

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 mathematical symbols based on the Euclidean, Manhattan, and elastic matching distances, as well as the distance to the convex hull of nearest neighbors. We show experimentally that 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. Leading developers of computer algebra systems are already actively targeting mobile applications. However, unfortunately, 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 [16]. 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.

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 [14] 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. Second, the elastic matching algorithm is sensitive to the number of points sampled from the curve, and therefore is not device-independent, 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 sub-quadratic 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). 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.

The Euclidean distance between two curves could 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. It has been shown [3] 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 [2], 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 [15, 6], performs significantly better than elastic matching. This method effectively takes advantage of the *local convexity* [10] of the curve classes (see Section 4.4). The distance to the convex hull of a few points can be computed fast [13, 6]. 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 [3] that, for any of the above distance measures, parameterization by the Euclidean arc length yields best results.

To summarize, among the considered distance measures on parametric curves, the Manhattan distance between the Legendre-Sobolev coefficient vectors of the coordinate functions parameterized by arc length, refined by the distance to the convex hull of nearest neighbors for the closest candidate classes, shows the best performance in terms of classification accuracy, time, and space complexity.

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 5 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 $x(t)$ and $y(t)$, $t \in [0, T]$, 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. 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 of two functions $f(t)$ and $g(t)$ on $[0, T]$ is defined as

$$\langle f(t), g(t) \rangle = \int_0^T f(t)g(t) dt + \mu \int_0^T f'(t)g'(t) dt,$$

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

$$f(t) = \sum_{i=0}^{\infty} c_i B_i(t),$$

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

$$c_i = \langle f(t), B_i(t) \rangle.$$

Then the truncated Legendre-Sobolev series of order d

$$\tilde{f}(t) = \sum_{i=0}^d c_i B_i(t)$$

is a polynomial of degree at most d , which can be thought of as the projection of the function $f(t)$ 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 of 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 [2, 3]. Multi-stroke symbols can also be accurately approximated by the truncated Legendre-Sobolev series, after consecutive strokes are joined [4].

Let $c_0^x, c_1^x, \dots, c_d^x$ be the first $d + 1$ Legendre-Sobolev coefficients of $x(t)$, and similarly for $y(t)$. Since $B_0(t) = 1$ (for any inner product), point (c_0^x, c_0^y) can be thought of as the curve’s center, and therefore we can normalize the curve’s position by simply omitting these coefficients. Next, the norm of the vector formed by the remaining coefficients,

$$(c_1^x, \dots, c_d^x, c_1^y, \dots, c_d^y)$$

is proportional to the size of the symbol, and therefore we can normalize the 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 [2, 3]. As the curve is written and a sequence of points is sampled from it, accumulate the moment integrals

$$m_i^x = \int_0^T x(t)t^i dt, \quad m_i^y = \int_0^T y(t)t^i dt, \quad i = 0, \dots, d.$$

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

$$\langle f(t), B_i(t) \rangle = \int_0^1 f(t)B_i(t) dt + \mu f(t)B_i'(t) \Big|_0^1 - \mu \int_0^1 f(t)B_i''(t) dt.$$

Observe that, since $B_i(t)$ are polynomials, the Legendre-Sobolev coefficients of $f(t)$ can be obtained as linear combinations of the moments of $f(t)$ and values of $f(t)$ at $t = 0$ and $t = 1$. If $f(t)$ is defined on an interval $[0, T]$, rather than $[0, 1]$, we can first apply a linear substitution $t \mapsto t/T$ to the moments. Note that both, the substitution and the linear transformation require a number of operations that only depends on the truncation order (quadratically). As we have seen earlier, the latter is small value, intrinsic to the character set (in our experiments we will try different values from 6 to 12). Thus, the time spent to compute the representation *after pen-up* can be neglected, even on the most inexpensive hardware. The challenge is to classify the curve fast among several hundreds of classes.

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 [7], and 9,762 samples from the LaViola database [11], which includes digits, Latin letters, and a few common mathematical symbols. The data was converted to a uniform InkML format [8] 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 have been visually inspected, in order to correct accidental labelling errors. Samples that appeared unrecognizable were discarded. Samples that belong to more than one class were attributed with all applicable labels. Some classes were so similar that it was decided not to attempt to distinguish between 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 resulted in 378 composite labels. Furthermore, we 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 numbers apply to the resulting data set.

In practice, it may 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. 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. This does not appear to impose a significant burden on the user.

