Distance-Based Classification of Handwritten Symbols

Oleg Golubitsky and Stephen M. Watt

University of Western Ontario, London, Ontario, Canada n6a 5b7
http://publish.uwo.ca/~ogolubit
http://www.csd.uwo.ca/~watt

Abstract. We study online classification of isolated handwritten symbols using distance measures on spaces of curves. We compare three distance-based measures on a vector space representation of curves to elastic matching and ensembles of SVM. We consider the Euclidean and Manhattan distances and the distance to the convex hull of nearest neighbors. We show experimentally that of all these methods the distance to the convex hull of nearest neighbors yields the best classification accuracy of about 97.5%. Any of the above distance measures can be used to find the nearest neighbors and prune totally irrelevant classes, but the Manhattan distance is preferable for this because it admits a very efficient implementation. We use the first few Legendre-Sobolev coefficients of the coordinate functions to represent the symbol curves in a finite-dimensional vector space and choose the optimal dimension and number of bits per coefficient by cross-validation. We discuss an implementation of the proposed classification scheme that will allow classification of a sample among hundreds of classes in a setting with strict time and storage limitations.

1 Introduction

Distance measures play a central role in most pattern recognition problems. The choice of the distance measure affects the accuracy, as well as the time and space complexity of the classification algorithms. The latter aspect is particularly important for online handwriting recognition, since it is desirable to perform this task in real time and on inexpensive hardware.

We are particularly interested in developing methods for recognition of handwritten mathematics suitable for implementation on mobile devices. Such devices have already become widespread and provide an excellent opportunity for communicating mathematics. Developers of computer algebra systems are already actively targeting mobile applications [2]. Unfortunately, however, the existing software for recognizing handwritten mathematics does not cover a sufficiently wide range of subjects and/or is not sufficiently fast and accurate, and therefore is hardly ever used in mobile user interfaces.

The problem of recognition of handwritten mathematics is substantially different from that of handwritten text. The mathematical alphabet is larger than that of European languages by an order of magnitude. Mathematical expressions have a two-dimensional layout. Moreover, there is no fixed vocabulary of formulae, and frequencies of various subformulae strongly depend on the area of mathematics [25]. On the other

hand, mathematical expressions are often hand-printed, rather than being hand-written, which means mathematical symbols are generally better segmented. These properties render the methods developed for handwritten natural language recognition ineffective in a mathematical context. They also suggest that recognition of isolated handwritten mathematical symbols is a central and well-posed problem. Of course, there are other important issues, such as character segmentation and rotation/slant correction. These problems are essentially context-dependent, and mainly for this reason are beyond the scope of this paper. As mentioned above, exact repeated contexts are rare in mathematical expressions, which means that contextual information is mostly statistical. One possible approach to incorporating such information into the classifier, via confidence measures, has been proposed in [8]. The rotation/slant correction problem has been recently addressed in [9].

The objective of this paper is to compare the accuracy and computational complexity of various distance measures and choose the most suitable one for the purpose of recognition of handwritten mathematical symbols. The distance measures considered in this paper are briefly described below.

Among the distance measures used for classifying handwritten mathematical symbols, the elastic matching distance [22] is known to be one of the most accurate. This distance, however, has several drawbacks.

First, the size and position of the symbol must be normalized before the distance is computed. This means that the computation cannot begin until the entire symbol curve is traced. Directional features, e.g., approximations of the first and second derivatives at each point, are translation-invariant, but size normalization is still required. It is quite difficult to construct *local* scale- and position-invariant features that would be resistant to noise, as they would involve higher-order derivatives or their ratios. Perhaps for this reason, to our knowledge, scale- and position-invariant local features have not been used in online character recognition. The Legendre-Sobolev coefficients of the coordinate functions, which we use as features in this paper, are not translation- or scale-invariant, but can be instantly normalized with respect to these transformations as discussed in Section 2.

Second, the elastic matching algorithm is sensitive to the number of points sampled from the curve, and therefore is device-dependent, which means that, in practice, the points must be re-sampled. Also, the complexity of the elastic matching algorithm is quadratic in the number of points. The number of re-sampled points is an arbitrary parameter which must be tuned experimentally. There exist modifications of the elastic matching algorithm of subquadratic complexity (also considered in this paper), which involve additional arbitrary parameters. However, even with all parameters carefully chosen, the elastic matching distance remains several times slower than the Euclidean distance (described below).

Third, and more importantly, the elastic matching distance is not a metric, as it is not symmetric and does not satisfy the triangle inequality. Therefore, it is difficult to use this distance in conjunction with other methods that rely on the properties of the vector space. We will show that faster and more accurate distance measures can be obtained by utilizing such properties.

One useful metric distance is the integral of the Euclidean distance between curves. This can be computed directly, from the point sequences. However, there is a much more efficient and device-independent way, based on the representation of the curve by the coefficients of the truncated series of its coordinate functions, with respect to an orthogonal functional basis [1]. There are actually many different Euclidean distance measures in the space of parametric curves, which are induced by various functional inner products. Among them, Chebyshev, Legendre, and Legendre-Sobolev inner products are widely used in signal processing [15]. It has been shown [4] that the Legendre-Sobolev distance provides the most accurate comparison of handwritten symbol curves, among the Euclidean distance measures.

The Legendre-Sobolev coefficients can be computed by an on-line algorithm [3], as the curve is written, and allow fast normalization of the size of the symbol after the pen is lifted. Moreover, as we shall see, only a few coefficients (10–12) and a few bits per coefficient (7–8) yield a sufficiently accurate representation of the curve. It turns out that, once the Legendre-Sobolev coefficient vectors are computed, it does not matter much which distance is used to compare them. The Manhattan distance is especially attractive, because it allows for an efficient implementation in about 50 machine instructions, or 20 nanoseconds on a modern PC.

The accuracy of nearest neighbor classification based on the Euclidean distance measures is lower than that of elastic matching. However, a refined method, based on the distance to the convex hull of several nearest neighbors [7, 24], performs significantly better than elastic matching. This method effectively takes advantage of the *local convexity* [14] of the curve classes (see Section 4.4). The distance to the convex hull of a few points can be computed fast [7, 18]. More importantly, this needs to be done only for a few most relevant classes, which can be reliably determined using nearest neighbor ranking with Manhattan distance.

The choice of the parameterization of the curve is also known to significantly affect the classification accuracy. It has been shown [4] that, for any of the above distance measures, parameterization by the Euclidean arc length yields best results when compared to parameterization by time or affine arc length.

To summarize the main results, among the distance measures we have considered, a combination of Manhattan and Euclidean distances gives the best performance in terms of classification accuracy, time and space complexity. The best curve representation is as vectors of Legendre-Sobolev coefficients of the coordinates, as functions of arc length. The Euclidean distance to the convex hulls of nearest neighbor classes (representing curves as described) gives the best classification accuracy. The Manhattan distance to select candidate classes for consideration (using Euclidean distance to convex hulls, as described) gives the fastest classification.

