction, called Weighted Distance Nearest Neighbor (WDNN) [11] is based on retaining the informative instances and learning their weights for classification. The algorithm assigns a non negative weight to each training instance tuple at the training phase. Only the training instances with positive weight are retained (as the prototypes) in the test phase. Although the WDNN algorithm is well formulated and shows encouraging performance, in practice it can only work with $K = 1$. A more recent approach WDkNN [20] tries to reduce the time complexity of WDNN and extend it to work for values of K greater than 1.

Chawla and Liu in one of their recent work [14] presented a novel K-Nearest Neighbors weighting strategy for handling the problem of class imbalance. They proposed CCW (class confidence weights) that uses the probability of attribute values given class labels to weight prototypes in kNN. While the regular kNN directly uses the probabilities of class labels in the neighborhood of the query instance, they used conditional probabilities of classes. They have also shown how to calculate CCW weights using mixture modeling and Bayesian networks. The method performed more accurately than the existing state-of-art algorithms.

KaiYan Feng and others [7] defined a new neighborhood relationship known as passive nearest neighbors. For two points A and B belonging to class L, point B is the local passive k^{th}-order nearest neighbor of A, only and only if A is the k^{th} nearest neighbor of B among all data of class L. For each query point, its k actual nearest neighbor and k passive nearest neighbors are first calculated and based on it, a overall score is calculated for each class. The class score determines the likelihood that the query points belong to that class.

In another recent work [12], Evan and others proposes to use geometric structure of data to mitigate the effects of class imbalance. The method even works, when the level of imbalance changes in the training data, such as online streaming data. For each query point, a k dimensional vector is calculated for each of the classes present in the data. The vector consist of distances of the query point to it's k nearest neighbors in that class. Based on this vector probability that the query point belongs to a particular class is calculated. However the approach is not studied in depth.

Yang Song and others proposes [18] two different versions of kNN based on the idea of informativeness. According to them, a point is treated to be informative, if it is close to the query point and far away from the points with different class labels. One of the proposed versions LI-KNN takes two parameters k and I, It first find the k nearest neighbor of the query point and then among them it find the I most informative points. Based on the class label of the informative points, class label is assigned to the query point. They also showed that the value of k and I have very less effect on the final result. The other version GI-KNN works on the assumption that some points are more informative then others. It tries to find global informative points and then assigns a weight to each of the points in training data based on their informativeness. It then uses weighted euclidean metric to calculate distances.

In another recent work [13], a k Exemplar-based Nearest Neighbor (kENN) classifier was proposed which is more sensitive to the minority class. The main idea is to first identify the exemplar minority class instances in the training data and then generalize them to Gaussian balls as concept for the minority class. The approach is based on extending the decision boundary for the minority class.

3 Problem Formulation and Mathematical Model of kNN

In this section, we present a mathematical model for kNN algorithm along with the notation used to model the dataset. We also show that kNN only makes use of local prior probabilities for classification.

The problem of classification is to estimate the value of the class variable based on the values of one or more independent variables (known as feature variables). We model the tuple as $\{x, y\}$ where x is an ordered set of attribute values and y is the class variable to be predicted. There are d attributes overall corresponding to a d-dimensional space.

Formally, the problem has the following inputs:

- A set of n tuples called the training dataset, $D = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$.
- A query tuple x_t.

The output is an estimated value of the class variable for the given query x_t, mathematically it can be expressed as:

$$y_t = f(x_t, D, parameters), \tag{1}$$

Where *parameters* are the arguments that the function $f()$ takes. These are generally set by the user or are learned by some method.

