Optimizing the *k*-NN Metric Weights Using Differential Evolution

Akram AlSukker, Rami Khushaba, and Ahmed Al-Ani
University of Technology, Sydney, PO Box 123, Broadway, NSW 2007, Australia
{alsukker, rkhushab, ahmed}@eng.uts.edu.au

Abstract

*Traditional k-*NN *classifier poses many limitations including that it does not take into account each class distribution, importance of each feature, contribution of each neighbor, and the number of instances for each class. A Differential evolution (DE) optimization technique is utilized to enhance the performance of k-*NN *through optimizing the* metric weights of features, neighbors and classes. *Several datasets are used to evaluate the performance of the proposed DE based metrics and to compare it to some k-*NN *variants from the literature. Practical experiments indicate that in most cases, incorporating DE in k-*NN *classification can provide more accurate performance.*

1. Introduction

The *k* nearest neighbor (*k*-NN) classifier is considered as one of the most widely used techniques in machine learning [1-6]. In a simple *k*-NN algorithm the distances between the testing pattern and all training patterns are calculated and then a voting criterion is applied using the class label of nearest *k* training vectors to obtain the output classes. The number of neighbors, *k*, should be chosen carefully, where based on the distribution of classes in the feature space, a certain value of *k* may give good results for one classification problem and fail for another.

Cover & Hart [7] have shown that when *k/N* approaches infinity, the optimal Bayes error rate can be obtained, where *N* represents the number of patterns. Hence, traditional *k*-NN classifier can provide good results when dealing with large dataset with evenly distributed patterns among different classes [8, 9]. Unfortunately, in most real life applications this is not always the case. As a result, different variants of *k*-NN were developed to overcome the traditional *k*-NN limitations [8-10]. Most these modifications lie into two categories: modifying the distance measure or proposing a weighting measure; where different weights can be assigned to the neighbors, classes, features or a combination of all of these categories.

In this paper, four versions of *k*-NN variants optimized with differential evolution (DE) optimization technique are implemented and evaluated according to their class-wise accuracy.

The paper is structured as follows: Section 2 gives an overview of *k*-NN classification algorithms. In Section 3, optimization of *k*-NN weights using Differential Evolution is explored. Experimental results are presented in Section 4, and a conclusion is given in Section 5.

2. An Overview of *k*-NN Classification Algorithms

A k-nearest neighbor (*k*-NN) classifier depends on the closest training example to predict the unknown class labels, where only a specific number of closest neighbors (*k*) are taken into account for this purpose. The nearest neighbors can be found by means of distance measures, which can be as simple as the Euclidean distance. The Euclidean distance *d* between two points $X(x_1,\ldots x_n)$ and $Y(y_1,\ldots y_n)$ is given by the following formula:

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

Other distance based measures can also be used, for example the Minkowski distance, which is a more general measure that can be expressed as:

$$d_{x,y} = \sqrt[\lambda]{\sum_{i=1}^{n}\left|x_i - y_i\right|^{\lambda}} \qquad \ldots (2)$$

Note that the Euclidean distance is a special case of the Minkowski distance where $\lambda=2$, while the Manhattan distance is obtained by assigning $\lambda=1$.

Voting can be implemented using either un-weighted or weighted scenarios. In un-weighted voting class labels are assigned according to the simple majority vote, where all neighbors have the same weight. In weighted voting the weight assigned to the neighbor *i* given as (w_i) is proportional to its distance from the test sample, which can be implemented as follows [10]:

$$w_i = \begin{cases} \dfrac{d(x_k,x) - d(x_i,x)}{d(x_k,x) - d(x_1,x)} & if\ , d(x_k,x) \neq d(x_i,x) \\ 1 & if\ , d(x_k,x) \neq d(x_i,x) \end{cases} \quad ..(3)$$

Where x_1 and x_k represents the nearest and farthest *k* neighbors to the current pattern x_i respectively. It has been found that when dealing with limited number of samples, *k*-NN may not give optimal results. Moreover *k*-NN usually gives poor performance when dealing with unbalanced data [8] (training patterns are not evenly distributed among data). Consequently variants of *k*-NN were developed to overcome its weakness.