This paper is organized as follows. In Section 2, we introduce the representation of a parametric curve by the Legendre-Sobolev coefficient vector and show how to compute this vector on-line, as the curve is written. In Section 3, we describe the dataset used in our experiments. In Section 4, we present the results of comparison of various distance measures by cross-validation and choose the optimal distance measure, taking into account classification accuracy, time, and space complexity. Section 6 concludes the paper.

2 Representation of parametric curves

Most existing methods of curve classification are based on the representation of a parametric curve by a sequence of points. This representation is readily available from the digital pen, which samples points from a curve with a certain frequency and outputs the sequence of their coordinates in real time. We disregard the additional information about the curve such as, for example, pen tip pressure or angle, in order to remain device-independent. Since the sampling frequency and resolution also depend on the device, the sequence must be size-normalized and resampled, in order to obtain a device-independent representation. Because the size and length of the curve are unknown until the entire curve is traced, these operations must be postponed until pen-up. Thus, the *time* of computing a device-invariant representation of the curve by a sequence of points inevitably depends on the device (and on the curve). Therefore, we propose to use another representation, which is also device-independent and can be computed mostly on-line, as the curve is written, with a small constant-time overhead after pen-up.

Let $x(\lambda)$ and $y(\lambda)$, $\lambda \in [0, L]$, be the coordinate functions of a parametric curve. We will represent the curve by the first $d + 1$ Legendre-Sobolev coefficients of the coordinate functions. Note that the idea is very similar to that of Fourier coefficients, the only difference being the choice of the functional inner product. The Legendre-Sobolev inner product is a natural choice, for the following reasons:

1. Chebyshev, Legendre, and Legendre-Sobolev series show similar convergence rates [1, 3, 4].
2. Legendre and Legendre-Sobolev inner products are the easiest ones to compute and, unlike the other inner products with non-linear weighting functions (e.g., Chebyshev or Fourier), can be computed on-line, as the curve is written (see [3] for Legendre and [4] for Legendre-Sobolev).
3. Classification accuracy improves when we replace Legendre inner product by Legendre-Sobolev, because the latter better incorporates the information about the derivatives of the coordinate functions, *i.e.*, the directional features of the curve.

The Legendre-Sobolev inner product of two functions $f(\lambda)$ and $g(\lambda)$ on $[0, L]$ is defined as

$$\langle f(\lambda), g(\lambda) \rangle = \int_0^L f(\lambda) g(\lambda)\, \mathrm{d}\lambda + \mu \int_0^L f'(\lambda) g'(\lambda)\, \mathrm{d}\lambda,$$

where μ is a real parameter (which we assume to be fixed). Consider the orthonormal polynomial basis $B_0(\lambda), B_1(\lambda), B_2(\lambda), \ldots$, with $\deg B_i(\lambda) = i$, which is uniquely determined by the inner product and can be obtained through Gram-Schmidt orthonormalization of the monomial basis $1, \lambda, \lambda^2, \ldots$. Then any function satisfying suitable smoothness conditions can be represented by its infinite Legendre-Sobolev series

$$f(\lambda) = \sum_{i=0}^{\infty} f_i B_i(\lambda),$$

where the Legendre-Sobolev coefficients f_i are given by the inner products

$$f_i = \langle f(\lambda), B_i(\lambda) \rangle.$$

Then the truncated Legendre-Sobolev series of order d

$$\tilde{f}(\lambda) = \sum_{i=0}^{d} f_i B_i(\lambda)$$

is a polynomial of degree at most d, which can be thought of as the projection of the function $f(\lambda)$ on the subspace of polynomials of degree at most d.

If the function is sufficiently smooth and has a small number of extrema, then it can be well approximated by a polynomial of a low degree, and so the projection on a low-dimensional polynomial subspace will be a close approximation to the function. The coordinate functions of symbol curves, as functions of time, do look smooth (even where the curve has cusps or corners) and have few extrema (at most 10–15). It is not surprising therefore that they can be accurately approximated by truncated series of order about 10, to the extent that the approximation error is unnoticeable to a human eye. When the curve is parameterized by arc length, the coordinate functions are no longer smooth, and the approximation will tend to "cut" cusps and corners; however, truncation order 10–15 still appears to be sufficient to approximate most symbol curves so that they can be recognized unambiguously by a human reader [3, 4]. Multi-stroke symbols can also be accurately approximated by the truncated Legendre-Sobolev series, after consecutive strokes are joined [5].

Let $x_0, x_1, \ldots, x_d$ be the first $d + 1$ Legendre-Sobolev coefficients of $x(\lambda)$, and similarly for $y(\lambda)$. Since $B_0(\lambda) = 1$ (for any inner product), point (x_0, y_0) can be thought of as the curve's center, and therefore we can normalize the curve's position by simply omitting these coefficients. To complete normalization, we consider size as any non-zero linear dimension of the symbol, such as width, height, length, diameter, or perimeter. In our context, this definition of size up to a constant factor is sufficient. The norm of the vector formed by the remaining coefficients,

$$(x_1, \ldots, x_d, y_1, \ldots, y_d)$$

is proportional to the size of the symbol and therefore we can normalize size by normalizing this vector. The resulting normalized vector represents the curve by a point in a $2d$-dimensional vector space and will be used for classification.

We next turn our attention to a fast method for computing the Legendre-Sobolev coefficients [3, 4]. As the curve is written and a sequence of points is sampled from it, accumulate the moment integrals

$$m_i(x, L) = \int_0^L x(\lambda)\lambda^i \,\mathrm{d}\lambda, \quad m_i(y, L) = \int_0^L y(\lambda)\lambda^i \,\mathrm{d}\lambda, \quad i = 0, \ldots, d.$$

As pointed out in [3], general-purpose numerical integration schemes, such as the trapezoidal rule, fail to approximate higher-order moments with sufficient precision. Instead, on each sub-interval, we compute the integral of λ^i exactly and multiply the result by average value of the coordinate function at the end points; see [3, Section 3] for details.

Consider the Legendre-Sobolev inner product on $[0, 1]$ and apply integration by parts:

$$\langle f(\lambda), B_i(\lambda)\rangle = \int_0^1 f(\lambda)B_i(\lambda) \,\mathrm{d}\lambda + \mu f(\lambda)B_i'(\lambda)\Big|_0^1 - \mu \int_0^1 f(\lambda)B_i''(\lambda) \,\mathrm{d}\lambda.$$

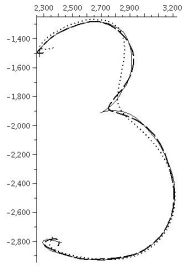

Fig. 1. Approximations of symbol '3' (solid) by Legendre series, degree 8 (dots) and 17 (dashes).

We observe that since $B_i(\lambda)$ are polynomials, the Legendre-Sobolev coefficients of $f(\lambda)$ can be obtained as linear combinations of the moments of $f(\lambda)$ and values of $f(\lambda)$ at $\lambda = 0$ and $\lambda = 1$. The matrix of the linear transformation from moments to the Legendre-Sobolev coefficients can be computed in advance. For the particular case of $\mu = 0$, which corresponds to the Legendre inner product, the explicit expressions for its entries are given in [3, Section 6].