3.1 Mathematical Model of *k*NN

For a given query instance x_t, kNN algorithm works as follows:

$$y_t = \underset{c \in \{c_1, c_2, \ldots, c_m\}}{\arg\max} \sum_{x_i \in N(x_t, k)} E(y_i, c) \tag{2}$$

Where y_t is the predicted class for the query instance x_t and m is the number of classes present in the data. Also

$$E(a, b) = \begin{cases} 1 & if a = b \\ 0 & else \end{cases} \tag{3}$$

$$N(x, k) = Set\ of\ k\ nearest\ neighbor\ of\ x$$

Eq. (2) can also be written as

$$y_t = \arg\max \left\{ \sum_{x_i \in N(x_t, k)} E(y_i, c_1), \sum_{x_i \in N(x_t, k)} E(y_i, c_2), \quad \ldots \quad , \sum_{x_i \in N(x_t, k)} E(y_i, c_m) \right\} \tag{4}$$

$$y_t = \arg\max \left\{ \sum_{x_i \in N(x_t, k)} \frac{E(y_i, c_1)}{k}, \sum_{x_i \in N(x_t, k)} \frac{E(y_i, c_2)}{k}, \quad \ldots \quad , \sum_{x_i \in N(x_t, k)} \frac{E(y_i, c_m)}{k} \right\} \tag{5}$$

and we know that

$$p(c_j)_{(x_t,k)} = \sum_{x_i \in N(x_t,k)} \frac{E(y_i, c_j)}{k} \tag{6}$$

Where $p(c_j)_{(x_t,k)}$ is the probability of occurrence of j^{th} class in the neighborhood of x_t . Hence Eq. 5 turns out to be

$$y_t = \arg\max\{p(c_1)_{(x_t,k)}, p(c_2)_{(x_t,k)}, \ldots, p(c_m)_{(x_t,k)}\} \tag{7}$$

It is clear from Eq. 7, that kNN algorithm uses only prior probabilities to calculate the class of the query instance. It ignores the class distribution around the neighborhood of query point.

4 Algorithm

In this section, we will explain in detail our proposed algorithm. To tune the existing kNN algorithm , we introduce a weighting factor for each class. Our algorithm can be formally expressed as follows, for a given query instance x_t:

$$y_t = \underset{c \in \{c_1, c_2, \ldots, c_m\}}{\arg\max} \sum_{x_i \in N(x_t,k)} W[c, x_t] * E(y_i, c) \tag{8}$$

Where $W[c, x_t]$ denotes the weighting factor for the class c, while classifying query instance x_t. For Weighting factor equal to 1 for all the classes, our algorithm reduces to the existing kNN classifier. This weighting factor is introduced to take into account, class distribution around the query instance. The proposed algorithm is sensitive to the value of weighting factor.

Now we discuss on how to learn the value of weighting factor for each of the classes. Fig. 2 illustrates an imbalance dataset, in which data points are present in clusters with each cluster having exactly one major class. In this case, regular kNN algorithm would fail to classify the minority class instances present at the boundary of the cluster region (for example query instance 1).

To design the weights, we considered both query dependent and query independent weighting factor. If our learned weighting factors have a constant value for each of the class through out the dataset i.e. they do not depend on the query instance, and favors the minority class then, our algorithm would have classified the minority class instances present at the boundary of the clusters correctly, but have not classified points like instance 2 correctly. Having only class dependent weighting factor values would not capture the data distribution around the neighborhood of the query instance.

Our weighting factor value $W[c, x_t]$ can be denoted as :

$$W[c, x_t] = \frac{alpha(c, x_t)}{1 + alpha(c, x_t)} \tag{9}$$

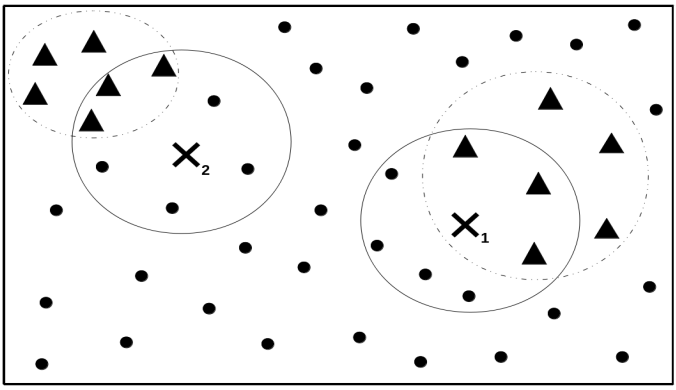

Fig. 2. A sample scenario where data points are present in clusters (dotted circle represent clusters containing minority class) with each cluster having one major class

