ADVANCES IN MANIPULATION AND RECOGNITION OF DIGITAL INK

(Thesis format: Monograph)

by

Vadim Mazalov

Graduate Program in Computer Science

A thesis submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy

The School of Graduate and Postdoctoral Studies
The University of Western Ontario
London, Ontario, Canada

Abstract

Handwriting is one of the most natural ways for a human to record knowledge. Recently, this type of human-computer interaction has received increasing attention due to the rapid evolution of touch-based hardware and software. While hardware support for digital ink reached its maturity, algorithms for recognition of handwriting in certain domains, including mathematics, are lacking robustness. Simultaneously, users may possess several pen-based devices and sharing of training data in adaptive recognition setting can be challenging. In addition, resolution of pen-based devices keeps improving making the ink cumbersome to process and store. This thesis develops several advances for efficient processing, storage and recognition of handwriting, which are applicable to the classification methods based on functional approximation. In particular, we propose improvements to classification of isolated characters and groups of rotated characters, as well as symbols of substantially different size. We then develop an algorithm for adaptive classification of handwritten mathematical characters of a user. The adaptive algorithm can be especially useful in the cloud-based recognition framework, which is described further in the thesis. We investigate whether the training data available in the cloud can be useful to a new writer during the training phase by extracting styles of individuals with similar handwriting and recommending styles to the writer. We also perform factorial analysis of the algorithm for recognition of *n*-grams of rotated characters. Finally, we show a fast method for compression of linear pieces of handwritten strokes and compare it with an enhanced version of the algorithm based on functional approximation of strokes. Experimental results demonstrate validity of the theoretical contributions, which form a solid foundation for the next generation handwriting recognition systems.

Keywords: Handwriting recognition, compression of digital ink, cloud computing

Acknowledgements

I would like to express deep appreciation to my advisor, Dr. Stephen M. Watt, for guidance and support throughout the research. His enthusiasm and welcoming attitude fostered the creativity and productivity needed to complete this thesis.

I thank my colleagues and friends Rui Hu and Paul Vrbik for consistent help they have been offering throughout the years. I also thank all other members of the Ontario Research Centre for Computer Algebra for the environment that encourages contribution and personal growth.

Most importantly, I thank all of my family for so much I was given.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Efficient processing and classification of digital ink becomes especially relevant with the recent popularity of tablet devices and touch-based interfaces. One of the sub-problems in handwriting recognition is handwritten mathematics, which allows two-dimensional input and communication of mathematical knowledge in a more natural way, compared with conventional typesetting systems. Writing mathematics on a digital canvas is similar to traditional pen-on-paper input. It does not require learning any languages and can be efficient, given a robust and reliable implementation. According to a study [4], pen-based input of mathematics is about three times faster and two times less error-prone than standard keyboard- and mouse-driven techniques. The hardware support of digital ink has reached its maturity, while algorithms for recognition of characters and spatial analysis of expressions are still the subject for improvement.

Accuracy of the mathematical equations recognizer is clearly based on the classification rate of individual characters. Although considerable progress has been achieved in the field of handwriting recognition, classification of mathematical symbols requires further improvement. Among the factors that make classification of mathematics especially challenging, compared to normal text recognition, is the relatively large set of similar looking few-stroke symbols that can be subjected to transformations. The absence of a fixed dictionary of multi-symbol "words" makes the syntactic verification of recognized formulas challenging. The two-dimensional nature of mathematical expressions requires an accurate differentiation between fluctuations in positioning and intentional super- or sub-scripting over a baseline.

We make the following contributions to the art of online recognition of characters and test them on the dataset of handwritten mathematical symbols. We improve classification of segmented symbols and develop a method for in-context recognition of samples. We propose a technique for classification of small characters based on the relative size of the samples with respect to other symbols in the collection and demonstrate the superior performance of the

method compared to our previous results [50]. We further note that the recognition error can be decreased if the technique is able to adapt to the handwriting of the user, since each individual has their own writing style. We propose an adaptive algorithm that allows rapid improvement of the classification rate by adjusting the weights of training samples.

Increasing reliability and decreasing cost of cloud technologies open new perspectives for improvement of pattern recognition applications available to public. The training samples and correction history produced by individuals can benefit others, asymptotically decreasing the recognition error. In addition, modern users don't limit themselves with a single device and typically rely on cloud services to synchronize data across different platforms. Handwriting recognition applications can certainly enhance their usability if they take advantage of the opportunities available in the cloud. We present a cloud-based framework for recognition of handwritten characters. Finally, we perform factorial analysis of the algorithm for recognition of rotated n-grams of characters to estimate the influence of the configuration parameters of the algorithm on its performance.

Efficient processing of digital ink is the foundation for all of the discussed algorithms and most other pen-based applications. Modern devices typically collect ink in high resolution, which is important for certain purposes, e.g. authentication of a user based on the signature. In general, however, the high precision makes the ink data cumbersome and costly to process, transmit over networks and store. We investigate two compression schemes that allow one to decrease the volume of data, while losing very little knowledge about the curve. The first method is based on functional approximation of strokes with high-degree orthogonal polynomials, and the second is based on piecewise linear approximation of the strokes. We demonstrate that the size of an ink database can be reduced to a large degree, while preserving the shape of the strokes.

Overall, this thesis addresses several facets of handwriting recognition systems. First, it significantly enhances the existing recognition algorithm by improving its performance and making it more robust for special characters. Then it develops a cloud architecture that allows sharing of the training data and correction history across devices. Then it proposes methods for compression of digital ink to facilitate compact storage and fast transmission. These contributions can be naturally integrated in the cloud environment. Some ideas can also be considered as the basis for cloud-based classification systems in other pattern recognition and machine learning domains, where public knowledge is useful for improving individual performance. The presented results form a valuable asset to developers of frameworks for manipulation and recognition of digital ink.

1.1 Outline of Chapters

The thesis is organized into the following Chapters:

Chapter 1 presents an overview of the problems addressed in the thesis.

Chapter 2 introduces the necessary preliminaries and the previous research.

Chapter 3 improves isolated and in-context classification of handwritten characters.

Chapter 4 explains how to make the character recognition algorithm robust against samples of substantially different size.

Chapter 5 discusses a structure for adaptive handwriting recognition by assigning weights to training samples.

Chapter 6 presents the cloud-based handwriting recognition framework.

Chapter 7 demonstrates factorial analysis of the in-context rotation invariant classification algorithm with respect to the most important parameters.

Chapter 8 draws the algorithm for linear piecewise compression of digital ink and improves the earlier developed method of approximation of strokes with higher degree orthogonal polynomials.

Chapter 9 concludes the thesis and proposes directions for future work.

Chapter 2

Previous Work and Preliminaries

2.1 Introduction to Digital Handwriting

In online handwriting, a curve is given as an ordered set of points in a Euclidean plane. Pen-based devices capture coordinates of a stylus as functions of time. Additionally, some hardware can collect other data – the degree of pressure, pen angle or coordinates of pen-up points, i.e. when a stylus does not touch the screen. However, we disregard this information to remain device-independent.

A curve is given as a sequence of points

$$(x_0, y_0, t_0), (x_1, y_1, t_1), ..., (x_n, y_n, t_n)$$

where $x_i, y_i, t_i \in \mathbb{R}, i = 0..n$, and $t_0 < t_1 < ... < t_n$. Devices typically collect points equally spaced in time and therefore t_i can be omitted. Most often, coordinates are represented as integers, and indeed that is how our experimental dataset is stored [48].

Writing on the canvas would not be that useful without the ability for a machine to understand the handwriting. Researchers have been tackling the problem of handwriting recognition for about half a century [57]. A variety of methods have been proposed, e.g. based on Markov chains and functional approximation. We mostly build on the foundation of the classification algorithm with functional approximation of strokes, described in [26]. However, a brief description of the general idea behind the approach with Markov chains is given below [48].

Markov chain methods typically model behaviour of elements of a handwritten curve, assuming those elements satisfy the Markov property (future states of the process depend only upon the present state). In the early papers, e.g. see [15], a curve is encoded as a sequence of directions selected from the set of s states. The probability distribution of the first piece of the curve is encoded as a vector $M_1 = [P_0, P_1, ...P_s]$, where P_i is the probability that the first stroke

is pointing in the i-th direction, and $\sum_{i=0}^{s} P_i = 1, 0 \leq P_i \leq 1$. Then the stochastic transition matrix at the k-th piece is

$$\begin{pmatrix} P_{00} & P_{01} & \cdots & P_{0s} \\ P_{10} & P_{11} & \cdots & P_{1s} \\ \vdots & \vdots & \ddots & \vdots \\ P_{s0} & P_{s1} & \cdots & P_{ss} \end{pmatrix}$$

where P_{ij} is the probability of transitioning from the direction i at the $k-1$ piece to the direction j at the k-th piece. Therefore a handwritten curve is represented as a collection of n stochastic transition matrices, and each matrix represents a piece of the curve. During the classification phase, a curve R to be classified is split into n pieces with corresponding directions $R_1, ..., R_n$. The probability of the curve to belong to a training class C is computed as

$$p^{(1)}_{R_1} \cdot p^{(2)}_{R_1 R_2} \cdot ... \cdot p^{(n)}_{R_{n-1} R_n}$$

where $p^{(1)}_{R_1}$ is the probability of the first piece of the class C to point in the direction R_1, and $p^{(k)}_{R_{k-1} R_k}$ is the probability of the k-th piece of the class C to point from the direction R_{k-1} to the direction R_k. Then the curve R is classified based on the maximum probability among training classes.

Besides considering the direction of a stroke, sometimes other measures are introduced, e.g. the length of the stroke and direction of the pen up movements [71]. In addition, a context analysis is included by considering substrokes in sets, rather than independently.

2.2 Orthogonal Series

Two functions $f(\lambda)$ and $g(\lambda)$ defined on the domain $[a, b]$ are orthogonal on this interval with respect to a given continuous weight function $w(\lambda)$, if their inner product

$$\langle f, g \rangle \equiv \int_a^b f(\lambda) g(\lambda) w(\lambda) d\lambda = 0.$$

One method to approximate a function $f : R \to R$ is as a linear combination of polynomials up to some degree d from a given basis $P = \{P_i : R \to R, i = 0, 1, ..., d\}$:

$$f(\lambda) \approx \sum_{i=0}^{d} f_i P_i(\lambda), \quad f_i \in R, \ P_i \in P$$

where polynomials $P_i, i = 0, 1, ..., d$ are orthogonal with respect to an inner product $\langle \cdot, \cdot \rangle$. Gram-Schmidt orthogonalization of the monomial basis $\left\{1, \lambda, \lambda^2, ...\right\}$ can be used to generate the system of orthogonal polynomials $\{P_0, P_1, ...\}$ with respect to a given inner product. The coefficients f_i can be found as [38]

$$f_i = \frac{\langle f, P_i \rangle}{\langle P_i, P_i \rangle}.$$

With this method, one is able to obtain representation of coordinate functions as follows:

$$X(\lambda) \approx \sum_{i=0}^{d} x_i P_i(\lambda), \qquad Y(\lambda) \approx \sum_{i=0}^{d} y_i P_i(\lambda).$$

The coordinate functions $X(\lambda)$ and $Y(\lambda)$ may be parameterized in various ways, such as by time or by arc length. Parameterization by arc length is preferable, since it provides independence of speed of writing of curves. A possible problem, though, is that arc length is not invariant under all affine transformations. For transformation-invariant parameterization one could use time, invariant under affine transformations, or special affine arc length, invariant under area preserving transformations [3]:

$$F(L) = \int_0^L \sqrt[3]{x'(\lambda)y''(\lambda) - x''(\lambda)y'(\lambda)}d\lambda.$$

Char and Watt proposed to represent a character as a vector of coefficients of the approximation of the curve coordinates with truncated orthogonal series [9]. They used Chebyshev polynomials of the first kind

$$T_n(\lambda) = \cos(n \arccos \lambda).$$

These are orthogonal on the interval $[-1, 1]$ for $w(\lambda) = 1/\sqrt{1 - \lambda^2}$. Even though Chebyshev polynomials are fast to calculate and allow accurate approximation of a curve with low degree series, the form of its weight function creates difficulties for online computation of approximation. Therefore it was proposed to use Legendre polynomials that allow recovering a function online [23] from its moments [70]. It was described how to compute the first d coefficients of the truncated Legendre series for a function $f(\lambda)$, normalized to a desired range and domain, in online time $OL_n[O(d), O(d^2)]$, where n is the number of known equally-spaced values of f [48].

In later work [24], the authors demonstrated that Legendre-Sobolev polynomials perform better than Legendre polynomials for recognition purposes. The Legendre-Sobolev polynomi-

als are orthogonal with respect to the inner product

$$\langle f, g\rangle = \int_a^b f(\lambda)g(\lambda)d\lambda + \mu \int_a^b f'(\lambda)g'(\lambda)d\lambda.$$

where μ is a parameter, that we call the "jet scale".

Legendre-Sobolev polynomials are suitable for online computation of coefficients and provide a more accurate description of a curve for a lesser degree of approximation than Legendre polynomials (due to the presence of derivatives in the inner product). Classification is based on Euclidean distance between coefficient vectors of subject and training samples. The authors presented results that demonstrate that classification rates with elastic matching and Legendre-Sobolev approximation are similar, while the latter is more efficient [48].

2.3 Classification with Convex Hulls

The technique of recognition via convex hulls represents classes by some fixed number of nearest neighbours and is similar to the recognition with SVMs. However, a subject sample is assigned to the class with a corresponding convex hull located on the smallest distance to the sample. Nearest neighbours are selected with the Manhattan distance, which is among the fastest distances known, requiring $2d - 1$ arithmetic operations, where d is the dimension. Distance to convex hulls is evaluated with the squared Euclidean distance, which takes $3d - 1$ operations [20, 48].

Computing the distance from a point to a convex hull is generally expensive. However, one can represent a convex hull as a simplex if the number of nearest neighbours is less than the dimension of the vector space and the points are in generic position. If the points happen to not be in generic position, a slight perturbation is done with a little affect on the distance [20]. Details of the algorithm are explained in Section 2.4.

Classification of multi-stroke characters can be implemented similarly to the classification of a single-stroke with functional approximation, see [25]. In the case of a multi-stroke sample, consecutive strokes are joined to obtain the function to approximate, and the number of strokes is included in the class label [20].

2.4 Distance to Simplex

The task of computing the distance from a point to a convex hull of k nearest neighbours occurs in various applications of machine learning. This section is based on the poster presented at the East Coast Computer Algebra Day 2012, abstract of which is published in [22].

2.4.1 Problem Definition

The classification algorithm is based on computation of the distance from a point to a simplex in n-dimensional Euclidean space. The distance is computed by the means of recursive projections onto linear subspaces containing lower-dimensional subsimplexes, proceeding until the projection is in the interior. Since any generic set of points of size not exceeding the dimension of the vector space plus one forms a simplex, we view the distance to the convex hull of k nearest neighbors as the distance to a simplex. When the k nearest neighbors are not in generic position one may perform simplicial decomposition of their convex hull.

Several methods have been proposed with various degrees of efficiency that either solve a more general problem, such as finding the distance from a point to a polytope, or a more specialized problem, such as computing the distance to a canonical simplex. The paper [72] relies on the observation that the distance from a point to a convex object is twice the distance to a maximal margin SVM hyperplane. In [78], the minimum distance from a point Y to a *polytope* on vertices $\{P_1, \ldots, P_m\}$ is found using quadratic programming to minimize $X^T X$ for $X = \sum_{k=1}^{m}(P_k - Y)w_k$ and $\sum_{k=1}^{m} w_k = 1$ for all $w_k \geq 0$. The paper [12] proposes an algorithm for computing the distance from a point to a *canonical simplex*. Similarly to the method in [78], the authors construct a function to be minimized, introduce the Moreau's proximity operator to the function and derive several properties that allow them to obtain a succinct algorithm.

Another technique for a *canonical simplex* was developed in [58]. The work presents a recursive algorithm that locates a solution in a strictly lower-dimensional space. The solution is found using Lagrange multipliers and properties of a canonical simplex. The dimension of the problem is decreased by removing the points that have negative corresponding coefficients. If all of the coefficients are non-negative then the solution is found. A similar logic was used to devise a method to find the distance to a more general type of convex objects – *convex polyhedral cones* [76]. The method finds the nearest point P in the cone K to a point Q as a positive linear sum of a subset of vectors from the generating set. The cone K is split in subcones $K_1, K_2, \ldots$. On each iteration the algorithm finds a point $P_j \in K_j$ which is closer to Q than is P_{j-1}. Since there is a finite number of cones, the algorithm terminates at some step.

The method we use is similar to that of Michelot [58], except that it computes the distance to a general simplex. While this could be accomplished by finding a linear transformation mapping the general simplex to the canonical one, and conjugating the Michelot's method, it is simpler to perform the computation directly.

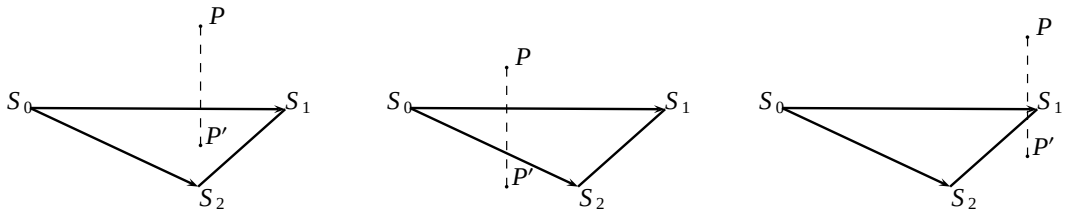


Figure 2.1: Possible projection scenarios

Recursive call	S	α
1	$\{S_0(0,0), S_1(3,0), S_2(2,-1)\}$	$\alpha_1 = 2, \alpha_2 = -1$
2	$\{S_0(0,0), S_1(3,0)\}$	$\alpha_1 = \frac{4}{3}$
3	$\{S_0(3,0)\}$	

Table 2.1: An example of the execution flow

2.4.2 Distance to a Simplex

The method below is based on recursive computation of the projection from the point to the smallest linear subspace that contains the simplex. The projection is expressed as a linear combination of the generating vectors of the simplex. The vectors with corresponding positive coefficients are used as the input for the next recursive call. The algorithm stops when all of the coefficients of a projection are non-negative or when the simplex contains only one vertex.

Let $S_i \in \mathbb{R}^n, i = 0..d, d \geq 0, d \leq n$ be points of a simplex and $P \in \mathbb{R}^n$ is the point from which the distance should be computed, where $\mathbb{R}^n$ is the n-dimensional Euclidean space. We assume that the points of the simplex are in generic position, i.e. the vectors $\mathbf{S_1} - \mathbf{S_0}, \mathbf{S_2} - \mathbf{S_0}, ..., \mathbf{S_d} - \mathbf{S_0}$ are linearly independent. For a detailed description, see Algorithm 1. The complexity of the algorithm is $O(d^4)$, where d is the number of vertices. In practice, however, the algorithm performs much faster, since on each recursive call the dimension drops by more than one.

An example of the execution flow of the algorithm for a point $P(4,1)$ and a simplex $S = \{S_0(0,0), S_1(3,0), S_2(2,-1)\}$ is shown in Table 2.1. During the third recursive call, the algorithm returns the Euclidean distance between $(4,1)$ and $(3,0)$.

2.5 Integral Invariants

To recognize characters invariant with respect to certain transformations, e.g. rotation, we investigated integral invariant functions [16]. These invariants are computed from the coordinate functions, which are then also functions of the curve parameterization. Exposing the sample

Algorithm 1 DistanceToSimplex($P, \{S_0, ..., S_d\}$).

Input: A point P and a simplex with vertices $\{S_0, ..., S_d\}$.
Output: Distance from P to the simplex.

if $d = 0$ **then**
  return Euclidean distance between P and S_0.
end if
Translate so that S_0 is the origin.
Find projection P' of P to the linear subspace with the set of basis vectors $\mathbf{S} = \mathbf{S_1}, ..., \mathbf{S_d}$. The projection can be computed as a solution of the system

$$\sum_{i=1}^{d} \alpha_i \langle \mathbf{S_i}, \mathbf{S_j} \rangle = \langle \mathbf{P}, \mathbf{S_j} \rangle, \; j = 1, 2, ..., d$$

and expressed as $\mathbf{P'} = \sum_{i=1}^{d} \alpha_i \mathbf{S_i}$.
if $\sum_{i=1}^{d} \alpha_i \leq 1$ **and** $\alpha_i \geq 0, \forall i = 1..d$ **then**
  {The projection is inside the simplex, see Figure 2.1(a)}
  return Euclidean distance between P and P'.
else if $\exists i$ such that $\alpha_i < 0$ **then**
  {See Figure 2.1(b)}
  $S' \leftarrow S_0 \cup \{S_i | \alpha_i > 0\}$.
  return DistanceToSimplex(P, S')
else
  {$\sum_{i=1}^{d} \alpha_i > 1$ **and** $\alpha_i \geq 0, \forall i = 1..d$, see Figure 2.1(c)}
  return DistanceToSimplex($P, S \setminus S_0$)
end if

to transformations results in the same invariant functions. As opposed to differential invariants, such integral invariants are relatively insensitive to small perturbations, and are therefore applicable to classification of handwritten characters with sampling noise [20].

As the name suggests, integral invariants are obtained by integration. Out of the infinite family of invariants we studied the first three [16], which we defined in terms of the coordinate functions $X(\lambda)$ and $Y(\lambda)$:

$$I_0(\lambda) = \sqrt{X^2(\lambda) + Y^2(\lambda)} = R(\lambda),$$
$$I_1(\lambda) = \int_0^{\lambda} X(\tau) dY(\tau) - \frac{1}{2} X(\lambda) Y(\lambda),$$
$$I_2(\lambda) = X(\lambda) \int_0^{\lambda} X(\tau) Y(\tau) dY(\tau) - \frac{1}{2} Y(\lambda) \int_0^{\lambda} X^2(\tau) dY(\tau) - \frac{1}{6} X^2(\lambda) Y^2(\lambda).$$

Functions $X(\lambda)$, $Y(\lambda)$ can be of any desired parameterization. The function $I_1(\lambda)$ can be geo-

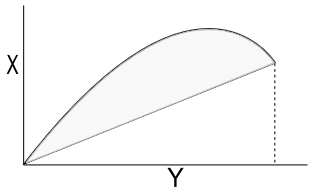


Figure 2.2: Geometric representation of the first order integral invariant

metrically represented as the area between the curve and its secant (Figure 2.2).

Function $I_0(\lambda)$ is independent of transformations of the special orthogonal group $SO(2)$, while $I_1(\lambda)$ and $I_2(\lambda)$ are invariant under the group of special linear transformations, $SL(2)$.

2.6 Approximation of Invariants

The coordinate functions are represented by the truncated sum of Legendre-Sobolev orthogonal series (for details, see Section 2.2). Therefore, we can write the approximation of invariants introduced above as

$$
\begin{aligned}
I_0(\lambda) &\approx \sqrt{\left(\sum_{i=1}^{d} \overline{x}_i P_i(\lambda)\right)^2 + \left(\sum_{i=1}^{d} \overline{y}_i P_i(\lambda)\right)^2} \\
I_1(\lambda) &\approx \sum_{i,j=1}^{d} \overline{x}_i \overline{y}_j \left[\int_0^{\lambda} P_i(\tau) P_j'(\tau) d\tau - \frac{1}{2} P_i(\lambda) P_j(\lambda)\right] \\
I_2(\lambda) &\approx \sum_{i,j,k,l=1}^{d} x_i x_j y_k y_l \mu_{ijkl}
\end{aligned}
$$

where

$$
\begin{aligned}
\mu_{ijkl} = {} & P_i(\lambda) \int_0^{\lambda} P_j(\tau) P_k(\tau) P_l'(\tau) d\tau \\
& - \frac{1}{2} P_l(\lambda) \int_0^{\lambda} P_i(\tau) P_j(\tau) P_k'(\tau) d\tau - \frac{1}{6} P_i(\lambda) P_j(\lambda) P_k(\lambda) P_l(\lambda).
\end{aligned}
$$

Here P_i denotes the i-th Legendre-Sobolev polynomial.

In our algorithms, these functions are, in turn, approximated with the orthogonal series. Therefore, it is reasonable to estimate how well the invariants can be approximated. To evaluate the quality of approximation, we compared coefficients of an original sample and the same sample sheared by 1 radian. The error of the 12-th degree approximation provides sufficient accuracy for our algorithms and such invariants can be successfully deployed for our

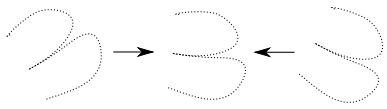


Figure 2.3: Rotation of a symbol

purposes [21, 48].

2.7 Rotation-Invariant Recognition

This section describes the main ideas presented in "Orientation-Independent Recognition of Handwritten Characters with Integral Invariants" co-authored with Watt.

Different solutions have been proposed, usually dealing with *ad hoc* rotation of a character after it is completely written (Figure 2.3). This rotation, as well as symbol resizing, are performed during the normalization stage in most of the online techniques. We proposed a different approach: rather than rotating a sample by some estimated amount, we computed from the sample certain functions that are invariant under rotation. We examined to what extent these transformations affect the classification rate and presented new algorithms for classifying symbols in the presence of such transformations. The following methods were considered: classification with integral invariants (CII) and classification with coordinate functions and integral invariants (CCFII). For these we used the theory of integral invariants of parametric curves [16]. To objectively evaluate recognition rate of these techniques, we compared to a similar algorithm that uses geometric moment functions for the rotation-independent classification. We called this last method classification with coordinate functions and moment invariants (CCFMI). For CII, we take the integral invariants as the curves to be approximated and look for nearest classes in a manner we describe below. For CCFII, the top N classes are selected with integral invariants, then the sample is rotated to determine the angle which gives minimal distance based on coordinate curves. The CCFMI method is similar to CCFII except that it computes geometric moment invariants to obtain top N candidates. These algorithms are online in the sense that most of the computation is performed while the sample is written, with minor overhead after pen-up. The algorithms are as well independent of translation and scaling, which is achieved by dropping the constant terms from the series and by normalizing the coefficient vectors respectively.

2.7.1 Geometric Moments

Similar to integral invariants, moment invariants provide a framework to describe curves independently of orientation. Among moment functions one can select geometric, Zernike, radial

and Legendre moments [60]. For the purpose of online curve classification under pressure of computational constraints, geometric moments are of special interest since they are easy to calculate, while invariant under scaling, translation and rotation.

Having been introduced by Hu [30], geometric moments are widely used for shape and pattern classification [44, 56, 60]. A $(p + q)$-th order moment of f can be expressed as

$$m_{pq} = \sum_x \sum_y x^p y^q f(x, y)$$

In general, translation invariance is achieved by computing central moments

$$\mu_{pq} = \sum_x \sum_y (x - x_0)^p (y - y_0)^q f(x, y),\ x_0 = \frac{m_{10}}{m_{00}} \text{ and } y_0 = \frac{m_{01}}{m_{00}}$$

and scale normalization is performed as

$$\eta_{pq} = \mu_{pq} / (\mu_{00})^{(p+q+2)/2}$$

The first three moment invariants are derived from algebraic invariants and can be represented as

$$M_1 = \eta_{20} + \eta_{02}, \quad M_2 = (\eta_{20} - \eta_{02})^2 + 4\eta_{11}^2, \quad M_3 = \eta_{20}\eta_{02} - \eta_{11}^2.$$

Independence of orientation of the above expressions can be verified by substitution with the geometric moments obtained after rotation transformation

$$\begin{aligned}
m'_{20} &= \frac{1 + \cos 2\alpha}{2} m_{20} - \sin 2\alpha\ m_{11} + \frac{1 - \cos 2\alpha}{2} m_{02}, \\
m'_{11} &= \frac{\sin 2\alpha}{2}\ m_{20} + \cos 2\alpha\ m_{11} - \frac{\sin 2\alpha}{2}\ m_{02}, \\
m'_{02} &= \frac{1 - \cos 2\alpha}{2} m_{20} + \sin 2\alpha\ m_{11} + \frac{1 + \cos 2\alpha}{2} m_{02}.
\end{aligned}$$

One can omit translation and scale normalization of moments by normalizing a sample's coordinates first. In this case the moment invariants are derived in terms of moments m_{pq}.

2.7.2 CII and CCFII

Consider the coordinate functions $X(\lambda)$ and $Y(\lambda)$ of a single- or multi-stroke sample. The first step is to approximate $X(\lambda)$ and $Y(\lambda)$ as truncated series in basis of Legendre-Sobolev polynomials. Let x_0, x_1,...,x_d be the coefficients of the approximation for $X(\lambda)$ and similarly for $Y(\lambda)$. Note, that these coefficients are computed while the curve is written with a small constant time overhead after pen-up [23].

Since the first polynomial (for any inner product) is 1, point (x_0, y_0) can be thought of as the curve's center. We can therefore normalize the curve with respect to position by simply discarding the first coefficients. Scale normalization is performed by normalizing the vector $(x_1, ..., x_d, y_1, ..., y_d)$, taking advantage of the fact that the norm of the vector is proportional to the size of the curve, to obtain $(\bar{x}_1, ..., \bar{x}_d, \bar{y}_1, ..., \bar{y}_d)$.

