A Scanning n-tuple Classifier for Online Recognition of Handwritten Digits

Eugene H. Ratzlaff
IBM T.J. Watson Research Center
Yorktown Heights, NY 10598, USA
ratzlaff@us.ibm.com

Abstract

A scanning n-tuple classifier is applied to the task of recognizing online handwritten isolated digits. Various aspects of preprocessing, feature extraction, training and application of the scanning n-tuple method are examined. These include: distortion transformations of training data, test data perturbations, variations in bitmap generation and scaling, chain code extraction and concatenation, various static and dynamic features, and scanning n-tuple combinations. Results are reported for both the UNIPEN Train-R01/V07 and DevTest-R01/V02 subset 1a isolated digits databases.

1. Introduction

The scanning n-tuple (or sn-tuple) classifier first described by Lucas and Amiri [1-3] has been demonstrated to be a good classifier for handwritten offline character recognition [1-4]. The sn-tuple classifier trains and classifies quickly and can be implemented with relative ease. As will be shown, it can also be applied with various static or dynamic features, or combinations of both. For these reasons, and because it is implemented in a probabilistic framework, the sn-tuple classifier merits consideration for use independently or in concert with other classifiers in online recognition tasks.

The sn-tuple method applies a novel combination of chain code feature extraction with a probabilistic n-tuple classifier. Unlike earlier n-tuple classification methods, the information-rich outline features of the image are first extracted by generating a chain code. The chain code is then sectioned into multiple, overlapping substrings of fixed length, each offset by 1 code element. Each substring is subsampled (decimated) to generate a short (typically 4 to 6 elements) string of length n - a "scanned" n-tuple - to be used as the feature or "address" for an n-tuple model [1].

This study describes an implementation of the sn-tuple classifier for recognition of isolated online handwritten digits drawn from the UNIPEN [5] data set. Preprocessing steps for scaling, generating a raster image, and chain code generation and handling are considered. The application of distortion transformations (also known as defect models) to training data [3, 6, 7] for training data augmentation and the use of smoothing [8] are described. Test data perturbations [7] and combinations of various sn-tuples are examined.

2. The scanning n-tuple classifier

The scanning n-tuple classifier is a variant of the memory-based standard n-tuple method [9]. The critical difference of the sn-tuple is the feature extraction [1]. A chain code is extracted from a binary image. The value of each chain code element may range from 0 to σ-1. The chain code for each training exemplar is reduced to y substrings, each of length n, by sampling n sequential locations in the chain code with a fixed offset $\delta \geq 1$ between each sample. These substrings are the sn-tuples described by this method:

Let $\boldsymbol{Y} = \{Y_0, \ldots, Y_{m-1}\}$ be a chain code of length m from which y sn-tuples are extracted, where $y \leq m$ (typically, $y = m$ and it is assumed that the end of the chain code can be defined to wrap back onto its beginning such that all indices into $\boldsymbol{Y}$ are evaluated modulus m). Then a set of y sn-tuples $\boldsymbol{S}_k$ are extracted from $\boldsymbol{Y}$ as follows:

$$\boldsymbol{S}_k = \{\, Y_l \mid l = (i\delta + k) \bmod m,\ 0 \leq i < n \,\} : 0 \leq k < y \quad (1)$$

In training, for each class $c \mid 1 \leq c \leq Q$ each sn-tuple S_{ck} is a single, discrete observation with σ^n different sn-tuples possible. A memory address $A_{cj} \mid 0 \leq j \leq \sigma^n - 1$ records the number of times t_{cj} each possible sn-tuple S_{cj} is observed for all training tokens. The total number of training tokens is retained as $\{T_1, \ldots, T_Q\}$. Hereafter assuming all class prior probabilities $P(c)$ equal, the posterior probability that a digit is a member of class c given randomly observed sn-tuple S_j is shown in (2):

$$P(c \mid S_j) = \frac{(t_{cj}/T_c)}{\sum_{i=1}^{Q} (t_{ij}/T_i)} \quad (2)$$

0-7695-1263-1/01/$10.00 © 2001 IEEE

In the classification step, noting that $P(S_k)$ is constant with respect to c and applying Bayes rule, the maximum likelihood classification result for observed chain code $\boldsymbol{Y}$ is assigned to the class c that maximizes $P(\boldsymbol{S}_k|c)$ as shown in (3):

$$\underset{c}{\operatorname{argmax}}\left[P(\boldsymbol{S}_k|c)\right] = \underset{c}{\operatorname{argmax}} \prod_{k=0}^{y} P(c \mid S_k) \quad (3)$$