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

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

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

(780)	—	=	578	≠	323	\bar{a}	10	≐	81
{	248	}	37	≡	102	≠	71	\tilde{a}	10	\approx	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
∈	143	∋	70	∉	76	∋	69	−	1212		130
⊂	127	⊃	111	⊄	66	⊄	74	×	555		
⊆	100	⊇	96	⊄	59	⊄	79	÷	89		
⊈	74	⊉	75	⊄	68	⊄	61	!	106	∠	—
⊊	76	⊋	76	≠	61	≠	74	?	44	∞	194
<	324	>	351	≠	73	≠	82	:	44	...	44
≤	104	≥	84					;	17	%	11
				⊥	58	⊤	135			∂	413
≪	100	≫	101	∴	17	∴	17	∀	35	∇	11
<	44	>	51	±	116	∓	81	∃	27	∫	412
≠	20	≠	14	↑	58	↓	68	⊙	77	√	397
+	56	+	68	↑	66	↓	73	⊕	71	∏	13
/	142	\	88	∩	153	∪	173	⊗	84	'	62
				∩	65	∪	80	⊗	63	★	23
→	138	←	70	∧	186	∨	462	⊕	91	*	18
⇒	93	⇐	87					⇔	86	†	10
↔	55	↔	41					↔	14	✓	33
↔	77	↔	56								
⇒	65	⇒	54								
↗	69	↖	55								
↘	63	↙	80								

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

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, 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.

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 [6] that this value yields best results for various classification methods).

Our experiments were performed on 5 identical PCs with a 2.4 GHz Intel Dual Core processor, 2GB of RAM, 2MB of cache, running Linux Ubuntu 8.04.1. The algorithms were implemented in C++ and compiled using g++ version 4.2.4, with option -O3. The distance to the convex hull was 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:

1. Elastic matching distance.
2. Euclidean distance on Legendre-Sobolev coefficient vectors.
3. Manhattan distance on Legendre-Sobolev coefficient vectors.
4. Distance to the convex hull of several nearest neighbors.

4.1 Elastic matching

The square elastic matching distance from a point sequence $A = a_1, \dots, a_n$ to a point sequence $B = b_1, \dots, b_n$ (we assume that the sequences have equal lengths) is defined as the minimum over all non-decreasing functions $r : \{1, \dots, n\} \rightarrow \{1, \dots, n\}$ 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 [14]. A *sliding window* version of elastic matching is often used, in which a parameter k , $1 \leq k \leq n$ is introduced, and the values $r(i)$ are restricted to the interval $[i - k, i + k]$, for each $i = 1, \dots, n$. With this restriction, the distance can be computed in $O(nk)$ operations.

It has been established earlier [3] that the elastic matching distance yields best results when the points a_1, \dots, a_n are sampled from the curve so that the arc lengths between the adjacent points are equal. 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 $k = 2$. The error rates for other values of n and k are shown in Table 3. Observe that the optimal value of k is proportional to n and approximately equal to $n/15$. There is no improvement observed for $n > 30$.

$k \setminus n$	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 %.

The algorithm for computing the elastic matching distance with the sliding window, in the special cases of small values of k , admits an efficient implementation with no dynamic memory allocation and completely inlined code. For example, the computation of a single distance for $n = 20$ and $k = 1$ takes 0.25 microseconds on our PC. Assuming

that a mobile device may be up to 50 times slower and that the maximal affordable delay is 100 milliseconds, we obtain that about 8,000 training samples 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 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 $d = 24$, 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 a mobile device (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

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 machine words as follows (a similar scheme applies to 64-bit architectures). For $k = 0, 1, 2, 3$, use bits $8k, \dots, 8k + 6$ 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 between two coefficient vectors (a_0, \dots, a_{d-1}) and (b_0, \dots, b_{d-1}) , given by

$$\sum_{i=0}^{d-1} |a_i - b_i|,$$

can be efficiently computed as follows.

Consider word 0, in which coefficients for $i = 0, 1, 2, 3$ are stored. Assume that a_i and $-b_i$ are stored in a 32-bit word each, as described above, 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 follows:

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

Note that the resulting absolute values again occupy at most 7 bits in each byte. To accumulate their sum, we can first compute

$$w = (w + (w \gg 8)) \& 0x00FF00FF$$

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

$$w = (w + ((w \gg 16)) \& 0xFFFF$$

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 likely to be due to the effects of packing, which ensures that the computation of the Manhattan distance proceeds in machine registers, while the Euclidean distance requires to retrieve the data from memory. The timings suggest that, on a mobile device that is about 50 times slower than our 64-bit machine, we can compute the Manhattan distance from a test sample to more than 80,000 training samples, for dimension 24, in under 100 milliseconds.