3. Differential Evolution (DE)

Differential Evolution (DE) is a simple, parallel, direct search, and easy to use optimization method having good convergence and fast implementation properties [11]. The crucial idea behind DE is a new scheme for generating trial parameter vectors by adding the weighted difference vector between two population members to a third member. The following equation shows how to combine three different, randomly chosen vectors (X, X_a and X_b) to create a mutant vector X':

$$X' = X + F(X_a + X_b) \qquad \ldots (4)$$

Where $F \in (0, 1)$ is a scale factor that controls the rate at which the population evolves. In addition, DE employs a uniform crossover, also known as discrete recombination, in order to build trial vectors out of parameter values that have been copied from two different vectors. In particular, DE crosses each vector with a mutant vector Y:

$$U_{j,i} = \begin{cases} X'_{j,i} & \text{if } rand(0,1) \leq Cr \\ Y_{j,i} & Otherwise \end{cases} \qquad \ldots (5)$$

The crossover probability $C_r \in [0,1]$ is a user defined value that controls the fraction of parameter values that are copied from the mutant. If the newly generated vector results in a lower objective function value (better fitness) than the predetermined population member, then the resulting vector replaces the vector with which it was compared. More description about DE can be found in [11].

Four different types of weight have been evaluated using DE, the length of the string is based on the selected method as follow:

- **Feature weighting (DE1)**: each member of the population is represented by a weight matrix of a length *f*, $W(w_1,w_2,\ldots w_f)$, where *f* represents the total number of available features. Calculating the new distance measure is done as shown in eq. (6) [12].

$$d_{x,y} = \sqrt{\sum_{i=1}^{n} w_i (x_i - y_i)^2} \qquad \ldots (6)$$

- **Neighbor weighting (DE2)**: each member of the population is represented by a weight matrix of a length *k*, $W(w_1,w_2,\ldots w_k)$, where *k* represents the total number of nearest neighbor. A weighted voting is then applied to assign a class label to each sample.

- **Class weighting (DE3)**: each member of the population is represented by a weight matrix of a length *c*, $W(w_1,w_2,\ldots w_c)$, where *c* represents the total number of classes. After finding the *k* nearest neighbor a weighted distance sum WS_c is obtained for each class *c* as in eq (7). For a given test pattern, the class label with the minimum weighted sum will be chose.

$$WS_c = \sum_{j=1}^{k} \delta w_c \, d_j \qquad \ldots (7)$$

Where d_j represents the distance from neighbor *j*.

$$\delta = \begin{cases} 1 & d_j \in c \\ 0 & d_j \notin c \end{cases} \qquad \ldots (8)$$

- **Hybrid weighting (DE4)**: each member of the population is represented by a weight matrix of a length *f*+*c*, $W(w_1,w_2,\ldots w_{f+c})$, where the first *f* elements represent the feature weights and the last *c* elements represent the class weights. This method is implemented by applying a feature weighting step, which will be followed by class weighting.

4. Experimental Results

Two sets of experiments were conducted. In the first one, 2-class synthetic data was specifically designed to generate different class ratios. Patterns were distributed between the two classes in two different scenarios, as shown in Fig. 1. The reason behind using the second data is to consider the case when the patterns of one class are concentrated in a certain region, which will have an effect on the identification of the nearest neighbors.

When implementing the DE based *k*-NN the objective function was set to maximize the average

classification accuracy of the two classes. This is justified by the fact that some methods might provide a high classification performance for one of the classes and a relatively low performance for the other. The *k*-NN variants that were considered here include: majority voting *k*-NN (MVKNN), weighted *k*-NN (WKNN) [13], pseudo nearest neighbor (PNN) [10] and neighbor-weighted *k*-NN (NWKNN) [8]. These methods were all ran only once due to their static output.

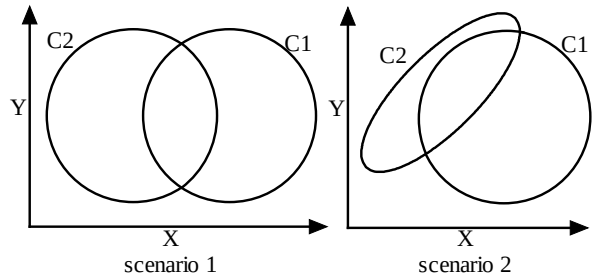

Fig. 1. Two class synthetic data distributed using two scenarios.

For each scenario, data was generated 30 times and the average class-wise accuracy and standard deviation were calculated. The number of neighbors (*k*) was set to 5, which was found to provide, on average, good performance for all methods.

Results for running the proposed DE based *k*-NN in comparison with other *k*-NN variants are shown in Figs. 2, 3 and 4. For scenario 1, the performance of all methods is comparable with DE-based *k*-NN achieving slightly better results, especially DE2.