A similar process of approximation is then applied to the invariant functions, yielding a $2d$-dimensional vector for each sample $(\bar{I}_{0,1}, ..., \bar{I}_{0,d}, \bar{I}_{1,1}, ..., \bar{I}_{1,d})$. Taking the second term in the expression for $I_1(\lambda)$ as precomputed, the Legendre-Sobolev coefficients can be calculated quickly, in time quadratic in d. The coefficients for $I_0(\lambda)$ are computed in the same way.

The CII algorithm relies on approximation of the invariant functions, as described above. We select the class closest to the sample in the space of coefficients of truncated polynomial series. The algorithm does not depend on the number of classes, since only one class is considered.

As an alternative, in CCFII the coefficients $(\bar{I}_{0,0}, \ldots, \bar{I}_{0,d}, \bar{I}_{1,0}, \ldots, \bar{I}_{1,d})$ are used to select the closest N candidate classes. The value for N may be determined empirically to ensure high probability of the correct class being within the ones chosen. Having a fixed small number of classes with the correct class among them, we evaluate the minimal distance from the sample to each class with respect to various sample rotations. This procedure gives correct class as well as the rotation angle. The angle is determined as the solution to the minimization problem

$$\min_{\alpha} \left(\sum_k (X_k - (x_k \cos\alpha + y_k \sin\alpha))^2 + \sum_k (Y_k - (-x_k \sin\alpha + y_k \cos\alpha))^2 \right),$$

where X_k, Y_k are the coefficients of the Legendre-Sobolev approximation of the coordinate functions of the training symbols, and x_k, y_k are the coefficients of the test sample.

2.7.3 CCFMI

The $(p + q)$-th moment functions of a sample's coordinates can be expressed as

$$m_{pq}(\lambda_\ell) = \sum_{i=1}^{\ell} \sum_{j=1}^{\ell} X(\lambda_i)^p Y(\lambda_j)^q f(X(\lambda_i), Y(\lambda_j))$$

where $X(\lambda_i)$ and $Y(\lambda_i)$ are the coordinates X and Y at sample point i. We take the intensity function to be of the form $f(X(\lambda_i), Y(\lambda_j)) = \sqrt{X(\lambda_i)^2 + Y(\lambda_j)^2}$ and work directly with moments, since normalization with respect to size and position is already performed in the algorithm.

Specifically, we tested the following rotation invariants

$$M_0(\lambda) = m_{00}(\lambda),$$
$$M_1(\lambda) = m_{20}(\lambda) + m_{02}(\lambda),$$
$$M_2(\lambda) = (m_{20}(\lambda) - m_{02}(\lambda))^2 + 4m_{11}(\lambda)^2.$$

As in CCFII, CCFMI selects the top N classes with rotation invariant functions. To make a fair comparison, we considered the classification rate for two combinations of moment invariants: $M_0(\lambda)$, $M_1(\lambda)$ and $M_1(\lambda)$, $M_2(\lambda)$. Classification with $M_1(\lambda)$, $M_2(\lambda)$ in general gave 3% higher error rate. We therefore focused on improving the recognition rate of $M_0(\lambda)$ and $M_1(\lambda)$ by variation of number of classes and number of nearest neighbours.

Our tests showed that CII gives a 88% recognition rate. This recognition rate does not depend on the angle to which test samples are rotated. Neither does the frequency of occurrence of the correct class in the top N classes depend on rotation angle.

It was found that CCFII has a better error rate than CCFMI. CCFII also requires fewer number of candidate classes and fewer nearest neighbour computations. Both methods, however, show better performance than CII, see details in [20].

As expected, we noticed an increase in error rate with the rotation angle for CCFII and CCFMI. The typical misclassifications that arise are when symbols have similar shape and are normally distinguished by their orientation, for example "1" and "/", "+" and "×", "U" and "⊂". As a possible solution to this, a system could consider the tendency to write characters in similar orientations and restrict the range of angles for nearby symbols. This is investigated in Chapter 3.

2.8 Shear-Invariant Recognition

We addressed another class of transformations that often occur in practice: shear, or "skew", transformations. This may be seen as a theoretically sound form of "de-slanting". Samples that have been sheared seem to be quite common in handwriting, compared with other transformations. Also, the maximal shear angle, for which a character is still readable by a human can be quite large (Figure 2.4), compared to the corresponding maximal rotation angle. We therefore expect that, in practice, a large amount of shear can occur and consider shear invariance as a useful addition to the set of tools for character recognition [20]. This section is based on the paper "Toward Affine Recognition of Handwritten Mathematical Characters" [21] co-authored with Golubitsky and Watt.

Shear is harder to deal with than rotation. Since shear does not preserve the length of

strokes, parameterization by the Euclidean arc length is no longer robust. Size normalization requires special attention as well. We developed an algorithm, invariant with respect to shear, rotation, scale and translation, and then proposed a way to extend the invariance of the method to the full affine group, while keeping the recognition rate higher than that of classification with the affine integral invariants alone [20].

2.8.1 Overview of Affine Methods

In this section we briefly describe some of the existing methods invariant under the group of affine transformations and the differences with our approach.

Stroke-Based Affine Transformation This approach was proposed in [3] to minimize distortions in handwriting by applying stroke-based affine transformation. The algorithm denotes stroke-wise uniform affine transformation for a stroke i with A_i and b_i, where A_i is a 2×2 matrix for shear, rotation and scale and b_i is a 2-dimensional translation vector. For a sample, a set of N strokes is selected to construct the objective function in the form of least-squares data fitting to determine the optimal A_i and b_i as

$$F_i = \sum_k \|A_i t_k + b_i - r_k\|^2 \rightarrow \min \text{ for } A_i, b_i, (1 \leq i \leq N)$$

where t_k and r_k are k-th feature points of the sample to be classified and a reference sample respectively, and $\|\cdot\|$ is the Euclidean norm. This yields the affine transformation with the least distance between corresponding strokes of the input and a reference sample. This procedure is repeated for each reference sample and then distance-based classification takes place. Recognition of handwritten characters as gray scale images was proposed in [73], using similar ideas.

Minimax Classification with HMM An online method, robust against affine distortions, was developed in [34], based on continuous-density hidden Markov models (CDHMM). Let N be the number of character classes C_i, $i = 1, ..., N$, each containing M_i CDHMMs

$$\left\{\lambda_i^{(m)}, m = 1, ..., M_i\right\}.$$

In the non-affine approach, an input sample I is classified as member of class C_i in terms of the joint likelihood of the observation I and the associated hidden state sequence S given CDHMM $\lambda_j^{(m)}$, $p(I, S|\lambda_j^{(m)})$, as follows

$$\arg\max_j \left\{\max_m \left[\max_S \log p(I, S|\lambda_j^{(m)})\right]\right\}.$$

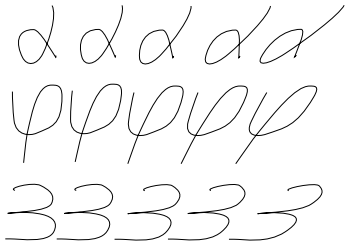


Figure 2.4: Skews of 0.0, 0.2, 0.4, 0.6 and 0.8 radians.

In order to eliminate affine distortions between the input and training samples, the authors use

$$\arg\max_j \left\{ \max_m \left[\max_S \log p(I, S|\Gamma_{\hat{A}}(\lambda_j^{(m)})) \right] \right\},$$

where Γ_A is a specific transformation of $\lambda_j^{(m)}$ with parameters A, and $\hat{A} = \arg\max_A p(I, \Gamma_A(\lambda_j^{(m)}))$. The authors propose solving this optimization problem with three iterations of the EM algorithm described in [43].

Affine Moment Invariants Affine moment invariants (AMIs) are independent of actions of the general affine group and can be used in recognition of handwritten characters [18]. A central moment of order $p + q$ for a 2-dimensional object O can be represented as

$$\mu_{pq} = \iint_O (x - x_c)^p (y - y_c)^q \mathrm{d}x\mathrm{d}y$$

where (x_c, y_c) is the center of gravity of the object O. In the work, the first four affine moments were calculated to obtain a description of an isolated character in the form of a 4-dimensional vector. Samples were classified by the minimum Euclidean distance to the training samples. The performance of AMIs is compared with that of the geometric moment invariants, which are invariant under rotation, scale and translation. It was concluded that AMIs gave a better recognition rate than geometric moments.

As opposed to the first two methods described above, we proposed a technique of classifying handwritten characters with integral invariants. Similarly to the classification with AMIs, affine-invariant quantities are computed from the original curve, without using any specific transformations of the input sample. However, AMIs, as originally defined, provide curve-to-value correspondence, unlike the curve-to-curve correspondence with the integral invariants. This allows to obtain a richer description of a curve without excessive computation. In fact, we considered only 2 invariants and found them sufficient for an acceptable classification accuracy of samples under different transformations. In order to further improve the classification rate, an analysis is still performed on a sample to obtain a numerical measure of the distortion (i.e.

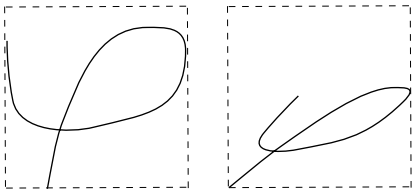


Figure 2.5: Aspect ratio size normalization.

the angle of rotation or shear). However, only small subset of top classes is the subject of this analysis, and it is computationally inexpensive.

2.8.2 A Shear-Invariant Algorithm

In this section we discuss an algorithm, invariant under shear, in addition to rotation, scale and translation [20]. We consider different size normalization methods and parameterizations of the coordinate functions to ensure appropriate setting for the method. The algorithm itself is given at the end of the section.

Size Normalization Size normalization is traditionally implemented by rescaling a sample to achieve standard values of certain parameters. Earlier, this parameter was the Euclidean norm of the vector of Legendre-Sobolev coefficients of the coordinate functions [26]. While this norm can still be used to rescale rotated samples [20], it is not invariant under shear and affine transformations in general. Instead, we look at the norm of the Legendre-Sobolev coefficient vector of I_1. We can then normalize the coefficient vectors of the coordinate functions by multiplying them by $1/\sqrt{\|I_1\|}$. Finally, we compute the coefficients of I_2 from the normalized coefficients of the coordinate functions. Computing the norm of I_1 allows us to extend the invariance of I_1 and I_2 from the special linear group, $SL(2,R)$, to the general linear group, $GL(2,R)$. Invariance under the general affine group, $\mathrm{Aff}(2,R)$, is obtained by dropping the first (order-0) coefficients from the coefficient vectors of the coordinate functions [26].

To evaluate the performance of $\|I_1\|$ for normalization, we consider two other normalization approaches typical in handwriting recognition: height and aspect ratio [44]. Both of these are not perfect in the presence of affine transformations. While normalization by height is invariant under horizontal shear, it becomes inaccurate if samples are subjected to rotation. Aspect ratio is suitable for rotation, but becomes inaccurate for larger degrees of shear (Figure 2.5).

Parameterization of the Coordinate Functions Parameterization by time and arc length are among the most popular choices in online handwriting recognition. Parameterization by arc length is usually preferable, since it is not affected by variations in writing speed and is

invariant under Euclidean transformation. It may be expressed as

$$AL(\lambda) = \int_0^\lambda \sqrt{(X'(\tau))^2 + (Y'(\tau))^2}\mathrm{d}\tau.$$

When one looks at the group of affine transformations, however, parameterization by arc length may no longer be the best choice, since it is no longer invariant. For example, it is changed by shear distortion. Instead, we may consider parameterization by special affine arc length. We use affine arc length in the form

$$AAL(\lambda) = \int_0^\lambda \sqrt[3]{|X'(\tau)Y''(\tau) - X''(\tau)Y'(\tau)|}\mathrm{d}\tau.$$

The Algorithm In online classification algorithms, a symbol is given as a continuous curve defined by a discrete sequence of points. When a symbol is given by multiple strokes, they are joined. The curve is parameterized with an appropriate function (Section 4.2) and the Legendre-Sobolev coefficients of the coordinate functions are computed online, as points are accumulated [23]. Using the representation of $I_1(\lambda)$ in Section 2.7.2, coefficients of the invariant are computed as

$$I_{1,i} = \langle I_1, P_i\rangle \,/\, \langle P_i, P_i\rangle, \quad i = 1..d.$$

Here $\langle P_i, P_i\rangle$ is the Legendre-Sobolev inner product. Similarly, we calculate coefficients for $I_2(\lambda)$ and obtain a $2d$-dimensional vector for each sample

$$(I_{1,1}, ..., I_{1,d}, I_{2,1}, ..., I_{2,d}).$$

Taking the second term in the expression for I_1 as precomputed, each coefficient of the approximation can be computed in time quadratic in d. Each coefficient of I_2 is computed in $O(d^4)$ operations. Note that one can also compute invariants of higher degree [16]. We expect, however, higher degree invariants to affect the classification rate only slightly, while introducing a noticeable computational overhead. For example, it would take $O(d^7)$ operations to calculate the coefficients of I_3 [16].

Given the representation of a character in terms of Legendre-Sobolev coefficients of the invariant functions, we classify the sample based on the distance to the convex hulls of nearest neighbours in the same representation.

We select N classes closest to the Legendre-Sobolev coefficient vector of the integral invariants. To find the correct class among these, we solve the following minimization problem for each of these classes C_i:

$$\min_\phi \mathrm{CHNN}_k(X(\phi), C_i),$$

where $X(\phi)$ is the sheared image of the test sample curve X and $\mathrm{CHNN}_k(X, C)$ is the distance from a point X (in the Legendre-Sobolev space) to the convex hull of k nearest neighbors in class C.

It is not infeasible to solve the minimization problem by trying all possible angles, given that the precision of 1 degree is certainly sufficient for our purposes. Our error rates were calculated using this method. However, there are also more efficient methods. If we replaced the class C by a single point $(X_0, \ldots, X_d, Y_0, \ldots, Y_d)$ in the Legendre-Sobolev space of the coordinate functions, then we could find the minimum among the values of the distance at the boundary points of the interval of shear (i.e. the smallest and the largest admissible shear angles) and the stationary point

$$\varphi = \arctan \frac{\sum_k (X_k - x_k)}{\sum_k y_k}.$$

Experimental results demonstrated that the approximation error of integral invariants is negligible: the absolute error for the 12-degree approximation is of the order of 10^{-5}.

The error rate of the algorithm was examined for different choices of parameterization of the coordinate functions: arc length, time and affine arc length. We also compared size normalization techniques described above.

Parameterization by arc length, which is not invariant under shear, gives a lower recognition rate than parameterization by time for large distortions. However, for shear up to about 0.45 radians ($\approx$25 degrees) it yields a noticeably better classification rate. Parameterization of the coordinate functions by affine arc length results in relatively low recognition rate. Presence of second order derivatives makes it sensitive to sampling perturbations, even though it is invariant under special affine transformation, see details in [21].

We have discovered that size normalization by height gives the best classification rate under shear transformation. Such normalization, however, is not suitable for some other affine transformations, e.g. rotation. Normalization with aspect ratio performs similarly to normalization by height at smaller degrees of shear, but the difference in classification rates becomes noticeable with the increase of deformation. Normalization by I_1 performs just as well as normalization by height and remains invariant under affine transformations. We therefore consider size normalization with I_1 as the most suitable approach, if transformation of characters takes place.

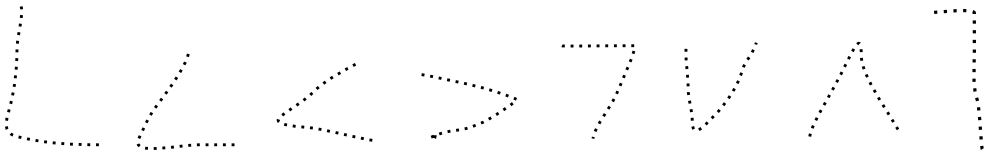


Figure 2.6: Ambiguity introduced by shear and rotation

2.9 Digital Ink Compression via Functional Approximation

We studied whether it is feasible to apply the theory of functional approximation to describe a stroke up to some given threshold of the maximal pointwise error and root mean square error. If so, what is the compression one could expect as the result of such approximation? This section is based on the paper "Digital Ink Compression via Functional Approximation" [50] co-authored with Stephen M. Watt, that appeared in the proceedings of the 12th International Conference on Frontiers in Handwriting Recognition.

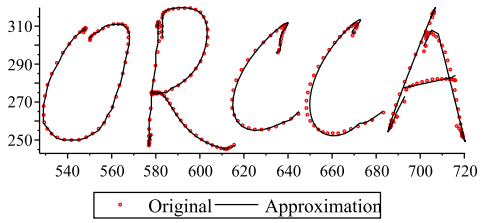
We empirically investigated different approaches to obtain the minimal overall size of coefficients of an approximation that satisfies the given error constraints. We considered compression of handwritten regular text, since it commonly occurs in pen-based computing and incorporates different kinds of patterns. An example word and its approximation with different thresholds are shown in Table 2.2 and the corresponding figure. We have observed that limiting the maximum error also limits the root mean square error, but not vice versa. Therefore, in our experiments we have limited the maximum error.

2.9.1 Ink Representation

A variety of digital ink standards are in use today. Among these one can mention vendor-specific or special-purpose formats: Jot [66], Unipen [27], Ink Serialized Format (ISF) [59] or Scalable Vector Graphics (SVG) [17]. In 2003, W3C introduced a first public draft of an XML-based markup language for digital trace description, InkML. This has evolved to the current standard definition in 2010 [11]. InkML has received an increasing attention due to its vendor neutrality and XML base. In general, a trace is given in InkML as a sequence of multidimensional points. Each coordinate gives the value of a particular channel at that point. Pre-defined channels include x and y coordinates, pressure and various angles.

2.9.2 Bases for Approximation

We wish to determine which bases will be suitable for compression. We have investigated three families of orthogonal polynomials with useful properties and have included Fourier series for



	O	R	C	C	A
Max. pt-wise err., %	1	2	3	4	5
RMSE, %	0.33	0.67	1	1.33	1.67

Table 2.2: Different approximation thresholds.

comparison. *Chebyshev polynomials* of the first kind, defined as $T_n(\lambda) = \cos(n \arccos \lambda)$, have weight function $w(\lambda) = \frac{1}{\sqrt{1-\lambda^2}}$ and are used in numerical approximation for their property of minimizing the maximum error. In [9] it was reported that Chebyshev polynomials are suitable for succinct approximation of strokes and perform better than Bernstein polynomials. *Legendre polynomials* are defined as

$$P_n(t) = \frac{1}{2^n n!} \frac{d^n}{dt^n}(t^2 - 1)^n$$

and have weight function $w(\lambda) = 1$. *Legendre-Sobolev polynomials* are constructed by applying the Gram-Schmidt orthogonalization to the monomial basis $\{\lambda^i\}$ using the inner product

$$\langle f, g \rangle = \int_a^b f(\lambda)g(\lambda)d\lambda + \mu \int_a^b f'(\lambda)g'(\lambda)d\lambda$$

where $\mu = 1/8$ as described in [26].

A property of the Legendre and Legendre-Sobolev orthogonal bases, as applied to online stroke modeling, is the ability to recover a curve from moments computed in real time, while the stroke is being written. The coefficients of the stroke may then be calculated on pen-up in constant time depending only on the degree of approximation [23]. *Fourier series* on $[-L, L]$ are provided for comparison, since we are not restricted in our selection of approximation basis.

$$f(x) \approx \frac{\alpha_0}{2} + \sum_{n=1}^{d} (\alpha_n \cos(\frac{n\pi x}{L}) + \beta_n \sin(\frac{n\pi x}{L}))$$

where

$$\begin{bmatrix} \alpha_n \\ \beta_n \end{bmatrix} = \frac{1}{2L} \int_{-L}^{L} f(x) \begin{bmatrix} \cos \\ \sin \end{bmatrix} (\frac{n\pi x}{L}) dx.$$

2.9.3 Algorithms

Overview At a high level, our compression method takes the following steps for each stroke:

1. Segment the stroke using one of the methods described below. Ensure the segments overlap by an amount at segmentation points.
2. For each segment, compute the orthogonal series coefficients for each coordinate function (*e.g.* x, y, p, where p is pressure).
3. Compress the stream of coefficients.

To reconstruct a stroke, the process is reversed:

1. Decompress the coefficient stream to obtain the curve segments.
2. Blend the curves on the overlaps to obtain the piecewise coordinate functions.
3. Obtain traces by evaluating the coordinate functions with the desired sample frequency.

On a given segment, the series coefficients are computed by numerical integration of the required inner products. The cost to compute the compression is linear in the number of trace sample points and in the number of coefficient size/approximation degree combinations allowed.

To obtain a more compact form for the coefficient stream, it may be compressed with a deflation tool. In the experiments below we use gzip, which implements a combination of LZ77 [79] and Huffman coding [33].This is for convenience only — a more specialized method would be used in a production setting.

Parameterization Choice We tested two used choices for curve parameterization widely used in pen-based computing: time and arc length. We observed that parameterization by time, while being easier to compute, also gives better compression. Comparison of the results is presented in [52] for approximation with Chebyshev polynomials with integer coefficients.

Segmentation We cannot expect long, complex strokes to be well approximated by low degree polynomials. Instead of varying the degree to suit any stroke, we segment strokes into parts that we can separately approximate. We have explored the three methods to segment traces, described here.

Fixed Degree Segmentation We fix the degree of the approximating functions. Intervals of approximation are constructed to allow the maximal length within the given error threshold. If the available interval can be approximated with a lower degree (*e.g.* the end of the curve has been reached), it is handled appropriately.

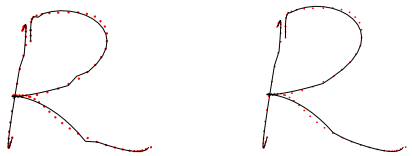


Figure 2.7: Example of blending.

Fixed Length Segmentation We fix the length of intervals and approximate each interval with the minimal degree possible, but not greater than 20 (to keep the algorithm computationally feasible).

Adaptive Segmentation The most comprehensive variant is to fix a maximum permissible degree and maximum permissible coefficient size (digits for text, bits for binary), and to perform fixed degree segmentation for each combination. Then the combination of degree and coefficient size that gives the smallest resulting total size is selected. The degree and coefficient size are saved together with the coefficient data.

Segment Blending If we allow a large error threshold (*e.g.* 4%), then it becomes possible to notice naïve segmentation because we do not match derivatives at the segmentation points. This can be observed in the Table 2.2. To make the stroke smooth, and to improve the approximation, we blend the transition from one piece to another by overlapping the segments slightly and transitioning linearly from one segment to the next on the overlap. Therefore, the approximation is given in segments, f_j, and takes form

$$f(\lambda) = \sum_{j=1}^{N} W_j(\lambda) f_j(\lambda) \approx \sum_{j=1}^{N} W_j(\lambda) \sum_{i=0}^{d} c_{ij} P_i(\lambda)$$

with the weight function

$$W_j(\lambda) = \begin{cases} 0, & \lambda \leq \lambda_j - a \\ \frac{\lambda - (\lambda_j - a)}{a}, & \lambda_j - a < \lambda \leq \lambda_j \\ 1, & \lambda_j < \lambda \leq \lambda_{j+1} - a \\ \frac{-\lambda + \lambda_{j+1}}{a}, & \lambda_{j+1} - a < \lambda \leq \lambda_{j+1} \\ 0, & \lambda > \lambda_{j+1} \end{cases}$$

where a is a proportion of approximation pieces and λ_j are the segment transition points. The value of a may be estimated empirically, but different types of curves will have a certain portion of overlap necessary for smooth transition. An example of a blended sample is given in the Figure 2.7.

2.10 Experimental Dataset

The described dataset of handwritten mathematical characters is used in the experiments throughout the thesis. The dataset currently comprises 50,703 samples from 242 classes. These samples have been collected from several sources: 26,139 characters were gathered at the Ontario Research Center for Computer Algebra (special mathematical characters, Latin letters and digits), 9,762 samples (digits, Latin letters and mathematical symbols) from the LaViola database [42], and 14,802 samples (mostly digits) from UNIPEN [27] handwriting database [21, 48].

All the samples are stored in a single file in InkML format. The number of strokes is included in the class labels. Thus, if a character, such as "7" is written with different number of strokes, it will be placed in different classes, even if the shape of the character is identical. Although this raises the total number of classes to 378, it was found [25] to give better recognition rates compared to when the number of strokes is included in the feature vector [21, 48].

To avoid confusion, all gathered characters had been visually inspected to discard symbols unrecognizable by a human. Symbols that look ambiguous to a human reader (those that may belong to more than one class) were labelled with all the corresponding classes. Classes that appear indistinguishable without context analysis were merged, such as x and $\times$; o, 0 and O. If there was at least one sample in the class that could be recognized by a human with confidence, we retained the label of the class. As a result, we obtained 38,493 samples assigned to single classes, 10,224 to 2 classes, 1,954 to 3 classes, 19 to 4 classes, and 13 samples to 5 classes. To increase the precision of the approximation of integral invariants, we precomputed some terms in the formulas for I_1 and I_2 in Maple [35] using rational arithmetic [21]. Additional details of the experimental setting are given in [26, 48].

The tests are implemented in 10-fold cross-validation. To conduct this process, symbols were split randomly in 10 parts, preserving the proportional sizes of the sets. The normalized Legendre-Sobolev coefficients of coordinate functions, integral invariants and moment invariants were precomputed for all symbols and stored in separate files [21].

Chapter 3

Improving Isolated and In-Context Classification of Handwritten Characters

It was shown in the previous chapter how to recognize handwritten characters by representing coordinate functions or integral invariants as truncated orthogonal series. The series basis functions are orthogonal polynomials defined by a Legendre-Sobolev inner product. The free parameter in the inner product, the "jet scale", has an impact on recognition both using coordinate functions and integral invariants.

In this chapter we develop methods of improving series-based recognition. For isolated classification, the first consideration is to identify optimal values for the jet scale in different settings. For the coordinate functions, we find the optimum to be in a small interval with the precise value not strongly correlated to the geometric complexity of the character. For integral invariants, used in orientation-independent recognition, we find the optimal value of the jet scale for each invariant. Furthermore, we examine the optimal degree for the truncated series. For in-context classification, we develop a rotation-invariant algorithm that takes advantage of sequences of samples that are subject to similar distortion. The algorithm yields significant improvement over orientation-independent isolated recognition and can be extended to shear and, more generally, affine transformations. This chapter is based on the paper "Improving isolated and in-context classification of handwritten characters" [51] co-authored with Stephen M. Watt, that appeared in proceedings of the 19th Conference on Document Recognition and Retrieval.

3.1 Introduction

It was proposed earlier [9] to represent an ink sample as a parameterized curve and to approximate the coordinate functions by truncated orthogonal polynomial series. Later, the Legendre-Sobolev (LS) basis was found to perform better, yielding 97.5% recognition rate [26] with a dataset of samples, most of which were collected as isolated symbols. Although the samples do exhibit certain amount of rotation and shear, it is expected that symbols written in a natural environment are more likely to be distorted in this way. To address the issue, we developed integral invariant methods for rotation- and shear-invariant classification [20, 21].

Our current goal is to improve the already good recognition rates obtained with these orthogonal basis methods. We do this by optimizing the choice of basis functions in two ways: We optimize the free parameter in the inner product definition in each of several settings, and we also optimize the series truncation order. Additionally, for orientation-independent recognition, we show that considering sequences of nearby characters avoids orientation ambiguities to a large extent.

We find optimal values for use with the coordinate functions and with integral invariants. We minimize classification error by investigating the role of the jet scale μ in description of coordinate and invariant functions. We also study whether there exists a dependence between *complexity* of a character and the optimal μ in its recognition. These optimizations are directly applied in the proposed algorithm for distortion-invariant classification, taking advantage of the natural property of human handwriting – writing characters with similar transformation. Experiments are performed for the case of rotation, and a similar setting can be used for shear- and, more generally, affine-independent recognition.

Some work has been done in context-dependent recognition of handwriting, mostly relying on statistical approaches. The context-aware classification of a symbol is often represented in some sort of a joint distribution function of the character and its neighbours. For example, some authors propose [71] to consider substrokes in sets, rather than independently, and encode them in HMMs. To keep the model computationally feasible, a hidden Markov network is used to share states of different HMMs. A similar approach is taken elsewhere [7], where the authors build trigraph models and share certain parameters between those trigraphs. Context can also be useful when dealing with ambiguous segmentation of handwritten words [74], where the classification task is represented as an optimization problem in a Bayesian framework by explicitly conditioning on the spatial configuration of the characters. As cited above, the context is typically taken into account for cursive words recognition. We find context useful in a different setting – classification of well-segmented symbols, subjected to certain distortion.

As it was discussed in Chapter 2, the main idea of the classification methods we consider is

to represent the coordinate functions in terms of an orthogonal basis and to use distance-based classification in the coefficient space [26]. The Legendre-Sobolev inner product used contains one free parameter, μ, which may be assigned any non-negative value. Because this parameter determines the relative weight of the coordinates and their derivatives (i.e. the weights in the jet space), we call μ the *jet scale*. In earlier work, $\mu = 1/8$ was taken as a suitable value.

The rotation-independent [20] and shear-invariant [21] algorithms compute special functions from coordinates. These functions are invariant to certain transformations and therefore describe curves in terms of values that remain relatively constant, even when samples are rotated or sheared on large angles, see Section 2.5.

The chapter is organized as follows: Section 3.2 describes the concepts and experimental methods for improving isolated character recognition via the coordinate functions. Section 3.3 presents a recognition approach for in-context classification of distorted characters. Experimental results are given in Section 3.4. Section 3.5 concludes the chapter.

