Reducing the computational demands for nearest centroid neighborhood classifiers

Szymon Grabowski

Computer Engineering Department, Technical University of Lodz,
Al. Politechniki 11, Lodz, 90-924
SGrabow@kis.p.lodz.pl

Abstract. The k Nearest Centroid Neighbor (k-NCN) is a relatively new powerful decision rule based on the concept of so-called surrounding neighborhood. Its main drawback is however slow classification, with complexity $O(nk)$ per sample. In this work, we try to alleviate this disadvantage of k-NCN by limiting the set of the candidates for NCN neighbors for a given sample. It is based on an intuition that in most cases the NCN neighbors are located relatively close to the given sample. During the learning phase we estimate the fraction of the training set which should be examined only to approximate the "real" k-NCN rule. Similar modifications are applied also to ensemble of NCN classifiers, called voting k-NCN. Experimental results indicate that the accuracy of the original k-NCN and voting k-NCN may be preserved while the classification costs significantly reduced.

1 Introduction

The standard k Nearest Neighbors (k-NN) rule is known for half a century, but still there are few classifiers superior in terms of accuracy. One of those rare successful proposals is the idea of *k Nearest Centroid Neighbors* (k-NCN) [9]. As in k-NN, a set of k neighbors of a test sample is selected, and the class with the greatest number of votes among those neighbors defines the output label.

The k nearest centroid neighbors of a query sample q are obtained as follows: the first neighbor of q is its nearest neighbor, n_1; the i-th neighbor, $n_i, i \geq 2$, is such that the centroid (i.e., the mean) c_i of this and previously selected neighbors, $n_1, \ldots, n_{i-1}$, is the closest to q.

Several works have pointed out high accuracy of k-NCN [9, 10, 2, 4]. Its main drawback of k-NCN is however slow classification, with complexity $O(nk)$ per sample. Finding each successive NCN neighbor requires examining all the samples from the learning set, which is often prohibitively costly.

In this paper, we propose a modified decision rule, which limits the set of the candidates for NCN neighbors for each sample. The rate of such reduction and hence the potential classification speedup rate is determined during the learning phase, i.e., is known before the actual classification. We adapt the idea not only to the original k-NCN, but also to its voting variant proposed in our earlier work [5, 4].

2 The Proposed Algorithms

Intuitively, the NCN neighbors should be located close to the query sample. However, the experimental results are somewhat counterintuitive. In [5], we counted the number of distance based neighbors within radius associated with the farthest among the k nearest centroid neighbors. Those nearest neighbor counts are surprisingly high: for example, for a single partition of the Ferrite dataset (to be described in Section 4), within the radius of the ball induced with 10 NCN neighbors there are on average about 140 distance based neighbors. Similar results were obtained on other datasets. This phenomenon has two consequences. First, it suggests that the importance of neighbors' proximity could be overrated. Moreover, if for a given NCN neighborhood so many samples are located closer to the query than the most distant NCN neighbor, then there may exist many ways to "improve" the set of neighbors to predict the label of the query sample [5]. The second conclusion is less optimistic: if NCN neighbors are often quite far from the tested sample, then there is not much hope for a significant speedup of the search process. We however decided to take a closer look at this question.

A basic idea for speeding up the classification of query sample q is to sort the learning set samples according to their distance to q, and then look for NCN neighbors only in a fraction of the learning set containing the closest samples. But what fraction? Datasets vary a lot in their characteristics and consequently any fixed percentage of samples (significantly less than 100%) may be inappropriate for many real datasets. Regarding that, it is advisable to determine the fraction in the learning phase.

The simplest idea is to find the farthest of k NCN neighbors for each sample in the learning set in a leave-one-out manner, calculate its rank according to the distance (for example, the third nearest neighbor would get rank 3) and finally find the maximum m over all those ranks. The ratio $m/(n-1)$, where n is the learning set count, is the desired fraction. Such an idea is however sensitive to the presence of such atypical training samples for which at least one of its k NCN neighbors has a large rank.

We devised a more robust variant of the presented idea. Let us define m_{robust} rank as the minimal value such that for, e.g., 95% samples from the training set the farthest of their k NCN neighbors has its distance rank not exceeding m_{robust}. Now the fraction of closest samples used in the classification phase will be equal to $m_{robust}/(n-1)$, which of course never exceeds the previously discussed fraction $m/(n-1)$. This variant will be called *limited k-NCN v1* in the test section.

Another extension is possible. Instead of finding the proper rank for all k neighbors, such a parameter may be estimated from the training set for each i-th neighbor (i=1, ..., k) separately. We denote this variant with *limited k-NCN v2*.

In an earlier work [4], we proposed the *voting k-NCN* classifier, which is an ensemble of k-NCN components with the decision obtained via plurality voting. The desired diversity in the ensemble is achieved by taking varying counts of neighbors k for the component k-NCN classifiers. The component classifiers are

trained on random partitions of the whole learning set. The learning set is divided L times at random into halves, and one half is used as the "real" training set and the other as the validation set. In this way, L values: $k_1, \ldots, k_L$, are obtained. The classification stage works with the k_i-NCN components, which refer (as opposed to the training phase) to the whole learning set. We set L to 10 in our experiments.