where

$$alpha(c, x_t) = \sum_{x_i \in N(x_t, \frac{k}{m}, c)} \frac{m * getcoef(x_i)}{k} \tag{10}$$

$$N(x, k, c) = Set\ of\ k\ nearest\ neighbor\ of\ x\ belonging\ to\ class\ c$$

$$getcoef(x_i) = \frac{N(x_i, k, c')}{N(x_i, k, y_i)} \tag{11}$$

where c' is the class to which x_i is classified by existing kNN classifier. if c' equals to y_i then $getcoef(x_i)$ turns out to be 1.

Hence for a query point the weighting factor of a class is calculated based on how the k/m nearest neighbors of query point belonging to that class are classified by the existing kNN classifier. If a instance is classified correctly $getcoef$ will return 1 for it, else it will return the value by which the class prior probability should be multiplied, so that it is classified correctly. The basic intuition about the above formulae is simple: "If instances of a specific class are poorly classified in a particular region, then that class is likely to be a minority class in that region and should be given a higher weight."

4.1 Properties of Weighting Factor

1. The weighting factor for each of the class is calculated based on how the k/m nearest neighbors of query point belonging to that class are classified by regular kNN classifier. If points belonging to a particular class in the neighborhood of the query points are classified incorrectly by regular kNN approach, then the weighting factor of that class will have a high value indicating that this class might be a minority class in that region. Hence the learned weighting factor takes into account the class distribution around neighborhood of the query point.

Algorithm 1. Pseudo code

Input: *Query instance* Q, *Training Data* D, *Parameter* k, *Set of all the class label* C
Output: *Class Label* Cl *for Query instance* Q

1: **for** *each class* j *in* C **do**
2: $Coefficient = (\frac{m}{k}) * \sum_{x_i \in N(Q, \frac{k}{m}, j)} (getcoef(x_i))$
3: $W[j, Q] = \frac{Coefficient}{1+Coefficient}$
4: **end for**
5: **for** *each neighbor* n *in* $N(Q, k)$ **do**
6: $Probability[class(n)] = Probability[class(n)] + W[class(n), Q]$
7: **end for**
8: $Cl = \arg\max Probability$

2. Value of weighting factor is bounded between 0.5 to 1, proof of which is :

   if x_i *is classified correctly by regular* kNN **then**
   $\Rightarrow getcoef(x_i) \leftarrow 1$
   else x_i *is classified as belonging to class* c'
   $\Rightarrow N(x_i, k, c') > N(x_i, k, y_i)$
   $\Rightarrow getcoef(x_i) > 1$ (from eq. 11)
   end if

   Hence $getcoef(x_i)$ value is always greater than or equal to 1, that implies $alpha(c, x_t)$ which is average of $getcoef$ values of k/m nearest neighbors of x_t is always greater than or equal to 1. Eq. 10 can also be written as

$$W[c, x_t] = \frac{1}{1 + \frac{1}{alpha(c, x_t)}}$$

   Which impliles that

$$0.5 \leq W[c, x_t] \leq 1$$

   If some outlier point is present in the neighborhood of the query point its $getcoef$ factor would have a high value, but as the weighting factor for a class is calculated by taking average of $getcoef$ values of k/m nearest neighbors of query point belonging to that class, its value would not be much affected by a outlier point. This makes our learned weighting factors resistant to outliers.

4.2 Complexity Analysis

The proposed algorithm needs to search for the k nearest neighbors of the query point (global nearest neighbors, line 5 of Algorithm 1.), same as the regular kNN algorithm. Apart from finding the global nearest neighbors, it also need to calculate the weighting factor for each of the class. Following calculations are involved in the calculation of weighting factors :

1. For each of the class, find k/m nearest neighbors of query point among that class (class neighbors). We can make use of the fact that the global k nearest

neighbors will be among the k nearest neighbors for each of the class, to optimize the search. Rather then finding k/m nearest neighbor for each class, we first get the k nearest neighbor for each class and then get the global k nearest neighbors, with some extra cost involved. For example, assuming that no index structure is present in the training data, while searching for global k nearest neighbors of query point a binary heap structure of size k is needed. For our proposed approach, we maintain such a heap structure one per class. While searching for neighbors, each of the points in the training data is inserted in the heap belonging to its class. When all the points are inserted in the heap, we have the k nearest neighbors of query points among each class in the respective class heap. The total *heapinsert* operations remains same as when finding global k nearest neighbors. Then we can find the global k nearest neighbors and class neighbors from this $k * (numberofclass)$ points.