3.2 Improving Isolated Symbol Classification

It is easily seen that the jet scale parameter, μ, in the LS inner product has an impact on recognition rate. We would like to understand this dependency better in order to optimize this parameter. For each value of μ considered in the experiments, LS polynomials are generated, orthogonal with respect to the corresponding inner product.

For these experiments we use a dataset of about 50,000 isolated handwritten mathematical symbols, identical to that described earlier 2.10.

Coordinate Functions To optimize μ for coordinate functions, we consider recognition of the original samples in our dataset without additional distortion. The coordinate functions of samples are approximated with LS polynomials for different μ. We test values of μ in the range from 0 to 0.10 with the step of 0.002 and from 0.10 to 0.20 with the step of $1/64$. Values outside this range give substantially worse results. Samples are classified with the distance to the CHNN_k in the space of the coefficients of coordinate functions [26].

Integral Invariants To study the impact of μ on integral invariants, we consider characters with unknown orientation. The whole collection of original samples is rotated by an angle α between $\pi/9$ and 2π. All multiples of $\pi/9$ are tested. For each angle, I_0 and I_1 are computed for the original and transformed samples. The invariants are then approximated with LS series for different values of $\mu \in (0, 0.2]$ with the step of 0.002. For each value of μ, the average

maximum approximation error with respect to angle is found as

$$\omega = \frac{1}{n}\sum_{k=1}^{n}\max_{ij}(c_{ij} - c_{ij}^{\frac{k\pi}{9}}) \tag{3.1}$$

where c_{ij} is the j-th coefficient of the i-th original sample, and $c_{ij}^{\frac{k\pi}{9}}$ is the corresponding coefficient of the sample, rotated on angle $\frac{k\pi}{9}$.

Complexity of Handwritten Characters We consider the possibility that the optimal value of μ may depend on the nature of the characters to be recognized. To understand this, we take the notion of a sample's complexity as

$$\eta = \sum_{i=1}^{d}(X_i^{1/i} + Y_i^{1/i}),$$

where X_i and Y_i are normalized coefficients of approximation of the sample with orthogonal polynomials. Coefficients of higher degree are typically greater for "complex" characters – characters that contain large number of loops and/or amount of curvature.

Degree of Approximation The degree of the truncated series, d, regulates how well curves are approximated. In general, higher degree polynomials provide lower error. Sometimes, however, higher order approximation of equidistant nodes may cause extreme oscillation at the edges of an interval (Runge's phenomenon). To find the optimal degree, we evaluate the recognition error, the maximum absolute and the average relative approximation error depending on d. The approximation errors are computed similar to the way, as shown in subsection 3.2, but instead of coefficients we compare original and approximated coordinates of samples.

3.3 Improving In-Context Invariant Classification

Context-Dependent Recognition There are two main approaches to recognition of handwritten mathematics: symbol-at-a-time and formula-at-a-time. Even though comprehensive semantic and syntactic verification of math is quite challenging, studies suggest that context can play an important role in accurate classification and grammatical information can be an asset [68]. Moreover, it has been shown [75] that n-grams provide useful information in a mathematical setting. These facts suggest that contextual information should be taken into account, especially considering large number of similar-shaped symbols that appear ambiguous on their own.

(a) (b) (c) (d) (e)

Figure 3.1: Distorted characters: (a) division vs. (b) modulus; (c) angle bracket vs. (d) angle vs. (e) less than

Figure 3.2: Characters from the training dataset

Figure 3.1 shows typical challenges that arise in classification of individual samples, but which are resolved by considering context.

Algorithm To improve classification of transformed characters, we propose to recognize a set of n samples at a time with the assumption that the characters in the set are transformed by approximately the same degree, see Algorithm 8. This assumption is justified, since samples written by a person are subject to similar distortions. Moreover, we find that symbols in our dataset exhibit various degrees of rotation and shear. Such initial transformations incorporate noise to the model and reflect real-life handwriting. We consider the case of rotation. Shear may be handled similarly. The algorithm is applied to a sequence of n characters rotated on a random angle $\gamma \in [-\beta, \beta]$. The value of ϵ_α, given in (3.2), can be interpreted as the error likelihood in recognition of the sequence distorted by an angle α. The value is derived from the observation: while the distance to the closest class is decreasing and sum of the distances to the closest p classes is increasing, the possibility of a recognition error is declining. Therefore, in the last step the algorithm finds and returns the angle γ of transformation that yields the least error likelihood of the whole sequence.

Complexity Analysis As has been shown [23], the coefficients of a d-dimensional approximation can be computed in online time $OL_n[O(d), O(d^2)]$, where $O(d)$ is the time complexity as each new point is observed and $O(d^2)$ is the cost at pen up. Sample normalization is performed in linear time. It was shown [21] how to compute each coefficient of approximation of I_1 in $O(d^2)$. Distance from a point to a CHNN_k is theoretically computed in $O(d^4)$. It performs much faster in practice, however, because at each recursive call the dimension often drops by more than one [20].

Algorithm 2 In-context rotation-invariant recognition

Input: A set of n rotated test samples and an angle β of the maximum possible rotation of the samples.

Output: A set of n recognized samples and the angle γ of rotation of the samples.

for $i = 1$ to n **do**

Approximate coordinate functions $X_i(\lambda)$ and $Y_i(\lambda)$, parameterized by arc length, with LS polynomials up to degree d

$$c^i_{xy} = (X_{i0}, ...X_{id}; Y_{i0}, ...Y_{id}).$$

Normalize the sample with respect to position by ignoring the 0-order coefficients X_{i0} and Y_{i0}, and with respect to size by dividing each coefficient by the norm $\sqrt{\sum_{j=1}^{d}(X_{ij}^2 + Y_{ij}^2)}$. Approximate I_0 and I_1 with LS polynomials, yielding

$$c^i_{II} = (I^0_{i0}, ...I^0_{id}; I^1_{i0}, ...I^1_{id}).$$

With Euclidean distance between vector c^i_{II} of the test sample and analogous vectors of training characters: Find T closest CHNN$_k$. These T classes serve as candidates for the i-th sample in the sequence.

end for

for $\alpha = -\beta$ to β by step of 1 degree **do**

Compute

$$\epsilon_\alpha = \prod_{i=1}^{n} \frac{D^1_{i\alpha}}{\sum_{j=1}^{p} D^j_{i\alpha}} \tag{3.2}$$

where $D^j_{i\alpha}$ is the Euclidean distance to the j-th *closest* CHNN$_k$ among the candidate classes T for the sample i in the sequence, rotated by angle α, and p is a parameter to be evaluated. Distance D is computed in the space of coefficients of LS polynomials of coordinate functions.

end for

Find $\epsilon_\gamma = \min_{-\beta \le \alpha \le \beta} \epsilon_\alpha$

return n and γ.

Experimental Setting The experimental setting is described in Section 2.10. The model is trained with non-transformed samples. For the recognition phase, sequences of n characters are taken from the dataset and each sequence is rotated by a random angle $\gamma \in [-\beta, \beta]$.

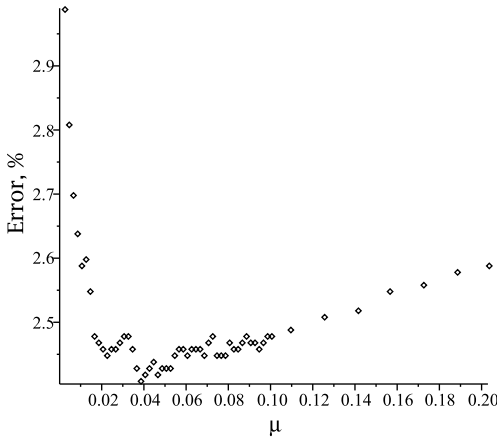


Figure 3.3: Recognition error of non-transformed characters for different values of μ

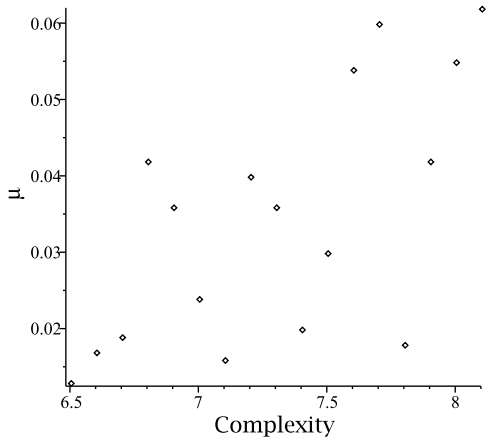


Figure 3.4: The optimal values of μ for samples with different complexity

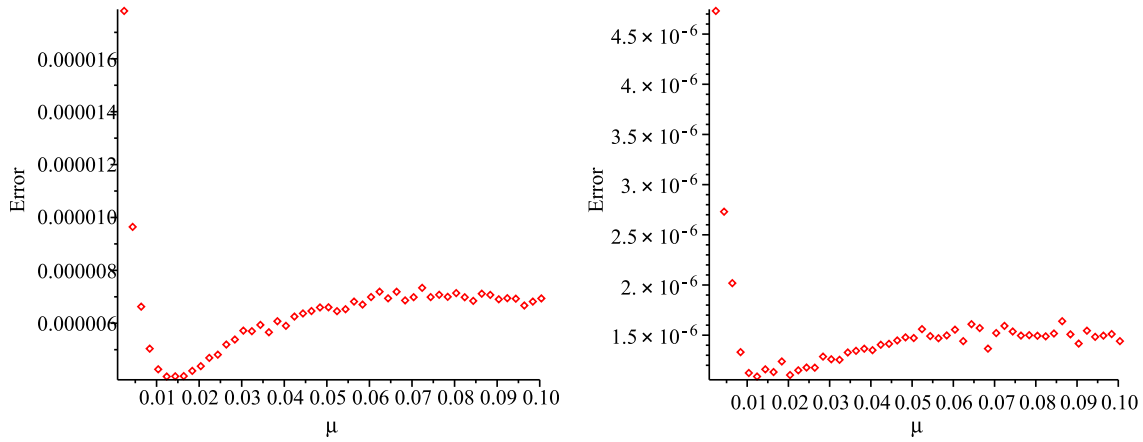
3.4 Experimental Results

3.4.1 Isolated Symbol Classification

Coordinate Functions Figure 3.3 shows the error rate for recognition using coefficients of the X and Y coordinate functions. An error rate of approximately 2.4% is reached for $\mu = 0.04$ and therefore this value is taken as the optimum for approximation of coordinate functions.

Optimal μ for Characters with Different Complexities We find that the optimal μ value is not strongly correlated with the complexity of characters. On the other hand, the recognition error is correlated with the complexity. Samples with small complexity $4 \leq \eta \leq 4.5$ (most of which are linear symbols such as "-") have 0% classification error for most of values of $\mu \in (0, 0.1]$. Recognition error is increasing with the increase of complexity and reaches 5.8% for samples that have the maximal value of $\eta \approx 8.2$ in our dataset, such as "g". The optimal values of μ for recognition of samples with different complexities are shown in Figure 3.4. Results of Spearman and Kendall tau-a correlation tests between complexity and μ are respectively: $\rho_{\mu,\eta}(13) = 0.52$, $p = 0.047$ and $\tau_{\mu,\eta}(13) = 0.38$, $p = 0.053$.

Integral Invariants Figure 3.5 shows the average maximum error of the coefficients of integral invariants with respect to different rotation angles as explained in section 3.2. The optimal value of μ, giving minimal error for I_0 and I_1, is found to be 0.012. Thus, for robust rotation-independent classification, each invariant should be approximated with the obtained μ. This value is preferable, since it provides the highest degree of invariance.



(a) (b)

Figure 3.5: Average maximum error in coefficients of (a) I_0 and (b) I_1 depending on μ

Table 3.1: The recognition error, the maximum approximation error and the average relative error for different degrees of approximation d, $\mu = 0.04$

Degree of approximation	9	10	11	12	13	14	15
Recognition error %	2.57	2.49	2.46	2.43	2.44	2.45	2.46
Maximum approximation error	707	539	539	484	475	494	500
Average relative error ($\times 10^{-3}$)	1.9	1.6	1.4	1.2	1.1	1.0	1.2

Evaluation of Degree of Approximation The recognition error, the maximum absolute approximation error and the average relative approximation error are presented in Table 3.1. We find degree 12 to be the optimum for recognition of symbols in our collection.

It is interesting that the recognition error starts to increase for $d > 12$. A similar trend applies to the maximum absolute and average relative errors. This confirms that higher order approximation may not be the optimal choice. On one hand it may lead to the Runge's phenomenon and on the other hand it may cause overfitting.

3.4.2 In-Context Classification

There are 3 parameters that in-context recognition rate can depend on, see Algorithm 8: the number p of closest classes in computation of error likelihood, the rotation angle, and the size n of the set of characters. To evaluate p, we fix the parameter $n = 3$ and perform classification for values p of 2, 3 and 4. We find that p has almost no effect on recognition error, and therefore, we take $p = 3$ and continue the experiments.

With the fixed value of p, evaluation is performed depending on n and the rotation angle, see Figure 3.6. A significant reduction in error rate is achieved compared to the results reported

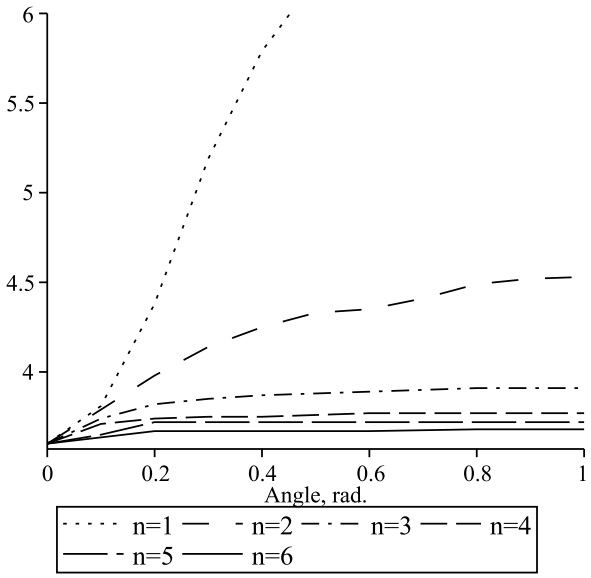


Figure 3.6: Recognition error (%) for different size of context n and different angles of rotation (in radians)

earlier [20], which are equivalent to $n = 1$. The major improvement is obtained if sequences of length $n = 3$ are recognized, rather than 1 or 2. For example, with rotation of 1 radian, $n = 3$ gives an error rate of 3.75% versus 8.2% reported previously. For more accurate classification and/or depending on an application, higher n can be used.

3.5 Conclusion

We have investigated several methods for improvement of orthogonal-series based character recognition, both in the case of isolated characters of known orientation and sequences of characters of unknown orientation. We have found (1) an optimal range of values for the jet scale for coordinate basis functions, (2) that this optimal value of μ, to a first approximation, does not depend on the complexity of the characters tested, (3) optimal values for the jet scale for the integral invariants I_0 and I_1, used for transformation-independent recognition, and (4) the optimal degree of the approximating series. In addition, we developed an in-context rotation-invariant algorithm that yields substantially better results than isolated recognition and can be extended to other transformations. These findings can be integrated into a character recognition system in the cloud.

Chapter 4

Recognition of Relatively Small Handwritten Characters, *or* "Size Matters"

Shape-based online handwriting recognition suffers on small characters, in which the distortions and variations are often commensurate in size with the characters themselves. This problem is emphasized in settings where characters may have widely different sizes and there is no absolute scale. We propose methods that use size information to adjust shape-based classification to take this phenomenon appropriately into account. These methods may be thought of as a pre-classification in a size-based feature space and are general in nature, avoiding hand-tuned heuristics based on particular characters. This chapter is based on the paper "Recognition of Relatively Small Handwritten Characters or "Size Matters"" co-authored with Stephen M. Watt [55], that appeared in the proceedings of the 13th International Conference on Frontiers in Handwriting Recognition.

4.1 Introduction

Size normalization is usually one of the early steps in the recognition of both handwritten and typeset characters, but can also be the source of errors. Characters can have different sizes for two reasons: First, the same symbol may appear in different sizes. An obvious example of this would be footnotes and titles having different sizes from normal text. Other examples would include: place names in map labels having greatly varying size, and the symbols of mathematics, which are smaller when written as superscripts or subscripts or larger when written as n-ary operators. Secondly, different symbols within the same symbol set may have different size rel-

ative to each other. For example, a period will be smaller than a lower case "o", which will in turn be smaller than a capital "M". When these two situations are combined, size normalization is a double-edged sword—it is required, but it can also lead to increased ambiguity.

We are motivated by the application of online mathematical handwriting recognition. Characters will be of greatly varying size and size can vary on a character-by-character basis, rather than word-by-word or sentence-by-sentence. In this setting, we have found it effective to use shape-based classification with orthogonal series representation of the curves traced, see Chapter 2. It was observed, however, that for very small traces the shape of the curve, when scaled, may be quite arbitrary. In these cases, the original size of a symbol is of high importance.

Recognition systems may adopt *ad hoc* rules to identify characters of unusual size, e.g. commas, long lines, arrows, *etc*. What is lacking in this approach are general principles by which such symbols requiring special treatment may be determined without any *a priori* knowledge of the symbol set, and how special rules to recognize them may be generated.

We propose a two-step processing method with samples being first pre-classified by size, and then recognized by shape. We take advantage of the usual cluster analysis techniques on a space of feature vectors computed from size measures. This may be used in two ways: first to do absolute classification based on size, and second, to do a blended classification, weighting unusually sized samples differently than samples whose size tends to the mean. These ideas can further be extended to literally any symbol set to identify those classes that are more easily separated by size measures than shape measures, e.g. lines, dots, etc.

We present three approaches to classification of small characters based on the relative size of the samples with respect to other symbols in the collection. The size of all samples is expressed in a metric unit, derived from the dataset. In the first method, that can be regarded as a 1-dimensional classifier, a feature is computed from a letter based on its width and height, regulated by a parameter. Given that the parameter is optimized, the method is shown to yield good results for our purposes. This method can be further extended to linear characters, such as "–", "|" with appropriate size measure. The second method is a generalized version of the first technique and it suggests to compute several parameters not only from the size of a letter, but also from its shape, e.g. the area of the convex hull of trace points of the character. Then, one-vs-one support vector machine (SVM) classification becomes a natural way to differentiate classes, if the number of classes is small. However, there are some dictionaries with large set of characters that have identical shape and can only be distinguished by its size. Examples include some capital and low-case characters from the Latin and Greek alphabets, e.g. Kk, Oo, most of the symbols from the Russian alphabet, e.g. Вв, Гг, Дд, Ии, musical notation, and Benesh notation. The third approach is the most robust and suitable for collections with large number of small classes. The distance to the convex hull of coefficients of approximation of

coordinate functions [26] is adjusted based on the size of the test sample and the average size of samples in the candidate class. All of the methods are shown to improve significantly the current state of our algorithm with respect to small characters.

The rest of the chapter is organized as follows. Some of the preliminaries are given in Section 4.2. Description of the size-sensitive classification schemes is given in Section 4.3, including the details of the measurement unit, the 1-dimensional and 3-dimensional classification algorithms, as well as the weight-based method. Experimental setting and results are reported in Section 4.4. Section 4.5 concludes the chapter.

4.2 Previous Work

Partially related problems have been studied in the past. A conventional approach to identification of small samples is by comparison with a fixed threshold, expressed in pixels. The adaptive normalization method developed in [45] adjusts the size of a character based on its aspect ratio. In [5] it is proposed to estimate the principal line, and correspondingly the size of symbols, using the pixel count histogram when projected on the vertical axis. Recognition rate of handwritten numerals depending on the size was investigated in [29]. In [63] it is proposed to perform size normalization with a Hough transform.

These methods are designed for either processing characters independently or for extraction of information from a set of characters. In contrast, we propose to apply special classification rules to *relatively* small symbols.

An efficient technique for online classification of characters has been described in Chapter 2. The technique is overall robust, but has a drawback, related to size-normalization – it does not take into account the initial size of a sample. As a result, small samples are scaled to the size of a regular character that leads to incorrect classification. Examples of small samples are shown in Figure 4.1, where it is easy to observe that, for instance, normalized period can be mis-classified as many other symbols, comma resembles a closing bracket, while quotes are hard to distinguish from “11”. Thus, the algorithm requires a robust adaptive size normalization approach.

4.3 Size-Sensitive Classification Schemes

4.3.1 The Unit of Measurement

To treat small samples efficiently, one has to identify what the small character is. The size of a small symbol should not be dependent on the device, nor identified as a constant amount of

Figure 4.1: Examples of scaled small characters from the top row to the bottom: period, comma, quotes.

pixels. Instead, the size should be expressed in terms of some properties of the dataset. Similar to the notion of *Ex-typography*, we choose to take the average height of lower-case x as the unit measure, and denote this value as *ex*, analogous to the *ex* measure in CSS [1]. In other words, *ex* can be understood as a metric unit for all characters in a database. In this setting, we can separate small classes from other classes based on dynamic size measures.

4.3.2 1-Dimensional Classification

The algorithm described in this section is the simplest form of a classifier, since only one feature is analyzed – the size of the sample. Despite its simplicity, in the experimental section we show that this technique has very low error in recognition of certain classes due to the dynamic nature of the size measure.

The Size Measure If the size of a character c is analyzed by its bounding box, there are essentially two types of size measures: perimeter-based and area-based. The perimeter-based measure is studied in this section $s(c) = \alpha w(c) + h(c)$, where α is a parameter, $w(c)$ and $h(c)$ are width and height of the bounding box of the character. We empirically find the α that gives the lowest classification error. The area-based feature is considered in Section 4.3.3.

Classification Consider a dataset with only two classes $\{\circ, \times\}$ that are to be classified with respect to size, and the average size of $\circ$ is less than the average size of $\times$. Let $s_{\{\circ,\times\}}$ be the size threshold that separates the classes. Then a sample from the class $\circ$ ($\times$) is considered to be classified incorrectly, if its size is greater (smaller) than $s_{\{\circ,\times\}}$. We denote with $I_\circ$ ($I_\times$) the set of

Algorithm 3 Find Separating Threshold($S_\circ$, $S_\times$, s)

Input: $S_\circ$ – the set of samples of the class $\circ$, $S_\times$ – the set of samples of the class $\times$, s – the array of sizes of samples from both classes, sorted in ascending order.

Output: $s_{\{\circ,\times\}}$.

Compute differences between consecutive elements of S as $\Delta_i = s[i] - s[i-1], i = 1, .., n$.

$D_{\{\circ,s[n]\}} \leftarrow 0$

for all $i = n - 1$ to 0 **do**

Compute the overlap for samples of the class $\circ$, if $s[i]$ is the threshold

$$D_{\{\circ,s[i]\}} \leftarrow k_{\{\circ,s[i]\}}\Delta_{i+1} + D_{\{\circ,s[i+1]\}}$$

where $k_{\{\circ,s[i]\}}$ is the number of incorrectly discriminated samples of $\circ$ for the threshold $s[i]$.

end for

$D_{\{\times,s[0]\}} \leftarrow 0$

for all $i = 1$ to n **do**

Compute the overlap for samples of the class $\times$, if $s[i]$ is the threshold

$$D_{\{\times,s[i]\}} \leftarrow k_{\{\times,s[i]\}}\Delta_i + D_{\{\times,s[i-1]\}}$$

where $k_{\{\times,s[i]\}}$ is the number of incorrectly discriminated samples of $\times$ for the threshold $s[i]$.

end for

for all $i = 0$ to n **do**

$$D_{\{\circ,\times,s[i]\}} \leftarrow D_{\{\circ,s[i]\}} + D_{\{\times,s[i]\}}$$

end for

return $\{s[m] \mid D_{\{\circ,\times,s[m]\}} = \min_{i=0..n} D_{\{\circ,\times,s[i]\}}\}$

incorrectly classified samples of $\circ$ ($\times$). Then, the overlap of the classes is computed as

$$D_{\{\circ,\times,s_{\{\circ,\times\}}\}} = \sum_{i\in I_\circ}(s(i) - s_{\{\circ,\times\}}) + \sum_{i\in I_\times}(s_{\{\circ,\times\}} - s(i))$$

The threshold $s_{\{\circ,\times\}}$ that minimizes the overlap can be found in $O(n)$ given that sizes have been computed and stored in a sorted array, where n is the total number of samples in $\circ$ and $\times$, see Algorithm 3 for details. The algorithm can be easily extended to an arbitrary amount of classes.

The classification error is measured as described in Algorithm 4.

Algorithm 4 classificationError(α)

Input: α - the parameter in the size measure.
Output: Classification error.

For the given α: Compute sizes of samples.
{In 10-fold cross-validation over the dataset}
for $i = 1$ to 10 **do**
Take the i-th training set and find $s_{\{\circ,\times\}}$ with Algorithm 3.
Test $s_{\{\circ,\times\}}$ with the i-th test set. The classification error is reported as the ratio of incorrectly discriminated samples to the total number of samples in the test set.
end for
return The average discrimination error over the 10 runs.

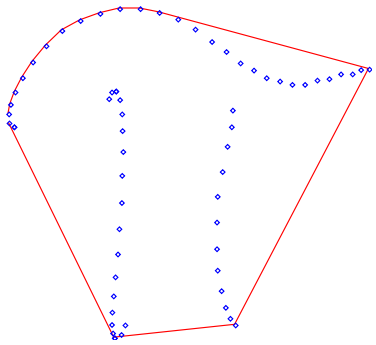


Figure 4.2: Convex hull of a sample

4.3.3 3-Dimensional Classification

In this scheme three features are extracted from characters: the height and the width of the bounding box, and the area of the convex hull of points of the sample, see Figure 4.2. We test whether these indicators are sufficiently discriminative with an SVM classifier.

4.3.4 Weight-Based Classification

The letter "." can usually be classified based on its size in *ex* units. By analyzing sizes of characters in a dataset, one can obtain the minimal size threshold of samples, other than ".". If the size of a test sample is smaller than the threshold, then it is automatically classified as ".". If the size is greater, the character still can be ".". Therefore, the class of "." is considered in computation of distances, described below.

Unlike ".", other small symbols, such as ",", preserve its initial shape after normalization, even though the letter maybe scaled significantly and appear as a different character. Thus, the shape and size should both be considered in classification. The distance to the small classes is

Algorithm 5 WeightedClassification(x)

Input: x - a test sample.
Output: The result of classification.

$s_x \leftarrow \text{width}(x) + \text{height}(x)$
{Select k nearest neighbours of candidate classes $C_1, ..., C_N$, as described in [26]}
for $i = 1$ to N **do**
  $d_i \leftarrow D(x, \text{CHNN}_k^i)$
  if C_i is a class of small symbols **then**
    $d_i \leftarrow (\omega(s_x) + \beta|\omega(\overline{s_i}) - \omega(s_x)|) \cdot d_i$
  end if
end for
return $C_j | d_j = \min_{i=1..N} d_i$

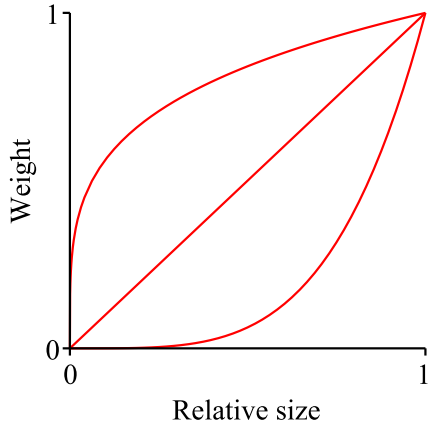


Figure 4.3: Examples of the weight function depending on the relative size: $\omega(s) = s^{1/4}$, $\omega(s) = s$, and $\omega(s) = s^4$

adjusted based on the average relative size of samples in the class and the relative size of the test sample

$$D_{adj} = (\omega(s_x) + \beta|\omega(\overline{s_i}) - \omega(s_x)|) \cdot D(x, \text{CHNN}_k^i)$$

where s_x is the relative size of the test sample x (the sum of its width and height), $\overline{s_i}$ is the average relative size of samples in the test class i, β is a parameter, $D/D_{adj}(x, \text{CHNN}_k^i)$ is the distance/adjusted distance from the test sample to the convex hull of k nearest neighbours of the class i [26], where i is one of the small classes. The distance to regular-size classes is computed without the weight adjustment. We take the function $\omega(s)$ to have the form s^γ where γ is a numeric parameter to be evaluated. See Figure 4.3 for examples of $\omega(s)$. This method is illustrated in Algorithm 5.

Besides their size, small characters can usually be differentiated by positioning, relative

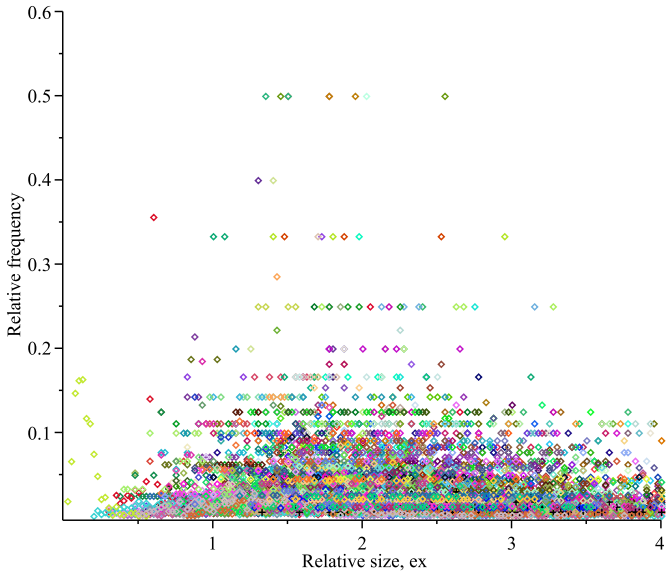


Figure 4.4: Relative frequency vs relative size for the different classes in the ORCCA dataset

to the baseline and mean line. However, we leave that analysis to another recognition layer, responsible for the spatial segmentation of formulas.