If $f(\lambda)$ is defined on an interval $[0, L]$, rather than $[0, 1]$, we can first apply a linear substitution $\lambda \mapsto \lambda/L$ to the moments, *i.e.*, divide $m_i(x, L)$ and $m_i(y, L)$ by L^i. Note that both the substitution and the linear transformation require a number of operations that depend only on the truncation order. As we have seen earlier, the truncation order has a small value, intrinsic to the character set (in our experiments we try different values from 6 to 12) so methods with quadratic complexity are perfectly acceptable. A thorough analysis of the approximation accuracy is carried out in [1, 3]. For an illustration of the convergence, please refer to Figure 1. Thus, the time spent to compute the representation *after pen-up* can be neglected, even on the most inexpensive hardware. The remaining challenge is therefore to classify the curve fast among several hundreds of classes.

Note that the truncated Legendre-Sobolev coefficient vector contains nearly complete information about most characters, because a human can easily recognize the character by looking at its approximation. Arguably, since there is no substantial loss of information at the representation stage, classification algorithms based on this representation should perform well. Another important property of the representation is that, mathematically, it is just a linear projection of the parametric curve onto a finite-dimensional subspace of truncated Legendre-Sobolev series. Under projections, the images of convex sets remain convex and, as is shown in Section 4.4, convexity is a useful property in our problem. Ultimately, a numerical representation that preserves essential information about the characters and allows high classification accuracy should give a new insight into the processes of handwriting recognition and generation as performed by humans.

Sym	#	Sym	#	Sym	#	Sym	#	Sym	#	Sym	#
a	507	A	50	$\mathcal{A}$	14	$\mathbb{R}$	14	α	184	A	—
b	431	B	54	$\mathcal{B}$	51	$\emptyset$	415	β	95	B	—
c	642	C	642	$\mathcal{C}$	87			γ	166	Γ	115
d	484	D	48	$\mathcal{D}$	26	$\hbar$	33	δ	125	Δ	82
e	463	E	30	$\mathcal{E}$	164	ℓ	376	ϵ	164	E	—
f	460	F	74	$\mathcal{F}$	10	∞	221	ζ	57	Z	—
g	359	G	69	$\mathcal{G}$	17			η	100	H	—
h	361	H	44	$\mathcal{H}$	—	1	2508	θ	336	Θ	71
i	697	I	52	$\mathcal{I}$	134	2	2784	ι	67	I	—
j	440	J	134	$\mathcal{J}$	—	3	2042	κ	65	K	—
k	392	K	294	$\mathcal{K}$	47	4	1887	λ	81	Λ	197
l	443	L	60	$\mathcal{L}$	66	5	1833	μ	89	M	—
m	382	M	53	$\mathcal{M}$	25	6	1785	ν	174	N	—
n	789	N	55	$\mathcal{N}$	12	7	1800	ξ	80	Ξ	73
o	—	O	—	$\mathcal{O}$	43	8	1773	o	—	O	—
p	410	P	183	$\mathcal{P}$	—	9	2070	π	249	Π	327
q	1704	Q	76	$\mathcal{Q}$	—	0	2675	ρ	168	P	—
r	347	R	71	$\mathcal{R}$	43			σ	74	Σ	407
s	841	S	773	$\mathcal{S}$	—			ς	64		
t	825	T	148	$\mathcal{T}$	13			τ	69	T	—
u	483	U	626	$\mathcal{U}$	—			υ	177	Υ	—
v	694	V	602	$\mathcal{V}$	—			ϕ	95	Φ	183
w	456	W	408	$\mathcal{W}$	—			χ	205	X	—
x	967	X	768	$\mathcal{X}$	—			ψ	164	Ψ	209
y	555	Y	152	$\mathcal{Y}$	—			ω	170	Ω	102
z	520	Z	477	$\mathcal{Z}$	14						

Table 1. Letter and digit classes used in the experiments.

3 Experimental setting

For our experiments we have assembled a dataset of 50,703 handwritten mathematical symbols from 242 symbol classes. 26,139 samples were collected at the Ontario Research Centre for Computer Algebra, 14,802 samples (of digits) were borrowed from the UNIPEN handwriting database [10], and 9,762 samples from the LaViola database [16], which includes digits, Latin letters, and a few common mathematical symbols. The data was converted to a uniform InkML format [11] and stored in a single file. For each symbol, the number of strokes and the x and y coordinates of the sample points are available; for some symbols, we also have information about the timing, pen pressure, pen-up strokes, and context (a MathML description of the formula containing the symbol), which was disregarded in the present experiments. The classes (chosen according to the MathML labelling standard, when possible), with the number of samples available for each class, are listed in Tables 1 and 2.

All samples were visually inspected in order to correct accidental labelling errors. Manual relabeling was necessary in order to obtain more accurate error rates for the classifiers. Indeed, for samples that are hardly recognizable by a human as members of the assigned class, the expected output of classification is essentially undefined, therefore such samples do not provide any valuable information and only increase the vari-

(	780	)	—	=	578	≠	323	$\bar{a}$	10	≐	81
{	248	}	37	≡	102	≢	71	$\tilde{a}$	10	≆	23
[	56	]	64	≈	88	≉	61	$\ddot{a}$	10	≠	44
⟨	26	⟩	23	≅	95	≇	27	$\dot{a}$	10		
⌈	—	⌉	26	∼	51	≁	77				
⌊	10	⌋	16	≃	91	≠	69	+	937	|	1809
								−	1212	‖	130
∈	143	∋	70	∉	76	∌	69	×	555		
⊂	127	⊃	111	⊄	66	⊅	74	÷	89		
⊆	100	⊇	96	⊈	59	⊉	79			∠	—
⊏	74	⊐	75	⋢	68	⋣	61	!	106	∝	194
⊑	76	⊒	76	⊀	61	⊁	74	?	44	…	44
<	324	>	351	≮	73	≯	82	:	44	%	11
≤	104	≥	84					;	17		
				⊥	58	⊤	135			∂	413
≪	100	≫	101	∴	17	∵	17	∀	35	∇	11
≺	44	≻	51	±	116	∓	81	∃	27	∫	412
≼	20	≽	14	⇑	58	⇓	68			√	397
⊣	56	⊢	68	↑	66	↓	73	⊙	77	∐	13
/	142	\	88	∩	153	∪	173	⊕	71	′	62
				⊓	65	⊔	80	⊗	84	⋆	23
→	138	←	70	∧	186	∨	462	⊎	63	*	18
⇒	93	⇐	87							†	10
⇝	55	⇜	41					⇔	91	✓	33
↪	77	↩	56					↔	86		
⇌	65	⇋	54					↦	14		
↗	69	↖	55								
↘	63	↙	80								

Table 2. Special mathematical symbol classes used in the experiments.

ance in the resulting error rates. Thus, samples that appeared totally unrecognizable were discarded. Samples that belong to more than one class, as determined by visual inspection, were attributed with all applicable labels (see Figure 2 for examples). Some classes were so similar (*i.e.*, had no examples of symbols that certainly belong to some but not all of them) that it was decided not to attempt to distinguish among them. Examples of such groups of similar classes include "Capital O, little o, omicron, zero", capital Greek letters with Latin analogues, and certain calligraphic letters without clearly pronounced calligraphic features. All samples of such classes were labeled with the same, most commonly used, label. Whenever a class contained at least one symbol that is clearly a member of this, and no other, classes, we retained the label. As a result, we obtained 38,493 samples belonging to a single class, 10,224 samples to 2 classes, 1,954 samples to 3 classes, 19 samples to 4 classes, and 13 samples to 5 classes.

