Comparison Between Two Prototype Representation Schemes for a Nearest Neighbor Classifier

Jari Kangas
Nokia China R&D Center
No.11, He Ping Li Dong Jie
Beijing, 100013 China
jari.a.kangas@nokia.com

Abstract

The paper is about the problem of finding good prototypes for a condensed nearest neighbor classifier in a recognition system. A comparison study is done between two prototype representation schemes. The prototype search is done by a genetic algorithm which is able to generate novel prototypes (i.e. prototypes which are not among the training samples). It is shown that the generalized representation scheme is more powerful, giving significantly larger normalized interclass distances. It is also shown that both representation schemes with genetic algorithm give significantly better prototypes than a direct prototype selection algorithm, which can select only among the training samples.

1. Introduction

Nearest neighbor classification algorithms give good results in many recognition problems. However, the memory requirements and the computational burden due to the large number of prototypes makes utilization of nearest neighbor classification difficult. Many algorithms have been developed to improve the situation (see [9]).

An approach to reduce the number of prototypes is to cluster the class samples, and use the center prototypes as class representatives in the classification stage. The center obviously depends on definition of the distance function, but the results also depend on the search algorithm. In [6] it was shown that a genetic algorithm produces better class centers than a selection algorithm for a simple character recognition algorithm

In this paper we compare two representation schemes for the character data. The representation used in [6] is enlarged so that a new "don't care" symbol is added to the symbol set used for direction coding. The new symbol never occurs in the data itself, but only in the prototypes. It is shown that the enlarged representation enables even better centers for the nearest neighbor classifier.

2. Character Recognition System

We applied a simple character recognition system, based on a nearest neighbor classifier. This chapter explains the basic principles. Somewhat similar character recognition system was explained in [11].

2.1 K-Nearest Neighbor Classifier

The classification decision was made by a k-nearest neighbor (k-nn) classifier [1-3], which is one of the simplest classification methods. The main principle is to make the decision by comparing an unknown sample against a large number of preclassified examples, and evaluating the class memberships of the k nearest examples. The power of the algorithm lies on the assumption that the most similar examples very probably belong to the correct class.

2.2 Distance Measure

The total distance between two characters was the sum of minimum distances between corresponding strokes and an extra term for every stroke in one character that had no corresponding strokes in the other character. Dynamic Time Warping algorithm [10] and Dynamic Programming implementation was used to compute the distances between direction sequences. Euclidean distances between stroke beginning and ending points, correspondingly, were added to that figure. A fixed cost table was defined for the direction code differences.

2.3 Input Data Representation

The input data was collected by a drawing tablet, and was a sequence of coordinate pairs in tablet coordinates. The strokes were separated; i.e. every separate line (from a pen down event to a pen up event) was a separate

0-7695-0750-6/00 $10.00 © 2000 IEEE

sequence. Preprocessing contained normalization to a fixed size, resampling, and transformation to direction codes (chain codes were used [4]). After preprocessing the data consisted of structures of the following form:

Char = { Stroke1 [dir. codes] [begin and end]
Stroke2 [dir. codes] [begin and end]
... },

where the direction codes were from a set {1,2,3,4,5,6,7,8}. Begin and end locations were given in the normalized coordinates of [1,100], in the order [begin-x, begin-y, end-x, end-y]. Location of coordinate origo was in the upper-left corner.

As an example one sample of a two stroke "A" character is shown below:

"A" = { [5 5 5 5 5 5 4 4 2 1 1 1 1 1 1] [0 100 59 100]
[4 3 3 3 3] [6 65 36 65] }.

2.4 Modified Prototype Representation

In the original system the prototypes used the same representation as the input data. To decrease the intraclass distances we considered modifications to that.

When the sample characters are evaluated, one notices that the beginnings and endings sometimes have extra symbols, which might result from accidental hooks in strokes. To counter the effect of those in the distance computations we added a special symbol that could be used as substitute for any symbol. I.e., there was a "don't care" symbol that could be replaced by any other symbol with a relatively small cost.

It was expected that the "don't care" symbol was useful in some locations of the prototype strings, like beginnings and endings. To discourage an excessive usage of the symbol in strings, however, a real cost of replacing the symbol to other symbols was attached.

As an example one extended prototype of an "A" character is shown below:

"A" = { [0 5 5 4 5 4 4 4 2 1 1 1 1 1 0] [2 93 63 99]
[3 3 3 3 3] [11 60 47 56] }.

3. Optimization by genetic algorithms

Genetic algorithms belong to a group of heuristic optimization algorithms, called evolutionary algorithms. The basic idea is to mimic the problem solving methods of evolution. For overviews, see, e.g. [7,8].

An evolutionary algorithm maintains a population of potential solutions to the optimization problem. Each solution is evaluated to get a measure of fitness, i.e. how good a solution it would be. A new population of solutions is initialized by selecting the more fit solutions. The selected solutions undergo transformations, thereby creating novel solutions. There exist several types of transformations, in some only one solution is modified (mutation), in some others several solutions are combined and modified by taking parts from each (crossover).