4.4 Experiments

4.4.1 Experimental Setting

The experimental dataset is based on the database of handwritten characters, collected at the Ontario Research Centre for Computer Algebra, a subset of the dataset described in [26]. Since the dataset does not contain classes with small characters, we obtained samples "." and ","/"'" by decomposing the following symbols: ":", "ä", "÷", "ȧ", "≐","!","...","i","j","⊙","?",";". Visual examination of the small characters written within the context of another character and the small letters written independently did not reveal significant differences. Therefore, we find this setting adequate. Overall, we have collected 803 samples of "." and 315 samples of ","/"'".

The physical size of an *ex* unit is 823. The relative frequency of sizes of samples, shown in Figure 4.4, was computed as follows:

1. Split the range of sizes in k intervals: $(s_0, s_1), (s_1, s_2), ..., (s_{k-1}, s_k)$. In the experiments, $k = 40$.

2. The relative frequency on an interval m is found as the ratio of the number n_m of samples in the interval to the total number of samples in the class: $n_m / \sum_{i=1}^{i=k} n_i$.

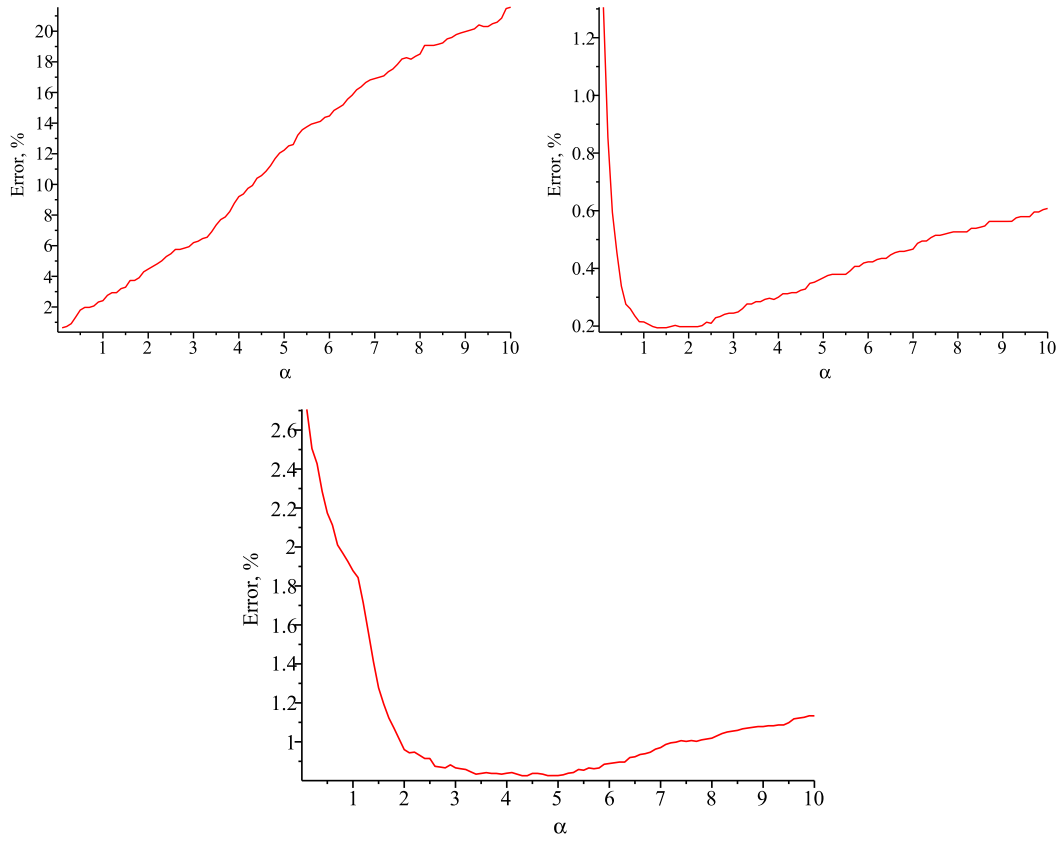


Figure 4.5: The classification rate depending on α for: "." and "," (left), "." and the rest of the classes (centre), "," and the rest of the classes (right)

3. Sizes are computed as the sum of width and height with $\alpha = 1$.

Note, that the most frequent size of "." is $\approx 0.02ex$. Therefore, the value of $0.01ex$ may be interpreted as *thickness* of digital ink and can be used in calligraphy of recognized characters or for beautification of scripts. Another interesting observation is that the frequencies seem to be centered approximately at $ex = 2$, which proves ex being the appropriate unit of measure for this type of analysis.

The recognition experiments were performed in 10-fold cross-validation: each collection has been split randomly in 10 approximately equal parts and the classification rate has been measured 10 times.

4.4.2 Performance before the Improvement

To estimate the performance of the methods developed in this chapter, we first measure recognition of small characters with the algorithm described in [26] and optimized in [51], where 97.6% classification rate was achieved. The recognizer is trained with all samples from our

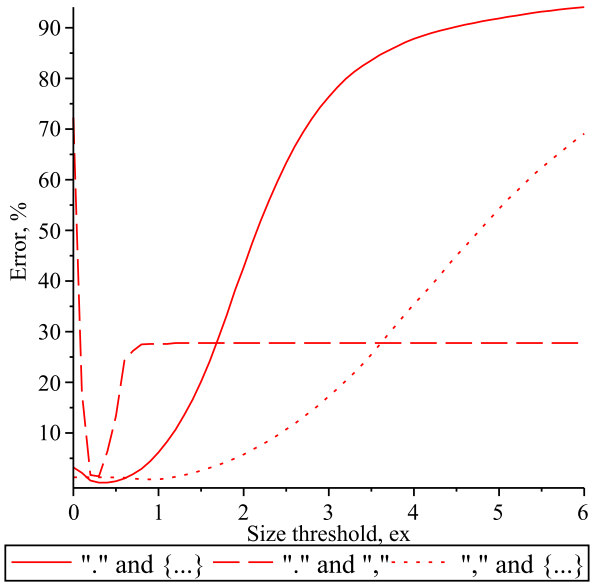


Figure 4.6: The recognition error depending on the size threshold for $s_{\{\text{“.”},\{...\}\}}$, $s_{\{\text{“.”},\text{“,”}\}}$, and $s_{\{\text{“,”},\{...\}\}}$

dataset (small and regular) and tested with small samples. The obtained classification error of the small samples is $\approx$ 17.5%, which is significantly higher than the classification error of regular sized characters reported in [51].

4.4.3 1-Dimensional Classification

In this experiment, all characters are divided in three parts: ".", ",", and the rest of the regular size classes in the dataset, denoted as {...}. The objective is to find optimal values of α that allow correct pair-wise discrimination between the parts. The recognition error as a function of α is shown in Figure 4.5. The values of α that yield the lowest classification error between "." and "," (0.6%), "." and {...} (0.2%), "," and {...} (0.8%) are respectively 0.1, 1.3, 4.4, and the values of the size threshold $s_{\{\circ,\times\}}$ are respectively 0.26*ex*, 0.34*ex* and 0.95*ex*. The stability of the recognition error depending on the threshold is shown in Figure 4.6.

4.4.4 3-Dimensional Classification

These experiments were performed with the SVM-Java [37], a Java implementation of SMO [62] technique for training an SVM. A subset of the collection of regular classes has been considered in this experiment: we randomly selected 1000 samples. The classes of "." and "," remained unchanged. The following respective error rates have been obtained for one-versus-one classification with the linear kernel for the classes "." and ",", "." and {...}, "," and {...}: 2.38%,

Table 4.1: Classification error, depending on β and γ

β	0.3	0.6	0.3	0.6	0.9	0.3	0.6	0.9
γ	2.4	2.4	2.7	2.7	2.7	3.0	3.0	3.0
Er.,%	2.75	3.48	2.76	2.94	3.21	2.06	2.23	2.60

1.44%, 4.92%. These results can be further improved by considering alternative kernels.

4.4.5 Weight-based classification

With optimization of the parameters β and γ, we obtained the classification error, as reported in Table 4.1. With the best result of 2.06% error, one can observe significant improvement over the original error of 17.5% of the algorithm on small samples.

4.5 Conclusion and Future Work

We have presented methods to address the large shape variations that can occur in small characters in handwritten samples. When there are only one or two classes which have much smaller characters than the rest, we have found that simple discrimination based on an optimized linear combination of width and height to be very effective. We have shown this can be combined effectively with shape-based methods by weighting shape and size depending on size of typical characters in the classes. We have found that using the area of the convex hull of characters, rather surprisingly, does not improve the accuracy over using a linear combination of width and height.

The presented work does not address differentiation between disconnected segments of a symbol and independent small characters. This is the question of recognition of groups of strokes that can be solved by construction of classification theories and computation of the confidence of each theory. In this chapter we have focused on devising general methods for very small characters. The developed contributions will make the cloud-based recognition engine more robust.

Chapter 5

A Structure for Adaptive Handwriting Recognition

We present an adaptive approach to the recognition of handwritten mathematical symbols, in which a recognition weight is associated with each training sample. The weight is computed from the distance to a test character in the space of coefficients of functional approximation of symbols. To determine the average size of the training set to achieve certain classification accuracy, we model the error drop as a function of the number of training samples in a class and compute the average parameters of the model with respect to all classes in the collection. The size is maintained by removing a training sample with the minimal average weight after each addition of a recognized symbol to the repository. Experiments show that the method allows rapid adaptation of a default training dataset to the handwriting of an author with efficient use of the storage space. This chapter is based on the paper "A Structure for Adaptive Handwriting Recognition" co-authored with Stephen M. Watt [53], that appeared in the proceedings of the 13th International Conference on Frontiers in Handwriting Recognition.

5.1 Introduction

It was described in Chapter 2 how samples are classified with the distance to the convex hull of k nearest neighbors in the space of coefficients of approximation. The method yields high accuracy, but has a significant drawback – it does not adapt to variations in writing style of trained classes. This is not acceptable in a production environment, since out of the box recognition applications are usually trained with a default dataset of samples. Such dataset relieves the user from an exhaustive training of a mathematical recognizer that may include several hundred classes. However, default training of some classes may differ from the writing style of

the user. This concern is aggravated for online algorithms that typically depend on the direction and order of writing of strokes. Therefore, instances that appear identical visually, but written in different styles, will be represented by points, positioned in absolutely different locations in the coefficients space. Thus, some training samples may represent noise and have negative impact on efficiency and accuracy.

The exemplar-based learning in higher dimensions is challenging due to the increase of sparsity of samples of a class. Therefore, selection of training exemplars has been thoroughly studied in instance-based machine learning and related applications [77]. Some methods suggest to retain a subset of the original instances [28, 2], while other techniques propose to compute prototypes from the training data [39]. Due to the nature of our classification method, we investigate the former approach. It can be divided in *incremental* (start from an empty training set and add instances one by one), and *decremental* (start from the training set with all samples and remove instances that are redundant or decrease accuracy). A decremental procedure *DROP1* [77] suggests to remove a point if all of its neighbors can still be correctly classified without the point. This and many other techniques [28, 2] study the local relationship between samples without taking into account that the training dataset may change over time, moving the underlying points in various directions.

We develop an online algorithm for adaptive recognition of handwritten characters that is based on reinforcement of samples that have positive impact on classification and removal of samples that cause error or are neutral. The method is suitable in both settings: When users train a recognizer from scratch or when they use the default dataset as the starting point. In the latter setting, to determine the average size of a training class, we model the error drop as a function of the number of samples and attempt to correlate parameters of the model with some spatial measurements of the class.

The proposed adaptive algorithm computes the participation weight of each of the k neighbors in a correct (incorrect) recognition and adds (subtracts) the value to (from) the total weight of the neighbor. In a sense, the method is similar to the *IB3* algorithm [2], in which removal or retaining of instances is based on counters. However, the *IB3* method is offline, meaning that it is run only once to select good classifiers out of the pool of training samples, while our algorithm is online and makes removal decisions with each new sample available from the input. The method presented has potential of asymptotic improvement in performance over the course of its use and is suitable for a variety of instance-based machine learning applications. Unlike some algorithms, based on neural networks or hidden Markov models, the proposed technique uses only gradual updates, making it suitable for real-time applications.

The main results of this chapter are

- an experimental analysis of how error rate drops as a function of the class size;

- an empirical model for the error rate, fitting the experimental data well, to determine the average size of a class for desired accuracy;

- an adaptive algorithm for distance-based symbol recognition, using the functional approximation framework.

This chapter is organized as follows. Section 5.2 explains our approach to modelling the recognition error. The adaptive recognition algorithm is presented in Section 5.3. Section 5.4 gives the experimental results that show good approximation of the error function and rapid adaptation of the recognition algorithm to the writing style of a user. Finally, Section 5.5 concludes the chapter.

5.2 Modelling the Recognition Error

In our classification paradigm, the concept of personalized recognition can be reformulated as continuous formation of the training set. A set of training characters of a class forms a cluster in the space. *A priori* knowledge of the average initial size of a training class to achieve a desired classification accuracy is important for compact storage. It has an additional usability-related benefit: When a new class is introduced to the dataset, the user can be informed about the expected error drop depending on the number of samples introduced to the class.

Here and below, we will use the following notation: n is the number of training samples that the class contains in a given moment and N is the maximal number of training samples available in the class. Based on our observation, convergence of the recognition error of samples of a class can be closely described by the models

$$\epsilon(n) = \frac{An + B}{n + C} \tag{5.1}$$

where A, B and C are parameters, or

$$\epsilon(n) = \alpha e^{-\beta f(n)} \tag{5.2}$$

where α and β are parameters, and $f(n)$ is a monotonically increasing function.

Our objective is to find values of the parameters for each class. We expect the parameters to be dependent on some inner properties of a class, as well as the positioning of the class relative to neighboring classes. Further, the mean parameters can be used to describe the average error drop.

5.3 Adaptive Recognition

Most commonly, misclassification of handwritten characters occurs when different samples are written similarly, since writing styles of users can vary significantly. On the other hand, classes of characters provided by one user can usually be discriminated well. As discussed in Chapter 2, only k samples of a candidate class are used in classification of a test symbol. Each of these k exemplars should be awarded a weight, computed as a function of the distance to the test sample. If the training symbol is located relatively close to the test character, the weight should have large absolute value, otherwise the weight should be close to zero. If the training sample is of the same class as the test symbol, the weight should be positive and otherwise – negative.

In general, distances between training samples within a class do not follow any of the major univariate distributions, since a class may contain several styles that group the exemplars. Therefore, basing the weight on statistical properties of a class can be quite challenging. Instead, we take the weight as follows: For a given test sample t_s and a training exemplar t_i, the recognition weight has the form

$$w_{t_i} = \frac{1}{d(t_s, t_i) + 1}$$

where $d(t_s, t_i)$ is the distance between the points. This weight is added to the total weight of the sample t_i, if t_s and t_i belong to the same class, and subtracted otherwise.

When a new sample is recognized, it is added to the class, and simultaneously a sample with the minimal average weight is removed from the dataset to prevent its growth. Nevertheless, at any given moment, the size of a class should not be less than k (the number of nearest neighbours that form convex hull during classification). The outline of the method is presented in Algorithm 6.

5.4 Experimental Results

This section presents experimental results of modelling the recognition error and the adaptive classification method. The experimental dataset is identical to the one described in [26].

5.4.1 Modelling the Recognition Error

We conducted a series of experiments to measure how the recognition rate changed as points were added to the classes. Each class was measured separately, in the following manner: All symbols from the class to be tested were removed from the training data set and the symbols from other classes were retained. Further, the samples from the test class were separated ran-

Algorithm 6 Adaptive recognition algorithm

Input: t_s – a test sample to be recognized.

{Recognize the sample as explained in Chapter 2} $Cl \leftarrow$ recognition class of t_s
{Recompute weights}
for $i = 1 \rightarrow T$ **do**
  {For each of the candidate classes}
  if $T_i = Cl$ **then**
    for $j = 1 \rightarrow k$ **do**
      {Increase the weight of each nearest neighbor t_{ij} in the correct class}
      $w_{t_{ij}} \leftarrow w_{t_{ij}} + \frac{1}{d(t_s,t_{ij})+1}$
    end for
  else
    for $j = 1 \rightarrow k$ **do**
      {Decrease the weight of each nearest neighbor t_{ij} in the incorrect class}
      $w_{t_{ij}} \leftarrow w_{t_{ij}} - \frac{1}{d(t_s,t_{ij})+1}$
    end for
  end if
  {Increase the counter}
  for $j = 1 \rightarrow k$ **do**
    $C_{t_{ij}} \leftarrow C_{t_{ij}} + 1$
  end for
end for
{Remove the exemplar with the minimal average weight among the classes with the number of samples $> k$}
Remove exemplar $t : \overline{w}_t = \min_{ij}\{\frac{w_{t_{ij}}}{C_{t_{ij}}}, |T_i| > k\}$
Assign an initial weight to t_s and add t_s to the recognized class.

domly into a test set P_i and a training set P_r. Then the symbols from P_r were added, initially one at a time and then in larger groups. After each addition of points, the recognition rate of the ensemble was measured using the test set. Thus, for each class, the recognition rate was tested first with 0 training points, then with 1 training point, then with 2, then after 3, 4, 5, 6, 7, 8, 10, 12, 14, 16, 20, 24, 28, 32, 40, 48, 56, 64, ... until all the training points were used. The number of training points ranged from 10 to 2048, depending on the class. This whole process was repeated ten times, and the recognition rate recorded for a class after a particular number of points was reported as the average of these ten measurements. The testing sets were selected randomly, but disjoint. The set of classes is denoted as Ω. The outline of experiments is given in Algorithm 8.

Algorithm 7 Outline of the experimental setting

```
for Each class ω in the set of classes Ω do
  Split samples in the class for 10-fold cross-validation.
  for i = 1 to 10 do
    Take the i-th part P_i for testing and the rest P_r for training.
    {Introduce integer variables used in splitting the training set.}
    s ← 0, k ← 3
    while s ≤ |P_r| do
      Clear the training set for the class ω.
      Conduct training with the first s samples from P_r.
      Conduct testing with samples from P_i.
      if s = 2^k then
        k ← k + 1
      end if
      {Increase the amount of training samples}
      s ← s + 2^(k−3)
      {where 2^(k−3) was selected heuristically, based on the observation that adding samples
      to a small training set has bigger impact than to a larger set}
    end while
  end for
end for
```

Some of the samples have several class labels. Therefore, the recognition error can be less than 100%, even if the class has zero training samples in it. Results of recognition for all classes, depending on n, are given in Figure 5.1.

We make a few observations: First, we see that for all classes the recognition rate improves dramatically with each of the first few symbols added. Most of the functions have shape that can be modelled with (5.1). For approximation, we used the Nonlinearfit Maple [35] command to evaluate A, B and C. In classes with more than a few dozen samples, the error rate appeared to drop off similarly to a negative exponential function, see Equation 5.2. In Equation 5.2,

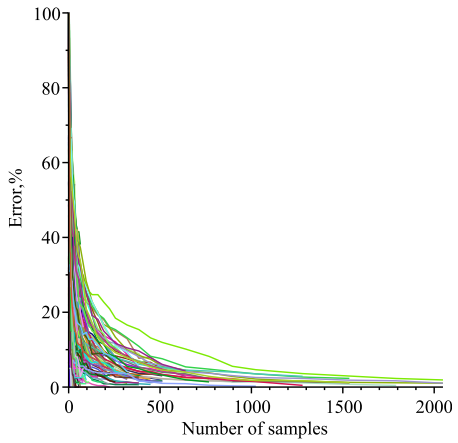


Figure 5.1: Recognition error for all classes, depending on n, the number of training samples in a class

	A	B	C
Mean	-0.007	11.718	23.398
σ	0.054	9.805	9.805

Table 5.1: The mean and the standard deviation of the parameters

$f(n) = \sqrt{n}$ was found to perform well. By taking the logarithm of both sides, the parameters can be evaluated as a linear regression between $\log(\epsilon(n))$ and $\sqrt{n}$. We used the LeastSquares Maple command to compute the least squares approximation.

We tested both models (5.1) and (5.2) and computed the average root mean square error (RMSE) among classes, obtaining respectively 0.03 and 0.87. Model (5.1) performed the better of the two, and so this is the one upon which we have concentrated. Examples of approximation with (5.1) for different values of N and the average model are given in Figure 5.2. We observed that classes of smaller size, with $N < 64$, are approximated not as good as larger classes, partially due to non-stable behaviour of the error function on the small testing set. Therefore, the mean parameters A, B and C were computed among classes with ≥ 64 training samples. The mean and the standard deviation of the variables are shown in Table 5.1. The small mean value of parameter A can be considered as an argument that the error model (5.1) can be simplified to $\epsilon(n) = \frac{B}{n+C}$. On the other hand, such simplification will make the model less robust and may have negative effect on the approximation accuracy. Therefore, we decided to keep the parameter.

The average RMSE between the modelled recognition rate and the actual recognition rate for classes of certain size is presented in Figure 5.3(a). Figure 5.3(b) shows the percentage of classes that are approximated with RMSE less or equal a given value.

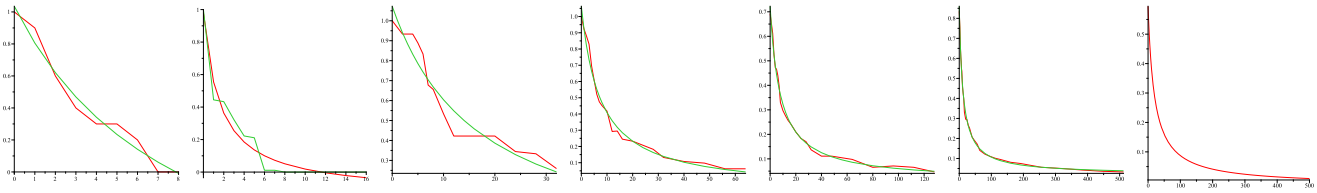
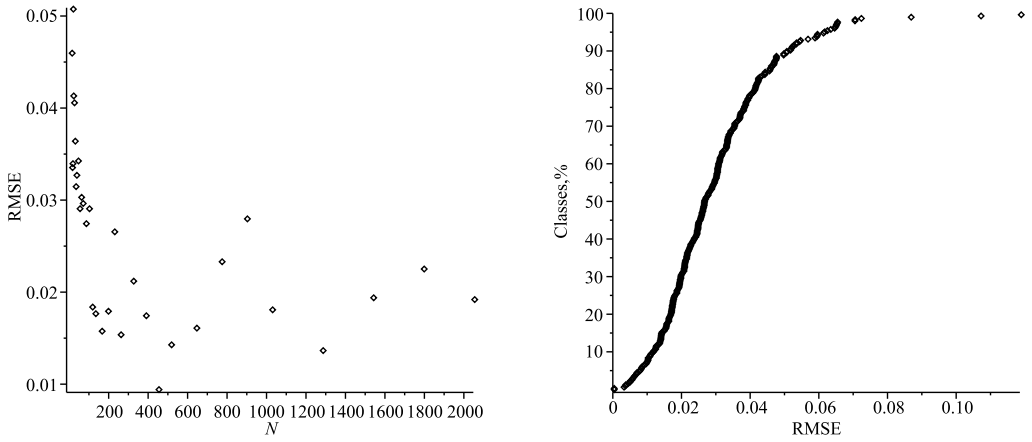


Figure 5.2: Examples of approximation of error for classes of different size N



(a) (b)

Figure 5.3: RMSE results: (a) Average RMSE for classes of different N, (b) Percentage of classes that are approximated with RMSE less or equal given RMSE

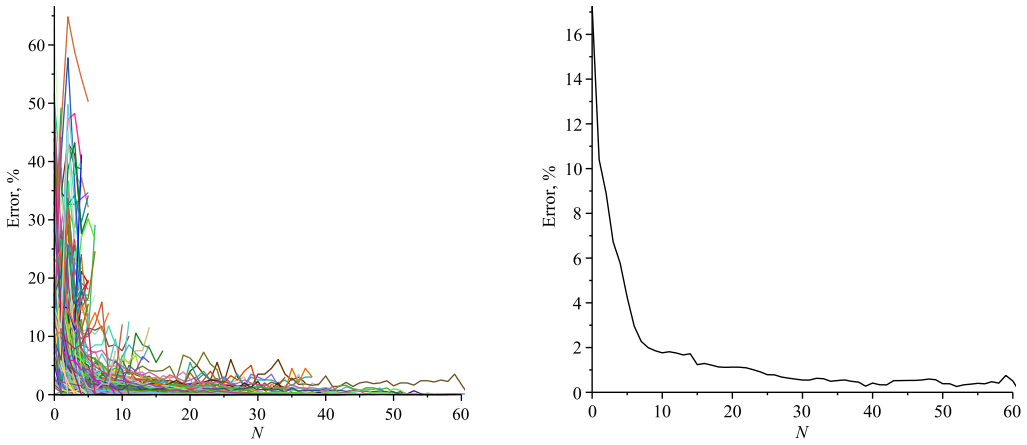
5.4.2 Correlation between class measurements and A, B and C

We question whether parameters A, B and C are related to spatial characteristics of the class, such as positioning of points within the class and distance to neighboring classes. For each class i, the following measurements are considered (in Euclidean distance)

- R_1^i - the maximal distance from the class center to any point in the class.
- $R_{.75}^i$ - the minimum radius of a ball centered at the class center that encloses 75% of points in the class.
- R_a^i - the average of radii from all points in the class to the class center.
- $R_\sigma^i = R_a^i + \sigma_i$, where σ_i is the standard deviation of the radii from points in the class to the class center.
- D_F^i - the minimum distance between points of the class to the closest neighboring class.

	Measurement	Spearman	Kendal tau-a
A	$\check{D}_1$	-0.29	-0.19
B	$\check{D}_\sigma$	-0.55	-0.39
C	$\check{D}_\sigma$	-0.59	-0.42

Table 5.2: The measurements with the largest absolute values of the correlation coefficients for each approximation variable



(a) (b)

Figure 5.4: Adaptive recognition error of the $(N + 1)$-th sample in a class: (a) For each author, (b) Average among the authors

In addition, we study the measurements

$$\check{D}_L^i = \min_{j \neq i}(d_{ij} - R_L^i - R_L^j),$$
$$\bar{D}_L^i = \operatorname{avg}_{j \neq i}(d_{ij} - R_L^i - R_L^j)$$

where L is any of the labels $1, .75, a, \sigma$ and d_{ij} is the distance between centers of classes i and j.

Spearman and Kendall tau-a tests did not demonstrate sufficient correlation of these measures with the model parameters. The largest absolute values of statistically significant correlation coefficients for corresponding class measurements are presented in Table 5.2.

5.4.3 Adaptive Recognition

For this experiment, each character in the collection is assigned to the author who provided the symbol. Then for each author, the dataset is split in two parts: samples provided by the author (used in testing) and the rest of the dataset. During the training phase, for each class,

we randomly select K samples and form the default training set. The value of K, the initial size of a training class, can be determined from the error modelling, and for this experiment we take $K = 30$. During the testing phase, a test sample is extracted from a randomly chosen class among those written by the test author and recognized. The recognition error of the N-th sample by the author is computed as the ratio of the number of misrecognitions of the N-th sample to the total number of N-th samples tested. This run is repeated 200 times and the average for each author is reported in Figure 5.4(a). Figure 5.4(b) shows the average error among all the writers. We observe that the adaptive algorithm on average results in a rapid decrease of error and converges to $\approx 99\%$ accuracy.

5.5 Conclusion

We have shown how handwriting recognition techniques based on functional approximation methods are well suited to adaptive setting. Rather than organizing the workflow as a training phase followed by a use phase, we see continuous improvement of recognition results taking advantage of correction history. In our setting, based on convex hulls of classes in the coefficient space, adaptation consists of weight-based evolvement of the shape of the class envelopes. The experiments have shown that the error rate drops approximately as $(An + B)/(n + C)$ as samples are seen, and that A, B and C slightly vary by class and correlate with class measurements to a minor degree. The average values of the parameters can be used to determine the size of each class in a default training dataset. The initial set assembled this way serves as an input to a weight-based adaptive classifier. The weight of an exemplar is computed from the distance to the test sample. With each recognition, the symbol with the minimal average weight gets deleted from the collection. Experiments show that the model allows rapid adjustment to the style of a particular writer and converges to approximately 99% accuracy. This model is an important element of the adaptive cloud-based recognition architecture.