We also included the number of strokes in the class labels. This decision was based on an experiment described in [5], where including the number of strokes in the class label yielded better classification results than using it as an entry in the feature vector. This is the case despite the fact that the number of classes raises to 378, as we include the number of strokes in the label. The reason why the number of strokes is not a good

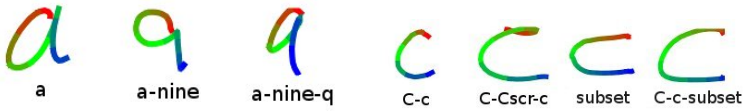

Fig. 2. Ambiguous symbols labeled with multiple class labels.

candidate for a coordinate in the feature vector is that, unlike the other features we consider, it violates the property of convexity, which we discuss below in Section 4.4.

We furthermore noticed that some symbols could be written in more than one way. It is reasonable to assume that a symbol can be classified correctly only if sufficiently many instances of this symbol form are available in the database. Therefore, we discarded all symbol forms which had fewer than 9 instances. However, we do *not* use allomorph labels in the classification. The classes of small symbols "`dot`" and "`comma`", as well as class "`rpar`" (closing parenthesis) which has a large overlap with the class of commas, were not considered. Arguably, small symbols should be classified separately, using a method that takes into account their relative size. All methods discussed in this paper apply size normalization before classification. The quoted results apply to the data set described.

The normalized Legendre-Sobolev coefficient vectors were pre-computed for all samples, with the value of the Legendre-Sobolev parameter μ set to $1/8$ (it has been determined earlier [7] that this value yields good results for various classification methods).

In order to evaluate the performance of various classification methods, we used a 10-fold cross-validation. The entire dataset of 50,703 symbols was randomly partitioned into 10 subsets, preserving the proportions of class sizes. Then, samples from each subset were classified, using the samples from the other 9 subsets as training data. A sample was considered to be classified correctly if the composite label attributed to it by the classification algorithm overlapped with the ground truth composite label. For example, if the sample is labelled "`4`" and the classifier outputs "`4` or `y`", the sample is correctly classified. We allow such ambiguous output from the classifier, because in many cases there is no meaningful way to assign a single label to a symbol. For example, a human cannot decide "`4`" or "`y`" for elements of "`4` or `y`" and the ambiguity can only be resolved by taking into account contextual information. Then, the percentage of mis-classified samples was calculated. Note that each mis-classified symbol contributes about 0.002% to the overall error rate.

Our experiments were performed on a PC with a 2.4 GHz Intel Dual Core processor, 2GB of RAM, running Ubuntu Linux 8.04. The algorithms were implemented in C++. The distances to the convex hulls were computed with the help of the linear algebra library `Lapack++`, release 2.5.3.

4 Classification

In this section we present the results of nearest neighbor classification using the following distance measures:

- elastic matching distance,
- Euclidean distance on Legendre-Sobolev coefficient vectors,
- Manhattan distance on Legendre-Sobolev coefficient vectors, and
- Euclidean distance to the convex hull of several nearest neighbors.

We compare the cost and accuracy these measures and present a combination that is efficient in both time and space. Our goal is to identify a classification method that is as fast, compact and accurate as possible in principle. We recognize that there are trade-offs between these objectives, so for concreteness we take our *reference mobile processor* as one with a clock of 100 MHz for 32 bit operations and at most 1 megabyte of memory available for use by recognition.

4.1 Elastic matching

The square elastic matching distance [19–22] from a point sequence $A = a_1, \ldots, a_n$ to a point sequence $B = b_1, \ldots, b_n$ (of equal length) is defined as the minimum over all non-decreasing functions $r : \{1, \ldots, n\} \to \{1, \ldots, n\}$ of the square Euclidean distance

$$\sum_{i=1}^{n} \|a_i - b_{r(i)}\|^2$$

It can be computed in $O(n^2)$ operations using a dynamic programming algorithm [22]. A *sliding window* version of elastic matching [13, 23] is often used, in which a parameter q, $1 \le q \le n$ is introduced, and the values $r(i)$ are restricted to the interval $[i - q, i + q]$, for each $i = 1, \ldots, n$. With this restriction, the distance can be computed in $O(nq)$ operations.

It has been established earlier [4] that the elastic matching distance yields best results when the points $a_1, \ldots, a_n$ are sampled from the curve so that the arc lengths between the adjacent points are equal. It has been also established [4, Table 6] that 1-nearest neighbor (NN) classification yields better results than 3-NN or 5-NN when many small classes (with about 10 samples per class) are present, which is our case. This applies to elastic matching and euclidean distance-based classification, therefore, we restrict our analysis to 1-NN in what follows.

By cross-validation, we determined that the lowest error rate of 1-nearest neighbor classification, equal to 3.0%, is obtained for $n = 30$ and $q = 2$. The error rates for other values of n and q are shown in Table 3. Observe that the optimal value of q is proportional to n and approximately equal to $n/15$. There is no improvement observed for $n > 30$.

The algorithm for computing the elastic matching distance with the sliding window, in the special cases of small values of q, admits an efficient implementation with no dynamic memory allocation and completely inlined code. For example, the computation

q \ n	10	12	14	16	18	20	22	24	26	28	30	40	50	60
1	5.1	4.7	4.1	3.7	3.6	3.5	3.4	3.4	3.3	3.4	3.4	3.7	3.9	4.0
2	5.3	5.0	4.5	4.2	3.8	3.8	3.4	3.4	3.2	3.1	**3.0**	3.2	3.3	3.4
3	5.4	5.1	4.7	4.4	4.0	4.1	3.8	3.7	3.5	3.5	3.4	3.2	3.1	3.1
4												3.5	3.2	3.1
5													3.4	3.2

Table 3. Elastic matching error rates in %.

of a single distance for $n = 20$ and $q = 1$ takes 0.25 microseconds on our experimental PC. Assuming that the maximal affordable delay is 100 milliseconds and our reference mobile processor to be up to 50 times slower than our experimental PC, we obtain that about 8,000 models can be processed. We conclude that the elastic matching distance is not accurate enough and not fast enough for our purposes and proceed to the study of alternatives.