1. Initialize a solution population
While termination condition is not satisfied do
{
2. Evaluate all individuals
3. Select the best individuals
4. Generate new individuals from the selected
}

4. Prototype Selection Problem

K-nearest neighbor classifier is simple in principle, but all the prototypes must be stored and the comparison phase often becomes time consuming.

In many classification problems it is only necessary to store a small subset of samples and still achieve comparable performance to the full prototype set. A well-known algorithm to reduce prototype number along those lines is the Condensed Nearest Neighbor (CNN) rule [5]. The basic principle in CNN is to find a subset of prototypes, which classify correctly all the remaining samples in the training set.

In this paper we used two prototype selection algorithms to find one average prototype for each class in the training samples. The first algorithm is a straightforward selection among the example cases; i.e. one of the example cases is used as a prototype. The second algorithm can use a novel prototype, which is not among the example cases.

4.1 Prototype Selection among the Cases

For every sample s_i the sum S_i of distances from the sample to every other sample was computed,

$$S_i = \sum_{i,j \in A, i \neq j} D_{Tot}(s_i, s_j),$$

where s_i and s_j both belong to the same class. The sample, which gave the minimum sum, was used as the prototype. Because the search is restricted to the training samples only, we used quite many samples (30 samples of each class) just to make sure that there is enough variation to find a reasonable average.

4.2 Prototype Search by A Genetic Algorithm

The prototype search by a genetic algorithm followed the basic function optimization approaches [7,8]. The prototype representations were directly taken as the chromosomes. The parameters were encoded as discrete symbols for the directions, and real valued parameters for the locations. The length of strokes (the number of

direction codes in each stroke) was naturally variable. The number of strokes was fixed for all samples in each class; that followed directly from the knowledge that all the character samples were collected from one person that had used consistently the same writing style. The fitness values were found by computing the cumulative distance of each prototype from all training samples.

We used crossover and mutation. To generate new individuals for the next generation we used a mixture of techniques based on a rank order. First we used an "elitist" selection by saving N_1 fittest prototypes directly. Then the N_2 next fit individuals were mutated and saved as such. Last, the already selected $N_1 + N_2$ individuals were used to generate new individuals for the next generation by crossover operation. The number of individuals in each generation was fixed. Also the number of generations was fixed; we used a number that was found to be large enough for saturation.

4.2.1 Mutation operation. The mutation operation was separately applied for the direction code parameters, and for the location parameters. The mutated strokes were selected by a random operation.

Each direction code in a sequence was separately mutated given a certain small probability. In mutation a code was replaced by a random code among all direction codes. In enlarged representation the candidate symbol set thus included also the "don't care" symbol. Mutation was the only source of "don't care" symbols. The prototype search was started from original sample strings, which did not contain any "don't care" symbols. It was up to the search algorithm to find positions for the symbols, if any.

In mutation the location parameters were deviated by a small random values each.

4.2.2 Crossover operation. One-point crossover was used, and applied only in the direction code section. The location parameters were copied from the first parent.

The crossover location was selected in one parent sequence and a "nearby" location was selected in the other parent sequence. Because they were not exactly the same (as measured from the beginning of sequence) the length of resulting offspring sequence could vary as well. That was very convenient, because the sample sequences were not of the same length and defining an optimal sequence length analytically would have been difficult.

5. Experiments

5.1 Character Data

For the prototype search experiments we collected data of 10 handprinted characters (uppercase characters "A"-"J"), 30 samples of each from one writer only. One writer was used to keep the sample sets rather compact. It was expected that data from one writer will form a unimodal set (of each character), with which the usage of one class prototype would be well motivated. If there were several modes in a character set, the usage of selection method especially would be questionable.

We used a Wacom ArtPad II drawing tablet connected to a Windows NT machine to collect the data. The drawing area was a square of about 2-cm times 2-cm. Because the character sizes were later normalized the character size was not critical. The sampling frequency was approximately 100 Hz, and the resolution in both directions was 100 points at most.

5.2 Indicator variables

We looked for two prototype quality variables. The first variable D_1 was an estimate of how far from the correct class prototypes the samples are located on the average. That variable value should be minimized. The variable, however, is sensitive to overall scale changes. To compensate for that we used another variable D_2 that was a ratio of the interclass distance of each sample from the nearest incorrect class prototype and the average intraclass distance computed over all classes. Both variables are computed classwise; i.e. samples from only one class are grouped together.

$$D_1 = \frac{1}{N}\sum_i d_i^c, \quad D_2 = \frac{1}{N \times \overline{D_1}}\sum_i d_i^w,$$

where N is the number of test samples, d_i^c is the distance of i:th test sample to the correct prototype, and d_i^w is the distance of i:th test sample to the nearest wrong prototype. The normalizing value $\overline{D_i}$ is an average of the first variable over all classes.