Chapter 6

A Cloud-Based Recognition Framework

While writer-independent handwriting recognition systems are now achieving good recognition rates, writer-dependent systems will always do better. We expect this difference in performance to be even larger for certain applications, such as mathematical handwriting recognition, with large symbol sets, symbols that are often poorly written, and no fixed dictionary. In the past, to use writer-dependent recognition software, a writer would train the system on a particular computing device without too much inconvenience. Today, however, each user will typically have multiple devices used in different settings, or even simultaneously. We present an architecture to share training data among devices and, as a side benefit, to collect writer corrections over time to improve personal writing recognition. This is done with the aid of a handwriting profile server to which various handwriting applications connect, reference, and update. The user's handwriting profile consists of a cloud of sample points, each representing one character in a functional basis. This provides compact storage on the server, rapid recognition on the client, and support for handwriting neatening. In this chapter we use the word "cloud" in two senses. First, it is used in the sense of cloud storage for information to be shared across several devices. Secondly, it is used to mean clouds of handwriting sample points in the function space representing curve traces. We "write on clouds" in both these senses. This chapter is based on the paper "Writing on Clouds" co-authored with Stephen M. Watt [54], that appeared in the proceedings of the 2012 Conferences on Intelligent Computer Mathematics.

6.1 Introduction

The recognition method described in Chapter 2 does not require many training samples to discriminate a class. However, because there are a large number of classes in handwritten mathematics, the training dataset may contain tens of thousands of characters. The underlying recognition model allows the dataset to evolve over the course of normal use. Furthermore,

as a user makes corrections to mis-recognized input, new training data is obtained. Therefore, synchronization of the dataset across several pen-based devices may become tiresome. To address this aspect, we propose to delegate the storage of the training database, as well as some of the recognition tasks to a cloud.

In this chapter we describe a cloud-based recognition architecture. It has potential to be beneficial not only to end users, but also to researchers in the field. A cloud infrastructure can assist in the capture of recognition history. The "knowledge" obtained from the public usage of the recognition software can help to improve the accuracy continuously. This serves as a basis for an adaptive recognition that results in asymptotic increase of user-, region-, or country-centered classification rate. Additionally, such a model has a number of other advantages: First, it allows the writer to train the model only once and then use the cloud with any device connected to the Internet. Secondly, it gives the user access to various default collections of training samples across different alphabets (e.g. Cyrillic, Greek, Latin), languages (e.g. English, French, Russian), and domains (e.g. regular text, mathematics, musical notation, chemical formulae). Thirdly, it provides a higher level of control over the classification results and correction history.

The architecture we present may be applicable to a variety of recognition methods across different applications, including voice recognition, document analysis, or computer vision. To demonstrate its use in recognizing handwritten mathematical characters, we have performed an experiment to measure the error convergence as a function of the input size and find an average number of personal samples in a class to achieve high accuracy.

Cloud computing allows remote, distributed storage and execution. The economic stimuli for providing software services in a cloud infrastructure are similar to those for centralized supply of water or electricity. This relieves consumers from a number of issues associated with software maintenance, while the provider may continuously improve the service. Agility of a cloud service is usually achieved by its internal organization according to the principles of the Service-Oriented Architecture (SOA). SOA allows splitting computational tasks into loosely coupled units, services, that can be used in multiple unassociated software packages. An external application executes a service by making a call through the network. The service consumer remains independent of the platform of the service provider and the technology with which the service was developed.

Several related projects have been described that mostly target development of managed experimental repositories and resource sharing in the context of: document analysis [41], astronomical observations [69], or environmental research [6]. In contrast, our primary objective is improvement of usability of recognition software across different pen-based devices. Collecting a comprehensive database that facilitates research is the second priority.

The rest of the chapter is organized as follows. Section 6.2 describes the cloud–based recognition framework, starting by giving an overview of the components. Then the flow of recognition and correction, as well as possible manipulations of clusters, are presented. Section 6.3 describes the implementation of projection of samples from high-dimensional space to the plane. This is important for visual analysis of the descriptiveness of the classification indicators. Section 6.4 describes details of the implementation of the system, the structure of a personal profile, the interface for training and recognition, the server side, as well as calligraphic representation of recognized characters. Section 6.5 presents experimental evaluation. In Section 6.6 we show that the cloud environment can improve recognition flow by semi-automated training of the recognizer, based on conditional probability of writing styles of the user. Section 6.7 concludes the chapter.

6.2 Clouds Serving Clouds

Touch screens with the ability to handle digital ink are becoming *de facto* standards of smart phones and tablet computers. The variety of such platforms challenges conventional recognition applications because:

- Certain mobile devices have limited storage capacity and computational power, restricting ink storage and processing. Recognition of handwritten math requires extra resources to build classification theories and to calculate the confidence of each theory [8].

- Development of a single recognition engine that runs efficiently across all the platforms is not easy, and in most cases a trade off has to be made, affecting classification performance.

- The evolving personal training datasets and correction histories are not synchronized across the devices.

Similar to the software as a service delivery model, we propose to have digital ink collected and, possibly, processed through a thin client, but its storage and some computationally intensive procedures are performed centrally in the cloud.

From the high-level, the system contains the following elements

- *Canvas* of a pen-based device, that can collect digital ink.

- *HLR* (High-Level Recognizer) accepts raw ink from the canvas and performs initial preprocessing of the ink.

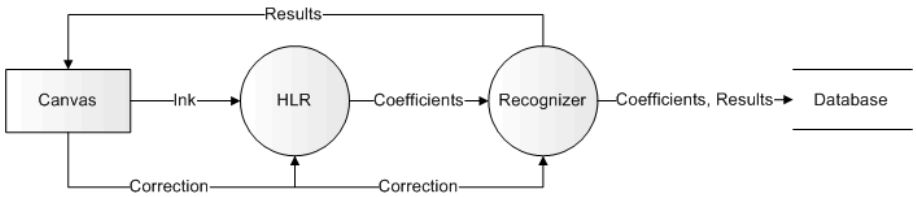


Figure 6.1: The data flow diagram for recognition and correction

- *Recognizer* is a character recognition engine, developed according to the principles described in [26].

- *Database* stores personal handwriting data, profiles of samples, correction history, etc.

Profiles of training samples are clouds of points in the space of approximated curves, each point being one character. These points are saved in a database in the cloud. When users sign up for the service, they are assigned a default dataset of training samples. If a person has several handwriting domains (e.g. different fields using mathematics, physics, music, etc), each domain should have a separate dataset, and the recognition application should allow switching between the subjects. The user shapes the datasets through a series of recognitions and corrections. Below, we show experimentally that the number of corrections decreases over time and eventually becomes quite small.

6.2.1 Recognition Flow

The overall recognition flow is shown in Figure 6.1. The High-Level Recognizer (HLR) accepts raw ink from the canvas and preprocesses it. The output of the HLR is available to the recognizer in the form of normalized coefficients. The coefficients are recognized. The results of classification are sent to the canvas and saved in the database.

Representation of Characters For a single-stroke character, after approximation of coordinates with truncated orthogonal series, the sample can be represented as

$$\frac{1}{||x,y||}, x_0, y_0, x'_1, y'_1, ..., x'_d, y'_d \tag{6.1}$$

where x_0, y_0 are Legendre-Sobolev coefficients that control the initial position of the character, $x'_1, y'_1, ..., x'_d, y'_d$ are normalized coefficients, and $||x,y||$ is the Euclidean norm of the vector [26]

$$x_1, ..., x_d, y_1, ..., y_d$$

coefficients ::= $\frac{1}{\|x,y\|}; x_0; y_0; x'_1; y'_1; ...; x'_d; y'_d$
msg ::= <m:Process>
<m:mt>*coefficients*</m:mt>
(<m:tr>*coefficients*</m:tr> <m:tr>*coefficients*</m:tr> *+)?*
</m:Process>

Figure 6.2: The format of the SOAP message sent to the cloud

The first three elements in (6.1) are ignored during recognition, but used in restoring the initial size and location of the character.

For a multi-stroke symbol, coefficients are computed for every stoke, as described for a single-stroke character, and also for all strokes joined sequentially. Coefficients of strokes are used for display of the sample and normalized coefficients of joined strokes are used for classification.

The described representation of samples allows significant saving on storage space and computations, since coefficients of symbols can be directly used in recognition without repetitive approximation [50]. However, this compression scheme is lossy and should not be used when precision of digital ink is of high importance, e.g. in applications that involve processing of personal signatures.

Recognition Individual handwriting can differ significantly from the default collection of training samples. This is illustrated by the historical use of a personal signature as a form of authentication of documents. It is to be expected that a successful recognition system should adapt to personal writing style. With *k*-nearest neighbors and related methods, the test sample can be easily introduced to the training set after classification. This facilitates adaptive recognition, since the model remains synchronized with the writer's style.

Two modes of recognition are possible, *local* and *remote*.

Local recognition is suitable for devices with sufficient computational capabilities. In this mode, the points that form the convex hulls of classes are stored on the device locally and periodically synchronized with the server. Synchronization can be performed through a profile of samples. The local recognition mode is useful when the user does not have a network connection and therefore can not take advantage of the remote recognition described below.

In *remote recognition* mode, digital curves are collected and preprocessed locally, and the coefficients are sent to a remote recognition engine. Having recognized the character, the server returns encoding of the symbol and nearest candidates. This mode allows to minimize the load on the bandwidth, since the training dataset does not have to be synchronized with the device. Coefficients can be transmitted in the body of a SOAP message, using the syntax shown in

```
...
<soap:Body xmlns:m="http://www.inkml.org/processing">
  <m:Process>
    <m:mt>0.005;94;-91;11;2;-14;64;-70;
      -18;1;-75;14;14;8;4;-2;4;0;-9;5;10;-11;5;</m:mt>
  </m:Process>
</soap:Body>
...
```

Listing 1: An example of the body of a SOAP message for a single-stroke character

```
...
<soap:Body xmlns:m="http://www.inkml.org/processing">
  <m:Process>
    <m:mt>1;0;0;-5;-22;-14;-15;-44;-72;20;13;-27;43;4;
      -28;48;-1;-10;16;-32;-17;-1;-12;</m:mt>
     <m:tr>0.005;92;-85;-1;3;-7;62;-79;-30;
      4;-61;32;4;-2;15;-4;-4;6;-3;0;6;-9;0;</m:tr>
     <m:tr>0.009;115;-100;-71;-102;-10;-1;11;1;
      -6;-8;5;6;-5;-9;2;3;-2;-5;6;6;-5;-9;</m:tr>
  </m:Process>
</soap:Body>
...
```

Listing 2: An example of the body of a SOAP message for a multi-stroke character

Figure 6.2. The element <m:mt> contains the normalization weight, the original coefficients of the 0-degree polynomials, and the normalized coefficients used in recognition. Additionally, for a multi-stroke sample, the <m:tr> element is used to represent each stroke independently. Examples of messages for a single-stroke and a multi-stroke character are shown in Listing 1 and Listing 2 respectively. The bodies of the SOAP messages contain enough information for both recognition and restoring approximate representation of a character in its initial position.

The results of recognition can be returned in a SOAP message, as shown in Listing 3. The body contains Unicode values of the top candidates to enable the client application to visualize recognized characters in a printed format. For calligraphic rendering, corresponding coefficients can be included as well.

When the recognition is incorrect, the user can fix the result on the canvas. A correction message is sent from the canvas to the recognizer and the database, see Figure 6.1. The correction message may contain Unicode value of the new character and the ID of the sample. After correction, if the recognition engine is context-sensitive, neighboring characters can be reclassified. Implementation of sensitivity to the context depends on the domain. With handwritten text, this task is solved by comparing a recognized word with entries in a dictionary. With mathematics, it is a harder problem, since expressions are represented as trees. Progress

```
...
<soap:Body xmlns:m="http://www.inkml.org/processing">
  <m:Response>
    <m:Unicode>0030, 004F, 006F</m:Unicode>
  </m:Response>
</soap:Body>
...
```

Listing 3: An example of the body of a SOAP response from the recognition service

Figure 6.3: A sample that belongs to classes “q” and “9”

can be achieved by considering the most popular expressions in the subject and their empirical or grammatical properties, see for example [47].

6.2.2 Manipulation of Clouds

With the discussed representation of samples as clouds in high dimensional space, they can also be treated as sets. In this context, corresponding theoretical domains become applicable, such as the set theory or some elements of computational geometry. Consider training characters from two classes, say i and j, forming sets S_i and S_j respectively. Then $S_i \cap S_j$ will produce samples written in an ambiguous way: If classes i and j represent characters 9 and q then a sample that belongs to both classes can look as the one shown in Figure 6.3. A naïve approach to compute such an intersection is to find the subset of points in each cluster with the distance to the second cluster being zero. To make the clouds linearly separable, the samples that belong to both clusters can be deleted or assigned a specific label. A similar operation is to find $S_q \setminus S_9$. This will give the samples that look different from samples of the adjacent class.

Another example is computing the “average” character, as the center of mass of samples in a style, and using the character in calligraphic rendering of recognized samples.

These and other operations can be expressed naturally as operations on the classes represented as clouds of points. With some other machine learning frameworks the analogous procedures can be more awkward.

6.3 Orthogonal Projection of Cloud Samples

6.3.1 Related Work

High-dimensional data samples, very frequently used in machine learning applications, can be hard to analyze without visual representation. A number of attempts have been made to map N-dimensional clusters to a plane. One of the most popular methods is Sammon projection [64], named after John W. Sammon. Sammon proposed to map high-dimensional space to a lower dimensionality (typically two or three) with the objective to preserve the distances and their ratios between points in high dimension and corresponding points in low dimension.

More formally, if d_{ij} is the distance between points i and j in high dimension and d'_{ij} is the distance between projections of the points then the function that Sammon proposed to minize is as follows

$$E = \frac{1}{\sum_{i<j} d_{ij}} \sum_{i<j} \frac{(d_{ij} - d'_{ij})^2}{d_{ij}}.$$

Sammon proposed to solve the minimization problem with gradient descent algorithm.

We use Sammon projection for 2-dimensional representation of relationship between three classes: Characters “8” (with 1453 samples), “1” (with 504 samples) and “C” (with 642 samples). Sammon mapping of samples in these classes is shown in Figure 6.4

We find this projection not truly reflect the relationship between classes and, therefore, look for an alternative way of representation of clusters.

6.3.2 Orthogonal Projection

In this section we develop another method for projecting points from high dimension to a plane. In the algorithm we consider three classes c_1, c_2 and c_3, with no requirement for the clusters to be linearly-separable. The overview of the method is as follows:

1. Find an Support Vector Machine (SVM) hyperplane that separates two of the classes, say c_1 and c_2

   $$\vec{w}_1 \cdot \vec{x} - b_1 = 0$$

   where $\cdot$ denotes the dot product and $\vec{w}_1$ is the normal vector.

2. Find another SVM hyperplane that separates the class c_3 from one of the classes c_1 or c_2,

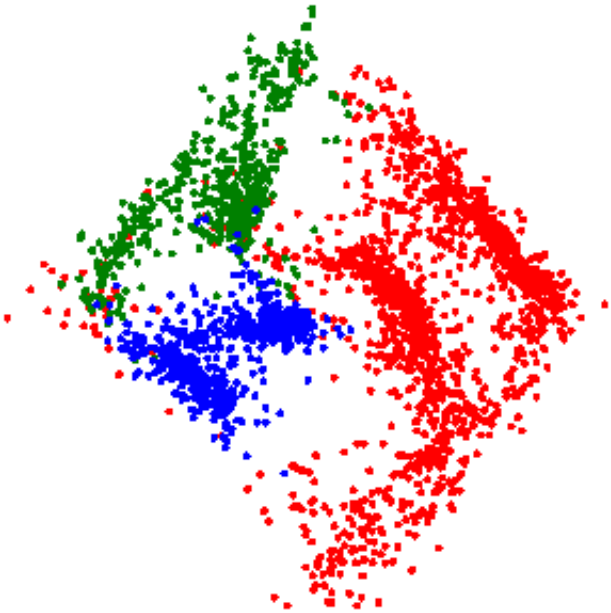


Figure 6.4: Sammon projection of the classes: “8” (red), “1” (green) and “C” (blue)

say between classes c_2 and c_3 (in practice, it is better to separate the closest classes)

$$\vec{w}_2 \cdot \vec{x} - b_2 = 0$$

3. Consider intersection of the hyperplanes

$$\begin{cases} \vec{w}_1 \cdot \vec{x} - b_1 = 0 \\ \vec{w}_2 \cdot \vec{x} - b_2 = 0 \end{cases}$$

Find translation of the hyperplanes, so that their intersection goes through the origin. After substitution $x_1 = x_1' + t_1$ and $x_2 = x_2' + t_2$, where $t_1, t_2 \in \mathbb{R}$, the translation can be obtained by solving the following system

$$\begin{cases} w_1^1 t_1 + w_1^2 t_2 - b_1 = 0 \\ w_2^1 t_1 + w_2^2 t_2 - b_1 = 0 \end{cases}$$

Having found the translation parameters t_1 and t_2, all samples in the clusters c_1, c_2 and c_3 should undergo the same translation.

4. After the translation, each of the hyperplanes forms a vector space over $\mathbb{R}$: V_1 and V_2.

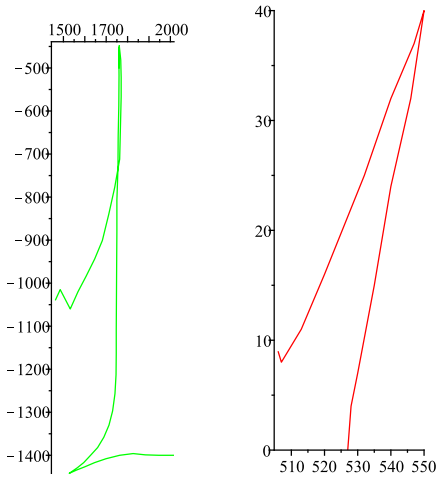
We denote

$$V_0 = V_1 \cap V_2$$
$$V = V_1 \cup V_2$$
$$V = V_0 \cup V'$$

One can consecutively generate a basis for V_0, V_1, V_2 and V. With Gram-Schmidt orthogonalization process, an orthogonal and then orthonormal basis **e** of V can be obtained.

5. Coordinates of each sample are computed in the basis **e** and projected to the subspace V' yielding 2-dimensional representation of the sample.

This algorithm results in a mapping, shown in Figure 6.6. In the Figure, it is easy to notice that the samples of characters "1" are somewhat separated in two sub-clusters. The reason is that each sub-cluster forms samples written with certain style. Indeed, Figure 6.5 shows two samples written in different styles available in the cluster: Figure 6.5(a) for the north-west sub-cluster and Figure 6.5(b) for the south-east.



(a) (b)

Figure 6.5: Samples of different sub-clusters of the character "1"

6.4 Implementation

From a high level viewpoint, the system contains the following parts, as shown in Figure 6.7.

- A user interface for training (used to collect profiles of characters).

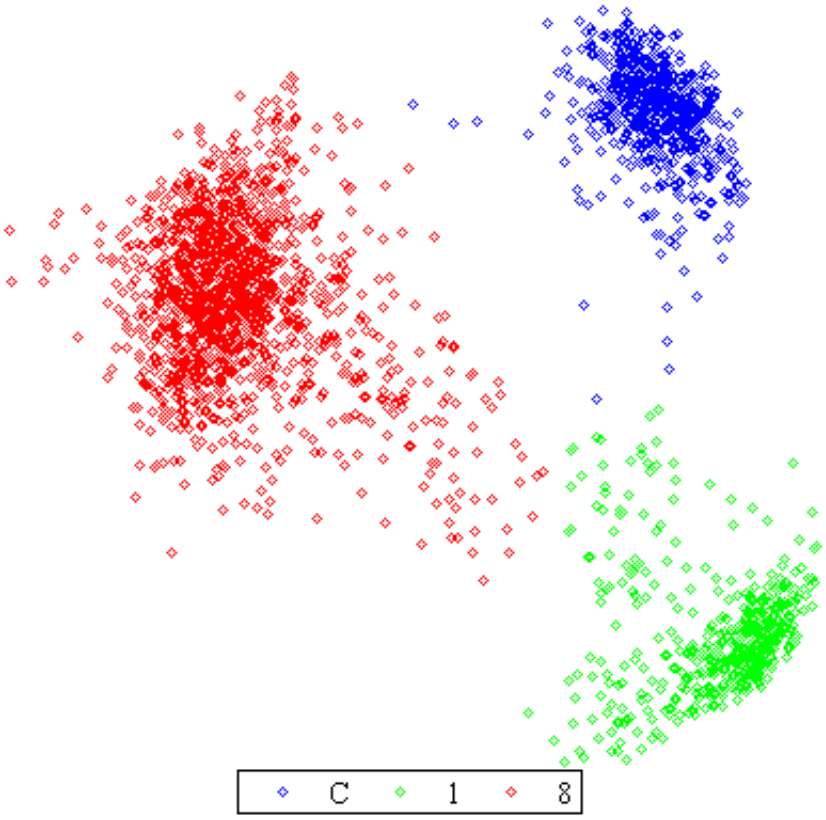


Figure 6.6: 2-dimensional orthogonal projection of points

- A user interface for recognition (ink canvas, HLR, and recognizer).
- A cloud – a web infrastructure that serves as a recognizer (in the remote recognition mode) and as an efficient storage of user-specific training data, allowing access, update, sharing, continuous adaptation of the shapes of clusters, etc. In the current prototype implementation, the back end consists of a web server, an application server, and a DBMS.

Communication between the client application for training and the cloud is performed through sending profiles, i.e. zipped XML documents that contain personal catalogs (clouds of points). The application server communicates with the database through SQL.

6.4.1 Initial Training

In an adaptive recognition environment, the training phase is not required. However, having some number of training samples in each class can significantly improve the initial recognition. Training is normally performed before usage of the application or after introducing a new character to the repository. Once training is finished the profile is synchronized with the cloud.

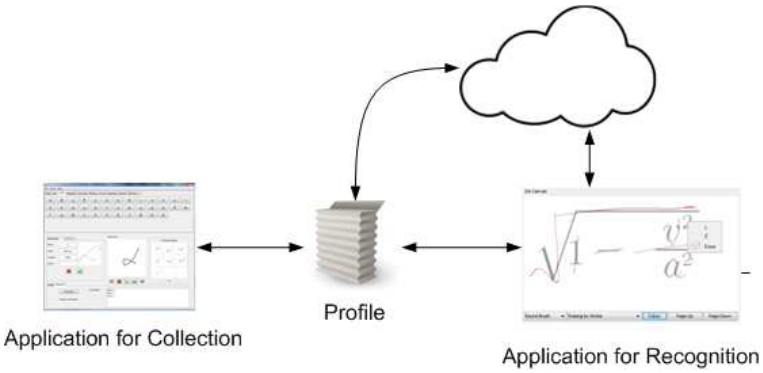


Figure 6.7: Interaction of user interfaces for collection and recognition with the cloud

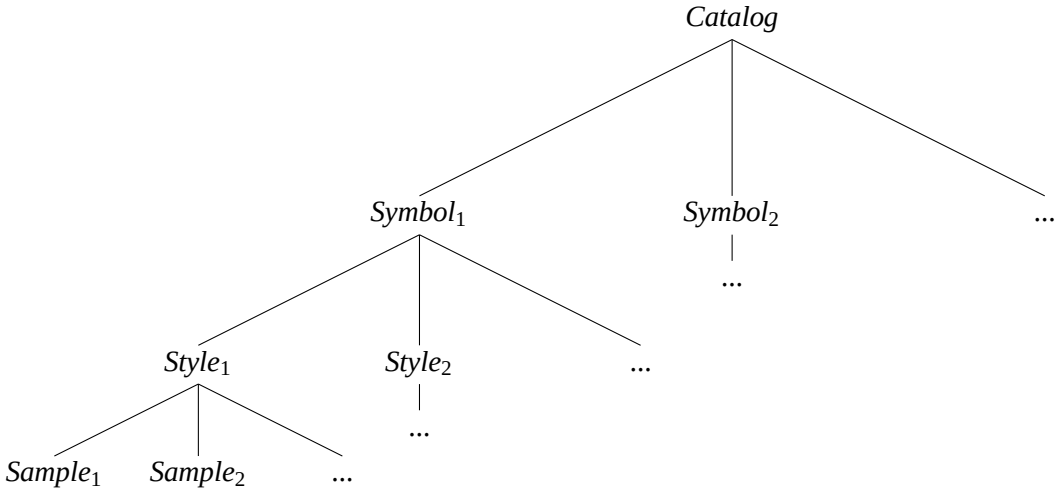


Figure 6.8: The structure of a catalog

A profile is a dataset of training characters used in recognition. The dataset is a collection of catalogs. Each catalog is a hierarchical container of symbols, styles, and samples. Figure 6.8 shows a structure of a catalog where

- *Catalog* is a catalog of related symbols, e.g. Latin characters, digits, mathematical operators, etc.
- $Symbol_i$ is a recognition class, e.g. "*a*", "1" or "±".
- $Style_i$ is a style, i.e. one of the possible ways to write the symbol. Our recognition algorithm is dependent on the direction of writing and the number of pen-ups of a character. For example, symbol *l* can have two styles: one style represents writing the character from the top to the bottom and another style – from the bottom to the top.
- $Sample_i$ is a training sample, written according to the corresponding style.

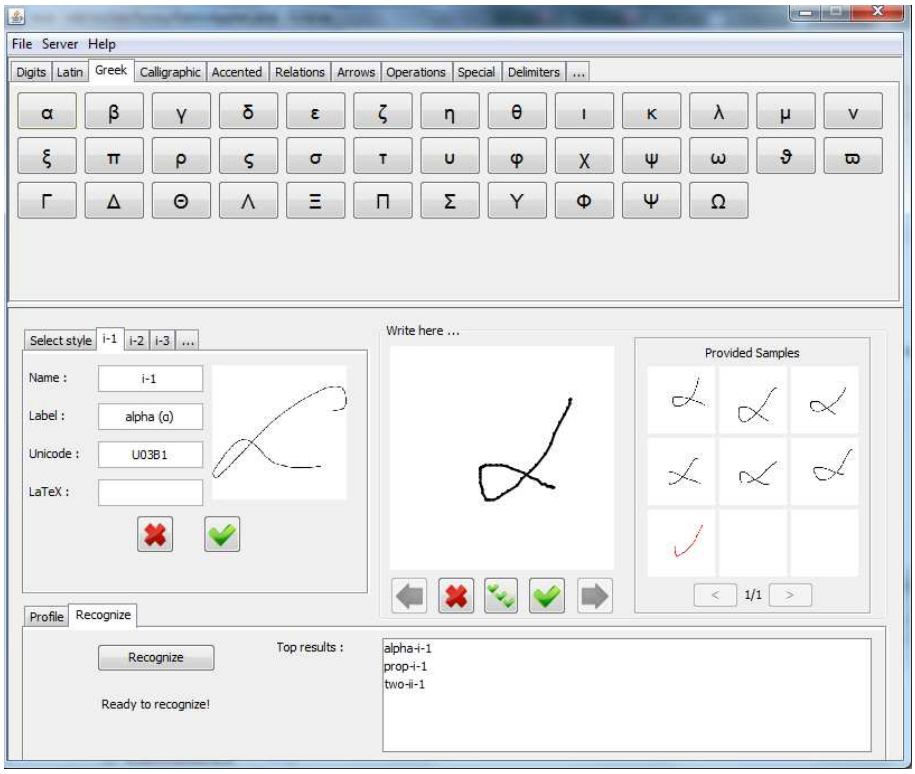


Figure 6.9: The main window of the training application

Each user can have several profiles used together or independently, representing, for example, different areas of mathematics, chemistry or music. *System* profiles should also be available – the default collections of typical symbols, styles, and samples in a domain.

The XML tree of a profile corresponds to the hierarchy of a catalog: It should contain symbols, styles, samples, and coefficients. The normalized coefficients $c_i \in [-1, 1]$ can be compactly stored in a byte variable as $[127c_i]$, where $[x]$ is rounding of x to an integer [26].

6.4.2 Implementation of the Application

For simplicity, our current model is implemented in three-tier architecture. The client applications for collection, recognition, and the application server have been developed in Java. Requests to the application server are routed through a web server.

Client Application for Collection of Characters The front end provides a convenient interface for the user to input and manage training samples. The interface comes along with the structure of the user profile. Specifically, the main window of the application is a tabbed panel with each tab representing a catalog of samples, as shown in Figure 6.9. A tab contains a list of symbols of the catalog. Once the user selects a symbol, the panel with styles becomes

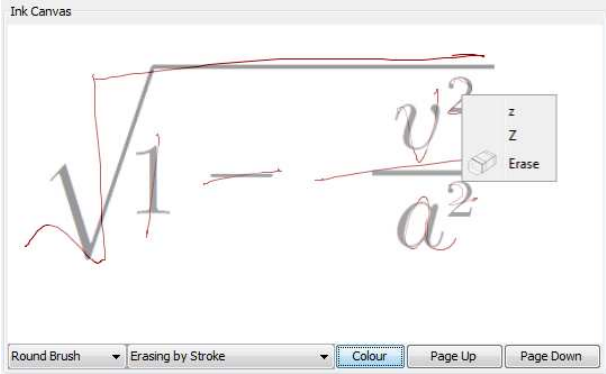


Figure 6.10: Client interface for recognition

available. Styles are shown as animated images for visualization of stroke order and direction. The discussed elements of the interface (catalogs, symbols, styles, and samples) are highly dynamic: A context menu is available that allows to create, to delete or to merge with another element. A profile can be saved on a local hard drive and reopened, as well as synchronized with the server.

Each provided sample should be assigned to a style. If a style has not been selected, it is determined automatically based on its shape and the number of strokes. This recognition is usually of high accuracy, since the candidate classes are styles of the selected symbol and the number of styles is typically small.