4.2 Euclidean distance

In this and the following sections, we consider the distance measures that are perhaps among the fastest possible. As we shall see, these distances are less accurate than the elastic matching distance. However, they are perfectly suitable for selecting the top T candidate classes, as well as finding the nearest neighbors in each class. Then, the distance to the convex hull of these neighbors will be used to select from among the top T.

Using the usual formula for the square Euclidean distance, we obtain an algorithm that computes the distance in $3D - 1$ arithmetic operations, or 0.12 microseconds (for dimension $D = 24$, with floating point arithmetic). This is twice faster than elastic matching, but still not fast enough, as it will allow to process about 16,000 samples on our reference mobile processor (see end of previous section for the calculations). Therefore, a faster distance is needed.

A significant improvement can be achieved by using integer arithmetic. This is particularly useful on mobile devices, many of which do not support floating point instructions. It turns out that Legendre-Sobolev coefficients can be computed with a very low precision, without compromising the correct retrieval rates. As Table 5 suggests, 7 bits per coefficient are enough. However, the computation of the squares in the Euclidean distance requires each coefficient to be stored in a separate machine word, which makes it difficult to take full advantage of the low precision. For this reason, we consider the Manhattan distance as an alternative.

4.3 Manhattan distance

The *Manhattan distance*, or L_1 distance, between two coefficient vectors a and b is the sum of the lengths of the projections of $a - b$ on the coordinate axes:

$$\|a - b\|_1 = \sum_{i=0}^{D-1} |a_i - b_i|.$$

We show how to compute the Manhattan distance quickly by operating on several coordinates at once in a machine word. Here we give the details for 32-bit words and simply note that the same approach can be applied with words of other sizes.

In our setting we can assume that the number of coefficients $D = 4m$ is a multiple of 4 and that each coefficient is an integer number between -63 and 63, inclusive. Then D coefficients can be stored in m 32-bit words as follows. For $k = 0, 1, 2, 3$, use bits $8k, \ldots, 8k + 6$ to store a coefficient using ones' complement binary representation, in which negative numbers are represented by the bitwise complement of their absolute values. Bits $8k + 7$ are assumed to be zeroes.

With this representation the Manhattan distance can be efficiently computed as follows. Consider word 0, in which coefficients for $i = 0, 1, 2, 3$ are stored. Assume that all four of a_i, $i = 0, 1, 2, 3$ are stored in a 32-bit word, as described above, as are $-b_i$, and let a and b denote these two words, respectively. Note that the negations $-b_i$ can be pre-computed in advance. Then the word containing $|a_i - b_i|$, for $i = 0, 1, 2, 3$, can be computed as shown in Figure 3. Note that the resulting absolute values again occupy at most 7 bits in each byte. To accumulate their sum, we can first compute

$$w = (w + (w \gg 8)) \;\&\; \texttt{0x00FF00FF}$$

for each word, then add the m words and, finally, apply

$$w = (w + ((w \gg 16)) \;\&\; \texttt{0xFFFF}$$

to the result. The total number of instructions is $6m + 3m + (m - 1) + 3 = 10m + 2$.

In comparison, the Euclidean distance requires $3D - 1 = 12m - 1$ operations. The timings for the Euclidean and Manhattan distances on 32-bit and 64-bit machines are shown in Table 4. Note that the real speed-up is much greater than 12/10: the Manhattan distance turns out to be about 3 times faster on a 32-bit machine and about 4–5 times faster on a 64-bit machine. This speed-up is largely due to the effects of packing, which ensures that the computation of the Manhattan distance proceeds in machine registers, while the Euclidean distance requires retrieval of data from memory. The timings suggest that, on our reference mobile processor, we can compute the Manhattan distance from a test sample to more than 80,000 training samples, for dimension 24, in under 100 milliseconds. This rough estimate suggests that, with the proposed methods, recognition speed is not an issue even on mobile devices, and the main concern becomes the memory required to store the samples. We propose various methods for compact representation in Section 4.4.

As Table 5 shows, the error rates of nearest neighbor classification with Manhattan distance are very similar to those with the Euclidean distance. These error rates are

Procedure 1

INPUT:

Two 32-bit words, a and b, storing a_i and $-b_i$, $i = 0, 1, 2, 3$ in sequential 8-bit regions.

OUTPUT:

One 32-bit word, w, storing $a_i - b_i$, $i = 0, 1, 2, 3$ in sequential 8-bit regions.

METHOD:

1. $w = a + b$ *(where + denotes unsigned integer addition)*
2. $s = (w \;\&\; \texttt{0x80808080}) \gg 7$ *(mask carries and shift them to position 0)*
3. $w = w + s$ *(values of $|a_i - b_i|$ in ones' complement)*
4. $p = s * \texttt{0xFF}$ *(propagate carries to the higher bits, for each byte)*
5. $w = w \oplus p$ *($\oplus$ denotes bitwise exclusive or).*

Fig. 3. Parallel absolute differences with bit fields

Distance \ Dimension	12	14	16	18	20	22	24
Euclidean (double), 32 bit	96	108	127	142	164	183	201
Euclidean (integer), 32 bit	61	68	76	82	92	99	107
Manhattan, 32 bit	20		24		28		34
Euclidean (double), 64 bit	79	86	93	99	106	113	121
Euclidean (integer), 64 bit	65	69	73	76	79	82	85
Manhattan, 64 bit	13		17		20		24

Table 4. Timings of the Euclidean and Manhattan distances in nanoseconds.

higher by about 0.5% than the best rate for the elastic matching distance, but the accuracy is sufficient to determine the top 3 classes. In order to break the ties among these most relevant classes, we will employ a more accurate distance, as described below.

4.4 Convex hull of nearest neighbors

It has been observed earlier [5–7] that classes of handwritten symbol curves satisfy the following property of *convexity*. Consider two curves that belong to the same class. If we gradually morph one curve into the other, using a linear homotopy, then the intermediate curves are expected to belong to the same class as well, as shown in Figure 4.

This means that, whenever two points belong to a curve class in the infinite-dimensional space of parametric curves, the entire straight line segment connecting these points is contained in this class. In other words, curve classes are convex sets. This property may fail to hold if the two curves are different allomorphs of the same symbol; such counter-examples can be found in [7]. However, it is safe to assume that, if the two curves from the same class are sufficiently close to each other (*e.g.*, in terms of the Euclidean distance), then linear homotopy between them will always produce similar curves.

