Incorporating Farthest Neighbours in Instance Space Classification

Daniel Vaccaro-Senna and Mohamed Medhat Gaber

School of Computing, University of Portsmouth, Hampshire, England, PO1 3HE, UK
{daniel.vaccaro-senna,mohamed.gaber}@port.ac.uk

Abstract. The nearest neighbour (*NN*) classifier is often known as a 'lazy' approach but it is still widely used particularly in the systems that require pattern matching. Many algorithms have been developed based on NN in an attempt to improve classification accuracy and to reduce the time taken, especially in large data sets. This paper proposes a new classification technique based on k-Nearest Neighbour (*k-NN*), called *k-Nearest & Farthest Neighbours* (*k-NFN*). Farthest neighbours are used to identify classes that an unseen record may not belong to and are considered with the nearest neighbours in the classification decision. Two neighbour voting systems are also proposed to further improve k-*NN* and *k-NFN* accuracy. The first uses a ranking system and the second uses a spectrum to consider how near or far a neighbour actually is. The accuracy of our three proposed *k-NFN* techniques and *k-NN* are compared using the standard ten cross fold validation experiments on a number of real data sets, evidencing the superiority of our proposed techniques in terms of accuracy.

Keywords. k-Nearest Neighbour, Farthest Neighbour, classification accuracy

1 Introduction

The surge in data collection and analysis has been met with a new field of interest to computer science known as data mining. A core aspect of this field is data classification. This can be defined as dividing objects into classes [1]. For example, patients in a hospital database may be classified by their illness or injury. The process often involves building a classification model that can identify the class an unseen instance belongs to. A tree model is a commonly used structure for decision making that has been used in classification techniques [2-5]. Constructing models on large data sets can be complicated and increase the time of classification, especially when a model needs to be rebuilt with each new instance.

Different approaches exist which use pattern recognition techniques. A 'benchmark method' [6] considered to be one of the top 10 data mining algorithms is the *k-Nearest Neighbour (k-NN)* classifier [7]. The idea is to estimate the class of an instance by studying the instances that are defined as the closest to it. The algorithm is considered simple and commonly used in pattern recognition [8]. As *k-NN* does not build a classification model in advance, it is considered a method of 'lazy learning' which can affect its efficiency on large or complex data sets [9]. The method is totally

focused on the similar instances, and consequently does not consider the wider picture. It is possible for instances that are not neighbours to still have the same class. This would be considered a vague class label. In this paper, we set out to extend the *k-NN* algorithm to consider its nearest and farthest neighbours to classify data without adding a model or rules, in order to maintain the original algorithm's simplicity. We also look at different ideas for voting systems amongst the neighbours and how the farthest neighbours may be best used for classification.

Three voting systems involving farthest neighbours are proposed. The first takes the standard *k-NN* and applies the farthest neighbours as direct opposites. In practice, each farthest neighbour cancels out a nearest neighbour with the same class. The second ranks the nearest neighbours in order of 'closeness' and scores higher values for classes that appear top of the rank. The same is applied to the farthest neighbours except that a class's majority score is decreased. The third creates a spectrum-like model. The single nearest and single farthest neighbours are used as the minimum and maximum on the scale. The remaining neighbours are given a score based on their relative position on the spectrum.

The use of farthest neighbours is labeled as the *K Nearest & Farthest Neighbour* algorithm (*k-NFN*). The three methods proposed based on *k-NFN* in this paper are named; *k-NFN Original*, *k-NFN Rank* (*k-NFNR*) and *k-NFN Spectrum* (*k-NFNS*) respectively.