The Client Interface for Recognition Classification of handwritten characters takes place when a user performs handwritten input through a separate application. The current implementation is integrated with the InkChat [31], a whiteboard software that facilitates engineering, scientific, or educational pen-based collaboration online. Nevertheless, a number of alternative applications can be used as the recognition front end, e.g. MathBrush [40], a pen-based system for interactive mathematics, or MathInk [67], a mathematical pen-based plug-in that can run inside computer algebra systems, such as Maple [35], or document processing software, such as Microsoft Word.

There can be two approaches to recognition – character-at-a-time (each character is recognized as it is written) and formula-at-a-time (characters are recognized in a sequence, taking advantage of the context and common deformation of samples). Classification results can be displayed super-imposed on the digital ink or replace it. For each entered character, a context menu is available that lists the top recognition candidates, as shown in Figure 6.10. If the user chooses another class from the candidates listed in the context menu, adjacent characters should be reclassified based on the new context information.

(a) (b)

Figure 6.11: (a) A set of provided samples, and (b) the average sample

The Server Side The server side has the following interacting parts: the Apache web server, an application server, and MySQL DBMS. The user uploads a profile to the application server as a zipped file. The profile is unzipped and parsed. Information is inserted in the database.

Upon download of a profile, the process is reversed – the user sends a request to the application server over the web server. The application server selects data from the database, forms an XML profile, performs compression, and sends it to the client.

In the current implementation, a client communicates with the application server over HTTP, but an encrypted communication channel is suggested in a production environment. Furthermore, profiles are recommended to be stored in the database in an encrypted format.

6.4.3 Attractive Display of Recognized Characters

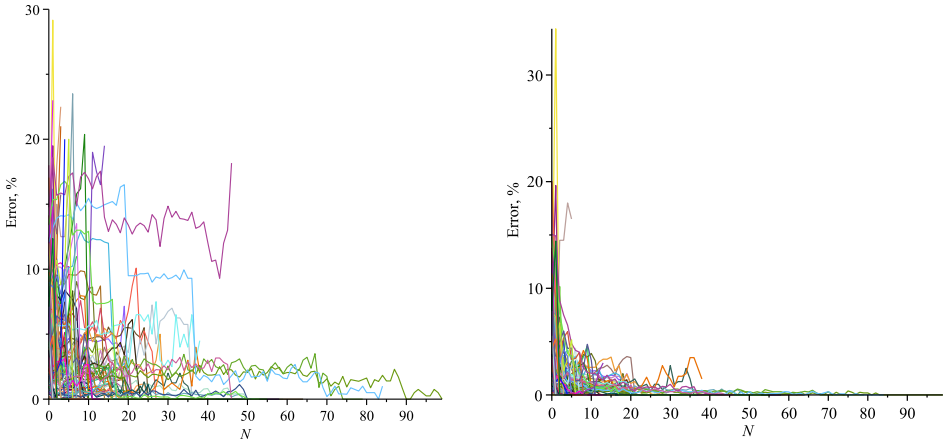
Some research has shown that averaging can be used to make faces look attractive [61]. We adopt a similar approach to generate visually appealing output. The shape of each output stroke is obtained by taking the average of coefficients of approximation of corresponding strokes of samples in the style

$$\overline{c_i} = \frac{\sum_{j=1}^{n} c_{ij}}{n}$$

where $\overline{c_i}$ is the i-th average coefficient of a stroke and n is the number of samples in the style. The traces of the average character are then computed from the average series. This approach allows personalized output, representing samples in a visually appealing form and yet preserving the original style of the writer, as illustrated in Figure 6.14.

6.5 Experimental Evaluation

We describe results of an experiment that shows performance of adaptive author-centered recognition that can be implemented in the cloud infrastructure. The experimental setting aims to simulate decrease in the classification error depending on a user's input size, given that the application is initially trained with a default dataset.



(a) The Null strategy (b) The Basic strategy

Figure 6.12: The average recognition error of the (N+1)-th sample in a class among all classes by an author. All authors are shown in the plot.

6.5.1 Setting

The experimental dataset is identical to the one described in Section 2.10. Further, each sample is assigned to one of the 369 authors. Then for each author, the dataset is split in two parts: samples provided by the author (used in testing) and the rest of the dataset (used in training). A test sample is extracted from a randomly chosen class among those written by the test author and recognized. The recognition error of the N-th sample by the author is computed as the ratio of the number of misrecognitions of the N-th sample to the total number of N-th samples tested. This run is repeated 200 times and the average is reported. We consider two strategies for processing the recognized character

- *Null* strategy: The test sample is disregarded after recognition. This strategy is implemented for comparison with the Basic strategy.

- *Basic* strategy: The test sample is added to the corresponding training class. This facilitates adaptive recognition when the training cluster is adjusted to the style of the current user with each new sample provided.

  The Basic strategy does not provide a mechanism to remove training samples that have negative impact on recognition. In Chapter 5, we developed an adaptive instance-based classifier that assigns a dynamic weight to each training exemplar. If the exemplar participates in a correct (incorrect) classification, the weight is increased (decreased). Samples with the minimal average weight are removed from the dataset.

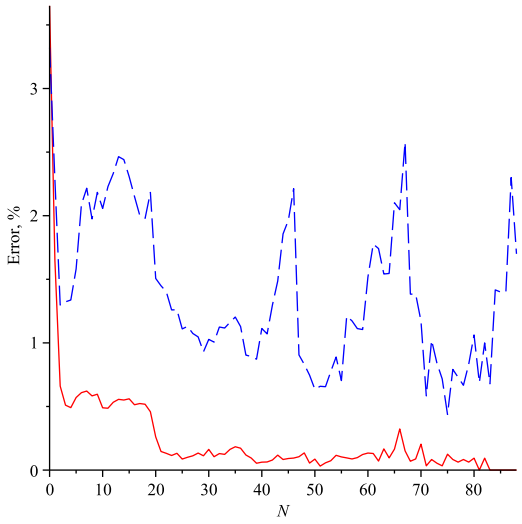


Figure 6.13: The average recognition error among all authors of the (N+1)-th sample in a class for the Basic strategy (solid) and the Null strategy (dash).

6.5.2 Results

Figures 6.12(a) and 6.12(b) demonstrate the average recognition error of the N-th sample in a class among all classes by an author for the Null and the Basic strategies respectively. Authors are shown in the plot in different colors. These figures show that the approach gives consistent results for different authors. The average recognition error among all authors is presented in Figure 6.13 for the Basic and the Null strategies.

On average, the Basic strategy demonstrates improvement over the course of use, which is most noticeable for less than 20 samples in a class by an author. Given that the dataset contains several hundred classes, synchronization of samples across devices is a valuable advantage and can make the recognition workflow efficient and smooth.

6.6 Semi-Automated Training of the Recognizer

Handwriting is believed to be individual, and therefore it has been used as the primary form of authentication for centuries. However, the general shape of handwritten samples may look alike among groups of individuals, especially those that have similar background, e.g. nationality, native language, etc.

The cloud dataset contains samples representing different styles of writing the same character, some of which are likely to be similar to the handwriting of the user. However, the samples that represent handwritten styles different from those of the user make the training dataset noisy and may cause misclassification.

(a) (b)

Figure 6.14: An example of characters written in a similar style (a) “9” and “a” are written clockwise, and (b) “a” and “9” are written counterclockwise

Typically, the classification method that we use does not require many training samples to discriminate a class. However, because there are a large number of classes in handwritten mathematics, the training dataset may contain tens of thousands of characters. Recommendation of styles applicable to the current user can be a valuable asset in this setting.

In this section, we implement semi-automated training of the recognizer by suggesting styles that are likely to be applicable to the handwriting of the user, based on the styles the user has already provided and the styles of writers with similar handwriting. This research is based on the assumption that if a group of users write some characters in the same style, it is likely that they will write certain other characters in the same style as well, see Figure 6.14. We are motivated by the wide and successful usage of recommendation systems on the Internet that are designed to recommend products to consumers, based on their purchasing history and the history of individuals with similar behaviour.

Assume that a user is starting to train the recognizer and he/she would like to reuse the training samples of other users who write characters in the same style. In this setting, all the user needs to do is to identify the styles for corresponding characters. Given that the alphabet of symbols can be extensive, suggestion of styles to the user can save time and efforts during the training phase.

6.6.1 User-Style Similarity

Let $P(S_0|S_1, S_2, ..., S_n)$ be the conditional probability that the character C_0 is written in style S_0 given that we already know a set of other styles provided by the user $S_1, S_2, ..., S_n$. Then for a given character, the style that is suggested to the user at the training phase can be found as

$$\max_{S' \in S} P(S'|S_1, S_2, ..., S_n) \tag{6.2}$$

where S is the set of styles with which the subject character can be written.

The value of Equation 6.2 can be computed with the chain rule

$$P(\cap_{k=1}^{n} S_k) = \prod_{k=1}^{n} P(S_k| \cap_{j=1}^{k-1} S_j)$$

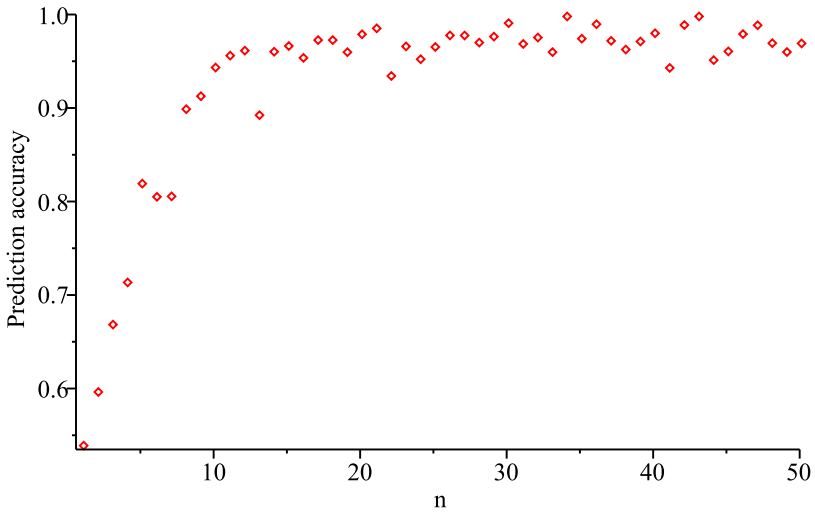


Figure 6.15: The style prediction accuracy

The probability of the user to write n given styles can be recorded as

$$P(\cap_{k=1}^{n} S_k)$$

and computed as the ratio of the number of authors who write each of the corresponding characters in one of the given styles to the total number of authors who provided samples for all of the corresponding classes.

6.6.2 Experimental Evaluation

Our experimental dataset is based on the one described in Section 2.10. Further, each sample is labeled with its style and the author who provided the sample. There are 369 writers in total. The experimental runs are organized as follows. For each author, we randomize the list of styles that the author provided and process the styles in the list. For each style we compute the conditional probability that the corresponding character is written in given style. Figure 6.15 presents the average prediction accuracy among all writers depending on the number of styles n available from the author. From the results we can conclude that once an author provided more than 10 styles, we can predict with high accuracy what corresponding styles the author will be using for other characters. This can speed up the training of a recognizer, because the new writer can simply select the styles that are similar to the styles he or she is using, and use samples from those styles to train the recognizer.

6.7 Conclusion

We have shown how online handwriting recognition systems can take advantage of centralized, cloud-based repositories. Incremental training data, ground truth annotations, and the machine learning framework can usefully reside on a server for the benefit of multiple client devices. We find this particularly effective for symbol sets that occur in mathematical handwriting.

With another meaning of the word “cloud”, our character recognition methods rely on clouds of points in an orthogonal series coefficient space. The representation of these clouds of training and recognition support data is quite compact, allowing collections of data sets to be cached locally even on small devices or transmitted over slow network connections. These clouds can evolve as new data is received by the server, improving recognition. These clouds also provide a simple but effective method for handwriting neatening, by taking an average point for each style.

We find that placing recognition point sets (“clouds” in one sense) in distributed storage and computing environments (“clouds” in another sense) to be a particularly fruitful combination.

We also demonstrated how the cloud data can be used to improve usability of the recognizer by performing semi-automated training based on the data available from other users.

Chapter 7

Factorial Analysis of the Recognition Algorithm

To understand the influence of factors in the algorithm of recognition of sequences of rotated characters, we implement its factorial analysis. This chapter is based on the paper "Pen-Based Computing in Medicine: Factorial Analysis of the Rotation-Invariant Recognition Algorithm" [49] that appeared in the proceedings of the 6th Canadian Student Conference on Biomedical Computing and Engineering.

7.1 Introduction

A 2^k factorial design allows to evaluate performance of a system depending on k factors with each factor taking 2 values. This type of analysis received significant attention due to its simplicity and sufficient power in sorting out factors depending on their impact on performance. A 2^k factorial design can be applied in a setting, when effect of factors is unidirectional – performance decreases or increases continuously while a factor is being changed. Therefore, selecting two significantly different values of a factor and measuring difference in performance is a good starting point in performance evaluation. If the difference in performance is significant enough, a detailed examination may take place. A $2^k r$ factorial design is useful to isolate experimental errors. In this design each of the experiments is repeated r times and it allows to introduce the error term to the model [36].

We propose to apply the factorial analysis technique to investigate recognition of n-grams of characters rotated on approximately the same angle. The difference in rotation between every pair of characters in an n-gram is $\leq 2\beta$ (where β is the noise angle). Having n characters in an n-gram, we look for an angle that allows to minimize likelihood of an error in recognition

of the samples. We define likelihood of recognition error as

$$\gamma_\alpha = \frac{d_\alpha}{\sum_{i=1}^{p} d^i_{min}}$$

$$\gamma_{[\alpha_1;\alpha_2]} = \min_\alpha \{\gamma_\alpha | \alpha \in [\alpha_1; \alpha_2]\}$$

where d_α is the minimal distance of the test sample (rotated on angle α) among the distances to convex hulls of nearest neighbours of all training classes, and $\sum_{i=1}^{p} d^i_{min}$ is the sum of p minimal distances for all angles, e.g.

$$d^1_{min} = \min_\alpha \{d_\alpha | \alpha \in [\alpha_{\min}; \alpha_{\max}]\}$$

$$d^2_{min} = \min_\alpha \left\{d_\alpha | d_\alpha > d^1_{\min} \;\&\; \alpha \in [\alpha_{\min}; \alpha_{\max}]\right\}$$

where $\alpha_{\min} = -\alpha_{\max}$, and angle $\alpha_{\max}$ is one of the parameters in the factorial design.

Total error likelihood of samples in n-gram is computed as

$$\gamma^t_\alpha = \sum_{i=1}^{n} \gamma^i_{[\alpha-\beta;\alpha+\beta]}$$

where $\gamma^i_{[\alpha-\beta;\alpha+\beta]}$ is the minimal likelihood of the i-th sample in the n-gram on $[\alpha - \beta; \alpha + \beta]$.

Having found the rotation angle that yields the minimal error likelihood, we normalize samples with respect to rotation and recognize them with regular techniques, see Chapter 2.

The rest of the chapter is organized as follows. In Section 7.2 we present the results obtained in factorial design: sign table, estimation of experimental error, allocation of variation, confidence intervals for effects, confidence intervals for predicted responses and verification of assumptions. Section 7.3 is devoted to the analysis of error rate depending on the parameter μ in the Legendre-Sobolev inner product and different datasets. Section 7.4 concludes the report.

7.2 A 2^4 Factorial Design with 5 Replications

Sign Table We select the following factors for analysis (corresponding values are given for each of the factors)

- The rotation angle (α): 0.3 and 0.6 radians
- The noise angle (β): 0.0 and 0.1 radians
- The size of the n-gram (n): 3 and 5

e_1	e_2	e_3	e_4	e_5
0.2	-0.3	0.1	-0.1	0.0
0.1	-0.3	0.1	0.1	-0.1
0.4	-0.5	0.0	-0.1	0.3
0.3	-0.5	0.0	0.0	0.1
0.3	-0.2	0.0	-0.1	0.0
0.3	-0.1	-0.1	-0.2	0.1
0.1	0.0	0.0	-0.2	0.1
0.3	0.0	-0.1	-0.2	0.1
0.1	-0.3	0.1	0.2	-0.1
0.1	-0.3	0.1	0.2	-0.1
0.2	-0.4	0.0	0.2	0.0
0.3	-0.4	0.1	0.1	-0.1
0.3	-0.2	0.0	-0.1	0.0
0.3	-0.1	-0.1	-0.1	0.1
0.2	-0.1	-0.1	0.0	0.0
0.2	-0.1	-0.1	-0.1	0.1

Table 7.1: Experimental errors

- The number of distances to consider in $\sum_{i=1}^{p} d_{min}^{i}$ (p): 3 and 5.

We chose to implement a $2^4 5$ factorial design, since we expect all the factors to have significant impact on performance and we would like to perform careful evaluation of the parameters. The number of experiments in the $2^4 5$ factorial design is feasible.

The model has the form, as discussed in [36] (for all arguments of the sum, refer to the sign Table A.1)

$$y = q_0 + q_\alpha x_\alpha + q_\beta x_\beta + q_n x_n + q_p x_p + q_{\alpha\beta} x_\alpha x\beta + ... + q_{\alpha\beta np} x_\alpha x_\beta x_n x_p + e$$

where q's are effects and e is experimental error. Computation of effects is quite intuitive and is represented in Table A.1.

Estimation of Experimental Errors Having computed the effects, we can evaluate the response $\hat{y}_i$ for each combination of factors $(x_{\alpha_i}, x_{\beta_i}, x_{n_i}, x_{p_i})$ as

$$\hat{y}_i = q_0 + q_\alpha x_{\alpha_i} + q_\beta x_{\beta_i} + q_n x_{n_i} + q_p x_{p_i} + q_{\alpha\beta} x_{\alpha_i} x_{\beta_i} + ... + q_{\alpha\beta np} x_{\alpha_i} x_{\beta_i} x_{n_i} x_{p_i}.$$

The experimental errors are computed as the difference between the measured and estimated values $e_{ij} = y_{ij} - \hat{y}_i$. Experimental errors are given in Table 7.1.

$y_1 - \bar{y}_{..}$	$y_2 - \bar{y}_{..}$	$y_3 - \bar{y}_{..}$	$y_4 - \bar{y}_{..}$	$y_5 - \bar{y}_{..}$
-0.4	-0.9	-0.5	-0.7	-0.6
-0.3	-0.8	-0.4	-0.3	-0.5
-0.9	-1.9	-1.4	-1.4	-1.1
-0.6	-1.4	-0.9	-0.9	-0.8
1.1	0.6	0.8	0.7	0.8
1.1	0.7	0.7	0.6	0.9
0.6	0.5	0.5	0.3	0.6
0.9	0.6	0.6	0.4	0.7
-0.3	-0.8	-0.4	-0.3	-0.6
-0.3	-0.7	-0.3	-0.2	-0.5
-0.7	-1.3	-0.9	-0.8	-1.0
-0.4	-1.1	-0.5	-0.6	-0.7
1.1	0.6	0.8	0.7	0.8
1.1	0.7	0.7	0.7	1.0
0.9	0.6	0.6	0.7	0.7
1.0	0.7	0.7	0.7	0.9

Table 7.2: Values of $y_{ij} - \bar{y}_{..}$

Allocation of Variation The total variation or the total sum of squares is computed as

$$SST = \sum_{ij} (y_{ij} - \bar{y}_{..})^2$$

where $\bar{y}_{..}$ is the average response for all replications of all combinations of factors. Values of $y_{ij} - \bar{y}_{..}$ are represented in Table 7.2.

SST can be divided into parts as

$$SST = SS_\alpha + SS_\beta + SS_n + SS_p + ... + SS_{\alpha\beta np} + SSE$$

where $SS_\alpha = 2^k r q_\alpha^2$, etc.

Values for percentage variation are presented in Table 7.3

Confidence intervals for Effects The standard deviation of errors, σ_e, and terms, σ_{terms}:

$$\sigma_e = \sqrt{\frac{SSE}{2^k(r-1)}}, \sigma_{terms} = \frac{\sigma}{\sqrt{2^k r}}.$$

Confidence intervals, computed as $q_i \mp t \cdot \sigma_{terms}$, are presented in Table 7.4. From the table we can conclude with 95% confidence that all of the factors are significant and some of the interactions are insignificant.

Parameters	Variation
α	1.06%
β	86.08%
n	3.99%
p	1.00%
α and β	0.16%
α and n	0.35%
β and n	0.95%
α and p	0.03%
n and p	0.27%
β and p	0.23%
α, β and n	0.03%
α, β and p	0.03%
α, n and p	0.02%
β, n and p	0.04%
α, β, n and p	0.00%
Error	5.76%

Table 7.3: Percentage variation

Confidence Intervals for Predicted Responses We compute confidence intervals for responses for combinations of factors. The standard deviation of the mean response, depending on the number of replications, m, is

$$s_{\hat{y}_m} = s_e(\frac{1}{n_{eff}} + \frac{1}{m})^{1/2},$$

where n_{eff} is the effective number of degrees of freedom (DFs) computed as

$$n_{eff} = \frac{\text{total number of runs}}{1 + \text{sum of DFs of parameters used in } \hat{y}}.$$

Confidence intervals for $m = 1, m = 5$ and $m = \infty$ are given in Table 7.5.

Verification of Assumptions The expressions for effects shown above are based on the following assumptions:

1. Errors are statistically independent.

2. Errors are Normally Distributed.

3. Errors have constant standard deviation.

These assumptions are validated through visual tests.

Confidence Intervals	lower	upper	Significance
α	95.30	95.40	Significant
β	-0.13	-0.03	Significant
n	-0.77	-0.68	Significant
p	0.11	0.20	Significant
α and β	-0.13	-0.03	Significant
α and n	-0.08	0.02	Insignificant
β and n	0.00	0.09	Insignificant
α and p	0.03	0.12	Significant
n and p	-0.06	0.03	Insignificant
β and p	-0.01	0.09	Insignificant
α, β and n	-0.08	0.01	Insignificant
α, β and p	-0.03	0.06	Insignificant
α, n and p	-0.06	0.03	Insignificant
β, n and p	-0.04	0.06	Insignificant
α, β, n and p	-0.03	0.06	Insignificant

Table 7.4: Confidence intervals for the factors and interactions

Independent Errors We want to make sure that the errors are independently and identically distributed in the model. We plot residuals versus the predicted response. The scaterlpot is given in Figure 7.1. The scaterplot testifies that our assumption is correct, since there is no visible trend in the points.

Normally distributed errors To verify this assumption, we build normal quantile-quantile plot of residuals, shown in Figure 7.2. As we can observe from the plot, the graph is approximately linear, which testifies that the assumption is correct.

Constant standard deviation of errors (homoscedasticity) We analyze the scaterplot 7.1, see section 14.7 of [36], and look if the spread of points in different parts of the graph is significantly different. We observe that in fact the points are distributed approximately the same and, therefore, we conclude that the errors satisfy the assumption of constant standard deviation.

Exper.	$\hat{y}$	Low(1)	High (1)	Low (5)	High (5)	Low (∞)	High (∞)
1	94.73	94.26	95.19	94.46	95.00	94.53	94.92
2	94.88	94.42	95.34	94.61	95.15	94.69	95.08
3	94.01	93.55	94.48	93.74	94.28	93.82	94.21
4	94.43	93.96	94.89	94.16	94.70	94.23	94.62
5	96.14	95.68	96.60	95.87	96.41	95.95	96.33
6	96.14	95.68	96.61	95.87	96.41	95.95	96.34
7	95.84	95.37	96.30	95.57	96.11	95.64	96.03
8	95.99	95.53	96.46	95.72	96.26	95.80	96.19
9	94.87	94.41	95.33	94.60	95.14	94.68	95.06
10	94.95	94.49	95.41	94.68	95.22	94.76	95.14
11	94.42	93.96	94.89	94.15	94.69	94.23	94.62
12	94.70	94.24	95.17	94.43	94.97	94.51	94.90
13	96.14	95.68	96.61	95.87	96.41	95.95	96.34
14	96.20	95.74	96.67	95.93	96.47	96.01	96.40
15	96.03	95.57	96.49	95.76	96.30	95.83	96.22
16	96.13	95.67	96.59	95.86	96.40	95.94	96.33

Table 7.5: Confidence intervals for $m = 1, m = 5$ and $m = \infty$

7.3 Evaluation of the Parameter μ in the Legendre-Sobolev Inner Product

7.3.1 Experimental Error

To estimate the jet scale μ in the Legendre-Sobolev inner product, we implement the model in the form of two-factor full factorial design without replications. We run the experiments on the original dataset of samples (samples are not subjected to any transformation intentionally). Parameter μ is evaluated as follows. We first evaluate the average error rate among all datasets for $\mu = \frac{1}{16}$, $\mu = \frac{1}{8}$ and $\mu = \frac{1}{4}$. Having found that $\mu = \frac{1}{16}$ performs slightly better than other alternatives, we evaluate error for $\mu = \frac{1}{32}$, $\mu = \frac{3}{32}$. We observe $\mu = \frac{1}{32}$ to perform slightly better and therefore we evaluate error for $\mu = \frac{1}{64}$ and $\mu = \frac{3}{64}$. Both of the last two alternatives perform worse than $\mu = \frac{1}{32}$ and therefore we stop evaluation. Error rate for different values of μ for different datasets and the average error are given in Table 7.6.

7.3.2 Two-Factor Full Factorial Design

Introduction The factors considered are

1. Parameter μ (denoted as A and effects as α) with the number of observations $a = 9$.

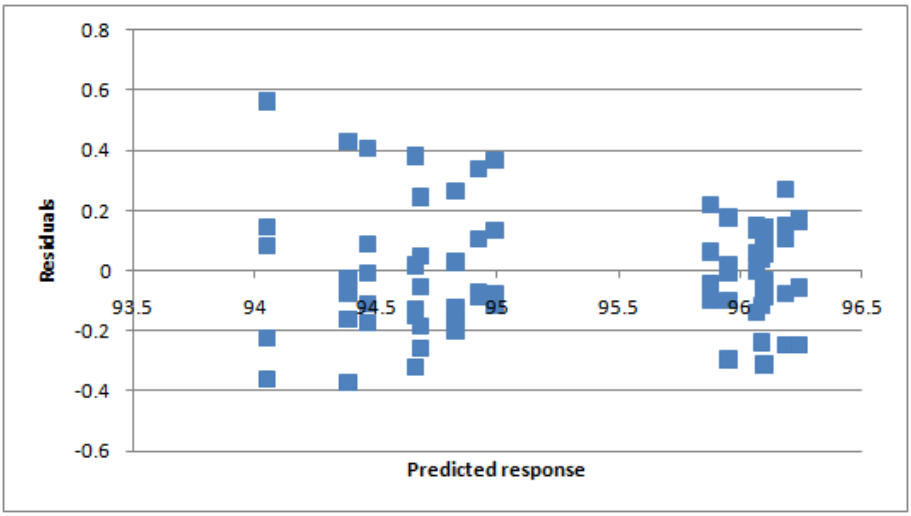


Figure 7.1: Scatter plot of residuals versus the predicted response

Dataset \ μ	$\frac{1}{64}$	$\frac{1}{32}$	$\frac{3}{64}$	$\frac{1}{16}$	$\frac{3}{32}$	$\frac{1}{8}$	$\frac{3}{16}$	$\frac{1}{4}$	$\frac{5}{16}$
1	2.12	2.16	1.93	2.12	2.10	2.24	2.36	2.32	2.34
2	2.47	2.57	2.49	2.55	2.61	2.47	2.55	2.57	2.59
3	2.63	2.63	2.67	2.63	2.55	2.65	2.77	2.75	2.79
4	2.56	2.34	2.40	2.42	2.44	2.50	2.60	2.58	2.62
5	2.44	2.34	2.60	2.72	2.68	2.82	2.76	2.80	2.78
6	2.51	2.22	2.32	2.26	2.24	2.34	2.36	2.40	2.36
7	2.54	2.37	2.43	2.37	2.43	2.41	2.48	2.58	2.64
8	2.47	2.56	2.52	2.54	2.52	2.54	2.68	2.73	2.79
9	2.90	2.69	2.75	2.77	2.77	2.75	2.77	2.73	2.75
10	2.67	2.53	2.40	2.50	2.50	2.46	2.52	2.46	2.38
mean	2.53	2.44	2.45	2.49	2.48	2.52	2.58	2.59	2.60

Table 7.6: Recognition error for different values of μ for different datasets

2. Datasets (denoted as B and effects β) with the number of observations $b = 10$.

Variation of the factors is shown in Table 7.6.

According to [36], the additive model for a two-factor design without replications is

$$y_{ij} = \bar{y}_{..} + \alpha_j + \beta_i + e_{ij}$$

where y_{ij} is the observation in the experiment with the first factor α being at level j and the second factor β being at level i, $\bar{y}_{..}$ is the mean response, α_j is the effect of factor α at level j, β_i is the effect of factor β at level i, and e_{ij} is the error term.