For scenario 2, the first four methods provided high standard deviation, as shown in Figs. 3 and 4. On the other hand, the variation is much lower for the DE-based *k*-NN, except for DE4 (the hybrid case). This can be due to the increased number of parameters that needed to be optimized using limited number of samples. The good performance that was achieved be DE1, DE2 and DE3 indicate that even when using small number of samples (200 and 150 respectively), an improved performance can be achieved.

It is worth mentioning that for the unbalanced case of scenario 2, the first four *k*-NN variants tend to give very low classification accuracy for the under-represented class and higher accuracy for the over-represented class, which reduces the average class-wise accuracy.

In the second experiment, a number of datasets from the UCI Machine Learning Repository were considered (as shown in Table I). The data from each set was divided into three divisions: the first two were used for training and validation, while the third one for testing. The reported UCI results represent an average of ten runs for the DE based *k*-NN, as the weight vector can be different for each run.

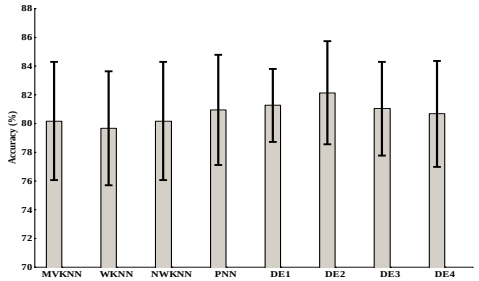

Fig. 2. Classification accuracy of the synthetic data generated using scenario 1 (2 classes, 2 features, 100 training points for each class).

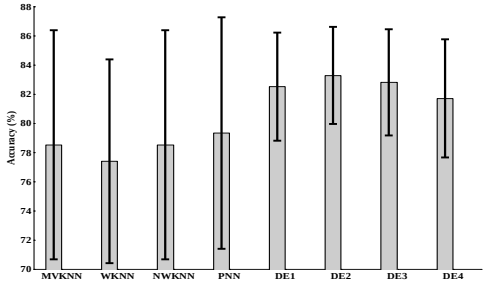

Fig. 3. Classification accuracy of the synthetic data generated using scenario 2 (2 classes, 2 features, 100 training points for each class).

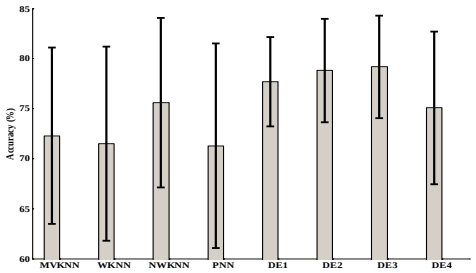

Fig. 4. Classification accuracy of the synthetic data generated using scenario 2 (2 classes, 2 features, 100 training points for the first class & 50 for the other).

TABLE I
DATASET DESCRIPTION

Name	No. of Patterns	No. of Attributes	No. of Classes
Pima	768	8	2
Cancer	683	9	2
Wdbc	569	30	2
Ion	351	34	2
Sonar	208	60	2

Table II shows the classification results of the different *k*-NN variants. The first thing that can be noticed is that DE1 and DE4 achieved better accuracy than other methods. This clearly indicates that assigning appropriate weights to both features and classes can lead to better performance. Note that with sufficient number of training samples to optimize the

weights for both features and classes (all datasets except sonar), DE4 achieved the highest performance when compared to all other methods. On the other hand, DE3 did not perform that well in all datasets, except the ionosphere, which represents an unbalanced dataset. We also have to mention that DE2 achieved the worst results compared to the other three DE-based *k*-NN variants. This indicates that only assigning weights to the neighbors may not be sufficient.

5. Conclusion

This paper presented a new approach to optimizing the weight metric of the *k*-NN classifier. The differential evolution was used to optimize the weights of features, nearest neighbors, and classes. Results conducted on both synthetic and real-life data indicated that improved performance can be achieved when considering the traditional *k*-NN and number of its variants. Results have in particular suggested that optimizing the weights of both features and classes, when have sufficient number of samples, can lead to better performance for both balanced and imbalanced data.

6. References

[1] T. Ananthakrishna, K. Shama, and U. C. Niranjan, "k-means nearest neighbor classifier for voice pathology," presented at India Annual Conference, 2004. Proceedings of the IEEE INDICON 2004. First, 2004.

[2] V. Athitsos, J. Alon, S. Sclaroff, and G. Kollios, "BoostMap: An Embedding Method for Efficient Nearest Neighbor Retrieval," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 30, pp. 89-104, 2008.