2 Background

The *k-Nearest Neighbour* algorithm gets its name from classifying instances by the integer *k* number of instances that are most similar or closest from the training set. To do this, the distance between instances must be determined based upon their attributes. Each training instance can be considered a point in *n* dimensional space where *n* is the number of descriptive attributes [10]. As a result of this, graph theory can be applied to the instances and the City Block or Euclidean distance function can be used. The City Block distance [1] between two points is based on the idea that you cannot usually take a direct route from one place to another. Considering two points in a graph, you can find the distance by taking the sum of horizontal and vertical distances. A more popular distance measurement is the Euclidean distance. This calculates the direct route between two points. The Euclidean distance between two instances *x* and *y* can be described as in equation (1)

$$D(x, y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2} \quad (1)$$

The Euclidean distance finds the difference between the values of *x* and *y* for each attribute from *i* to *n* and applies the Pythagoras theorem to find the overall distance. A potential problem with using Euclidean, or even the City Block distance function, is that attributes with large domains may dominate. A solution to this is normalisation.

$$N(value) = \frac{(value - minimum)}{(maximum - minimum)} \quad (2)$$

As shown in equation (2), to normalise data, the maximum and minimum values of each attribute are identified for the calculation to take place. Distances among categorical attributes have been set to 0 for the same values and 1 if the values are different. In this paper, we have handled missing values by replacing them by the average.

3 Related Work

Classification methods have been the focus of a large body of the data mining literature. The *k-Nearest Neighbour* technique is a popular algorithm noted as one of the best [7]. The *k-NN* algorithm is highly rated, but it also has its issues including; computational complexity [8] and the curse of dimensionality [11]. Some extensions of *k-NN* have been made that successfully improved the classification accuracy in some data sets by improving the distance function or neighbourhood size. These include the weight adjusted k-nearest neighbour (*WAKNN*) and dynamic k-nearest neighbour (*DKNN*) [11]. Research into the importance of the value for k resulted in Variable k-NN (*V-kNN*) which discovers the optimum for each training set [12]. Another idea for improving *k-NN's* performance was developed through Class Based k-NN (*CB-kNN*) [12]. This algorithm considers how there may be an unbalanced set of classes in the data. Another proposed algorithm is the cluster-based trees [13]. The K Best Cluster Based Neighbour (*KB-CB-N*) [10] is a classification technique that also uses a cluster based approach combined with three different similarity measures rather than just the Euclidean distance.

Most advancements on *k-NN* change the method of calculating neighbours. None of them consider the use of more than just the nearest neighbours to classify instances. The distance between an unseen instance and every instance in the training set are calculated but only a small sample, defined by *k*, is used. The Density Based k-NN (*DB-kNN*) algorithm is a notable classification technique that evaluates the neighbours rather than just counting them. Because of this it has been labeled an 'important step forward' [12]. Nearest neighbour classifications are considered lazy [9], even with weight adjustments and different values of *k*. Using more neighbours, such as the farthest, may increase the knowledge base of classification. As there is no considerable research into the use of farthest neighbours, there is also no defined method of using them.

4 K-NFN Classification

In our novel algorithm the nearest *k* neighbours are not the only factor in the classification of an unseen instance. A new integer value labeled *fk* is introduced (also *k* is renamed *nk*). *fk* represents the number of farthest neighbours to include. A farthest neighbour is a record considered to be the farthest in distance from the instance. The Euclidean distance function is still used to calculate the distances.

The farthest neighbours are being considered in this paper because they may be used to identify vague classes or may be useful in deciding close decisions. If one class label beats another by only 1 appearance but also appears several times in the

farthest instances, it may be more accurate to classify the new instance with the second class label. In this situation this may be correct because the first class label is used on similar and dissimilar instances which show no real connection between the common attributes of the neighbours, to the class.

Algorithm 1: Neighbours(*data*: training data set, *nk*: integer of nearest neighbours, *fk*: integer of farthest neighbours, *r*: new data record)
begin
  for each instance *i* in *data* **do**
    calculate distance(*i*, *r*) as *D* on their attributes;
    store *D* in *arrayOfDistances*;
  endfor;
  Order *arrayOfDistances* from lowest to highest;
  Find *nk* nearest neighbours from *arrayOfDistances* as *nn*;
  Find *fk* farthest neighbours from *arrayOfDistances* as *fn*;
  r.Class = result of vote(*nn*,*fn*) using either voteOriginal, voteRank or voteSpectrum;
end;

Once the farthest neighbours have been found there are several ways they can be used in classification. In this paper we have tested 3 of those methods on several data sets and compared the results to *k-NN*. The new algorithm is labeled *k-NFN* (*k-Nearest & Farthest Neighbours*). The first method of *k-NFN* is considered the original *k-NFN*. 2 further algorithms were developed as extensions to the original. The same extensions were applied to *k-NN* to create similar *k-NN* functions which were experimented on so the use of farthest neighbours can be compared for each method. The base of the *k-NFN* algorithm is described in Algorithm 1.