2. For each of the points obtained above calculate $getcoef$ function, which needs the k nearest neighbors for each of the points (line 2). If $getcoef$ function is calculated for all the points present in the training data during the pre processing step then, this runtime overhead can be avoided. Else we need to find the k nearest neighbors of all the points obtained above i.e. k points.

5 Experimental Study

5.1 Performance Model

In this section, we demonstrate our experimental settings. The experiments were conducted on a wide variety of datasets obtained from UCI data repository [1] and Weka Datasets [9]. A short description of all the datasets is provided in Table 1. These datasets have been selected as they typically have a low minority class percentage and hence are imbalance. We have evaluated our algorithm against the existing state of art approaches. All the results have been obtained using 10-fold cross validation technique, except for SMOTE. For SMOTE each of the dataset is first randomized and then divided into training and testing data. SMOTE sampling is applied on training data to oversample the minority class and then regular kNN is used to classify instances present in testing data using the sampled training data.

As it is known that the use of overall accuracy is not an appropriate evaluation measure for imbalanced datasets, because of the dominating effect of the majority class, we have used F-Score as the evaluation metric. F-Score considers both the precision and the recall of the test to compute the score. We compared our performance against the following approaches: Regular K Nearest Neighbors (kNN) [1], Exemplar kNN (kENN) [2], SMOTE , HDDT [3] , C4.5 , CCPDT [4],

[1] For kNN , SMOTE, C4.5 (available as j48) and Naive, implementation available in Weka toolkit is used.

[2] The code is obtained from `http://goanna.cs.rmit.edu.au/$\sim$zhang/ENN/Weka-3-6_ENN.zip`

[3] The code is obtained from `http://nd.edu/$\sim$dial/software/hddt.tar.gz`

[4] The code is obtained from `http://www.cs.usyd.edu.au/$\sim$weiliu/CCPDT_src.zip`

Table 1. Dataset Description

Dataset	#Instances	#Attributes	#Class	Minority Class%
Balance	625	5	3	7.84
Cmc	1473	10	3	22.61
Diabetes	768	9	2	34.9
Glass	214	10	6	4.2
Heart	270	14	2	44.44
Hungarian	294	14	2	36.05
Ionosphere	351	34	2	35.9
Tranfusion	748	5	2	23.7
Wine	178	13	3	26.9

NaiveBayes (Naive). For all kNN based approaches (including SMOTE) F-Score obtained at maximum accuracy is mentioned.

5.2 Results and Discussion

Table 2. compares the result of our modified algorithm with existing state of the art algorithm. The number in the parenthesis indicates the rank of the respective algorithm. Also the top two algorithms are highlighted in bold. It can be seen that our approach produces consistently accurate classifier and outperforms other algorithms in most of the datasets. Also our proposed algorithm always outperform regular kNN on all the datasets, this confirms that the modified kNN algorithms takes into account the nature of the data to classify it. However for Ionosphere dataset decision tree based algorithms perform better than other state of the art algorithms.

