A Nearest Neighbor Approach to Letter Recognition

Aiyuan Ji
Department of Computer information Science
Clark Atlanta University
Atlanta, GA 30314, USA
aji@cau.edu

Roy George
Department of Computer information Science
Clark Atlanta University
Atlanta, GA 30314, USA
rgeorge@cau.edu

ABSTRACT

The nearest neighbor classifier (NNC) is a non-parametric classification technique that classifies a test pattern to the class of its nearest neighbor in the training data. In this research, we applied the NNC to a standard letter recognition data and obtained a superior classification rate in comparison to extant approaches. A prime drawback to the NNC technique has been the relative inefficiency of the model. A modified NNC was implemented and applied to the same recognition problem. It is found that if we choose a suitable threshold minimum difference for classification, we can reduce the CPU time by half without lowering the performance of the classifier.

1. INTRODUCTION

The nearest neighbor classifier (NNC) is used to classify a test pattern to the class of its nearest neighbor in the training data. NNC is a simple and common-used non-parametric classifier. It has been widely used in pattern recognition and machine learning, e.g., handwritten character recognition[1,2], fingerprint recognition[3], face recognition[4], and image classification[5].

The UCI letter recognition database has been widely used for testing machine learning algorithms[6]. In this study, we applied the NNC to the UCI letter recognition data set. To improve the efficiency of the NNC, we also proposed a modified NNC. The results of the application of the basic and modified NNC algorithms are presented and the improvements in efficiency and CPU time are discussed.

2. DATA

The dataset used in this study is called the "Letter Recognition Database" from the Machine Learning Repository at the University of California, Irvine. The dataset consists of 20000 total exemplars of the 26 letters of the English alphabet in the upper case. The letters are derived from 20 fonts that are randomly distorted to form black-and-white images. Each image is analyzed to produce an exemplar of 16 numerical attributes. Detailed information of these attributes can be found in UCI ML website[7].

3. METHOD

ACM SE'06, March, 10-12, 2006, Melbourne, Florida, USA

3.1 Data separation

In this study, we randomly picked 80% (i.e., 16000) data points from the original data to form a training dataset, and the left 20% (i.e., 4000) data points to form the test dataset. Cross validation on the technique was performed using the 10-fold test.

3.2 Nearest Neighbor Classifier (NNC)

The Nearest Neighbor Classifier (NNC) searches all training data points to find a training point with the minimum difference between the training point and the test point, and assign the class of the training point to the test point. *X[Ntrn][M]* is the training data, where *Ntrn* is the total number of the training data points, and *M* is the number of attributes. *Y[Ntest][M]* is the test data, where *Ntest* is the total number of the test data points. The difference between a test point *X[i][M]*, and a training point *Y[j][M]* is computed as: $diff_{i,j} = \frac{1}{M}\sqrt{\sum_{k=1}^{M}(X[i][k]-Y[j][k])^2}$.

The Basic NNC Algorithm:

---

for (each test point)

{Initiate *min-diff* to be a maximum possible number}

for (each training point) compute *diff* based on equation (1)

if (*diff*<*min_diff*) then

min_diff=diff

min_diff_index = index of training point

end for

class of the test point = class of the training point (*min_diff_index*)

end for

3.3 A Modified NNC Algorithm

According to the NNC algorithm discussed in 3.2, we understand that totally *Ntrn*×*Ntest* loops are needed to complete classification of *Ntest* test points using *Ntrn* training points, which are 4000×16000 loops for this study. To improve the classification efficiency, user can choose a threshold minimum difference (TMD) for classification. If *min_diff* is equal to TMD, we stop the loop of searching each training point and assign the class of the training point with the difference equal to TMD to the test point. If TMD is set to be zero, the modified NNC is reduced to the NNC. On the other hand, if TMD is greater than zero, the number of loops and computation time will be reduced. This modification can also influence the classification rate of the classification.

The Modified NNC Algorithm:

for (each test point)

{Initiate *min-diff* to be a maximum possible number}

for (each training point) compute *diff* based on equation (1)

if (*diff*<*min_diff*) then

min_diff=diff

min_diff_index = index of training point

if *(min_diff==TMD} break*

end for

class of the test point = class of the training point (*min_diff_index*)

end for

4. RESULTS AND DISCUSSIONS

We applied the NNC to each pairs of training-test dataset. To investigate the efficiency of the NNC, we also tracked the CPU time used for each classification. The classification rate and CPU time used for each dataset are listed in Table 1. According to Table 1, we can find that the performance and efficiency of the NNC are good, because the classification rates are all above 95.1% and the CPU time is less than 451 seconds.