Distance (dimension) \ # bits	6	7	8	9	10	12
Euclidean (12)	6.2	5.1	5.0	4.9	4.9	4.9
Manhattan (12)	6.6	5.5	5.3	5.3	5.2	5.2
Euclidean (16)	4.9	4.1	4.1	3.9	4.0	4.0
Manhattan (16)	5.3	4.5	4.3	4.2	4.2	4.2
Euclidean (20)	4.4	3.8	3.6	3.5	3.5	3.6
Manhattan (20)	5.1	4.2	3.9	3.8	3.8	3.8
Euclidean (24)	4.4	3.7	3.6	3.5	3.5	3.5
Manhattan (24)	5.0	4.1	4.0	3.9	3.8	3.8
Euclidean (12, top 2)	1.6	1.4	1.3	1.3	1.3	1.3
Manhattan (12, top 2)	1.6	1.4	1.2	1.1	1.2	1.1
Euclidean (24, top 2)	1.0	0.9	0.9	0.9	0.9	0.9
Manhattan (24, top 2)	0.5	0.3	0.3	0.3	0.3	0.3
Euclidean (12, top 3)	0.1	0.2	0.2	0.2	0.2	0.2
Manhattan (12, top 3)	0.2	0.2	0.2	0.2	0.2	0.2
Euclidean (24, top 3)	0.1	0.1	0.1	0.1	0.1	0.1
Manhattan (24, top 3)	0.1	0.0	0.0	0.0	0.0	0.0

Table 5. Euclidean and Manhattan distance error rates in % *vs* coefficient size in bits.

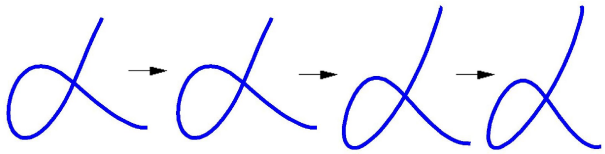

Fig. 4. Linear homotopy between two class members stays within the class.

The fact that the property of convexity holds only locally for the curve classes has implications for the choice of the classification method. Indeed, if the classes were globally convex, then they would be linearly separable (provided that the classes do not overlap). For such classes, classifiers based on linear support vector machines are a natural choice [5, 6]. On the other hand, for classes that do not exhibit any convexity, linear support vector machines get outperformed by distance-based nearest neighbor classification. The disadvantage of the latter family of methods is that they essentially bypass the learning stage and require very large training sets. The distance to the convex hull of nearest neighbors [7, 24] can be viewed as a combination of the two approaches, which is particularly suitable for classifying among locally convex classes.

When the number of points is less than the dimension of the vector space their convex hull is a simplex (provided that the points are in generic position). The distance from a point to a simplex can be computed by an efficient algorithm [7, 18] presented in Figure 5.

It has been shown experimentally [7] that, for the purpose of curve classification, the optimal number of nearest neighbors in the convex hull is indeed smaller than the

Procedure 2

INPUT:

Point x and vertices $\{s_1, \ldots, s_k\}$ *of a simplex.*

OUTPUT:

Distance from x to the simplex.

METHOD:

1. **if** $k = 1$ **then return** $\|x - s_1\|$
2. *Find* $v = \sum_{i=1}^{k} \alpha_i s_i$ *such that* $\langle v - x, v - s_i \rangle = 0$, $i = 1, \ldots, k$.
3. **if** $\alpha_i \geq 0$ *for all* $i = 1, \ldots, k$ **then return** $\|v - x\|$
4. *Let* $S_+ = \{\ s_i \mid \alpha_i \geq 0,\ i = 1, \ldots, k\ \}$
   return *DistToSimplex(x,* S_+*)*

Fig. 5. Algorithm to compute the distance to a simplex.

dimension. Given that the number of neighbors is small and the classes are locally convex, it is unlikely that convex hulls of nearest neighbors from different classes will overlap (unless the classes themselves overlap, which is again unlikely in the presence of multiple labels). An exact measurement of the volume of these overlaps, however, would require rather sophisticated computations, which are beyond the scope of this paper.

It has been also shown [7] that the distance to the convex hull of nearest neighbors successfully breaks the ties among the top few classes produced by a support vector machine classifier. We present the results of experiments, which demonstrate that the pre-classifier based on support vector machines can be replaced by the nearest neighbor classification with Manhattan distance. The latter method is advantageous in three respects: it does not require convexity of classes, and hence allomorph labels (which are very tedious to obtain), is faster, and requires less storage.

To summarize our combined classification method, we list all the steps and parameters involved in Figure 6. In order to reduce the time complexity and space requirements, one can use a certain subset of training samples in steps 2 and 3. Intuitively, the samples on the boundary of classes play a more important role than those in the interior. The *significance* S of a training sample X from a class C can be estimated by computing the number of samples from the other classes, for which X is the nearest neighbor, among the samples from C. It turns out that discarding the samples of low significance does not diminish classification accuracy and allows to reduce the size of the training set almost by a factor of 2.

Figure 8 shows the error rates of the combined classification method for various values of the parameters D, b, and S, together with the corresponding storage requirements. It has been established earlier [7] that the value of $k = 11$ yields best results for the distance to the convex hull of nearest neighbors. The error rates also reach their minimum for $T = 10$ (3.9% for $T = 1$, 2.6% for $T = 5$, 2.5% for $T \geq 10$). The reason why we need T higher than 3, even though the top-3 error rate for the Manhattan distance is close to zero, is because, in the presence of overlaps, it is sometimes beneficial

Procedure 3

INPUT:

A test curve C.
Positive integer parameters b, k, D and T.
A set of classified points $\mathbf{DB} \subset \mathbf{Cl} \times \mathbb{Z}_b^D$, where $\mathbf{Cl}$ is the set of classes and $\mathbb{Z}_b$ is the set of integers with b or fewer bits.

OUTPUT:

The classification of C (i.e. a member of $\mathbf{Cl}$.)

METHOD:

1. *Compute the D-dimensional normalized vector of Legendre-Sobolev coefficients of the coordinate functions of C. Store each coefficient as a b-bit integer.*
2. *Apply nearest neighbor classification and select the closest T classes using the Manhattan distance.*
3. *In each of the top T classes choose k nearest neighbors and find the Euclidean distance from the test point to their convex hulls. Output the class for which this distance is minimal.*

Fig. 6. Curve classification by convex hull of nearest neighbors (CHNN)

Top-1	Top-2	Top-3	Top-4	Top-5	Top-6	Top-7	Top-8	Top-9	Top-10
21.75%	6.25%	2.96%	2.33%	2.07%	1.94%	1.80%	1.76%	1.68%	1.62%

Table 6. Error rates when class labels are chosen randomly from a multi-class output.

to leave among the top T classes more than one intersection of the correct class with a neighboring class.

Note that the above rates were computed in the presence of multi-class labels. That is, a test symbol was considered to be correctly classified, if its multi-class label has at least one class in common with the multi-class label returned by the classifier. In the setting where we are going to apply this classifier, such ambiguities will be resolved by taking into account statistical contextual information. However, it could possibly be used in other settings, where a single class is needed on the output. In that case, when multiple classes are returned, we can choose the correct one at random, with the probabilities weighted by the corresponding numbers of samples per class. The resulting error rates, for test symbols labelled with their original single-class labels as presented to the writers, are shown in Table 6 (the parameters are the same as the ones that yielded 2.5% error rate for multi-class labels). The high top-1 error rate of about 20% is not very surprising, given that the classification is based on inherently ambiguous multi-class labels and a random choice to disambiguate them, and that about 20% of all samples are labeled with at least two classes.