[3] E. Blanzieri and F. Melgani, "Nearest Neighbor Classification of Remote Sensing Images With the Maximal Margin Principle," *Geoscience and Remote Sensing, IEEE Transactions on*, vol. 46, pp. 1804-1811, 2008.

[4] D. Masip and J. Vitria, "Shared Feature Extraction for Nearest Neighbor Face Recognition," *Neural Networks, IEEE Transactions on*, vol. 19, pp. 586-595, 2008.

[5] A. J. Nor'aini, P. Raveendran, and N. Selvanathan, "Human Face Recognition using Zernike moments and Nearest Neighbor classifier," presented at Research and Development, 2006. SCOReD 2006. 4th Student Conference on, 2006.

[6] L. Yue and T. Chew Lim, "Improved nearest neighbor based approach to accurate document skew estimation," presented at Document Analysis and Recognition, 2003. Proceedings. Seventh International Conference on, 2003.

[7] T. Cover and P. Hart, "Nearest neighbor pattern classification," *Information Theory, IEEE Transactions on*, vol. 13, pp. 21-27, 1967.

[8] S. Tan, "Neighbor-weighted K-nearest neighbor for unbalanced text corpus," *Expert Systems with Applications*, vol. 28, pp. 667-671, 2005.

[9] R. Paredes and E. Vidal, "Learning weighted metrics to minimize nearest-neighbor classification error," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 28, pp. 1100-1110, 2006.

[10] Y. Zeng, Y. Yang, and L. Zhao, "Pseudo nearest neighbor rule for pattern classification," *Expert Systems with Applications*, vol. In Press, Corrected Proof.

[11] K. V. Price, R. M. Storn, and J. A. Lampinen, *Differential Evolution: A practical approach to global optimization*: Springer, 2005.

[12] S. Qiang, L. Lv, and H. Chen, "Optimization of K-NN by feature weight learning," presented at Machine Learning and Cybernetics, 2005. Proceedings of 2005 International Conference on, 2005.

[13] S. A. Dudani," The Distance-Weighted k-Nearest-Neighbor Rule*", IEEE Transactions on Systems, Man and Cybernetics, Vol.6, No. 4, pp. 325–32,1976.*

TABLE II
DATASET RESULTS

Name	Method	Overall Acc.	Class 1 Acc.	Class 2 Acc.	Avg. class-wise Acc.
Pima					
	MVKNN	75.52	84	59.70	71.85
	WKNN	71.88	76	64.18	70.09
	NWKNN	74.48	72	79.10	75.55
	PNN	72.40	77.6	62.69	70.14
	DE1	81.51	88.67	69.58	79.92
	DE2	70.31	84.17	47.22	67.23
	DE3	69.27	60.83	83.33	71.15
	DE4	81.04	75.83	89.72	82.20
cancer					
	MVKNN	97.06	96.55	98.15	97.25
	WKNN	95.88	96.55	94.44	95.63
	NWKNN	97.06	96.55	98.15	97.25
	PNN	95.88	96.55	94.44	95.63
	DE1	99.88	99.90	99.85	99.88
	DE2	96.47	100	90.91	95.79
	DE3	99.41	100	98.49	99.30
	DE4	99.93	99.94	99.96	99.95
wdbc					
	MVKNN	95.07	87.93	100	93.97
	WKNN	95.78	89.66	100	94.83
	NWKNN	95.07	87.93	100	93.97
	PNN	95.07	87.93	100	93.97
	DE1	98.87	96.98	99.70	98.52
	DE2	96.48	95.35	96.97	96.27
	DE3	97.18	95.35	97.98	96.84
	DE4	98.80	96.05	100	98.28
Ion					
	MVKNN	86.21	100	63.64	81.82
	WKNN	90.81	98.15	78.79	88.47
	NWKNN	88.51	98.15	72.73	85.44
	PNN	89.66	100	72.73	86.36
	DE1	92.87	100	77.04	89.97
	DE2	87.13	100	58.52	81.88
	DE3	95.40	98.33	88.89	94.22
	DE4	98.85	98.33	100	99.06
sonar					
	MVKNN	82.69	66.67	93.55	80.11
	WKNN	90.39	80.95	96.77	88.86
	NWKNN	82.69	66.67	93.55	80.11
	PNN	88.46	76.19	96.77	86.48
	DE1	95.39	92.14	99.17	95.57
	DE2	73.08	67.86	79.17	73.37
	DE3	84.62	82.14	87.50	84.75
	DE4	90.19	90.36	90	90.18