3. Implementation

Stroke vectors from the raw data are mapped to a binary bitmap raster with black data pixels on a white background. Before rasterization to a ***WxH*** (terms later referenced in Table 1 are given in bold italic) width-by-height bitmap the vector data for the entire digit are pre-scaled to fit within a (W-2)x(H-2) area to allow the mapping to leave a 1-pixel white border inside the border limits. In many experiments a 3x3 convolution ***kernel*** is applied (Fig. 1) at each original pixel to broaden the bitmap stroke width. In such cases, the vector data are pre-scaled to (W-4)x(W-4) to maintain the white pixel border. Three different scaling methods were applied: ***S1***, in which the datum aspect ratio was not maintained and the width and height were independently scaled to fill the bitmap in both width and height; ***S2***, in which the original aspect ratio was preserved (the bitmap was typically filled in only 1 dimension); and ***S3***, in which the height was scaled to fit the bitmap height and the aspect ratio was preserved *unless* the scaled width dimension exceeded the bitmap dimensions, in which case the width was scaled down to fill the available space. Typically, all strokes were rasterized into one bitmap. Alternatively, each stroke (***iso stroke***) could be rasterized to a separate bitmap. Chain codes from these separate bitmaps were then treated as if they were derived from one bitmap.

Static chain codes were extracted (Fig. 1) from the contours of the black regions (static image information). Static chain codes are recorded in the order observed while scanning the bitmap in raster order (starting top-left; left-to-right, top-to-bottom); contours were mapped in a clockwise direction if first encountered at a white-to-black pixel transition (typically an exterior surface), or counter-clockwise if first found for a black-to-white edge. ***Dynamic*** chain codes were determined from the directional shift of the stroke as each pixel was placed on the bitmap (dynamic stroke information). Dynamic chain codes are recorded in the order given by the original stroke sequence. Contour ***profile*** chain codes were not given as directional shifts, but rather by the scaled distance from the side of the bitmap to the first observed ink. The maximum distance was scaled to σ-2; when no ink was observed the code σ-1 was used.

σ = 9 chain codes were extracted, where codes 0-7 were the conventional 8-direction codes as shown in Figure 1. The ninth code was used to extend the ends of open chains. This prevented sn-tuple folding, balanced code representation, and provided "end-of-chain" context information. End extension was always done for code chains that were very short (as with the chain corresponding to the hole in Figure 1) when necessary to prevent the wrapping of an sn-tuple sequence back onto itself. The number of end extension codes added depends

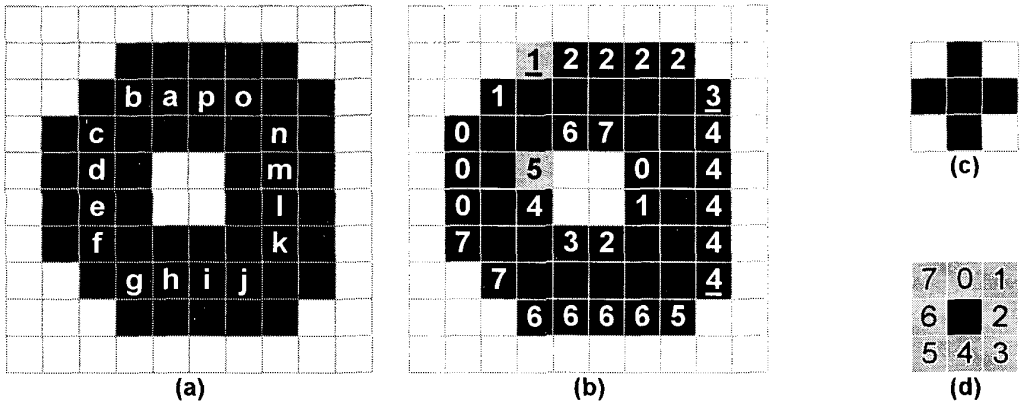


Figure 1: Rasterization and chain code extraction. (a) Example zero rendered in 10x10 bitmap with 3x3 kernel; original dynamic points given in alphabetic order, a-p. (b) Static chain code superimposed on corresponding bitmap locations from (a); 2 static chain codes starting at gray pixels: 122223444445666770001, 54321076; 1st scanning 3-tuple with offset δ=5 is 134 (underlined digits). (c) 3x3 binary convolution kernel. (d) Directional chain code key for gray pixels with respect to central black pixel. Dynamic chain code is 654443222100076. Left contour profile chain code is ...872100001278..., where ... represents a variable number of 8 codes (end extension codes).

on the coverage range (given by n and δ) of the sn-tuple: sufficient end extension codes were added to allow each original chain code element to be represented an equal number of times by an equal number of sn-tuples. The ninth chain code was always used to extend contour profile chain codes. It was optionally used (***end extend***) to extend static and dynamic chain codes that were not wrapped back on themselves.