Table 1. Classification rate and CPU time

	No.1	No.2	No.3	No.4	No.5
Rate	**96.2%**	**96.1%**	**95.9%**	**95.8%**	**96.0%**
CPU (s)	**493**	**450**	**450**	**438**	**439**
	No.6	**No.7**	**No.8**	**No.9**	**No.10**
Rate	**95.7%**	**95.6%**	**95.1%**	**96.0%**	**95.7%**
CPU (s)	**493**	**450**	**450**	**438**	**439**

To investigate the performance and efficiency of our modified NNC, we utilized the modified NNC to classifier training-test dataset No.1. We chose 10 different TMD values between 0 and 1.0. The results are shown in Figure 1.

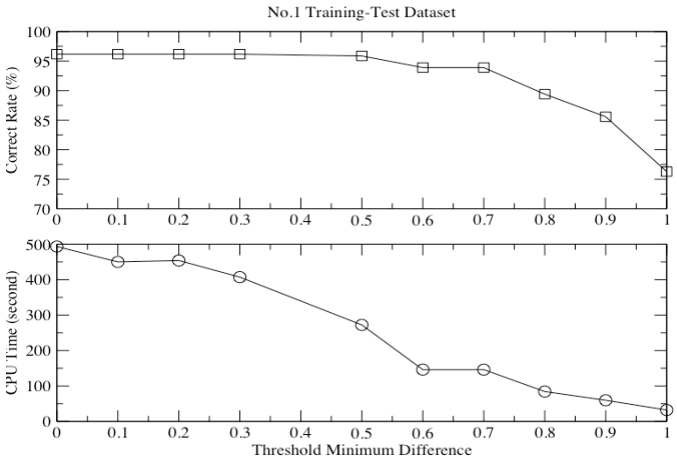

Figure.1 Plots of threshold minimum difference versus classification rate (top) and the threshold minimum difference versus CPU time (bottom)

Figure 1 indicates that as the TMD increases, the classification rate and CPU time all decrease. Figure 1 also shows that as the TMD is 0.5, although the classification rate decreases less than 1%, the CPU time is almost reduced to 50% of CPU time used from TMD=0 (i.e., the original NNC). Therefore, we can improve the efficiency of classification without worsen the performance by using our modified NNC and choosing a suitable TMD value, because the TMD determines the trade off between the correct classification rate and the classification efficiency., i.e., smaller TMD value, higher classification rate and lower efficiency, and verse versa.

5. SUMMARY

Through applying the nearest neighbor classifier (NNC) to the 10-fold CV datasets, we have demonstrated that the NNC provides a good classification rate on the letter recognition database. To improve the classification efficiency further, we proposed a modified NNC and applied the modified NNC algorithm to the same database. It is found that if we choose a suitable threshold minimum difference for classification, we can reduce the CPU time significantly without lowering the performance of the classifier.

In this study, the upper case data was used. Future work will apply the NNC and the modified algorithm to lower case data and upper and lower mixed case data. We will investigate the relationships along all attributes and apply these relationships to assist the classification and hence to improve the classification rate and efficiency.

Acknowledgements

This work is partially supported by NSF Grants DUE-0417079 and HRD-0401679, US Army Contract No: W911NF-04-2-0036, and DOD Grant No: DAAD19-01-2-0014. The content of this work does not reflect the position or policy of the sponsors and no official endorsement should be inferred.

6. REFERENCES

[1] Viswanath P., Murty M.N., and Bhatnagar S., Overlap pattern synthesis with an efficient nearest neighbor classifier, *Pattern Recognition*, 38, 1187-1195, 2005.

[2] Liu C.-L., and Nakagawa, Evaluation of prototype learning algorithms for nearest-neighbor classifier in application to handwritten character recognition. *Pattern Recognition*, 34, 601-615, 2001.

[3] Blue J., et al., Evaluation of pattern classifiers for fingerprint and OCR applications. *Pattern Recognition*, 27, 485-501, 1994.

[4] Wiskott L., Fellous J., Kruger N., and von der Malsburg C., Face recognition by elastic bunch graph matching. *Transactions on Pattern Analysis and Machine Intelligence*, 19, 775-779, 1997.

[5] Hastie T. and Tibshirani R., Discriminant adaptive nearest neighbor classification. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18, 607-615, 1996.

[6] P. W. Frey and D. J. Slate: Letter recognition using Holland-style adaptive classifiers. *Machine Learning*, 6(2), 161-182, 1991.

[7] P.M. Murphy, UCI repository of machine learning databases. Department of Information and Computer Science, UCI, 1994. http://www.ics.uci.edu/mlearn/MLRepository.html.