5.3 Experimental setup

The test runs were conducted using a leave-one-out principle. One test sample at a time was removed from the sample set, the prototypes were found out using the remaining samples, and the distances of the test sample to corresponding (the correct and the nearest incorrect) prototypes were computed. We made, therefore, 300 prototype searches, and acquired 300 pairs of distances (d_i^c and d_i^w) for both prototype search algorithms, and for two representation schemes. From these we computed the average values for both test variables.

The quality variables were computed classwise; i.e. the variable values were given separately for each class. The classwise differences between the methods were computed to see if there was any significant difference.

6. Results

For each of the ten classes we found out the average intraclass distances given by the 30 leave-one-out experiments. The results for both search methods and both representation schemes are given below.

Intraclass	**Selection**	**Genetic**	**Genetic+**
A	3.38	3.18	2.79
B	2.45	2.06	1.58
C	0.62	0.63	0.64
D	2.37	2.12	1.31
E	3.91	3.45	2.51
F	2.01	1.78	1.79
G	1.70	1.62	1.69
H	4.77	4.61	3.18
I	0.71	0.50	0.53
J	0.56	0.57	0.55

To see the significance of the differences in the intraclass distances we computed the means of the difference values between different methods, and the confidence intervals (for 99 % confidence level) of the means. Because zero was in two cases out of three outside the confidence interval, the intraclass distances given by both genetic search algorithm implementations are **significantly smaller** than the corresponding values for the selection method. There is no significant difference between the different representations.

Intra	Mean of diff.	Conf. Interv.	Significant
S – G	0.19	0.08 - 0.30	X
S – G+	0.59	0.10 – 1.09	X
G - G+	0.40	-0.03 – 0.83	

Corresponding results for the average interclass distances are given in the following table.

Interclass	**Selection**	**Genetic**	**Genetic+**
A	25.92	28.33	35.57
B	2.39	2.66	3.02
C	11.79	14.57	15.74
D	1.72	1.68	1.57
E	10.74	11.65	13.86
F	10.51	11.20	13.89
G	12.82	13.88	17.13
H	29.04	31.39	38.77
I	9.37	10.12	12.65
J	13.94	15.28	18.18

As above we computed the means of the difference values between different methods, and the confidence intervals (for 99 % confidence level) of the means. In all combinations the **differences are significant**, even between the two different representation schemes.

Inter	Mean of diff.	Conf. Interv.	Significant
S – G	-1.25	-2.03 - -0.47	X
S – G+	-4.21	-6.85 - -1.57	X
G – G+	-2.96	-5.03 - -0.89	X

7. Conclusions

It has been shown that the genetic algorithm can be used to find significantly better prototypes in a nearest neighbor classification scheme. The comparison is against a direct optimal selection scheme. The intraclass distances from the prototypes to independent samples decrease and the normalized interclass distances between the samples and the best incorrect prototypes increase significantly.

When the representation scheme of the prototypes is changed the genetic algorithm gives even better prototypes. The modified prototypes are significantly better than the unmodified prototypes in normalized interclass distances.

8. References

[1] Cover, T. M., and Hart, P. E., "Nearest Neighbor Pattern Classification," *IEEE Transactions on Information Theory*, 13(1), pp. 21-27, January 1967.

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

[3] Fix, E., and Hodges, J. L., *Discriminatory Analysis – Nonparametric Discrimination: Consistency Properties*, Tech. Report No 4, USAF School of Avia. Med., Texas, USA, 1951.

[4] Freeman, H., "On the Encoding of Arbitrary Geometric Configurations," *IRE Transactions on Electronic Computers*, pp. 260-268, June 1961.

[5] Hart, P. E., "The Condensed Nearest Neighbor Rule," *IEEE Transactions on Information Theory*, pp. 515-516, May, 1968.

[6] Kangas, J., "Prototype Search for a Nearest Neighbor Classifier by a Genetic Algorithm," *Proc. of 1999 Int. Conf. on Computational Intelligence and Multimedia Applications* ICCIMA'99, pp. 117-122, 1999.

[7] Michalewicz, Z., *Genetic Algorithms + Data Structures = Evolution Programs*, Springer-Verlag, 1996.

[8] Mitchell, M., *An Introduction to Genetic Algorithms*, MIT Press, 1996.

[9] Ripley, B. D., *Pattern Recognition and Neural Networks*, Cambridge University Press, 1996.

[10] Sankoff, D, and Kruskal, J. B., *Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison*, Addison-Wesley, 1983.

[11] Yuen, H., "A Chain Coding Approach for Real-time Recognition of On-line Handwritten Characters," *Proc. of Int. Conf. on Acoustics, Speech and Signal Processing*, vol. 6, pp. 3426-3429, 1996.