The Design of a Nearest-Neighbor Classifier and Its Use for Japanese Character Recognition

Tao Hong, Stephen W. Lam, Jonathan J. Hull* and Sargur N. Srihari

CEDAR, Computer Science Department
State University of New York at Buffalo
Buffalo, New York 14228-2567

*RICOH California Research Center
2882 Sand Hill Road, Suite 115
Menlo Park, CA 94025

Abstract

The nearest neighbor (NN) approach is a powerful nonparametric technique for pattern classification tasks. In this paper, algorithms for prototype reduction, hierarchical prototype organization and fast NN search are described. To remove redundant category prototypes and to avoid redundant comparisons, the algorithms exploit geometrical information of a given prototype set which is represented approximately by computing k-nearest/farthest neighbors of each prototype. The performance of a NN classifier using those algorithms for Japanese character recognition is reported.

KEYWORDS: Nearest-neighbor classifier, Japanese character recognition.

1 Introduction

Given a test sample, a NN classifier recognizes the sample by finding its k-nearest neighbors among a set of prototypes that have been labeled with category identity. Using the whole set of training samples as the prototype set, which is usually very large in size, the accuracy of a NN classifier is very high. However, such a classifier is very slow because it has to exhaustively match all the prototypes against the test sample to find it k-nearest neighbors.

To improve speed without much sacrifice of accuracy, different approaches have been proposed to make the NN approach more practical[1]. One of the approaches is to reduce the size of prototype set[3, 4]. Another important approach is to speed up the search procedure during classification[2].

To apply the NN approach to Japanese character recognition, two difficulties have to be addressed. First, the number of categories is large, e.g., there are more than 3,000 frequently used Japanese characters in Japanese document. Most characters in the character set are Chinese characters, or so-called *Kanji*. To deal with so many categories with variations of font and degradation, a training set is usually very large in size. Second, the feature description for a Kanji is usually high in dimension because the stroke structure of a Kanji character is complex. These difficulties make a brute-force NN classifier almost impractical because the computation load is so heavy.

In this paper, algorithms for prototype reduction, hierarchical prototype organization and fast NN search are described. The prototype reduction algorithm is a non-iterative process. To remove redundant prototypes and to avoid redundant comparisons, the algorithms exploit the geometrical information of the prototype set which is carried by k-nearest/farthest neighbors of each prototype. Then, we report our experiments on designing a NN classifier for Japanese character recognition by using those algorithms and discuss its performance on large data sets.

2 Using Nearest/Farthest Neighborhood as Geometric Information

Here, we propose to use k-nearest/farthest neighbor lists to estimate the geometric information, which is about the distribution of samples in the feature space of a given sample set. Given a set of samples, $P = \{p_0, p_1, ..., p_{n-1}\}$, for each sample p_i, its k-nearest neighbor list, $N_i = \{n_{io}, n_{i1}, ..., n_{i,k-1}\}$, can be computed by comparing the sample with the rest of samples; similarly, its k-farthest neighbor list, $F_i = \{f_{io}, f_{i1}, ..., f_{i,k-1}\}$, can be computed. For each n_{ij} and f_{ij}, we also keep their distances to p_i, $Dist(p_i, n_{ij})$ and $Dist(p_i, f_{ij})$ respectively. Those data will be used in the algorithms described later.

When the size of a training set becomes very large,

0-8186-7128-9/95 $4.00 © 1995 IEEE

it is time-consuming to calculate the exact k-nearest neighbor list for each sample. To speed up, an approximate algorithm was designed.

3 Prototype Reduction

Given a training set, the step of prototype reduction is to select a subset of samples which can represent the whole training set. The subset can be generated by deleting those redundant samples from the training set. In detail, an algorithm, $RNN2$, for the method is described in Figure 1. By checking the precomputed nearest neighbor list of each training sample, the algorithm decides whether a training sample can be deleted without negative effect on correct classification of itself and other samples if the sample is removed.

Given two training samples, x and y, which are from the same category, the function *redundant*() tests whether x is redundant in the presence of y inside the current prototype set P. This can be done by examining the k-nearest neighbor list of each prototype in P. If there exists a prototype p which has its k-nearest neighbor list as following,

$$..., x, ..., z, ..., y, ...$$

where z is a sample from a different category and it is very close to x or y in distance, then, x can not be redundant; otherwise, x is redundant and can be removed from the prototype set P. In the algorithm, α and β are two parameters to control the size of prototype set to be derived.