Table 2. Experimental results over several real world dataset

Dataset	Our Algo	kNN	kENN	SMOTE	HDDT	C4.5	CCPDT	Naive
Balance	**0.361 (1)**	0.077 (6)	**0.167 (2)**	0.154 (3)	0.089 (5)	0.000 (7)	0.092 (4)	0.000 (7)
Cmc	0.419 (3)	0.418 (4)	**0.424 (2)**	0.364 (7)	0.380 (6)	0.409 (5)	0.356 (8)	**0.445 (1)**
Pima	**0.628 (2)**	0.601 (6)	0.610 (5)	0.593 (7)	0.613 (4)	0.614 (3)	0.587 (8)	**0.643 (1)**
Glass	**0.778 (1)**	**0.778 (1)**	0.560 (7)	0.750 (3)	0.571 (6)	0.636 (5)	0.235 (8)	0.696 (4)
Heart	**0.812 (2)**	0.805 (5)	**0.812 (2)**	0.765 (7)	0.784 (6)	0.736 (8)	**0.828 (1)**	**0.812 (2)**
Hungarian	0.779 (3)	0.747 (6)	0.747 (6)	**0.805 (2)**	0.767 (4)	0.656 (8)	**0.815 (1)**	0.762 (5)
Ionosphere	0.824 (5)	0.779 (8)	0.793 (6)	0.833 (4)	**0.891 (2)**	0.874 (3)	**0.894 (1)**	0.781 (7)
Tranfusion	**0.489 (2)**	0.442 (7)	0.486 (4)	**0.509 (1)**	**0.489 (2)**	0.481 (6)	0.486 (4)	0.281 (8)
Wine	**0.980 (1)**	**0.980 (1)**	0.950 (6)	**0.980 (1)**	0.958 (5)	0.923 (7)	0.871 (8)	0.970 (4)
Average Rank	**2.22**	4.88	4.44	**3.88**	4.44	5.77	4.77	4.33

Fig. 3 compares performance of our algorithm with kNN in terms of overall accuracy and accuracy to classify minority class, as the value of k varies for Hungarian dataset. It becomes clear from the figure that our algorithm based

classifier are more sensitive to classify minority class and are still highly accurate. Also classifier learned from our approach are more accurate for larger values of k, this is evident from the fact that, for high value of k large region around the neighborhood of query point is considered to determine the local class distribution.

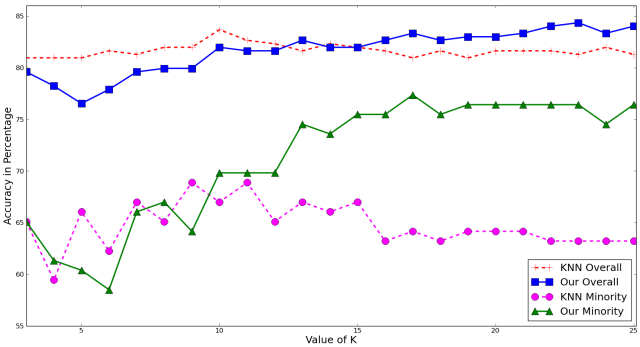

Fig. 3. Performance Comparison between Our Algorithm and kNN

6 Conclusion

In this paper, we have proposed a modified version of K-Nearest Neighbor algorithm so that it takes into account, class distribution around neighborhood of query instance during classification. In our modified algorithm, a weight is calculated for each class based on how its instances are classified by existing K-Nearest Neighbor classifier around the query instance and then a weighted kNN is applied. We have also evaluated our approach against the existing standard algorithms. Our work is focused on tuning the K-Nearest Neighbor for imbalance data, so that its performance on imbalance data is enhanced. As shown in the experimental section, our approaches more than often, outperforms existing state of the art approaches on a wide variety of datasets. Also our modified algorithm perform as good as the existing K-Nearest Neighbor classifier on balance data.

References

1. Asuncion, D.N.A.: UCI machine learning repository (2007)
2. Bunkhumpornpat, C., Sinapiromsaran, K., Lursinsap, C.: Safe-level-smote: Safe-level-synthetic minority over-sampling technique for handling the class imbalanced problem. In: Theeramunkong, T., Kijsirikul, B., Cercone, N., Ho, T.-B. (eds.) PAKDD 2009. LNCS, vol. 5476, pp. 475–482. Springer, Heidelberg (2009)
3. Chawla, N., Bowyer, K., Hall, L., Kegelmeyer, W.: Smote. Journal of Artificial Intelligence Research 16(1), 321–357 (2002)