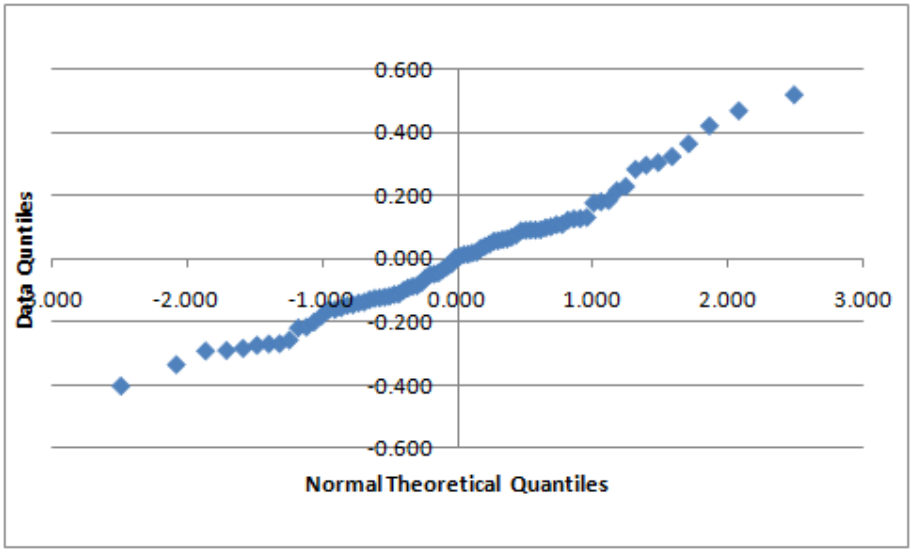


Figure 7.2: Normal quantile-quantile plot for residuals

μ	$\frac{1}{64}$	$\frac{1}{32}$	$\frac{3}{64}$	$\frac{1}{16}$	$\frac{3}{32}$	$\frac{1}{8}$	$\frac{3}{16}$	$\frac{1}{4}$	$\frac{5}{16}$
$\bar{y}_{\cdot j}$	2.53	2.44	2.45	2.49	2.48	2.52	2.58	2.59	2.60
α_j	0.01	-0.08	-0.07	-0.03	-0.04	0.00	0.06	0.07	0.08

Table 7.7: $\bar{y}_{\cdot j}$ and α_j values

Computation of Effects We find $\bar{y}_{\cdot\cdot}$ as the mean observation for all combinations of α and β: Then α_j and β_i are found as

$$\alpha_j = \bar{y}_{\cdot j} - \bar{y}_{\cdot\cdot}, \beta_j = \bar{y}_{i\cdot} - \bar{y}_{\cdot\cdot}$$

Values for $\bar{y}_{\cdot j}$ and α_j, $\bar{y}_{i\cdot}$ and β_i are given in Tables 7.7 and 7.8 respectively.

Experimental error Estimated response is computed as

$$\hat{y}_{ij} = \bar{y}_{\cdot\cdot} + \alpha_j + \beta_i.$$

Then experimental errors, see Table 7.9, are found as

$$e_{ij} = y_{ij} - \hat{y}_{ij}.$$

Allocation of Variation The following formula takes place

$$SSY = SS0 + SSA + SSB + SSE$$

Datasets	1	2	3	4	5	6	7	8	9	10
$\bar{y_{i\cdot}}$	2.19	2.54	2.67	2.50	2.66	2.33	2.47	2.60	2.77	2.49
β_i	-0.33	0.02	0.15	-0.02	0.14	-0.19	-0.05	0.07	0.24	-0.03

Table 7.8: $\bar{y_{i\cdot}}$ and β_i values

$i \backslash j$	1	2	3	4	5	6	7	8	9
1	-0.08	0.05	-0.19	-0.04	-0.05	0.06	0.11	0.06	0.07
2	-0.08	0.11	0.02	0.04	0.11	-0.07	-0.05	-0.04	-0.03
3	-0.05	0.03	0.07	-0.01	-0.09	-0.02	0.03	0.01	0.03
4	0.06	-0.08	-0.03	-0.04	-0.02	0.01	0.04	0.01	0.04
5	-0.23	-0.24	0.01	0.09	0.06	0.16	0.04	0.07	0.04
6	0.17	-0.03	0.06	-0.04	-0.06	0.01	-0.04	-0.01	-0.06
7	0.06	-0.02	0.03	-0.07	-0.01	-0.06	-0.05	0.04	0.08
8	-0.14	0.05	0.00	-0.02	-0.03	-0.05	0.02	0.07	0.11
9	0.13	0.01	0.06	0.04	0.04	-0.01	-0.06	-0.11	-0.10
10	0.17	0.12	-0.02	0.04	0.04	-0.03	-0.04	-0.10	-0.19

Table 7.9: Experimental error

where

$$SSY = \sum_{ij} y_{ij}^2, SS0 = ab\bar{y}_{\cdot\cdot}^2, SSA = b\sum_j \alpha_j^2, SSB = a\sum_i \beta_i^2, SSE = \sum_{ij} e_{ij}^2$$

Percentage of variation explained by factors A (parameter μ in LS inner product), B (different datasets) and error are 9.47, 72.90 and 17.63 respectively. Therefore, selection of the dataset has the greatest impact on performance.

Analysis of Variance We compute the degrees of freedom for various sums as follows

$$SSY = SS0 + SSA + SSB + SSE$$

$$ab = 1 + (a-1) + (b-1) + (a-1)(b-1)$$

Then we compute the mean squares as

$$MSA = \frac{SSA}{a-1}, MSB = \frac{SSB}{b-1}, MSE = \frac{SSE}{(a-1)(b-1)}$$

Then the F-ratios to test the significance of factors A and B are

$$F_A = \frac{MSA}{MSE}, F_B = \frac{MSB}{MSE}$$

Effect	Mean Effect	Stand. Dev.	Lower Bound	Upper Bound
μ	2.52	0.01	2.50	2.54
A				
$\frac{1}{64}$	2.53	0.26	2.01	3.06
$\frac{1}{32}$	2.44	0.26	1.92	2.97
$\frac{3}{64}$	2.45	0.26	1.93	2.98
$\frac{1}{16}$	2.49	0.26	1.96	3.01
$\frac{3}{32}$	2.48	0.26	1.96	3.01
$\frac{1}{8}$	2.52	0.26	1.99	3.04
$\frac{3}{16}$	2.58	0.26	2.06	3.11
$\frac{1}{4}$	2.59	0.26	2.07	3.12
$\frac{5}{16}$	2.60	0.26	2.08	3.13
B				
1	2.19	0.25	1.69	2.69
2	2.54	0.25	2.04	3.04
3	2.67	0.25	2.17	3.17
4	2.50	0.25	2.00	3.00
5	2.66	0.25	2.16	3.16
6	2.33	0.25	1.83	2.84
7	2.47	0.25	1.97	2.97
8	2.60	0.25	2.09	3.10
9	2.77	0.25	2.26	3.27
10	2.49	0.25	1.99	2.99

Table 7.10: Confidence intervals for effects

Both factors are significant.

Confidence intervals for effects The standard deviation is computed as

$$s_e = \sqrt{MSE}, s_\mu = \frac{s_e}{\sqrt{ab}}, s_{\alpha_j} = s_e\sqrt{\frac{a-1}{ab}}, s_{\beta_i} = s_e\sqrt{\frac{b-1}{ab}}$$

The confidence intervals are presented in Table 7.10.

7.3.3 Visual Verification of Assumptions

We verify the following assumptions

Independent Errors The plot of residuals versus the predicted response is given in Figure 7.3. The scaterplot testifies that our assumption is correct, since there is no visible trend in the points.

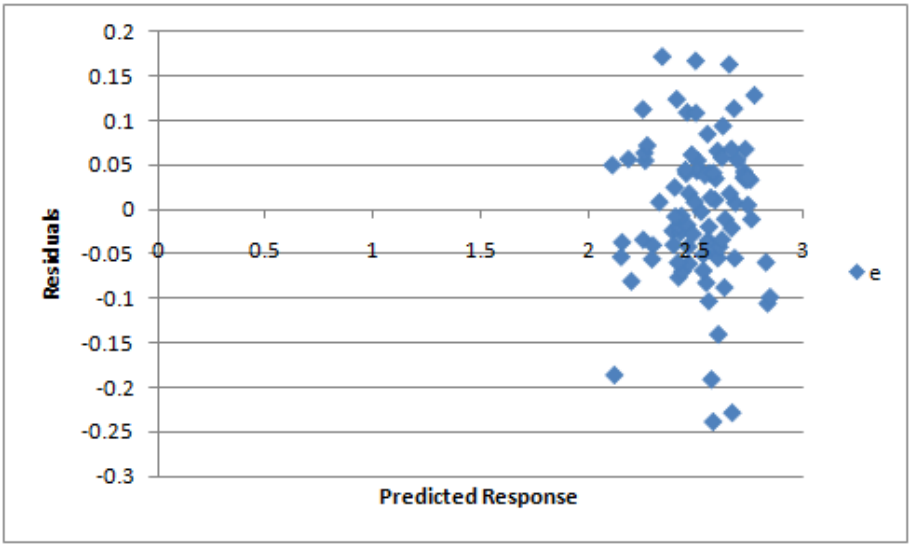


Figure 7.3: Scatter plot of residuals versus the predicted response

Normally distributed errors To verify this assumption, we build normal quantile-quantile plot of residuals, shown in Figure 7.4. As we observe from the plot, the graph is approximately linear, which testifies that the assumption is correct.

Constant standard deviation of errors (homoscedasticity) From the scaterplot 7.3 we observe that spread of the points is approximately the same and, therefore, the errors have approximately constant standard deviation.

7.4 Conclusion

In Section 7.2 we performed an investigation of performance of the recognition algorithm of rotated characters in an n-gram depending on 4 factors: angle of rotation, noise, n and the number of samples in computation of error likelihood. The experimental runs were repeated five times to estimate the error. We observed that the noise angle has the most effect on performance and causes about 86% in variation of the recognition rate, then comes the value of n, causing 4% variation, and then rotation angle causing only 1% of variation. These results demonstrate that the algorithm is highly invariant to rotation. We also determined with 95% confidence that all of the factors are significant, but some interactions are not. The confidence intervals for the predicted response for different values of m (the number of replications): 1, 5 and ∞ were shown as well. Finally, we presented visual verification of assumptions that are in the core of the factorial design.

In Section 7.3 we focused on evaluation of the classification error of the recognition algo-

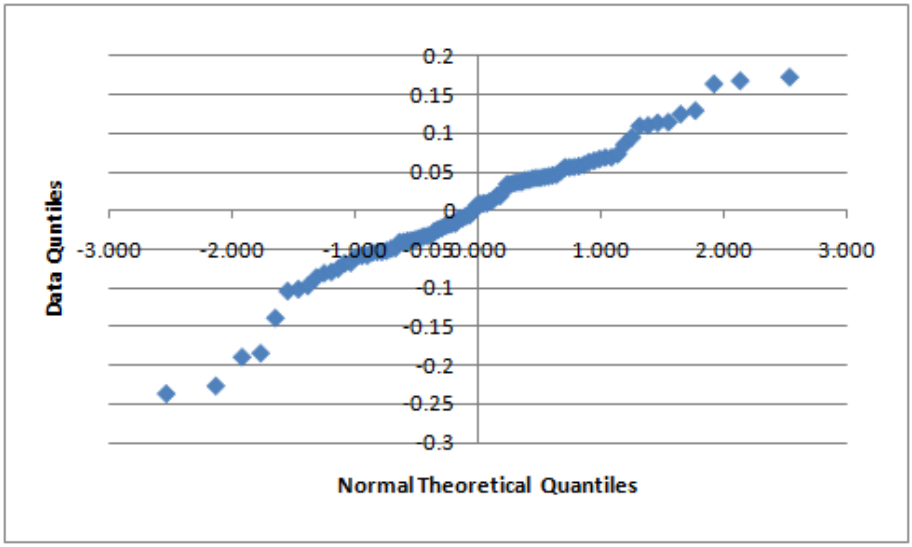


Figure 7.4: Normal quantile-quantile plot for residuals

rithm for different values of μ and different datasets. We computed allocation of variation and discovered that selection of the dataset has larger impact on performance than the jet scale μ. In the analysis of variance we found both of the factors (μ values and datasets) to be significant. We also computed confidence intervals for effects and performed visual verification of the assumptions.

The results obtained are valuable for better understanding of the character classification algorithm and the influence of the configuration parameters on its performance.

Chapter 8

Linear Compression of Digital Ink via Point Selection

Each year more devices support digital ink in some form, for capture, processing or recognition. These devices have a wide range of form factors and resources, from small hand-held devices to digital whiteboards. These devices are used in various configurations, individually, tethered for a single user, or in multi-party collaboration. Various vendor-specific binary formats are used to represent digital ink, and there is the vendor-neutral XML format InkML. With this increased use of digital ink, its efficient handling has become increasingly important. Small devices need to be able to handle it efficiently. On more powerful devices, ink-handling applications may need to store a significant amount of model data to support recognition [50].

In addition, the sampling frequency and spatial resolution of hardware has been increasing over time, creating opportunities and challenges for ink processing applications. The opportunities are associated with the possibility of more detailed analysis, since a device can capture in high precision variations of pen movement. On the other hand, such high volumes of ink data require extra resources for processing and storage.

We take the view that lossless compression at time of ink capture is not a meaningful objective as each ink capture device has a resolution limit and sampling accuracy. So long as the reconstructed curve lies within these limits, lossy and lossless compression are equivalent. For our own applications involving recognition, lossless compression has no benefit. Small perturbations in strokes give symbols that a human reader would recognize as the same [50].

This chapter is based on the paper “Linear Compression of Digital Ink via Point Selection” [52] co-authored with Stephen M. Watt, that appeared in the 10th IAPR International Workshop on Document Analysis Systems.

8.1 Introduction

We present a method to compress digital ink based on piecewise-linear approximation within a given error threshold. The objective is to achieve good compression ratio with very fast execution. The method is designed and especially effective on types of handwriting that have large portions with nearly linear parts, e.g. hand drawn geometric objects. In simple terms, this problem is solved by removing points that do not affect the shape of the curve significantly, while the error between the original and the approximating curves remains within a threshold. The method can be viewed as a dynamic adjustment of the density of points, depending on the shape of a stroke. More points are removed from straighter regions than regions with high curvature. Thus, we would expect geometric drawings with many lines to compress particularly well. We compare this method with an enhanced version of our earlier functional approximation method, finding the new technique to give slightly worse compression while performing significantly faster. This suggests the presented method can be used in applications where speed of processing is of higher priority than the compression ratio.

We have two subproblems that need to be solved:

1. decomposition of digital ink into pieces, suitable for compression, and

2. compression of the individual pieces.

We present fast, easy to implement solutions to both of these problems and show experimentally that the technique yields good compression for handwritten text and even better compression for hand drawn geometric objects. The discussed method is most useful for compression of linear pieces of a curve and can be implemented as a part of a multipurpose hybrid compression algorithm.

We also implement an enhanced version of the compression method described in Section 2.9 by representing coefficients in a more compact form. We measure the compression rate and time required to process the experimental datasets and compare with the performance of the linear method. While losing in compression, the linear method is found to perform more than 100× faster.

The chapter is organized as follows. An improvement to the functional approximation method is proposed in Section 8.3. The linear compression algorithm is explained in Section 8.4. Section 8.5 presents details about the experimental setting and the results obtained. Section 8.6 concludes the chapter.

8.2 Related Work

8.2.1 Digital ink compression

A number of digital ink compression algorithms have been developed to date. One of the most popular lossless schemes is to use second differences [59].

An efficient lossy method was developed in [50]. In this work we also present results of compression with the industry standard – the second difference method. This was based on piecewise functional approximation of curves by truncated orthogonal polynomial series and representation of the pieces by the approximating series coefficients. The desired approximation accuracy is achieved by dynamically changing the degree of approximation and the size of pieces.

Another lossy algorithm was presented in [46], based on stroke simplification. It suggests to eliminate excessive points, forming a skeleton of the original curve. The algorithm is based on iterative computation of chordal deviation – the distance between the original curve and its approximation. Points with the minimal distance are removed until the distance becomes larger than a threshold. A "substantially lossless" method was proposed in [10]. It allows the compression error's magnitude to be not greater than the sampling error's magnitude. In this approach, the original curve is split into segments and each segment is represented by some predefined shape, such as a polygon, ellipse, rectangle or Bezier curve. It is not mentioned how to obtain the shapes from a curve and what compression this approach gives.

A method for selection of the minimal number of points to represent a curve within an error bound was proposed in [32]. The technique is based on dynamic programming and has linear complexity in the number of points selected.

8.2.2 Approximation of univariate convex functions

Several "sandwich" algorithms have been proposed for approximation of univariate convex functions. For example, see [19] for a method that requires derivative information along with the function values, and [65] for an iterative algorithm when only function values are available. The latter technique can be briefly described as follows. Consider a convex function defined on an interval I and some threshold of approximation error δ. Approximation on the interval is obtained by joining its boundary points. Let the approximation error of the interval I be δ_I and $\delta_I > \delta$. Then the interval I is split into subintervals, according to a partitioning rule. The procedure is repeated until the approximation error becomes less than δ for each subinterval. Several partitioning rules are considered, e.g. the maximal error rule that selects the point located on the maximal distance to the approximation curve. The algorithm converges quadratically if

certain conditions on derivatives are satisfied, and linearly under other conditions.

8.2.3 Decomposition of digital curve in inflection-free parts

Several methods exist for decomposition of digital curves in segments without inflection, e.g. see [13, 14]. However, these algorithms are primarily designed for digital images to extract convex/concave pieces of an object to determine meaningful parts. In contrast, we are interested in the decomposition of digital ink. We note that the methods developed for binary images are in most cases not suitable for our purpose, since digital ink is represented as a sequence of points on a curve, rather than as a field of pixels in two dimensions.

8.3 Enhanced Compression via Functional Approximation

We propose a way to improve the functional approximation technique developed in [50]. As mentioned earlier, that method is based on piecewise approximation of curves by truncated series in an orthogonal polynomial basis. In [50] we experimented with Chebyshev, Legendre, Legendre-Sobolev polynomials and Fourier series. It was found empirically that Chebyshev polynomials yield the best compression, as reported in [50]. In the present work our goal is to improve performance of the method with Chebyshev polynomials as the orthogonal basis. The improvement is to be achieved by representing coefficients in a more compact form.

We consider the adaptive segmentation scheme of [50]. For each trace, the degree d of the approximation is selected dynamically. A higher degree provides a more accurate approximation of a curve, but increases the compressed size. In the adaptive scheme, the size of coefficients is also selected for each trace independently. Coefficients are recorded as floating-point numbers with base 2. The significand and the exponent are two's complement binary integers, since this is the representation that is most often used to represent integers in computing devices. The significand is encoded in a bits, while the exponent is in p bits. The value of p is fixed, and the value of a is dynamically adjusted for each stroke. The following representation of each information channel of a trace i is proposed:

- Encode the 0 order coefficient in $2a + p$ bits, since this coefficient regulates the initial position of the trace and is typically larger than the rest of the coefficients. This number of bits is device-dependent and for the test device this value appears appropriate with respect to the maximal non-zero coefficients occurred.

- Find the coefficient $c_M = \max |c_i|, i = 1..d$ and encode it in $a + p$ bits.

- Encode coefficients c_j, $j = 1..d$, as two's complement binary integers $r_j = \left\lfloor \frac{|c_M|}{c_j} \right\rceil$ in b_r bits, where $\lfloor x \rceil$ represents rounding of x to the integer.

Thus, a trace i is recorded as

$$a_i d_i \lambda_1 c_{10} c_{1M} r_{11} ... r_{1d_i} \lambda_2 c_{20} c_{2M} r_{21} ... r_{2d_i} ... \lambda_D$$

where a_i is the number of bits for encoding the significand; d_i is the degree of approximation; λ_j is the initial value of parameterization of a piece j; c_{j0} is the 0-order coefficient; $c_{jM} = \max |c_{jk}|, k = 1..d$; $r_{jk} = \left\lfloor \frac{|c_{jM}|}{c_{jk}} \right\rceil$, c_{jk} is the k-th coefficient of the j-th piece. This differs from the method of [50] by having the coefficients c_j represented as scalings rounded to integers rather than as significand-exponent pairs.

8.4 The Linear Compression Algorithm

8.4.1 Decomposition into Inflection-Free Parts

The method described in [65] is not suitable for digital ink as originally presented, since it requires parameterization and segmentation. We develop a method that does not require parameterization and can be used as the first step in processing.

Our compression method works with pieces locally curving in one direction or the other, but not changing back and forth. To be more precise, the curve should be decomposed into parts where the second derivative has constant sign, i.e the normal vector in the Frenet frame is pointing to the same side of the curve.

Definition We say that a sequence of points $(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)$ is an *inflection-free segment* if and only if the polygon formed by these points, after joining (x_1, y_1) and (x_n, y_n), is convex.

The property of a convex polygon that every internal angle is less than or equal to π is used in the *online* decomposition Algorithm 8. The algorithm, in the body of the **while** loop, lists operations performed on each incoming ink point to obtain a sequence of inflection-free segments. This takes into account that

- Two points are considered equal, if their coordinates are equal.
- $|P|$ denotes the number of points in the list P.

Algorithm 8 FormInflectionFreeSegments()

Input: *Points* – a stream of input points
Output: C – a list of inflection-free segments

```
C ← [ ] {list of inflection-free segments found}
S ← [ ] {current segment being collected}
i ← 0 {index of current point without duplication}
while Points.hasNext() do
  P ← Points.getNext()
  if i = 0 or P ≠ P_{i−1} then
    P_i ← P
    if |S| ≥ 2 then
      if P_i = P_0 then
        Append the list S to the end of the list C
        S ← [ ]
      else
        A_i   ← Angle(P_{i−2}, P_{i−1}, P_i) − π
        A_Beg ← Angle(P_i, P_0, P_1) − π
        A_End ← Angle(P_{i−1}, P_i, P_0) − π
        if A_i × A_{i−1} < 0
        or A_i × A_End < 0 or A_Beg × A_End < 0 then
          Append the list S to the end of the list C
          S ← [ ]
        end if
      end if
      Append P_i to the end of the list S
      i ← i + 1
    end if
  end if
end while
If S is non-empty, append it to the end of the list C
return C
```

- Angle(P, Q, R) is the "oriented" angle between vectors $\overrightarrow{QP}$ and $\overrightarrow{QR}$. In other words, Angle$(P, Q, R) = 2\pi -$ Angle(R, Q, P). These angles can be found with the dot and cross products of given vectors.

- A_{Beg} is the complement of the oriented angle made by the beginning vector $\overrightarrow{P_0P_1}$ and the last point. A_{End} is the complement of the oriented angle made by the ending vector $\overrightarrow{P_{i-1}P_i}$ and the first point. A_i is the complement of the oriented angle made by the most current three points.

- We test for products less than zero to detect changes in direction of curvature. Two angles in the same direction will give a positive product (either as $+ \times +$ or $- \times -$) and three

collinear points will give a zero product.

8.4.2 Compression of Inflection-Free Parts

Once the curve is decomposed as a collection of inflection-free segments, each piece is a subject to compression. Our compression technique is similar to the sandwich algorithm proposed in [65]. However, rather than looking at the lower and upper bounds of a function, we find the distance between a curve and its approximation. If either the maximal error $\|\cdot\|_{\max}$ or the root mean square error $\|\cdot\|_{\text{rms}}$ on an interval is greater than the respective thresholds $\epsilon_{\max}$ or ϵ_{rms}, the curve is split into two parts. Other norms on the space of curves could be used if desired. The steps are presented in Algorithm 9, considering that j.first and j.last are respectively the first and the last points of the interval j.

Definition We write pw(L) for the piecewise linear curve defined by the list of points L. If two points a and b occur in a list L, with a preceding b, then we say that $[a, b]$ is an *interval* in L. We write $L|I$ for the sublist of L restricted to the interval I.

The point of division is found with one of the partitioning rules:
Rule 1: Based on the maximal *distance*: the decomposition point is selected based on the distance from the point to the line that goes through the boundary points of the interval.
Rule 2: Based on the *angle* formed at the point: if all of the oriented angles within the segment are less than π then the minimal angle is considered, otherwise (when all of the angles are greater than π) the maximal angle is found.

8.4.3 Complexity

The decomposition algorithm processes each incoming point in constant time $O(1)$. There are no additional operations at the last input. It is online, in that after each point a valid decomposition is maintained.

The best case time complexity of compression of a piece is $O(n)$. If the splits always divide a segment into two equal parts, and the algorithm continues until there is a split at every point, the cost is $O(n \log n)$. If the splits are made unequally, always splitting n points as 1 and $n-1$, then the cost is $O(n^2)$.

8.4.4 Correctness

The termination condition of CompressCurve merits attention. If a function satisfies a maxnorm bound on each element of a partition, then it satisfies the maxnorm over the union of the

Algorithm 9 CompressCurve(S, R)

Input: S – a list of points for an inflection-free segment
R – a partitioning rule (rule 1 or 2)
Output: L – a list of points such that
$\|\text{pw}(S) - \text{pw}(L)\|_{\max} < \epsilon_{\max}$ and
$\|\text{pw}(S) - \text{pw}(L)\|_{\text{rms}} < \epsilon_{\text{rms}}$

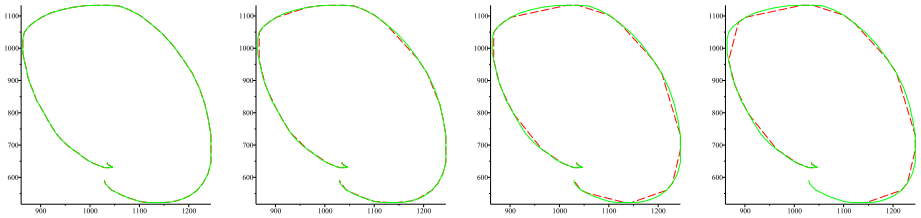
{J is a stack of intervals to be refined.}
$J \leftarrow$ [Interval with first and last point of S]
$L \leftarrow$ [ ]
while $J \neq$ [ ] **do**
  $j \leftarrow$ Pop an interval from J
  $a \leftarrow j$.first; $b \leftarrow j$.last
  if $\|\text{pw}(S|j) - \text{pw}(j)\|_{\max} > \epsilon_{\max}$
  or $\|\text{pw}(S|j) - \text{pw}(j)\|_{\text{rms}} > \epsilon_{\text{rms}}$ **then**
    {Split j according to rule R at some point c in S}
    $j_1 \leftarrow [a, c]$
    $j_2 \leftarrow [c, b]$
    Push j_2 and then j_1 onto the stack J
  else
    Append a and then b to the end of list L
  end if
  Remove element j from J
end while
return L

parts. For RMS, note that if a domain D is partitioned as $D_1, ..., D_n$ and $\sqrt{\sum_{a\in D_i} f(a)/|D_i|} < \epsilon$, then $(\sum_{a\in D_1} + \cdots + \sum_{a\in D_n}) f(a) < (|D_1| + \cdots |D_n|)\epsilon^2$ so $\sqrt{\sum_{a\in D} f(a)/|D|} < \epsilon$, and take $f(a) = (S(a) - S^*(a))^2$.

8.4.5 Discussion

Binary Encoding of Points The sequence of points of a compressed trace can be encoded in binary for compact representation. Coordinates in our dataset have absolute value not greater than 2^{13} and can be recorded as two's complement integers in a sequence of groups of 14 bits.

Drifting of Approximation The presented compression method is not suitable for repeated resampling. While the approximation to each inflection-free segment will lie within any required error bound, the approximation will lie completely on one side of the input curve. If the resulting piecewise linear function is then resampled and recompressed repeatedly, systematic drift may occur. To address the issue of drift under repetitive resampling and recompression, the line segments could be positioned to cross the original curve so that the error is equal on both sides of the original.



$\epsilon_{\max} = 1$ $\epsilon_{\max} = 5$ $\epsilon_{\max} = 10$ $\epsilon_{\max} = 15$

Figure 8.1: Approximation of a sample with different error thresholds (dash line) and the original curve (solid line)

8.5 Experiments

8.5.1 Experimental Setting

The experimental dataset was collected in the Ontario Research Centre for Computer Algebra with a tablet device with the following specifications: 2540 dpi resolution, 133 pps data rate, and ±.02 sampling error.

Two types of digital ink were collected for the experiments

- Handwriting. Different individuals have provided various parts of regular English text to ensure variations in length of strokes and writing styles. From the whole collection, we randomly selected 46 traces containing, on average, 51 points each. This number of traces is feasible for the adaptive functional approximation method, that requires significant amount of time.

- Geometric objects. We collected simple two-dimensional geometric objects, such as triangles, rectangles and lines. Then we randomly selected 33 traces containing, on average, 68 points each in order to achieve feasible running time of the functional approximation compression algorithm.

In the experiments, the root mean square error was taken heuristically as a portion of the maximal error $\epsilon_{\text{rms}} = \frac{3}{4}\epsilon_{\max}$. Unlike the results reported in [50], we look at the absolute, not relative, approximation error and the binary stream of coefficients does not undergo further gzip compression. The compressed size is reported as S_c/S_o where S_c is the size of the compressed dataset and S_o is the size of the original dataset.

The compression algorithms were implemented and run on Maple 13 on an Intel Core 2 Duo 2.40 GHz CPU with 2GB RAM, running Ubuntu Linux version 2.6.24-19-generic.