Previous methods, such as Hart's *condensed nearest neighbor rule* (CNN) and Gates' *reduced nearest neighbor rule* (RNN), have to call the procedure of classification iteratively to test whether the deletion(or adding) of a prototype affects the correct classification of the other prototypes[4, 3]. The advantage of the method here is to avoid such an iterative process.

4 Fast NN Search

To speed up NN classification, we proposed a fast NN search algorithm, FNN. The basic idea is to avoid unnecessary comparisons. Suppose there is a test sample X to be classified (see the triangle in Figure 2). It has to compare with prototypes from the prototype set. After the comparison between the input pattern X and a prototype p_i, we know that the distance between them is very small. For those prototypes in p_i's

```
RNN2(S, P, α, β)
    Assumption: training set is S = {s1, s2, ....., sn};
                prototype set to be generated is P;
                α <= k; β <= k.
BEGIN
1.  P ← S; /* initialize the prototype set P*/
2.  FOREACH si in P DO
3.     FOREACH sij in si's k-nn list Ni DO
4.        IF ( (category(sij) = category(si))
5.             AND (category(sij) =
6.                  category(sij 's nearest-neighbor))
7.             AND (redundant(sij, si, P, α, β) = TRUE) )
8.          P = P − {sij}; /* remove sij from P */
9.     ENDFOR;
10. ENDFOR;
END.

BOOLEAN redundant(x, y, P, α, β)
/* test whether x is redundant in the presence of y in P*/
BEGIN
1.  redundant = TRUE; /* assume x is redundant */
2.  FOREACH si in P DO
3.     get si's k-nn list Ni = (si1, .., sik);
4.     IF there exist a p and a q,
5.        such that sip = x AND siq = y
6.                 AND p < α AND p < q
7.        FOR m = p + 1 TO q − 1 DO
8.           IF ( (category(sim) ≠ category(x))
9.                AND ( (sim is in x's β-nn list)
10.                     OR (sim is in y's β-nn list) )
11.             redundant = FALSE;
12.             BREAK;
13.       ENDFOR;
14.    IF (redundant = FALSE)
15.       BREAK;
16. ENDFOR;
17. RETURN(redundant);
END.
```

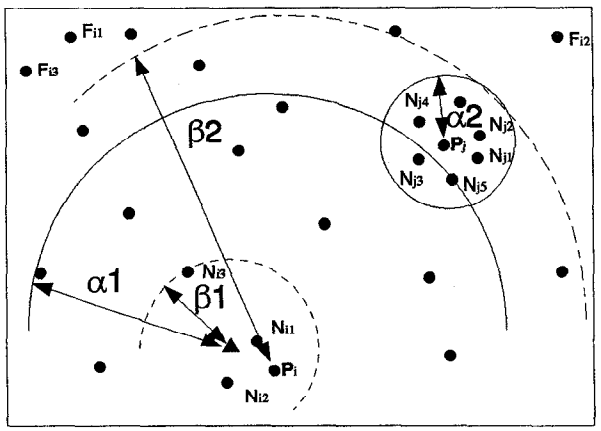
Figure 1: An algorithm for prototype reduction

k-farthest neighbor list, such as f_{i1},f_{i2} and f_{i3} shown in Figure 2, without going further to compare each of them with the input sample X, we know the distance will be very large and therefore they can not be in X's k-NN list. Similarly, after the comparison between X and a prototype p_j, we know that the distance between them is very large. For those prototypes in p_j's k-nearest neighbor list, such as n_{j1},n_{j2},n_{j3} and n_{j4} shown in Figure 2, without going further to compare each of them with the input sample X, we know the distance will be very large and therefore they can not be in X's k-NN list. In detail, the algorithm which is based on the discussion above is described in Figure 3.

5 Packing Prototypes and Fast NN Search

In order to further speed up the search process, The prototype set can be organized into a hierarchical representation. The prototype set is divided into two levels. The first level is the so-called "*centroid set.*" For a centroid, there may be a set of prototypes which are stored in the second level. The set for a centroid

Proceedings of the Third International Conference on Document Analysis and Recognition (ICDAR '95)
0-8186-7128-9/95 $10.00 © 1995 IEEE



Note: ● denotes prototype; ▲ is an input pattern x.

Figure 2: Speed up nearest neighbor search by avoiding redundant comparisons

is called as the centroid's "*package*." A centroid can approximately represent those prototypes in its package. The process to create such a hierarchy is to pack prototypes for selected centroids.