We examined the mis-classified symbols in order to characterize them. We found that in about 70% of the "misclassified" symbols, the symbol was unrecognizable out of context, even by a person. These could be viewed as labelling errors — that the symbol should have had multiple labels. Examples of these are shown in Figure 7 (a)–(o).

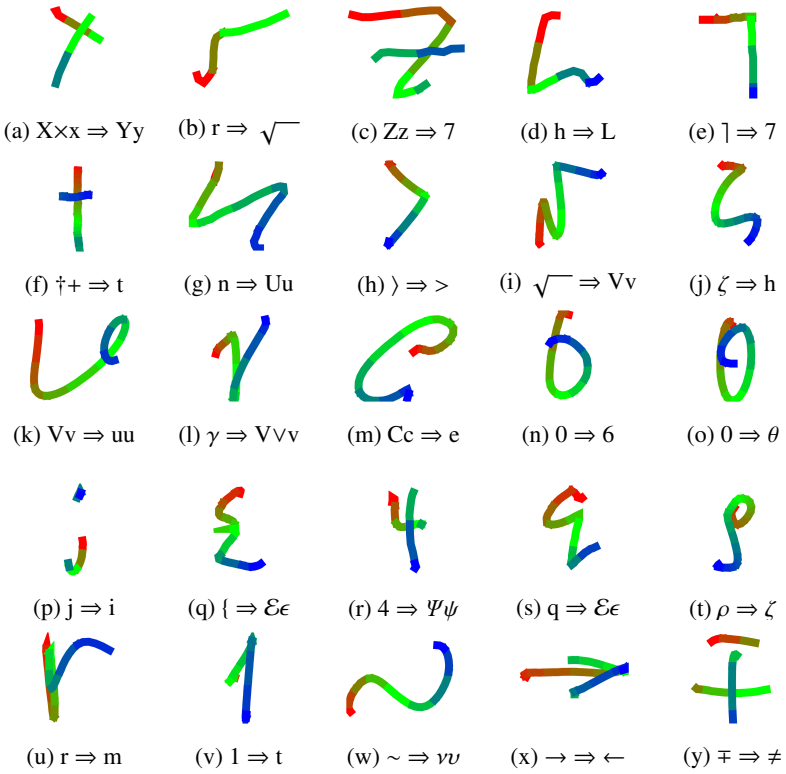

Fig. 7. Examples of mis-classified symbols. Label ⇒ assigned classification.

About 30% of of the misclassified symbols (30% of 2.5% = 0.75% overall) were genuine mis-classifications. Some of these were attributable to deficiencies in the training set, others would be correctly classified if baseline information were provided, some appear to arise when significant parts of the symbol are retraced, and others appear to be well formed symbols that were mis-identified. Typical examples of these are shown in Figure 7 (p)–(y).

The time needed to compute the distances to the convex hull of 11 points 10 times can be neglected (for any device). Therefore, on a modern PC or a more powerful mobile device, the entire available data set can be easily used, with the maximal dimension (24) and 7 bits per coefficient (there is no improvement for $b > 7$ when the dimension equals 24). For an inexpensive mobile device, on which storage is precious, we can use Figure 8 to choose the optimal values of D, b, and S, given the desired error rate and storage limitations. In order to be able to compute the Manhattan distance fast, one has to leave 1 empty bit for the carries, by reserving 1 byte but actually using 7 bits per coefficient. Taking into account this detail, a convenient choice of the parameters for a mobile device could be $D = 20$, $b = 7$, and $S = 4$, which yields an error rate of 2.8% with less than 30,000 samples, requiring less than 600 KB of storage. Experiments with

Error rate	Storage	Dimension	#bits	Significance	#Training samples
2.52%	958 KB	24	7	1	45,607
2.60%	693 KB	24	7	3	33,012
2.65%	616 KB	24	7	4	29,313
2.70%	586 KB	20	8	4	29,313
2.80%	513 KB	20	7	4	29,313
2.85%	470 KB	20	7	5	26,874
2.90%	440 KB	20	6	4	29,313
3.00%	403 KB	20	6	5	26,874

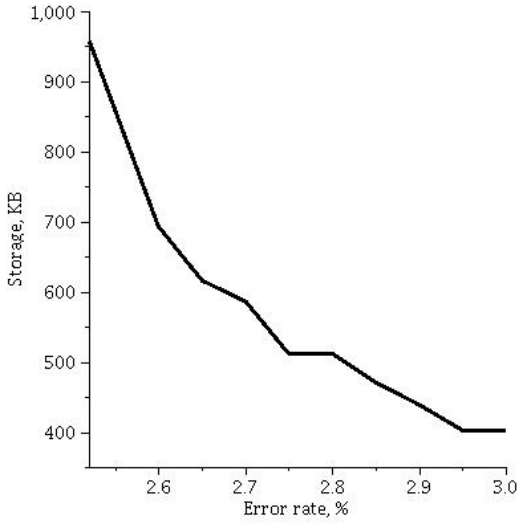

Fig. 8. Minimal storage to achieve given error rate with $\mu = 1/8$, $k = 11$, $T = 10$.

fast compression utilities suggest that the storage can be further reduced to about 450 KB. With these settings, classification is expected to take less than 50 milliseconds on our reference mobile processor.

Even though it is natural to ignore classes with few samples to improve classification accuracy, this approach presents a practical concern: in a real system, it may often happen that a user enters a new symbol which is not in the data set. In this case, the system may have to ask the user to provide additional samples, in order to be able to reliably identify members of the new class. This may be inconvenient to the user; however, since a particular writer is likely to use a particular allomorph of each symbol most of the time, it will only be necessary for the writer to provide about 10 samples of the new symbol, in order for the system to learn it well, so the burden does not appear to be too high.

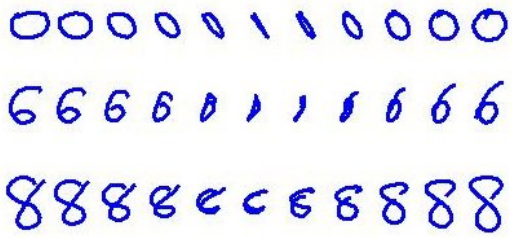

Fig. 9. Homotopy between different writing directions leads outside the class.

Top-1	Top-2	Top-3	Top-5	Top-10
4.0%	1.7%	1.0%	0.6%	0.3 %

Table 7. Error rates for linear SVM, $\mu = 1/8$ and SVM regularization constant $C = 10$.

5 Comparison with linear support vector machines

As mentioned above, symbol classes are linearly separable, but only to some extent. In fact, linear separability is consistently violated by the presence of allomorphs, i.e., different ways of writing the same symbol. There is no reason why linear homotopy between two different allomorphs should stay within the same class and, indeed, as Figure 9 illustrates, it often does not.