Three different *k-NFN* methods are proposed in this paper. They differ by vote function. *K-NFN Original* uses *voteOriginal*, *k-NFN Rank* (*k-NFNR*) uses *voteRank* and *k-NFN Spectrum* (*k-NFNS*) uses *voteSpectrum*. The three methods are discussed in the following subsections.

Algorithm 2: voteOriginal (*nn*: Ordered array of the nearest neighbours, *fn*: Ordered array of the farthest neighbours)
begin
  for each instance *i* in *nn* **do**
    find class label *C* of *i*;
    add 1 to *C* occurence in *arrayOfClassLabels;*
  endfor;
  for each instance *j* in *fn* **do**
    find class label *C* of *j*;
    subtract 1 from *C* occurence in *arrayOfClassLabels;*
  endfor;
  return *C* with highest occurrence in *arrayOfClassLabels;*
end;

4.1 K-NFN Original

The voting in the first version of the *k-NFN* algorithm treats the farthest neighbours as opposites to the nearest. This means that for each class that appears as a farthest neighbour, a point is deducted from its nearest total. In practice this would mean that if class A appeared 4 times in the nearest neighbours and 3 times in the farthest neighbours, the total appearance of class A would be considered as 1. The voting system used for *k-NFN* is described in Algorithm 2.

4.2 K-NFNR

In *k-NFNR*, there are changes to how neighbours influence the class decision. A ranking system is used. The nearest neighbours are ordered by their distances and given a rank value depending on the integer *nk*. The very nearest is given the value of *nk*. This value is decremented for every instance that precedes it until the last nearest neighbour is given the value of 1. Table 1 shows how this would work when 5 neighbours are used (*nk* is set to 5), with two class labels denoted A and B.

Table 1. Example of ranking neighbours

Instance	Class	Rank Value
Nearest	A	5
2nd Nearest	B	4
3rd Nearest	B	3
4th Nearest	B	2
5th Nearest	A	1

The rank value is then used to score the classes. Thus, in Table 1, class A has a total value of 6 and class B has a total value of 9. So if classification was applied using just nearest neighbours, class B would be chosen in this example (*k-NN Ranking*).

The *k-NFN Ranking* method applies the ranking to nearest neighbours and then applies the same technique to the farthest neighbours. The farthest instance is given a rank value of *fk* so that the farthest has the greatest influence. Each preceding instance is then given a value based on its position. Table 2 shows an example of this.

Table 2. Example of ranking Farthest Neighbours

Instance	Class	Rank Value
Farthest	B	5
2nd Farthest	A	4
3rd Farthest	A	3
4th Farthest	B	2
5th Farthest	A	1

The rank values of the farthest neighbours are totaled and subtracted from the values of the nearest for each class. Thus if an unseen instance being classified had the

nearest neighbours in Table 1 and farthest from Table, 2 it would be classified as follows:

1. Class A has a value of 6 (5+1) from nearest neighbours and a value of 8 (4+3+1) from the farthest. Applying nearest – farthest gives Class A the value of -2 (6-8).
2. Class B has a value of 9 (4+3+2) from nearest neighbours and a value of 7 (5+2) from the farthest. Applying nearest – farthest gives Class B the value of 2 (9-7).
3. As class B has a higher value than class A, the instance would be classified under class B.

The algorithm for *voteRank* is given in Algorithm 3.