Sn-tuples were extracted from each chain code either individually or by first concatenating all chain codes (***merge chains***) in the order observed into a single chain code representation. Typically, the end of each static chain code of length m was "wrapped" back to the start to allow m sn-tuples to be mapped. Alternatively, $m - (\delta \times (n\text{-}1))$ sn-tuples were extracted when neither wrapping (***wrap***) nor end extension was applied.

For those sn-tuples that were seldom or never observed in training for a single class, the corresponding regions of the class probability density were optionally estimated (***smooth***) using modified Kneser-Ney smoothing [8].

Distortion ***transformations*** on training data were investigated. The training data were augmented with 6 synthetic exemplars generated from each original training exemplar by means of the following distortions: ±5° slant (shear), ±10% drift in the x velocity, and ±5° baseline slope. These specific values were based on limited past experience with other classifiers; variations or optimizations were not explored.

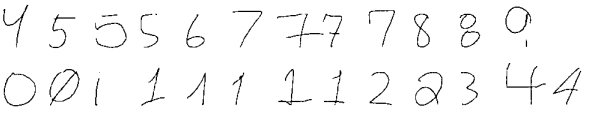
Similarly, these same transformations were applied as perturbations to the test data. Each perturbed datum was separately classified and the best score (no weighted voting as in [7]) among all ***perturbation*** results for each character was retained.

4. Experiments and results

Experimental results were obtained on isolated digits using the UNIPEN [5] Train-R01/V07 subset 1a database and the DevTest-R01/V02 subset 1a database. Because the DevTest set has only been released on a very limited basis, most results are evaluated using only the Train set. The 15953 digits of the Train set were segmented into 10 approximately equal-sized homogeneous subsets as well as 10 heterogeneous subsets. Each writer's data was equally distributed among all 10 homogeneous subsets, but in only 1 heterogeneous subset. Results were obtained for each set by combining 10 jackknife experiments. In each experiment, 8 different subsets were combined for training; a ninth subset was used for cross-validation and the tenth subset was used to test accuracy. Test results were obtained after retraining with the initial 8 subsets plus the cross-validation subset. Although it was observed that approximately 0.5% of the data is mislabeled, no data were excluded in either training or testing, with one exception: any datum with an aspect ratio greater than 2.5 was not used in training.

By visual inspection, 27 different allographs were selected as candidate archetypes for training. Approximately 2 to 3 exemplars of each archetype were manually clustered into subclasses in each jackknife training subset. Each training set of 8 subsets was then fully clustered into the final allograph subclasses in two steps. First, a bootstrap classifier was trained with the manually-clustered allographs. Then, the balance of the 8-subset training set was fully clustered using this bootstrap classifier. The tested classifier was then trained using the fully subclassed training set. Additional training/clustering cycles were tested, but found to reduce accuracy slightly. Observing that two of the final clusters typically had fewer than 20 exemplars, these two allographs were eliminated, leaving 25 archetypes, as shown by example in Figure 2. No further clustering algorithms or optimizations have been investigated.

Figure 2: Example allograph archetypes used for bootstrapping.



Results on the jackknifed subsets for 27 varied combinations of preprocessing, training, and sn-tuple combination are shown in Table 1. Sn-tuple lengths of 2-6 were investigated; lengths beyond 6 were considered to be impractical due to large memory requirements. Each of the single sn-tuple methods shown was first trained using sn-tuple lengths from 2 to 6 and offsets from 1 to 13. Test subset results are mean values based on conditions found optimal on the cross-validation subsets; 6-tuples generally performed best; the best-performing offset, δ, is presented. Standard deviations for the homogeneous results typically ranged from about 0.2% – 0.6% while those for heterogeneous results ranged from 1% - 3%. Rows 24-27 are for classification results using combinations of the aforementioned single sn-tuple methods. The highest accuracies for the homogenous and heterogeneous sets are for sn-tuple combination method 27 at 98.9% with a standard deviation (s.d.) of 0.2% and 98.0% with an s.d. of 1.0%, respectively.