4. Cieslak, D.A., Chawla, N.V.: Learning decision trees for unbalanced data. In: Daelemans, W., Goethals, B., Morik, K. (eds.) ECML PKDD 2008, Part I. LNCS (LNAI), vol. 5211, pp. 241–256. Springer, Heidelberg (2008)
5. Cortes, C., Vapnik, V.: Support-vector networks. Machine Learning 20, 273–297 (1995), doi:10.1007/BF00994018
6. Duda, R.O., Hart, P.E., Stork, D.G.: Pattern Classification, 2nd edn. Wiley, New York (2001)
7. Feng, K., Gao, J., Feng, K., Liu, L., Li, Y.: Active and passive nearest neighbor algorithm: A newly-developed supervised classifier. In: Huang, D.-S., Gan, Y., Gupta, P., Gromiha, M.M. (eds.) ICIC 2011. LNCS, vol. 6839, pp. 189–196. Springer, Heidelberg (2012)
8. Garcia, S., Herrera, F.: Evolutionary undersampling for classification with imbalanced datasets: proposals and taxonomy. Evolutionary Computation 17(3), 275–306 (2009)
9. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.H.: The weka data mining software: an update. SIGKDD Explor. Newsl. 11, 10–18 (2009)
10. Haykin, S.: Neural Networks: A Comprehensive Foundation. Prentice-Hall (1999)
11. Jahromi, M.Z., Parvinnia, E., John, R.: A method of learning weighted similarity function to improve the performance of nearest neighbor. Inf. Sci. 179, 2964–2973 (2009)
12. Kriminger, E., Principe, J., Lakshminarayan, C.: Nearest neighbor distributions for imbalanced classification. In: The 2012 International Joint Conference on Neural Networks (IJCNN), pp. 1–5 (June 2012)
13. Li, Y., Zhang, X.: Improving k nearest neighbor with exemplar generalization for imbalanced classification. In: Huang, J.Z., Cao, L., Srivastava, J. (eds.) PAKDD 2011, Part II. LNCS, vol. 6635, pp. 321–332. Springer, Heidelberg (2011)
14. Liu, W., Chawla, S.: Class confidence weighted knn algorithms for imbalanced data sets. In: Huang, J.Z., Cao, L., Srivastava, J. (eds.) PAKDD 2011, Part II. LNCS, vol. 6635, pp. 345–356. Springer, Heidelberg (2011)
15. Cieslak, D., Liu, W., Chawla, S., Chawla, N.: A robust decision tree algorithms for imbalanced data sets. In: Proceedings of the Tenth SIAM International Conference on Data Mining, pp. 766–777 (2010)
16. Loizou, G., Maybank, S.J.: The nearest neighbor and the bayes error rates. IEEE Transactions on Pattern Analysis and Machine Intelligence PAMI-9(2), 254–262 (1987)
17. Yang, Q., Wu, X.: 10 challenging problems in data mining research. International Journal of Information Technology and Decision Making 5(4), 597–604 (2006)
18. Song, Y., Huang, J., Zhou, D., Zha, H., Giles, C.L.: Iknn: Informative k-nearest neighbor pattern classification. In: Kok, J.N., Koronacki, J., Lopez de Mantaras, R., Matwin, S., Mladenič, D., Skowron, A. (eds.) PKDD 2007. LNCS (LNAI), vol. 4702, pp. 248–264. Springer, Heidelberg (2007)
19. Wu, X., Kumar, V., Ross Quinlan, J., Ghosh, J., Yang, Q., Motoda, H., McLachlan, G.J., Ng, A., Liu, B., Yu, P.S., Zhou, Z.-H., Steinbach, M., Hand, D.J., Steinberg, D.: Top 10 algorithms in data mining. Knowl. Inf. Syst. 14, 1–37 (2007)
20. Yang, T., Cao, L., Zhang, C.: A novel prototype reduction method for the K-nearest neighbor algorithm with $K \geq 1$. In: Zaki, M.J., Yu, J.X., Ravindran, B., Pudi, V. (eds.) PAKDD 2010. LNCS, vol. 6119, pp. 89–100. Springer, Heidelberg (2010)