Algorithm 3: voteRank (*nn*: Ordered array of the nearest neighbours, *fn*: Ordered array of the farthest neighbours)

```
begin
    rankScore := nn.length;
    for each instance i in nn do
        find class label C of i;
        add rankScore to C occurence in arrayOfClassLabels;
        rankScore := rankScore-1;
    endfor;
    rankScore := fn.length;
    for each instance j in fn do
        find class label C of j;
        subtract rankScore from C occurence in arrayOfClassLabels;
        rankScore := rankScore-1;
    endfor;
    return C with highest occurrence in arrayOfClassLabels;
end;
```

Algorithm 4: plotSpectrum (*nn*: Instance of nearest neighbour, *fn*: Instance of farthest neighbour, *neighbour*: neighbour to be plotted)

```
begin
    spectrumLength := 2;  nd := nn.distance;   fd := fn.distance;
    n := neighbour.distance;
    //First calculate distance between Nearest and Farthest
    totalDistance := |fd – nd|;
    //If nearest n being plotted, find distance of n compared to nearest else find
    //distance of neighbour to farthest
    if n = a nearest neighbour then distance = |n- nd|
    else distance = |n - fd|;  end if;
    // Find this distance as a percentage of the gap between nearest and farthest
    pd := distance / totalDistance * 100;
    // Find the distance on spectrum by using percentage on spectrum length
    sd := spectrumLength / 100 * pd;
    //If plotting nearest neighbour, get neighbour position by subtracting distance
    //from 1 else add distance to -1
    if n = a nearest neighbour then return 1-sd;
    else return -1+sd; end if;
end;
```

4.3 K-NFNS

The last extension of the original *k-NFN* algorithm presented in this paper creates a spectrum-like measurement based on the single nearest and the single farthest neighbours (*k-NFNS*). All other neighbours are then plotted on the spectrum based on how near or far they are to the unseen instance.

The single nearest neighbour is given the value of 1. The single farthest neighbour is given the value of -1. Each of the nearest neighbours up to the value of *nk*, are given a value that represents how near they are based on the difference to the nearest distance. This is calculated using a percentage difference. The percentage of a neighbours distance from the nearest or farthest, on the total distance between the nearest and farthest is found and applied to the spectrum length. We can plot a neighbour on the spectrum by using Algorithm 4.

As an example, we take an instance with a nearest neighbour of distance 1.4 and a farthest neighbour of distance 6. To plot the second nearest neighbour of distance 2.1 we follow the steps in Figure 1.

Initial Values: fd = 6. nd = 1.4. n = 2.1. spectrumlength = 2
Distance between Nearest and Farthest: 6 – 1.4 = 4.6
Distance of neighbour to nearest: 2.1 – 1.4 = 0.7
Percentage of difference: $\frac{0.7}{4.6} * 100 = \ 15.2173$ ....
Find distance on spectrum from nearest: $\frac{2}{100} * 15.2173 \ldots . = \ 0.3043$ ....
Plot neighbour by subtracting from 1: 1 – 0.3043.... = 0.6957 (4dp)

Fig. 1. Plotting neighbour with 2.1 distances on a spectrum

Thus the second nearest neighbour in this example will have a position of 0.6957 on the spectrum. Its position is shown on the spectrum in Figure 2.

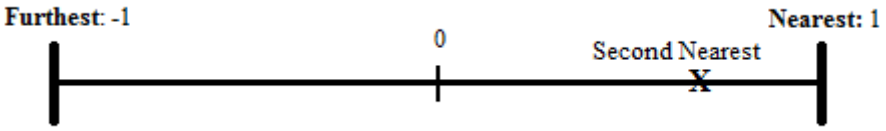

Fig. 2. Plotting neighbour with distance of 2.1 on a spectrum diagram

The position on the spectrum is the value used to total up appearances of a class. So in the example of Figure 2, if both the nearest neighbour and second nearest neighbour belong to class A then the value of class A would be 1+0.6957.