Typically, only 10 to 20 percent of all possible sn-tuples were observed while training. Posterior probabilities were stored as a set of tables of 16-bit scaled log-probabilities indexed by allograph, one such table for each sn-tuple; omitting empty tables conserved memory space. Referring to Table 1 methods 22, 26, and 27, memory usage was 1.0, 19.3, and 4.4 megabytes, and classification speed was 440, 70, and 420 characters-per-second on a 300 MHz RS6000 workstation, respectively.

Table 1: Classification accuracies for various experimental conditions for single (rows 1-23) and combined (rows 24-27) scanning n-tuples; bold table entries indicate distinguishing parameters. Column headings as follows (see text for further details):

#	method index
WxH	raster bitmap total width by height (***WxH***)
Scale	***S1***, ***S2***, or ***S3***; blank = ***S3*** (see text for details)
n	tuple length for best results (last columns) on homo. and hetero. cross-valid. sets, respectively
δ	sn-tuple offset (best on training set for conditions)
Krnl	1 or 3; width of convolution ***kernel*** applied to broaden strokes; blank for dynamic chain codes
Type	D for ***dynamic*** chain code; PL, PR, PT, PB, for Left, Right, Top and Bottom contour ***profile***, respectively, and P4 for all four contour profiles combined; blank for ***static*** chain code
Iso Stroke	Y if each stroke was rasterized to a separate bitmap (***iso stroke***)
Merge Chains	Y if all chain codes concatenated to form 1 chain code (***merge chains***)
Wrap	N if chain codes were not wrapped, Y or blank if wrapped (***wrap***)
End Extend	Y if end-of-stroke code extensions were added at ends of chain code (***end extend***)
Trans	NT if no pre-training data ***transformations*** applied, NP if ***perturbation*** method not used
Smooth	N if modified Kneser-Ney smoothing of training data not applied (***smooth***)
Homo Train	classification accuracy for homogeneously jackknifed Train-R01/V07 data set (%)
Hetero Train	classification accuracy for heterogeneously jackknifed Train-R01/V07 data set (%)

#	WxH	Scale	*n*	δ	Krnl	Type	Iso Stroke	Merge Chains	Wrap	End Extend	Trans	Smooth	Homo Train	Hetero Train
1	24x32		6	7,6	3								98.0	96.9
2	24x32		6	7,6	3							**N**	97.7	96.9
3	24x32		6	7,7	3						**NT**		97.5	95.9
4	24x32		6	7,7	3						**NP**		97.8	96.5
5	24x32		6	7,7	**1**								97.7	96.2
6	24x32		**5**	7,7	3								97.9	96.6
7	24x32	**S1**	6	6,6	3								97.4	96.2
8	24x32	**S2**	6	6,6	3								97.6	96.4
9	24x32		6	7,7	3			**Y**					97.7	96.8
10	24x32		6	7,7	3		**Y**						97.4	95.8
11	**12x16**		6	3,3	1								97.7	96.6
12	**12x16**		6	2,2	3								96.7	95.2
13	**42x56**		6	11,11	3								97.7	95.9
14	**48x32**		6	7,7	3								97.8	96.6
15	**48x32**	**S2**	6	7,7	3								97.7	96.5
16	24x32		6	7,9		D	Y		N	Y			98.0	96.5
17	24x32		6	4,5		D	Y		N	**N**			93.1	91.2
18	24x32		6	8,7		D	Y		**Y**	**N**			97.2	95.4
19	24x32		**5**	7,8		D	Y		N	Y			97.9	96.6
20	24x32		6	3,2	1	PL							86.3	81.4
21	24x32		6	2,2	1	PR							78.5	75.0
22	24x32		6	2,2	1	PB							70.0	67.5
23	24x32		6	2,2	1	PT							75.3	71.7
24	24x32		6	2,2	1	P4							96.7	96.5
25	24x32		**4**	**5,5**	1	P4							96.7	95.5
26	24x32		6	7	3									
	24x32		6	9		D			N	Y			98.9	97.9
	24x32		6	2	3	P4								
27	24x32		**5**	7	3						**NP**			
	24x32		**5**	7		D			N	Y	**NP**		98.9	98.0
	24x32		**4**	5	3	P4					**NP**			

DevTest results are reported for comparative purposes. Because the ground-truth labels for the 8598 isolated digits of the DevTest set were unavailable, this set was manually ground-truth labeled. When the most likely label for a datum could not be inferred by appearance or apparent intent of the writer (based on stroke direction and sequence and in comparison to other related examples), or when the datum was clearly not a digit, the datum was truthed as a non-digit, but was *not* removed from the test set. Using the test conditions given for method 26 in Table 1 and the full Train set for training, the classification accuracy for the (mostly) heterogeneous DevTest set was 98.2%.