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.

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 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.

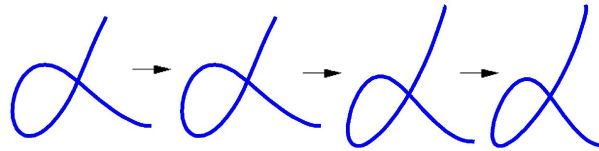
4.4 Convex hull of nearest neighbors

It has been observed earlier [4–6] that classes of handwritten symbol curves satisfy the following property of *convexity*. Consider two curves that belong to the same class. If we

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.

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.



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 [6]. 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.

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 [4, 5]. On the other hand, for classes that do not exhibit any convexity, distance-based nearest neighbor classification may be the only alternative. 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 [15, 6] 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 [13, 6]. It has been shown experimentally [6] that, for the purpose of curve classification, the optimal number of nearest neighbors in the convex hull is indeed smaller than the dimension. It has been also shown [6] 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:

1. Compute the d -dimensional normalized vector of Legendre-Sobolev coefficients of the coordinate functions of the test curve. Store each coefficient as a b -bit integer.
2. Apply nearest neighbor classification to select the top T classes.
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.

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 1 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 [6] 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 to leave among the top T classes more than one intersection of the correct class with a neighboring class.

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, 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 a mobile device, on which storage is precious, we can use Figure 1 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 would 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 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 a mobile device that is 50 times slower than our PC.

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

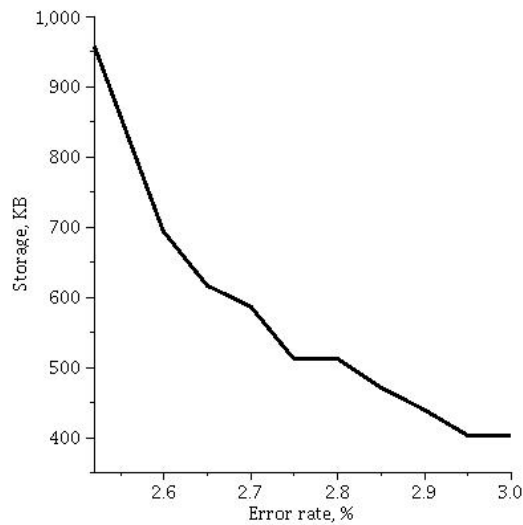


Fig. 1. Minimal storage required to achieve given error rate.

5 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 it turned out 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 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. 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.
3. Golubitsky, O., Watt, S.M.: Online Computation of Similarity between Handwritten Characters. Proc. Docum. Rec and Retrieval (DRR XVI) (2009) C1–C10.
4. Golubitsky, O., Watt, S.M.: Online Recognition of Multi-Stroke Symbols with Orthogonal Series. Submitted to ICDAR (2009).
5. Golubitsky, O., Watt, S.M.: Improved Character Recognition through Subclassing and Runoff Elections. Submitted to ICIAP (2009).
6. Golubitsky, O., Watt, S.M.: Tie Breaking for Curve Multiclassifiers. Submitted to ICCV (2009).
7. Guyon, I., Schomaker, L., Plamondon, R., Liberman, M., Janet, S: UNIPEN Project of On-line Data Exchange and Recognizer Benchmarks. Proc. 12th International Conference on Pattern Recognition (ICPR 1994), Jerusalem, Israel. IAPR-IEEE (1994) 29–33.
8. Ink Markup Language (InkML) W3C Working Draft (23 October 2006) <http://www.w3.org/TR/InkML/>
9. 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.
10. Klee, V.L., Jr.: Convex Sets in Linear Spaces. Duke Math. J. **18** (2) (1951) 443–466.
11. LaViola, J.J., Jr.: Symbol Recognition Dataset. Microsoft Center for Research on Pen-Centric Computing. <http://pen.cs.brown.edu/symbolRecognitionDataset.zip>
12. Li, M., Sethi, I.: Confidence-Based Classifier Design. Pattern Recognition **39** (7) (2006) 1230–1240.
13. 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.
14. 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.
15. 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.
16. Watt, S.M.: Mathematical Document Classification via Symbol Frequency Analysis. Proc. Towards Digital Mathematics Library (DML 2008) 29–40.