Our motivation to use the spectrum is that if the nearest neighbours are not that close in proximity to the instance, they should have less influence on the classification. For farthest neighbours, it is the case that neighbours that are not very far should have less influence. The voting system in *k-NFNS* is given in Algorithm 5.

```
Algorithm 5: voteSpectrum (nn: Ordered array of the nearest neighbours,
                           fn: Ordered array of the farthest neighbours)
begin
    nearest := first instance in nn;
    farthest:= first instance in fn;
    for each instance i in nn do
        find class label C of i;
        spectrumPoint := plotSpectrum(nearest, farthest, i);
        add spectrumPoint to C occurence in arrayOfClassLabels;
    endfor;
    for each instance j in fn do
        find class label C of j;
        spectrumPoint := plotSpectrum(nearest, farthest, j);
        add spectrumPoint from C occurence in arrayOfClassLabels;
    endfor;
    return C with highest occurrence in arrayOfClassLabels;
end;
```

5 Experimental Results

We implemented the three variations of the *K-NFN* algorithms and their *K-NN* counterparts using the Java programming language. Then we compared their accuracy using real data sets with a 10 fold cross validation technique [14]. The experimental study in this paper has two targets:

1. To evaluate the performance of *K-NFN* algorithms against *K-NN*.
2. To assess how the difference in *nk* and *fk* values may affect classification

All the data sets used in this paper have been retrieved from the UCI data repository [15]. Table 3 displays the details of the data sets used. Variations in the properties of the data in terms of size, dimensionality and number of classes have been the factors we used to choose the data sets for our experimental study.

Table 3. Data set details

Data set	No. Of Attributes	No. Of Instances	No. Of Classes
Iris	4	105	3
Haberman	3	306	2
Balance Scale (BS)	4	625	3
Tic-Tac-Toe (TTT)	9	958	2
Heart	44	267	2
Glass	9	214	7
Ecoli	7	336	8
Hayes-Roth	5	132	3
Sonar	60	208	2
Breast-Cancer-Wisconsin (BCW)	10	699	2

Each data set was validated *C* times, where *C* is the number of classes within the data set. For each method, the value of *nk* is set to 1 and *fk* to *C* for the first cross fold validation experiment. *nk* is then incremented and *fk* decremented and the set is validated again. This continues until *nk* equals *C* (and *fk* should finish at 1). This cycle of *nk* and *fk* is done for each algorithm on each data set. Table 4 shows the results of the average cross validation accuracy for each algorithm on all the data sets.

Table 4. Table of Cross Validation Accuracy Results as Percentages

Data set	k-NN	k-NNR	k-NNS	k-NFN	k-NFNR	k-NFNS
Iris	95.25926	95.25926	95.18519	95.33333	**95.7037**	95.55556
Haberman	67.25826	68.77778	68.81532	70.34685	**71.61411**	70.16967
BS	79.62847	79.64596	79.51822	81.76505	**81.91542**	81.27347
TTT	74.8605	**81.40504**	81.31749	78.37557	79.19486	79.22858
Heart	64.79167	67.5	**68.33333**	67.5	67.70833	67.29167
Glass	67.15646	**69.07392**	68.62313	60.75646	60.44717	63.64898
Ecoli	78.48769	78.48422	76.51263	**78.84059**	78.14015	76.19066
Hayes-Roth	68.90598	80.25071	**80.62108**	67.04274	73.66952	71.85755
Sonar	84.57184	85.67241	**86.29598**	85.59483	81.12356	85.14368
BCW	**95.91879**	95.66226	95.42958	95.29994	95.27273	95.18804

In Table 4, there are cases where a *k-NFN* based method consistently provides better accuracy than *k-NN*, notably in the Haberman and Balance Scale data sets. It can be observed that in 40% of the used data sets, incorporating farthest neighbours have enhanced the performance of the classification over all variations of k-NN. Furthermore, our proposed voting systems have proved their efficiency by having the highest accuracy of classification for 80% of the data sets.

In the balance scale data set, the biggest difference between a *k-NFN* method and a *k-NN* is when the ranking vote system is used, i.e., *k-NNR* & *k-NFNR*, as shown in Figure 3. The ranking system also proves to be noteworthy in the Haberman data set.