5. Discussion

The scanning n-tuple method as implemented herein is an effective classifier for online digits. The 98.1% classification accuracy for the DevTest-R01/V02 test set (though it may be slightly optimistic given a potentially cleaner ground-truth) compares favorably to the value of 96.1% reported for a hybrid KP neural network [10].

The benefits of several experimental options can be compared in Table 1. The use of a convolution kernel to broaden the strokes improves results except for the smaller bitmaps (compare methods 1, 5; 11, 12). Maintaining the aspect ratio when possible while always filling the height (S3) is preferred over filling the bitmap (S1) or always preserving the aspect ratio (S2) (methods 1, 7, 8). For the static sn-tuple, neither isolating strokes nor concatenating chain codes is best (methods 1, 9, 10). Perturbation, smoothing, and training data transformations are all useful (methods 1-4). For the dynamic sn-tuple, isolating each stroke, leaving each chain code unwrapped and extending the ends of the chain code are important (methods 16-19).

Combining sn-tuple methods based on different feature types is quite powerful (methods 1, 25-26). Not only does the error rate drop significantly when combining the best-performing static, dynamic and contour profile methods, but the performance of such combination methods is also maintained (methods 24, 25; 26, 27) while using fewer memory resources with methods that are lower-performing in isolation (methods 1, 4, 6; 16, 19). Somewhat surprisingly, combinations of only static sn-tuples (results not shown) with different values of n and δ did not outperform the single methods as has been observed for digit images [1-4].

6. Conclusions

This study suggests that the fast and uncomplicated scanning n-tuple classifier is a viable classifier for isolated online handwriting recognition. The conventional static sn-tuple method has been extended to include dynamic and contour profile features. The application of a broadening kernel, pre-transforming training data, perturbation, smoothing, and isolation of chain codes were all demonstrated to improve accuracy in this context. Stroke isolation and extension was introduced and shown to be valuable for the dynamic sn-tuple method. Contour profile sn-tuples were found useful in combination. The combination of static, dynamic and contour profile sn-tuples was especially powerful, where high accuracy and high speed can be achieved with reduced memory resources.

Future work will consider optimizations of memory resources, speed, and allograph clustering, as well as extension to problems with a larger number of classes.

Acknowledgments

The author thanks Michael Perrone and John Pitrelli for helpful comments and Jayashree Subrahmonia for helpful discussions and encouragement regarding this study.

7. References

[1] Lucas, S. & Amiri, A., "Statistical syntactic methods for high performance OCR," *IEE Proc.-Vis. Image Signal Process.*, 143(1), pp. 23-30, (1996).

[2] Lucas, S. & Amiri, A., "Recognition of chain-coded handwritten characters with the scanning n-tuple method," *Electronics Letters*, 31(24), 2088-2089 (1995).

[3] Lucas, S.M., "Improving scanning n-tuple classifiers by pre-transforming training data," *Proc. Fifth Int'l. Workshop on Frontiers in Handwriting Reco.*, 143-146tris (1996).

[4] Tambouratzis, G., "Improving the classification accuracy of the scanning n-tuple method," *ICPR2000*, 1050-1053 (2000).

[5] Guyon, I., Schomaker, L., Plamondon, R., Liberman, M. & Janet, S., UNIPEN project of on-line data exchange and recognizer benchmarks, *Proc. of the 12th Int'l. Conf. on Pattern Recognition, ICPR'94*, 29-33 (1994). IAPR-IEEE.

[6] Baird, H.S., "Document Image Defect Models," H.S. Baird, H. Bunke, and K. Yamamoto, eds., *Structured Document Image Analysis*. Springer Verlag, 1992, pp.546-556.

[7] Ha, Thien M. & Bunke, Horst, "Off-Line, Handwritten Numeral Recognition by Perturbation Method," *IEEE Trans. PAMI*, 19(5), 1997.

[8] Chen, S.F., Goodman, J., "An Empirical Study of Smoothing Techniques for Language Modeling," *Tech. Rep. TR-10-98*, Harvard University, 1998.

[9] Rohwer, R. & Morciniec, M., "The Theoretical and Experimental Status of the n-tuple Classifier," *Neural Networks*, 11(1) 1-14 (1998).

[10] Hébert, J.F., Parizeau, M., Ghazzali, N., "A new fuzzy geometric representation for on-line isolated character recognition," *Proc. 14th Int'l. Conference on Pattern Recognition, ICPR '98*, 1121-3 (1998).