In the algorithm $PackProto\_f$, by randomly picking up a prototype p_i in the prototype set P, p_i will be put into the $Centroid\_Set$. Its package will be formed by checking p_i's k-nearest neighbor list,

$$p_{i0}, p_{i1}, \cdots, p_{ij}, \cdots, p_{i,k-1}$$

From the list, the *farthest-like-neighbor*, p_{ij} can be found so that all those neighbors, $p_{i0}, p_{i1}, \ldots, p_{i,j-1}$, will be put into p_i's package. Here, *farthest-like-neighbor* means the last prototype in the k-nearest neighbor list, which has the same category as p_i's. In this way, it allows a centroid to have some prototypes in its package which are from different categories. The algorithm is therefore *farthest-like-neighbor-based.*

Similarly, a *nearest-unlike-neighbor-based* algorithm $PackProto\_n$ is designed. In this version, after choosing p_i as a centroid, all neighbors before the *nearest-unlike-neighbor* in p_i's k-nearest neighbor list will be put into p_i's package. Here, the *nearest-unlike-neighbor* is the first neighbor whose category identity is different from the p_i's.

After packing prototypes, the k-nearest neighbor list and k-farthest neighbor list of each prototype in the centroid set C will be computed. The NN classifier can be described as a two-step procedure. Given a test sample X, its approximate k-nearest neighbors $\overline{N}_X$ in the centroid set C can be calculated using the search algorithm described above. Then the prototypes in the packages of those centroids in $\overline{N}_X$ will be compared

```
FNN(X, P, α1, α2, β1, β2)
    X is input pattern;
    P is Prototype Set: {p0, p1, ...., pn-1}
    α1, α2, β1 and β2 are thresholds;
BEGIN
1.  Top_k_Candidate_Set ← ∅;
2.  kth_smallest_dist ← +∞;
3.  FOREACH prototype pi in P DO
4.      IF (Dist(X, pi) < kth_smallest_dist)
5.          update Top_k_Candidate_Set
6.                 and kth_smallest_dist;
        /* remove some prototypes from P if possible */
7.      IF (Dist(X, pi) > α1)
8.          FOREACH pij in pi's k-nn list Ni
9.              IF Dist(pi, pij) < α2
10.                 Remove pij from P;
11.     IF (Dist(X, pi) < β1)
12.         FOREACH pij in pi's k-farthest neighbor list Ni
13.             IF Dist(pi, pij) > β2
14.                 Remove pij from P;
15. ENDFOR
    /* to make the Top_k_Candidate_Set more accurate by
     * matching X to prototypes in k-nearest neighbor lists
     * of those prototypes from Top_k_Candidate_Set */
16. FOREACH prototype pi in Top_k_Candidate_Set DO
17.     FOREACH pij in pi's k-nn list Ni
18.         IF ( Dist(pi, pij) < α2
19.              AND pij is not in Top_k_Candidate_Set
20.              AND (Dist(X, pij) < kth_smallest_dist)
21.             update Top_k_Candidate_Set
22.                    and kth_smallest_dist;
23.     ENDFOR
24. ENDFOR
25. FOREACH prototype pi in Top_k_Candidate_Set DO
26.     output Category(pi);
27. ENDFOR
END.
```

Figure 3: A fast NN search algorithm

with X to generate the final k-nearest neighbors N_X for the sample.

6 A NN Classifier for Japanese Character Recognition

A NN classifier using the algorithms described above was implemented and tested on machine-printed Japanese character images. The feature used in the classifier is so-called *Local Stroke Direction*(LSD) feature[5]. The dimension of the feature is 256.

The first experiment is to use samples from ETL cdrom to do training and testing. There are totally 52,004 samples in ETL database. The number of categories is 2,147. Half of samples were used as training samples(the set ETL_{train}) and half of them were used for testing(the set ETL_{test}).

Using the whole training set ETL_{train} as prototype set, the performance of a brute-force NN classifier on the testing set ETL_{test} is shown in Table 1 (a).