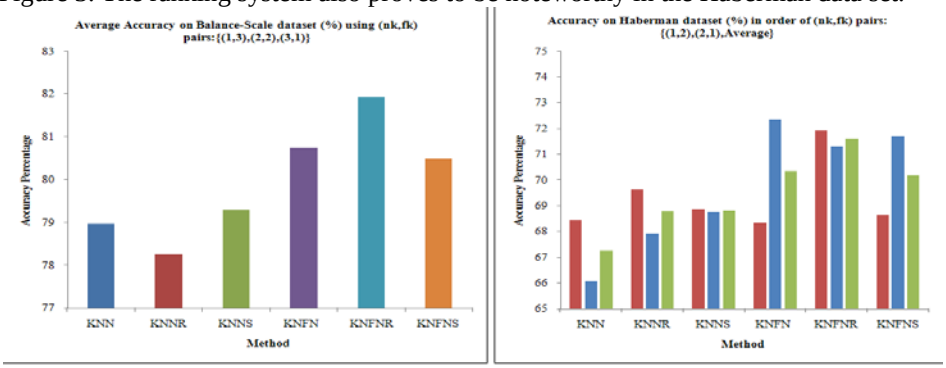

Fig. 3. Graphs for Balance Scale and Haberman data set

The graph to the left in Figure 3 shows the average accuracy of the Balance Scale data set. The graph to the right breaks down the average accuracy on the Haberman data set. For each method, the left bar represents a (*nk*,*fk*) pair of (1,2), the middle

represents (2,1) and the last is the average across the two. The *k-NFN* based methods on average outperform *k-NN* based ones. We can see how the values of *nk* and *fk* affect the accuracy and not just the methods. For example, using 1 nearest neighbour outperforms the use of 2 in *k-NN* original. However in *k-NFN*; the use of 2 nearest outperforms the use of 1, when combined with 1 farthest. Thus, using a farthest neighbour makes up for the reduction in the accuracy of *k-NN*, when using 2 nearest. Here we can see a farthest neighbour can greatly improve classification for the Haberman data set. Similar patterns are noticeable in *k-NFNS* and *k-NNR*.

The importance of selecting a value *k* is a recognised problem when using *k-NN* [12], which is inherited by *k-NFN*. In fact, with *k-NFN* the difference between *nk* and *fk* must be considered. This is noticeable in the Glass data set. On average, *k-NN* based methods outperform *k-NFN*. However the average of *k-NFN* is greatly affected by the use of 1 nearest neighbour and 7 farthest. Table 5 shows the cross validation accuracy of the 6 algorithms on the Glass data set.

Table 5. Cross Validation Accuracy for the Glass Data Set

nk	KNN	KNNRank	KNNSpec	nk	fk	KNFN	KNFNRank	KNFNSpec
1	68.44444	**69.07937**	68.46984	**1**	**7**	44.74921	45.35873	46.65397
2	66.34286	**67.99365**	67.69524	**2**	**6**	58.38095	47.78413	64.65397
3	69.28254	**70.21587**	69.93016	**3**	**5**	64.8381	56.09524	65.95556
4	69.70794	69.33968	**69.86667**	**4**	**4**	62.40635	65.95556	67.65079
5	66.90794	67.75873	69.60635	**5**	**3**	66.74286	**69.89206**	65.70159
6	65.07302	**69.58095**	66.27302	**6**	**2**	65.1873	69.20635	66.91429
7	64.33651	**69.54921**	68.52063	**7**	**1**	62.99048	68.8381	68.0127

The *k-NFN* methods perform worse when farthest neighbours are high. As the difference between *nk* and *fk* is reduced, the accuracy between *k-NN* and *k-NFN* become similar. The glass data set results also show that the original *k-NN* algorithm is outperformed by the proposed voting systems; rank and spectrum. The rank system is more accurate in most cases. The use of 7 neighbours produces the largest difference in accuracy between *k-NN* original and the new voting systems.