Linear support vector machines, naturally, will fail to distinguish between classes that are not linearly separable. One certain, albeit tedious, way to achieve linear separability is to label the allomorphs and use linear SVM to classify among the allomorph classes, rather than the original ones. This inevitably increases the number of classes and decreases the number of available training samples per class, but the classes become simpler. There are two reasons that influenced our choice of the distance to the convex hull of nearest neighbors in favor of an ensemble linear SVM: the latter classification method is based on pairwise classifiers and therefore its complexity grows quadratically in the number of classes (this is also the reason why we found ensembles of non-linear SVM impractical); the former also shows slightly lower classification rates (see Table 7 for comparison). Nevertheless, we do consider linear SVM as a competitive approach, which could be combined with the distance-based methods, for example, via measures of confidence, which are available in both cases [8]. For a detailed description of experiments with linear SVM, please refer to [7].

6 Conclusion

The analysis of various distance measures carried out in this paper suggests that a careful assessment of the quality of a distance measure cannot be based solely on the measurement of error rates produced by the corresponding nearest neighbor classifier. In the

case of handwritten symbol recognition, the elastic matching distance has been shown to yield a noticeably lower nearest neighbor classification error rate than the Euclidean distance in the space of curves. Yet we have shown that a more accurate classifier can be built using the vector-space-based representation, by working in a space with more structure and using its additional properties. By integrating a very fast pre-classifier based on the Manhattan distance with a more accurate tie-breaker based on the Euclidean distance to the convex hull of nearest neighbors, we have obtained a combined method that is clearly superior to elastic matching in terms of classification accuracy, time and space complexity. We have shown that this method allows classification of a test sample among several hundred classes with a high accuracy and is suitable for implementation in systems with very strict speed and memory requirements.

References

1. Char, B., Watt, S.M.: Representing and Characterizing Handwritten Mathematical Symbols through Succinct Functional Approximation. Proc. Intl. Conf. on Docum. Anal. and Rec. (ICDAR) (2007) 1198–1202.
2. Fujimoto, M., Suzuki, M.: AsirPad - A Computer Algebra System with a Pen-based Interface on PDA. Proc. 7th Asian Symposium on Computer Mathematics (ASCM2005), Korea Institute for Advanced Study (2005), 259–262.
3. Golubitsky, O., Watt, S.M.: Online Stroke Modeling for Handwriting Recognition. Proc. 18th Intl. Conf. on Comp. Sci. and Soft. Eng. (CASCON) (2008), 72–80.
4. Golubitsky, O., Watt, S.M.: Online Computation of Similarity between Handwritten Characters. Proc. Docum. Rec and Retrieval (DRR XVI) (2009), C1–C10.
5. Golubitsky, O., Watt, S.M.: Online Recognition of Multi-Stroke Symbols with Orthogonal Series. Proc. 10th International Conference on Document Analysis and Recognition, (ICDAR 2009), July 26-29 2009, Barcelona, Spain, IEEE Computer Society, pp. 1265–1269.
6. Golubitsky, O., Watt, S.M.: Improved Character Recognition through Subclassing and Runoff Elections. Ontario Research Centre for Computer Algebra Tech. Rep. TR-09-01. http://www.orcca.on.ca/TechReports/2009/TR-09-01.html
7. Golubitsky, O., Watt, S.M.: Tie Breaking for Curve Multiclassifiers. Ontario Research Centre for Computer Algebra Tech. Rep. TR-09-02. http://www.orcca.on.ca/TechReports/2009/TR-09-02.html
8. Golubitsky, O., Watt, S.M.: Confidence Measures in Recognizing Handwritten Mathematical Symbols. Proc. Conferences on Intelligent Computer Mathematics: 16th Symposium on the Integration of Symbolic Computation and Mechanized Reasoning and 8th International Conference on Mathematical Knowledge Management, (MKM 2009), July 10-12 2009, Grand Bend, Canada, Springer Verlag LNAI 5625, 460–466.
9. Golubitsky, O., Mazalov, V., Watt, S.M.: Orientation-Independent Recognition of Handwritten Characters with Integral Invariants. 9th Asian Symposium on Computer Mathematics (ASCM), Fukuoka, Japan, December 2009 (accepted).
10. Guyon, I., Schomaker, L., Plamondon, R., Liberman, M., Janet, S: UNIPEN Project of Online Data Exchange and Recognizer Benchmarks. Proc. 12th International Conference on Pattern Recognition (ICPR 1994), Jerusalem, Israel. IAPR-IEEE (1994) 29–33.
11. Ink Markup Language (InkML) W3C Working Draft (23 October 2006) http://www.w3.org/TR/InkML
12. Joshi, N., Sita, G., Ramakrishnan, A.G., Madhvanath, S.: Elastic Matching Algorithms for Online Tamil Character Recognition. Neural Information Processing, Lecture Notes in Computer Science **3316** (2004) 820–826.

13. Keogh, E.: Exact indexing of dynamic time warping. 28th International Conference on Very Large Data Bases (2002), Hong Kong, 406–417.
14. Klee, V.L., Jr.: Convex Sets in Linear Spaces. Duke Math. J. **18** (2) (1951) 443–466.
15. Krall, A.: Hilbert Space, Boundary Value Problems and Orthogonal Polynomials, Operator Theory: Advances and Applications, vol. 133, Birkhuser, Basel, 2002.
16. LaViola, J.J., Jr.: Symbol Recognition Dataset. Microsoft Center for Research on Pen-Centric Computing. `http://pen.cs.brown.edu/symbolRecognitionDataset.zip`
17. Li, M., Sethi, I.: Confidence-Based Classifier Design. Pattern Recognition **39** (7) (2006) 1230–1240.
18. Michelot, C.: A Finite Algorithm for Finding the Projection of a Point onto the Canonical Simplex of $\mathbb{R}^n$. J. Optimization Theory and Applications **50** (1) (1986) 195–200.
19. Munich, M. E., Perona, P.: Visual signature verification using affine arc-length. IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR99), **2** (1999).
20. Munich, M. E., Perona, P.: Visual identification by signature tracking, IEEE PAMI, vol. 25, no. 2, pp. 200–217, 2003.
21. Myers, C. S., Rabiner, L. R.: A comparative study of several dynamic time-warping algorithms for connected word recognition. The Bell System Technical Journal, 60(7): 1389–1409, September 1981.
22. Uchida, S., Sakoe, H.: A Survey of Elastic Matching Techniques for Handwritten Character Recognition. IEICE Transactions on Information and Systems E88-D(8) (2005) 1781–1790.
23. Sakoe, H., Chiba, S.: Dynamic programming algorithm optimization for spoken word recognition. IEEE Trans Acoustics Speech Signal Process ASSP (1978) **26**: 43–49.
24. Vincent, P., Bengio, Y.: K-local Hyperplane and Convex Distance Nearest Neighbor Algorithms. Adv. in Neural Inform. Proc. Systems, The MIT Press (2002) 985–992.
25. Watt, S.M.: Mathematical Document Classification via Symbol Frequency Analysis. Proc. Towards Digital Mathematics Library (DML 2008) 29–40.