Given the training set ETL_{train} with 26,002 samples, the RNN2 algorithm generated a reduced set ETL_P with 5,117 prototypes. Using ETL_P as the

Proceedings of the Third International Conference on Document Analysis and Recognition (ICDAR '95)
0-8186-7128-9/95 $10.00 © 1995 IEEE

prototype set and ETL_{test} as the test sample set, the performance of our classifiers and a brute-force NN classifier is shown in Table 1 (b). The performance of the brute-force NN indicates that the reduced prototype set ETL_P can represent the whole training set ETL_{train} quit well(see also Table 1 (a) for a comparison). By using our FNN, the accuracy does not drop, but the average number of comparisons needed for each test sample reduces to 3,325 from original 5,117. By using FNN together with *nearest-unlike-neighbor-based* PackProto algorithm($PackProto\_n$), the accuracy drops a little and the average number of comparisons reduces further to 2906. By using FNN together with *farthest-like-neighbor-based* PackProto algorithm($PackProto\_f$), the accuracy also drops a little, but the average number of comparisons reduces further to 2460, about 48% of the prototype set ETL_P.

	Accuracy top1	top2	top5	# Of Comparisons
Brute-force NN	99.19%	99.60%	99.81%	26,002

(a)

	Accuracy top1	top2	top5	Avg. # Of Comparisons
Brute-force NN	98.21% (25,539)	99.45% (25,859)	99.74% (25,936)	5,117
FNN	98.22% (25,540)	99.44% (25,858)	99.73% (25,933)	3,325 (64.98%)
FNN + ProtoPack-n	97.90% (25,458)	99.09% (25,766)	99.37% (25,839)	2,906 (56.79%)
FNN + ProtoPack-f	97.65% (25,391)	98.82% (25,697)	99.10% (25,768)	2,460 (48.08%)

(b)

Table 1: Performance of NN classifiers on the testing set ETL_{test} (size=26,002): (a) by using the training set as the prototype set(size = 26,002); (b) by using ETL_P as the prototype set(size = 5,117)

Another experiment was done on a much larger data set. The data set is from CEDAR's Japanese image database. Character images in the database were extracted from Japanese documents such as newspapers, books, journals and facsimiles. Many documents are highly degraded and present with many different typefaces and fonts. The number of categories included in the database is 3,230. In the database, the number of training samples is 114,596 and the number of test samples is 27,717. Training and testing samples are collected from different document pages.

Because it is time-consuming to calculate exact k-nearest neighbor lists for samples in such a large training set, the approximate method is utilized for this this experiment. There are 13,393 samples selected as prototypes($CEDAR_P$) after RNN2. Table 2 has the classification results on testing set. Three versions of our classifier were tested on the testing set using $CEDAR_P$ as the prototype set. 13,393.

	Accuracy top1	top2	top5	Avg. # of comparisons
Brute-Force	92.47% 25631	95.61% 26502	97.40% 26997	13,393
FNN	92.39% (25,608)	95.49% (26,466)	97.19% (26,937)	8,470 (63.24%)
FNN + ProtoPack-n	91.80% (25,446)	94.82% (26,282)	96.46% (26,735)	7,192 (53.70%)
FNN + ProtoPack-f	90.10% (24,972)	93.06% (25,793)	94.66% (26,237)	4,028 (30.08%)

Table 2: Performance of NN classifiers on CEDAR testing set(size = 27,717) using $CEDAR_P$ as prototype set(size = 13,393)

7 Conclusions

In this paper, algorithms for prototype reduction and fast NN search are presented. They use k-nearest/farthest neighbor lists to estimate the geometric information of a training set and a derived prototype set. A Japanese character classifier has been implemented using the algorithms. Its performance on two data sets was reported. To obtain a better prototype set on the large training set, the method to estimate k-nearest neighbor lists has to be more elaborate.

References

[1] B.V. Dasarathy, "Nearest Neighbor (NN) Norms: NN Pattern Classification Techniques," IEEE Computer Society Press, 1991.

[2] J.H. Friedman, F. Baskett and L.J. Shustek, "An Algorithm for Finding Nearest Neighbors," IEEE Transactions on Computers, Vol. C-24, No. 10, pp. 1000-1006, October 1975.

[3] G.W. Gates, "The Reduced Nearest Neighbor Rule," IEEE Transactions on Information Theory, Vol. IT-18, No.3 pp. 431-433, May 1972.

[4] P.E. Hart, "The Condensed Nearest Neighbor Rule," IEEE Transactions on Information Theory, Vol. IT-14, No.3 pp. 515-516, May 1967.

[5] S. Mori, K. Yamamoto and M. Yasuda, "Research on Machine Recognition of Handprinted Characters," in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-6, 4, pp. 386-405, July 1984.

Proceedings of the Third International Conference on Document Analysis and Recognition (ICDAR '95)
0-8186-7128-9/95 $10.00 © 1995 IEEE