Adapting the neighborhood limiting ideas to the voting k-NCN is straightforward. The rank m_{robust} is now defined as the minimal value such that for, e.g., 95% samples from the training set the farthest of their k_{max} NCN neighbors has its distance rank not exceeding m_{robust}, where k_{max} denotes the maximum among the $k_1, \ldots, k_L$ values. In this way, the variants v1 and v2 for limited voting k-NCN can be defined analogously to the respective variants for the limited k-NCN.

3 Implementation Issues

The set of candidate neighbors for a given query sample q may be obtained via sorting the training set samples according to the distance to q, but in fact full sorting is not necessary. In the limited k-NCN v1, it is enough to divide the learning set into two disjoint subsets of specified size, one such that any sample from this subset is located in the closer or equal distance to q than any sample from the other subset. The well-known *randomized-select* algorithm [1], based on quick sort, serves this purpose.

In the limited k-NCN v2 variant, more "sortedness" is required for the training set. For simplicity, we decided to sort fully the subset of $m_{robust} \cdot n/(n-1)$ nearest samples to q, where m_{robust} is the minimal value such that for f=95% samples from the training set the farthest of their k NCN neighbors has its distance rank not exceeding m_{robust}. As it appears, the computational increase in this phase of classification in v2 is more than compensated with the savings in distance calculations in the NCN search phrase.

During our experiments we noticed quite a surprising phenomenon. A very simple distance calculation trick, namely: rejecting the candidate sample if already after the first half of examined features it may be concluded the sample is located too far (a trick viable for both k-NN and k-NCN), had a major speed impact for the used datasets. For example, for Ferrite dataset and k-NCN, in about 90% of cases the given criterion was satisfied.

Another practical trick for k-NCN-like classification is to avoid multiplications/divisions as the real centroids of found-so-far neighbors are not actually necessary. Instead of the centroid of i neighbors, $i = 1 \ldots k$, we can store their sum (which obviously is the centroid vector multiplied by i) and consequently refer to the query vector multiplied by the same i. In this way, all the distances are scaled, which does not hinder finding the NCN neighbors. Additionally, some temporary values may be cached in k-NCN routines for further speedup of a few percent.

4 Experimental Results

We conducted experiments on two real datasets: one concerning quality control of ferrite cores [7, 8] and one taken from a remote sensing application [11]. For Ferrite dataset ten unbalanced partitions were made: 1400 samples (training sets) and 4503 samples (test sets). The dataset has 30 features and 8 classes. The available remote sensing dataset contained 5124 samples (5 classes, 9 features). Again, ten partitions were used: the learning sets had 1250 samples each time.

The tested classifiers were k-NN, k-NCN, voting k-NCN and the proposed limited k-NCN and limited voting k-NCN in two variants: v1 and v2. As we have mentioned the significant impact of the two simple distance calculation tricks, we decided to compare the classifiers in two implementations: *plain* and *fast* (both described tricks used for k-NCN based classifiers, and only the former one for k-NN). For the voting k-NCN variants, we present the results of the fast implementations only. Learning sessions are identical for k-NCN and its "limited" variants, and so is for the voting k-NCN case. That means the learned values of k are identical for a given task, which is a potential deficiency of our idea.

The average value of k for k-NN over 10 partitions was 4.5 for Ferrites and 10.8 for Remotes. For k-NCN, the respective values of k were 5.9 and 14.2.

In limited k-NCN v1, it was enough to consider on average 16.3% of all samples for NCN candidates for Ferrites. The respective fraction for Remotes was 45.1%. In v2, the fractions for individual i-th ($i = 1 \ldots k$) neighbors vary of course, but on average are much lower. Taking the average fraction for each partition, and again the average over all the partitions, we obtain 5.0% for Ferrites and 14.3% for Remotes.

All the program codes were compiled in Delphi 3.0. The tests were run on an Athlon 2400+ machine with 256 MB RAM under Windows 98 SE. Each 10-partition series was tested several times and the best total timing was taken for the comparison. Tables 1 and 2 present average errors and timings for both datasets. The rightmost column indicates the slowdown factor in comparison to the fast k-NN implementation.

It is easy to notice that the limited k-NCN v2 is much faster than original k-NCN (1.73x for Ferrites and 2.69x for Remotes, comparing the fast implementations) and also achieves (perhaps accidentally) slightly greater accuracies. The results are quite stable over the partitions showing that the ad hoc fraction parameter f=95% was quite a good choice. The limited k-NCN v1 variant was less beneficial, anyway in comparison with original k-NCN it still looks competitively. It is also noteworthy how profitable simple implementation tricks may be; a fair speed comparison of original k-NN and k-NCN was one of the purposes for this work. Nevertheless, k-NN is still much faster than any of the k-NCN based classifiers, so it depends on the application whether the expected accuracy boost with limited k-NCN is worth the additional classification time. This question pertains to the limited voting k-NCN as well, where further improvements in accuracy also imply classification 7-9x slower than k-NN. Still, the v2 variant