6 Conclusion

In this paper, we have proposed and developed a *k-NFN* classification algorithm with three different methods of voting. Our hypothesis is that using farthest neighbours to aid the nearest in identifying has improved the accuracy of classification. Once neighbours have been selected their classes are given a score from the vote system and the class with the greatest score at the end is selected. The first form of voting gives a +1 value for nearest neighbours' classes and -1 for farthest. The second system ranks the neighbours in order, so the single nearest and the single farthest neighbour have more influence. The third system is similar in that the nearest and farthest have a greater score. It uses the nearest and farthest as the minimum and maximum of a spectrum and all other neighbours are plotted in relation to them.

Experimental results show that the *k-NFN* algorithm can provide more accurate classification than *k-NN* in 40% of the data sets. Moreover, our proposed voting methods have shown their superiority by increasing the classification accuracy over *k-NN* for 80% of the data sets. From this, we plan for future work to include research into the affect of the difference between the nearest-k and the farthest-k. The discovery of the optimum values may improve the classification for even those data sets that were not improved in this paper.

References

1. Bramer, M.: Principles of Data Mining, Springer Verlag, New York, March 2007.
2. Friedl, M.A., Brodley, C.E.: Decision Tree Classification of Land Cover from Remotely Sensed Data. In: Remote Sensing of Environment, vol. 61, pp. 399-409. Elsevier Science Inc, New York (1997)
3. Kamber, M., Winstone, L., Gong, W., Cheng, S., Han, J.: Generalization and Decision Tree Induction: Efficient Classification in Data Mining. In: Research Issues in Data Engineering, pp. 111-120. Birmingham (1997)
4. Apte, C., Weiss, S.: Data mining with decision trees and decision rules. In: Future Generation Computer Systems 13, vol. 13, pp. 197-210. Elsevier Science Inc, New York (1997)
5. Quinlan, J.R.: Decision Trees and Decision making. In: Systems, Man and Cybernetics, IEEE Transactions, vol. 20, pp. 339-346, (1990).
6. Shah, J.K., Smolenski, B.Y., Yantorno, R.E., Iyer, A.N.: Sequential *k*-Nearest Neighbor Pattern Recognition for Usable Speech Classification. In: European Signal Processing Conference (2004), pp. 741-744.
7. Wu, X., Kumar, V., et al: Top 10 algorithms in data mining. In: Knowledge and Information Systems, vol. 14, pp. 1-37. Springer, London (2008).
8. Laaksonen, J., Oja, E.: Classification with learning k-nearest neighbors. In: Neural Networks, 1996., IEEE International Conference on , vol.3, pp. 1480-148. IEEE, Washington
9. Guo, G., Wang, H., Bell, D., Bi, Y., Greer, K.: KNN Model-Based Approach in Classification. In: On The Move to Meaningful Internet Systems 2003: CoopIS, DOA, and ODBASE, vol. 2888, pp. 986-996. Springer, Berlin.
10. Abdallah, Z., Gaber, M.: KB-CB-N classification: towards unsupervised approach for supervised learning. In: Proceedings of the IEEE Symposium on Computational Intelligence and Data Mining (CIDM 2011), Paris, France (2011).
11. Liangxiao, J., Zhihua, C., Dianhong, W., Siwei, J.: Survey of Improving K-Nearest-Neighbor for Classification. In: Fuzzy Systems and Knowledge Discovery, 2007. FSKD 2007. Fourth International Conference on , vol.1, pp.679-683
12. Voulgaris, V., Magoulas, G.D.: Extensions of the k Nearest Neighbour Methods for Classification Problems. In: AIA '08 Proceedings of the 26th IASTED International Conference on Artificial Intelligence and Applications. ACTA Press Anaheim, CA (2008)
13. Zhang, B., Srihari, S.N.: Fast k-Nearest Neighbor Classification Using Cluster-Based Trees. In: IEEE Transactions On Pattern Analysis And Machine Intelligence, vol. 26, no. 4, pp. 525-52. IEEE (2003)
14. Schaffer, C.: Selecting a classification method by cross-validation. In: Machine Learning, vol. 13, no. 1, pp. 135-143. Springer, Netherlands (2001)
15. Frank, A., Asuncion, A.: UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science. (2010).