Table 1. Ferrites. Classification errors and timings

classifier	error (%)	classif. time (s)	time rel. to k-NN, fast
k-NN, plain	10.96	2.50	2.66
k-NN, fast	10.96	0.94	1.00
k-NCN, plain	10.18	19.57	20.82
k-NCN, fast	10.18	5.99	6.37
limited k-NCN v1, plain	10.17	6.33	6.73
limited k-NCN v1, fast	10.17	4.33	4.61
limited k-NCN v2, plain	10.10	4.30	4.57
limited k-NCN v2, fast	10.10	3.47	3.69
voting k-NCN, fast	9.69	21.70	23.11
limited voting k-NCN, v1, fast	9.66	12.18	12.97
limited voting k-NCN, v2, fast	9.68	6.67	7.10

of limited voting k-NCN is about 3x faster than the original rule at a similar accuracy rate.

Table 2. Remotes. Classification errors and timings

classifier	error (%)	classif. time (s)	time rel. to k-NN, fast
k-NN, plain	21.63	0.38	1.36
k-NN, fast	21.63	0.28	1.00
k-NCN, plain	20.77	7.99	28.54
k-NCN, fast	20.77	3.98	14.21
limited k-NCN v1, plain	20.78	4.40	15.71
limited k-NCN v1, fast	20.78	2.78	9.93
limited k-NCN v2, plain	20.67	2.02	7.21
limited k-NCN v2, fast	20.67	1.48	5.29
voting k-NCN, fast	20.11	7.59	27.50
limited voting k-NCN, v1, fast	20.10	5.37	19.44
limited voting k-NCN, v2, fast	20.19	2.54	9.21

The appropriate value of the fraction f can possibly be estimated from the training set, nevertheless after the preliminary attempts we do not expect a breakthrough in accuracy or speed. Some more experiments are required with the *randomized-select* routine; currently it is implemented in the simplest form.

5 Conclusions and Future Plans

We presented the idea of limiting the set of neighbors for k-NCN decision rules. In practice, such a limitation does not change much in the classification, except for a significant speed improvement. We also presented several simple but surprisingly efficient implementation tricks for NN and NCN based classifiers. As

the next step, we intend to perform the comparison tests on a broader collection of datasets. Of interest should also be to incorporate the presented ideas into several cascade classifiers [4], where NCN components are expected to be triggered on relatively few samples only. It is our hope that the modified schemes will become faster, but the classification accuracy will be preserved.

References

[1] T. H. Cormen, Ch. E. Leiserson and R. L. Rivest: Introduction to Algorithms, MIT Press, 1990.

[2] Sz. Grabowski: Experiments with k-NCN decision rule, IX Konferencja Sieci i Systemy Informatyczne (9th Conf. "Networks and IT Systems"), Lodz, Poland, 2001, pp. 307–317.

[3] Sz. Grabowski: A family of cascade NN-like classifiers, Proc. 7th Int. IEEE Conf. TCSET'2002, Lviv-Slavsk, Ukraine, Feb. 2002, pp. 223–225.

[4] Sz. Grabowski, A. Jozwik and C.-H. Chen: Nearest neighbor decision rule for pixel classification in remote sensing, a chapter in Frontiers of Remote Sensing Info Processing, ed. S. Patt, World Scientific Publishing Co. Pte. Ltd., Singapore, July 2003.

[5] Sz. Grabowski: A family of cascade NN-like classifiers, Proc. 7th Int. IEEE Conf. on Experience of Designing and Application of CAD Systems in Microelectronics (CADSM), Lviv-Slavsk, Ukraine, Feb. 2003, pp. 503–506.

[6] Sz. Grabowski: Towards decision rule based on closer symmetric neighborhood, Biocybernetics and Biomedical Engineering, Vol. 23, No. 3, pp. 39–46, July 2003.

[7] A. Jozwik, L. Chmielewski, W. Cudny and M. Sklodowski: A 1-NN preclassifier for fuzzy k-NN rule, Proc. 13th Int. Conf. on Pattern Recognition, vol. IV, track D, Parallel and Connectionist Systems, Vienna, Austria, 1996, pp. 234–238.

[8] M. Nieniewski, L. Chmielewski, A. Jozwik and M. Sklodowski: Morphological detection and feature-based classification of cracked regions in ferrites, Machine Graphics and Vision, Vol. 8, No. 4, 1999.

[9] J. S. Sanchez, F. Pla and F. J. Ferri: On the use of neighbourhood-based non-parametric classifiers, Pattern Recognition Letters, Vol. 18, No. 11–13, pp. 1179–1186, 1997.

[10] J. S. Sanchez, F. Pla and F. J. Ferri: Improving the k-NCN classification rule through heuristic modifications, Pattern Recognition Letters, Vol. 19, No. 13, pp. 1165–1170, 1998.

[11] S. B. Serpico, F. Roli: Classification of multisensor remote sensing images by structured neural networks, IEEE Trans. on Geoscience Remote Sensing, Vol. 33, No. 3, pp. 562–578, 1995.