Table 8.1: Compressed size (%) as a function of the maximal error ($\epsilon_{\max}$) and the number of exponent bits (p) for 7 coefficient bits (b_r) for the handwriting dataset

p \ $\epsilon_{\max}$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
3	21.5	14.7	12.5	11.6	10.3	9.6	9.0	8.5	8.1	7.9	7.6	7.3	7.1	6.9	6.7
4	19.2	13.2	11.3	10.3	9.2	8.7	8.1	7.6	7.3	7.1	6.8	6.5	6.3	6.2	6.0
5	19.0	13.6	11.7	10.8	9.5	8.9	8.3	7.9	7.5	7.3	7.0	6.7	6.5	6.4	6.2

Table 8.2: Compressed size (%) as a function of the maximal error ($\epsilon_{\max}$) and the number of exponent bits (p) for 7 coefficient bits (b_r) for the dataset of geometric objects

p \ $\epsilon_{\max}$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
3	22.3	17.2	15.0	13.7	12.5	11.7	10.9	10.5	9.8	9.5	8.9	8.7	8.4	8.1	7.8
4	20.6	15.0	13.0	11.9	10.8	10.0	9.3	9.0	8.5	8.2	7.7	7.6	7.3	7.1	6.8
5	21.8	16.2	13.9	12.8	11.6	10.8	9.9	9.6	9.1	8.6	8.2	8.0	7.7	7.4	7.1

Table 8.3: Compressed size (%) as a function of the maximal error ($\epsilon_{\max}$) and the number of coefficient bits (b_r) for 4 exponent bits (p) for the handwriting dataset

b_r \ $\epsilon_{\max}$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
4	20.3	14.6	12.2	11.0	9.7	8.7	8.2	7.5	7.2	7.0	6.6	6.3	6.1	5.9	5.8
5	18.6	13.5	11.4	10.0	9.0	8.4	7.6	7.2	7.0	6.7	6.5	6.2	6.0	5.8	5.6
6	18.5	13.0	11.1	10.0	8.9	8.3	7.8	7.4	7.0	6.8	6.6	6.3	6.1	5.9	5.8
7	19.2	13.2	11.3	10.3	9.2	8.7	8.1	7.6	7.3	7.1	6.8	6.5	6.3	6.2	6.0
8	19.1	13.6	11.7	10.7	9.6	9.0	8.4	7.9	7.6	7.4	7.1	6.8	6.6	6.5	6.3

8.5.2 Experimental Results

Optimal values of p and b_r In the experiments we measure compressed size for different values of approximation error. Figure 8.1 shows an original curve and linear approximation for different maximal error thresholds. From the figure, one can observe that compressing the curve with the maximal error of up to 5 has almost no effect on representation of the curve and can be used in the applications that do not require high precision of ink, e.g. recognition.

In the first set of experiments, we look for the optimal values of p and b_r in a simplified manner, see Section 8.3. With fixed $b_r = 7$, the value of p was changed and the compressed size was measured for both datasets. Results for handwriting and geometric objects are shown in Tables 8.1 and 8.2 respectively. The value of $p = 4$ was found to be the most efficient. With fixed $p = 4$, b_r was changed to find the optimal value. The compressed sizes for the datasets of handwriting and geometric objects are shown in Tables 8.3 and 8.4 respectively. The value of $b_r = 5$ was selected. In a production setting, a more detailed evaluation, possibly a grid evaluation, should be performed.

One can observe that the compression rate as a function of p or b_r is not changing monotonically, which can be explained by the fact that large values of p or b_r may result in large approximation chunks and vice versa.

Table 8.4: Compressed size (%) as a function of the maximal error ($\epsilon_{\max}$) and the number of coefficient bits (b_r) for 4 exponent bits (p) for the dataset of geometric objects

b_r \ $\epsilon_{\max}$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
4	19.5	14.6	12.8	11.6	10.6	9.8	9.1	8.8	8.2	7.9	7.4	7.2	6.9	6.7	6.3
5	19.5	14.4	12.4	11.4	10.4	9.7	9.0	8.7	8.0	7.7	7.3	7.1	6.8	6.6	6.3
6	20.2	14.7	12.6	11.6	10.5	9.9	9.1	8.8	8.3	7.9	7.5	7.3	7.1	6.8	6.6
7	20.6	15.0	13.0	11.9	10.8	10.0	9.3	9.0	8.5	8.2	7.7	7.6	7.3	7.1	6.8
8	20.2	14.7	12.6	11.6	10.5	9.9	9.1	8.8	8.3	7.9	7.5	7.3	7.1	6.8	6.6

Table 8.5: Time (in seconds) for compression of the handwriting dataset

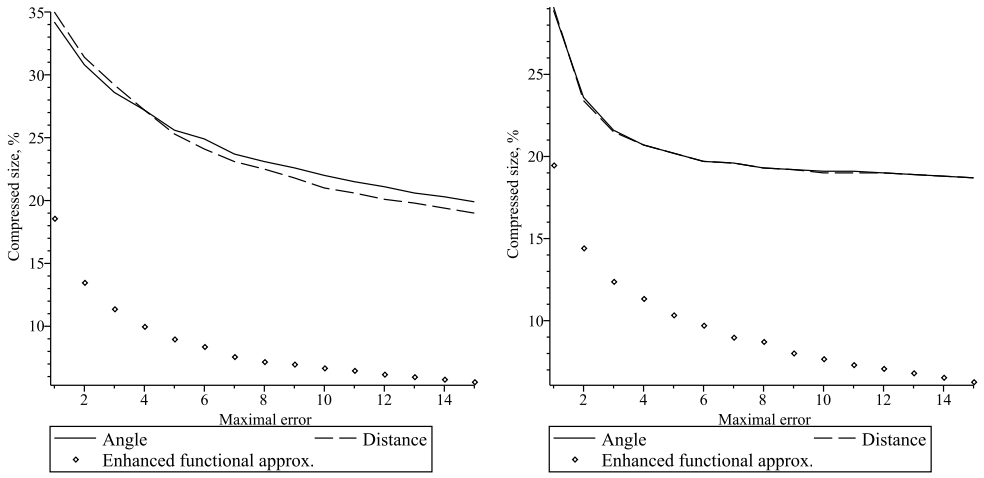
Method \ $\epsilon_{\max}$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
L	25	20	21	21	17	17	17	19	18	15	16	15	15	20	16
F	879	1083	1287	1498	1700	1982	2188	2326	2479	2618	2727	2915	3019	3138	3327

Table 8.6: Time (in seconds) for compression of the dataset of geometric objects

Method \ $\epsilon_{\max}$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
L	12	10	8	9	8	7	8	8	9	9	9	8	8	9	8
F	1188	1355	1781	2034	2185	2346	2475	2593	2710	2830	2980	3086	3180	3281	3333

Comparison of functional approximation with linear compression The compression rate of the linear method was measured for the two segmentation rules explained in Section 8.4.2 on both datasets. Figure 8.2 presents the results of the functional approximation and linear compression methods for different values of $\epsilon_{\max}$. The partitioning rules show similar performance on the handwriting dataset and almost identical on geometric objects. As expected, due to the nature of the linear algorithm, we obtained higher compression of geometric objects than handwritten text. The functional approximation method shows similar performance on both datasets.

The compression time is given in Table 8.5 for the dataset of handwriting and Table 8.6 for geometric objects. The linear method performs almost instantly, compared to the compression with higher-order functional approximation. One can observe a trend of increase of the execution time of the functional approximation technique with the increase of the error threshold. In fact, the running time is around three times higher for $\epsilon_{\max} = 15$ compared to the execution time for $\epsilon_{\max} = 1$. This growth arises because more combinations of approximation degree and number of coefficient bits become suitable for approximation of pieces. Evaluation of those combinations is computationally intensive and can require significantly more time for high resolution devices.



Handwriting Geometric objects

Figure 8.2: Compressed size depending on the maximal approximation error for handwriting and geometric objects: for Rule 1 (maximal distance) and Rule 2 (based on the angle), and for enhanced functional approximation

8.6 Conclusion

We have examined two methods for the compression of digital ink or, more generally, sampled curves in any dimension. One method selects a subset of the sample points to give a piecewise linear function that is within a given tolerance of the original. The second method adapts previous work based on orthogonal series approximation, representing the coefficients more efficiently. Our experiments show the piecewise linear approximation method to perform about 100× faster than the functional approximation algorithm, but it yields a less compact representation. The proposed piecewise linear compression technique can be used when simplicity or speed are important, such as for hardware implementation and data transmission. On the other hand, the functional approximation method is suitable for applications that require compact storage of ink. Depending on the application and the choice of functional basis, in this representation certain recognition operations may be performed without decompression.

Chapter 9

Conclusion

9.1 Retrospective

The theoretical and experimental results reported in this thesis are aimed to enhance performance of handwriting recognition systems based on orthogonal approximation of curves by making them more robust, more accurate and adaptive. We achieve compact representation of ink for efficient processing, transmission and storage. The presented results form a valuable asset to developers of frameworks for manipulation and recognition of digital ink. These contributions can be naturally integrated in the cloud environment. Some ideas can also be considered as the basis for cloud-based classification systems in other pattern recognition and machine learning domains, where public knowledge is useful for improving individual performance. In particular, we make the following contributions.

(1) We perform optimization of recognition of isolated characters by finding the value of the jet scale in the Legendre-Sobolev inner product and the degree of approximation of character strokes that result in the lowest classification error. We also propose an algorithm for recognition of groups of rotated characters, taking advantage of the natural habit of humans to write symbols with similar degree of transformation. This algorithm can be extended to shear and more general affine transformations.

(2) We propose methods for robust classification of samples of substantially smaller size. We propose a unit of measurement of the size and apply this measurement to classification algorithms based on the size only. We also propose a recognition method based on adjustment of the distance to convex hulls of nearest neighbours for classes with small samples.

(3) We develop an adaptive classification method. This method is applicable to the cloud environment since the training dataset of a user is constantly evolving and synchronized with

the cloud. All the pen-based devices used by the writer get the updates from the cloud, thereby making the continuous training of the application efficient. We show that the adaptive method allows significant asymptotic improvement of the author-centered classification rate.

(4) We describe the cloud-based architecture that is designed to simplify sharing of training data across devices and may incorporate other contributions of the thesis.

(5) We also perform factorial analysis of the algorithm for recognition of groups of rotated characters to analyze the impact of the configuration parameters of the algorithm. We also study the influence of parameter μ on recognition algorithm, concluding that it is relatively not influential.

(6) Finally, we propose and empirically estimate an enhancement to the compression algorithm, based on approximation of strokes with orthogonal polynomials. We also develop a fast compression scheme based on piecewise linear representation of curves.

9.2 Future Work

For the future work, we recognize the importance of developing an algorithm for recognition of mathematical expressions in the cloud environment. Besides the challenges associated with recognition of two-dimensional formulas, an important question posed – which tasks should be delegated to the cloud and which are to be computed on the client. The computational tasks may include segmentation of strokes into characters, recognition, spatial analysis of recognized characters, building the expression tree and analyzing recognition confidence.

The next interesting problem is syntactic and semantic verification of recognized formulas in the cloud. The main problem encountered is the absence of a fixed dictionary of “words” or “set” of rules that controls the evolution of existing words.

Study of applicability of the cloud-based recognition architecture to other pattern-recognition domains is probably the primary direction for further research. We believe that the future of recognition systems is in the cloud due to the number of advantages that it offers, related to storage of training data, collection of feedback, easy enhancement of the classification algorithm, reliability of service.

Bibliography

[1] Cascading style sheets (css) snapshot 2010, May 2011. W3C Working Group.

[2] David W. Aha, Dennis Kibler, and Marc K. Albert. Instance-based learning algorithms. In *Machine Learning*, pages 37–66, 1991.

[3] A. Ali, S. Gilani, and N. Memon. Affine invariant contour descriptors using independent component analysis and dyadic wavelet transform. *Journal of Computing and Information Technology*, 16(3):169–181, 2008.

[4] Lisa Anthony, Jie Yang, and Kenneth R. Koedinger. Evaluation of multimodal input for entering mathematical equations on the computer. In *CHI '05 extended abstracts on Human factors in computing systems*, CHI EA '05, pages 1184–1187, New York, NY, USA, 2005. ACM.

[5] Homayoon S.M. Beigi, Krishna Nathan, Gregory J. Clary, and Jayashree Subrahmonia. Size normalization in on-line unconstrained handwriting recognition. In *Proc. IEEE Int'l Conf. Acoustics, Speech and Signal Processing*, pages 169–172, 1994.

[6] Bora Beran, Catharine Ingen, and Dennis Robert Fatland. Sciscope: a participatory geoscientific web application. *Concurrency and Computation: Practice and Experience*, 22(17):2300–2312, 2010.

[7] Anne-Laure Bianne, Christopher Kermorvant, and Laurence Likforman-Sulem. Context-dependent hmm modeling using tree-based clustering for the recognition of handwritten words. In *Proc. of the Document Recognition & Retrieval XVII*, 2010.

[8] Kam-Fai Chan and Dit-Yan Yeung. Mathematical expression recognition: a survey. *IJDAR*, 3(1):3–15, 2000.

[9] Bruce W. Char and Stephen M. Watt. Representing and characterizing handwritten mathematical symbols through succinct functional approximation. In *Proc. International Con-*

ference on Document Analysis and Recognition, (ICDAR), pages 1198–1202, Curitiba, Brazil, September 2007. IEEE Computer Society.

[10] Manjirnath Chatterjee. System and method for ink or handwriting compression. *United States Patent No US 6,549,675 B2*, April 2003.

[11] Yi-Min Chee, Max Froumentin, and Stephen Watt. Ink markup language (InkML). `http://www.w3.org/TR/InkML/`. (valid on June 6, 2013).

[12] Y. Chen and X. Ye. Projection onto a simplex. *Arxiv preprint arXiv:1101.6081*, 2011.

[13] Isabelle Debled-Rennesson, Jean-Luc Remy, and Jocelyne Rouyer-Degli. Detection of the discrete convexity of polyominoes. In *Proceedings of the 9th International Conference on Discrete Geometry for Computer Imagery*, DGCI '00, pages 491–504, London, UK, 2000. Springer-Verlag.

[14] H. Dorksen-Reiter and I. Debled-Rennesson. Convex and concave parts of digital curves. In Reinhard Klette, Ryszard Kozera, Lyle Noakes, and Joachim Weickert, editors, *Geometric Properties for Incomplete data*, pages 145–159. Springer Netherlands, 2006.

[15] R.F.H. Farag. Word-level recognition of cursive script. *Computers, IEEE Transactions on*, C-28(2):172 –175, feb. 1979.

[16] S. Feng, I. Kogan, and H. Krim. Classification of curves in 2d and 3d via affine integral signatures. *Acta Applicandae Mathematicae*, pages 903–937, March 2010.

[17] Jon Ferraiolo, Fujisawa Jun, and Dean Jackson. *Scalable vector graphics (svg) 1.1 specification*. W3C, January 2003.

[18] Jan Flusser and Tomas Suk. Character recognition by affine moment invariants. In *Proc. of the 5th International Conference on Computer Analysis of Images and Patterns*, pages 572–577. Springer-Verlag, 2007.

[19] B. Fruhwirth, R. E. Burkard, and G. Rote. Approximation of convex curves with application to the bicriterial minimum cost flow problem. *European Journal of Operational Research*, 42:326–338, 1989.

[20] Oleg Golubitsky, Vadim Mazalov, and Stephen M. Watt. Orientation-independent recognition of handwritten characters with integral invariants. In *Proc. Joint Conference of ASCM 2009 and MACIS 2009: Asian Symposium of Computer Mathematics and Mathematical Aspects of Computer and Information Sciences, (ASCM 2009)*, pages 252–261,

Fukuoka, Japan, December 2009. COE Lecture Note Vol. 22, Kyushu University, ISSN 1881-4042.

[21] Oleg Golubitsky, Vadim Mazalov, and Stephen M. Watt. Toward affine recognition of handwritten mathematical characters. In *Proc. International Workshop on Document Analysis Systems, (DAS 2010)*, pages 35–42, Boston, USA, June 9-11 2010. ACM Press.

[22] Oleg Golubitsky, Vadim Mazalov, and Stephen M. Watt. An algorithm to compute the distance from a point to a simplex. *ACM Communications in Computer Algebra*, 46(2-180):57–57, June 2012.

[23] Oleg Golubitsky and Stephen M. Watt. Online stroke modeling for handwriting recognition. In *Proc. 18th Annual International Conference on Computer Science and Software Engineering, (CASCON 2008)*, pages 72–80, Toronto, Canada, October 2008. IBM Canada.

[24] Oleg Golubitsky and Stephen M. Watt. Online computation of similarity between handwritten characters. In *Proc. Document Recognition and Retrieval XVI, (DRR 2009)*, volume 7247, pages C1–C10, San Jose, California USA, January 2009. SPIE and IS&T.

[25] Oleg Golubitsky and Stephen M. Watt. Online recognition of multi-stroke symbols with orthogonal series. In *Proc. 10th International Conference on Document Analysis and Recognition, (ICDAR 2009)*, pages 1265–1269, Barcelona, Spain, July 2009. IEEE Computer Society.

[26] Oleg Golubitsky and Stephen M. Watt. Distance-based classification of handwritten symbols. *International Journal of Document Analysis and Recognition*, (doi 10.1007/s10032-009-0107-7), 2010.

[27] Isabelle Guyon, Lambert Schomaker, Rkjean Planiondon, Mark Liberman, and Stan Janet. Unipen project of on-line data exchange and recognizer benchmarks. In *Proc. 12th International Conference on Pattern Recognition (ICPR 1994)*, pages 29–33, Jerusalem, Israel, 1994. IAPR-IEEE.

[28] P. Hart. The condensed nearest neighbor rule (corresp.). *Information Theory, IEEE Transactions on*, 14(3):515 – 516, may 1968.

[29] Chun Lei He, Ping Zhang, Jianxiong Dong, Ching Y Suen, and Tien D Bui. The role of size normalization on the recognition rate of handwritten numerals. *The 1st IAPR TC3 NNLPAR*, 1:1–5, 2001.

[30] M.K. Hu. Visual pattern recognition by moment invariants. *IRE Transactions on Information Theory*, 8(2):179–187, 1962.

[31] Rui Hu. Portable implementation of digital ink: Collaboration and calligraphy. Master's thesis, The University of Western Ontario, London, Ontario, Canada, 2009.

[32] Rui Hu and Stephen M. Watt. Optimization of point selection on digital ink curves. In *ICFHR*, pages 527–532, 2012.

[33] David A. Huffman. A method for the construction of minimum-redundancy codes. In *Proceedings of the I.R.E.*, pages 1098–1102, September 1952.

[34] Qiang Huo and Tingting He. A minimax classification approach to hmm-based online handwritten chinese character recognition robust against affine distortions. In *Ninth International Conference on Document Analysis and Recognition (ICDAR 2007)*, pages 1226–1230. IEEE Computer Society, July 2007.

[35] Maplesoft Inc. Maple 13 user manual. Technical report, Maplesoft, 2009.

[36] Raj Jain. *The art of computer systems performance analysis*. John Wiley and Sons, Inc, 1991.

[37] X. Jiang and H. Yu. SVM-JAVA: A java implementation of the SMO (sequential minimal optimization) for training SVM, 2008.

[38] B.S. Kashin and A.A. Saakyan. *Orthogonal Series*. Translations of Mathematical Monographs. American Mathematical Society, 2005.

[39] T. Kohonen. *Self-organization and associative memory*. Springer-Verlag New York, Inc. New York, NY, USA, 1989.

[40] George Labahn, Scott Maclean, Mirette Marzouk, Ian Rutherford, and David Tausky. A preliminary report on the MathBrush pen-math system. In *Maple 2006 Conference*, pages 162–178, 2006.

[41] Bart Lamiroy, Daniel Lopresti, Hank Korth, and Jeff Heflin. How carefully designed open resource sharing can help and expand document analysis research. In *Document Recognition and Retrieval XVIII - DRR 2011*, volume 7874, San Francisco, United States, January 2011. SPIE.

[42] Joseph J. LaViola Jr. Symbol recognition dataset. Technical report, Microsoft Center for Research on Pen-Centric Computing.

[43] C. J. Leggetter and P. C. Woodland. Maximum likelihood linear regression for speaker adaptation of continuous density hidden markov models. *Computer Speech & Language*, 9(2):171 – 185, 1995.

[44] C. Liu, K. Nakashima, H. Sako, and H. Fujisawa. Handwritten digit recognition: investigation of normalization and feature extraction techniques. *Pattern Recognition*, 37:265–279, 2004.

[45] Cheng-Lin Liu, Masashi Koga, Hiroshi Sako, and Hiromichi Fujisawa. Aspect ratio adaptive normalization for handwritten character recognition. In *Proc. of the Third International Conference on Advances in Multimodal Interfaces*, ICMI '00, pages 418–425, London, UK, 2000. Springer-Verlag.

[46] Zicheng Liu, Henrique S. Malvar, and Zhengyou Zhang. System and method for ink or handwriting compression. *United States Patent No US 7,302,106 B2*, November 2007.

[47] Scott MacLean, George Labahn, Edward Lank, Mirette Marzouk, and David Tausky. Grammar-based techniques for creating ground-truthed sketch corpora. *Int. J. Doc. Anal. Recognit.*, 14:65–74, March 2011.

[48] Vadim Mazalov. Geometric techniques for digital ink. Master's thesis, University of Western Ontario, 2012.

[49] Vadim Mazalov, Dmitry Mazalov, and Anna Pauer. Pen-based computing in medicine: Factorial analysis of the rotation-invariant recognition algorithm. In *Proc. of the 6th Canadian Student Conference on Biomedical Computing and Engineering*, pages pp.85–89. University of Western Ontario, 2011.

[50] Vadim Mazalov and Stephen M. Watt. Digital ink compression via functional approximation. *Proc. of International Conference on Frontiers in Handwriting Recognition.*, pages 688–694, 2010.

[51] Vadim Mazalov and Stephen M. Watt. Improving isolated and in-context classication of handwritten characters. pages 82970B–82970B–8, 2012.

[52] Vadim Mazalov and Stephen M. Watt. Linear compression of digital ink via point selection. In *Proceedings of the 2012 10th IAPR International Workshop on Document Analysis Systems*, DAS '12, pages 429–434, Washington, DC, USA, 2012. IEEE Computer Society.

[53] Vadim Mazalov and Stephen M. Watt. A structure for adaptive handwriting recognition. In *ICFHR*, pages 692–697, 2012.

[54] Vadim Mazalov and Stephen M. Watt. Writing on clouds. In *Proceedings of the 11th international conference on Intelligent Computer Mathematics*, CICM'12, pages 402–416, Berlin, Heidelberg, 2012. Springer-Verlag.

[55] Vadim Mazalov and Stephen M. Watt. Recognition of relatively small handwritten characters or "size matters". In *Frontiers in Handwriting Recognition (ICFHR), 2012 International Conference on*, pages 319–324, Sept.

[56] M. Mercimek, K. Gulez, and T.V. Mumcu. Real object recognition using moment invariants. *Sadhana*, 37(6):765–775, 2005.

[57] Paul Mermelstein and Murray Eden. Experiments on computer recognition of connected handwritten words. *Information and Control*, 7(2):255 – 270, 1964.

[58] C Michelot. 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):195–200, July 1986.

[59] Microsoft Inc. *Ink serialized format specification*.

[60] R. Mukundan and K. R. Ramakrishnan. *Moment Functions in Image Analysis: Theory and Applications*. World Scientific, 1998.

[61] D.I. Perrett, K.A. May, and S. Yoshikawa. Facial shape and judgments of female attractiveness. *Nature*, 368:239–242, March 1994.

[62] J. Platt. Sequential minimal optimization: A fast algorithm for training support vector machines. *Advances in Kernel Methods-Support Vector Learning*, 208:98–112, 1999.

[63] A. Rosenthal, J. Hu, and M. Brown. Size and orientation normalization of on-line handwriting using hough transform. In *Proc. of the 1997 IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP '97) -Volume 4 - Volume 4*, ICASSP '97, pages 3077–, Washington, DC, USA, 1997. IEEE Computer Society.

[64] Jr. Sammon, J.W. A nonlinear mapping for data structure analysis. *IEEE Transactions on Computers*, page 401409, 1969.

[65] A. Y. D. Siem, D. den Hertog, and A. L. Hoffmann. A method for approximating univariate convex functions using only function value evaluations. *INFORMS J. on Computing*, 23:591–604, October 2011.

[66] Slate Corporation. *Jot - a specification for an ink storage and interchange format*, May 1996.

[67] Elena Smirnova and Stephen M. Watt. Communicating mathematics via pen-based computer interfaces. In *International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, (SYNASC 2008)*, pages 9–18, Timisoara Romania, September 2008. IEEE Computer Society.

[68] Elena Smirnova and Stephen M. Watt. Context-sensitive mathematical character recognition. In *Proc. IAPR International Conference on Frontiers in Handwriting Recognition, (ICFHR 2008)*, pages 604–610, Montreal, Canada, August 19-21 2008 2008. CENPARMI Concordia University, ISBN 1-895193-03-6.

[69] Alexander S. Szalay. The sloan digital sky survey and beyond. *SIGMOD Rec.*, 37:61–66, June 2008.

[70] G. Talenti. Recovering a function from a finite number of moments. *Inverse Problems*, (3):501–517, 1987.

[71] J. Tokuno, N. Inami, S. Matsuda, M. Nakai, H. Shimodaira, and S. Sagayama. Context-dependent substroke model for hmm-based on-line handwriting recognition. In *Frontiers in Handwriting Recognition, 2002. Proceedings. Eighth International Workshop on*, pages 78 – 83, 2002.

[72] Pascal Vincent and Yoshua Bengio. K-Local Hyperplane and Convex Distance Nearest Neighbor Algorithms. In Thomas G. Dietterich, Suzanna Becker, and Zoubin Ghahramani, editors, *Advances in Neural Information Processing Systems 14*, Cambridge, MA, September 2002. MIT Press.

[73] Toru Wakahara and Seiichi Uchida. Hierarchical decomposition of handwriting deformation vector field using 2dwarping and global/local affine transformation. In *10th International Conference on Document Analysis and Recognition*, pages 1141–1145. IEEE Computer Society, July 2009.

[74] J. Wang, P. Neskovic, and L.N. Cooper. A probabilistic model for cursive handwriting recognition using spatial context. In *Acoustics, Speech, and Signal Processing, 2005. Proceedings. (ICASSP '05). IEEE International Conference on*, volume 5, pages v/201 – v/204 Vol. 5, march 2005.

[75] Stephen M. Watt. An empirical measure on the set of symbols occurring in engineering mathematics texts. In *Proceedings of the 2008 The Eighth IAPR International Workshop on Document Analysis Systems*, pages 557–564, Washington, DC, USA, 2008. IEEE Computer Society.

[76] Don R. Wilhelmsen. A nearest point algorithm for convex polyhedral cones and applications to positive linear approximation. *Mathematics of Computation*, 30(133):48–57, January 1976.

[77] D. Randall Wilson and Tony R. Martinez. Reduction techniques for instance-based learning algorithms. In *Machine Learning*, pages 257–286, 2000.

[78] Philip Wolfe. Finding the nearest point in a polytope. *Mathematical Programming*, 11:128–149, 1976.

[79] Jacob Ziv and Abraham Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, 23:337–343, 1977.

Appendix A

Factorial Analysis Sign Table

Table A.1: Sign table of the factorial analysis of the algorithm for recognition of *n*-grams of rotated characters

Exp.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16		
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1525.6	95.4
α	1	1	1	1	1	1	1	1	-1	-1	-1	-1	-1	-1	-1	-1	-1.3	-0.1
β	1	1	1	1	-1	-1	-1	-1	1	1	1	1	-1	-1	-1	-1	-11.6	-0.7
n	1	1	-1	-1	1	1	-1	-1	1	1	-1	-1	1	1	-1	-1	2.5	0.2
p	1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	-1.3	-0.1
$\alpha * \beta$	1	1	1	1	-1	-1	-1	-1	-1	-1	-1	-1	1	1	1	1	-0.5	0.0
$\alpha * n$	1	1	-1	-1	1	1	-1	-1	-1	-1	1	1	-1	-1	1	1	0.7	0.0
$\beta * n$	1	1	-1	-1	-1	-1	1	1	1	1	-1	-1	-1	-1	1	1	1.2	0.1
$\alpha * p$	1	-1	1	-1	1	-1	1	-1	-1	1	-1	1	-1	1	-1	1	-0.2	0.0
$n * p$	1	-1	-1	1	1	-1	-1	1	1	-1	-1	1	1	-1	-1	1	0.7	0.0
$\beta * p$	1	-1	1	-1	-1	1	-1	1	1	-1	1	-1	-1	1	-1	1	-0.6	0.0
$\alpha * \beta * n$	1	1	-1	-1	-1	-1	1	1	-1	-1	1	1	1	1	-1	-1	0.2	0.0
$\alpha * \beta * p$	1	-1	1	-1	-1	1	-1	1	-1	1	-1	1	1	-1	1	-1	-0.2	0.0
$\alpha * n * p$	1	-1	-1	1	1	-1	-1	1	-1	1	1	-1	-1	1	1	-1	0.2	0.0
$\beta * n * p$	1	-1	-1	1	-1	1	1	-1	1	-1	-1	1	-1	1	1	-1	0.3	0.0
$\alpha * \beta * n * p$	1	-1	-1	1	-1	1	1	-1	-1	1	1	-1	1	-1	-1	1	0.0	0.0
y_1	94.9	95.0	94.4	94.8	96.4	96.4	95.9	96.2	95.0	95.1	94.6	95.0	96.4	96.5	96.2	96.3		
y_2	94.4	94.6	93.5	94.0	96.0	96.1	95.8	96.0	94.6	94.6	94.1	94.3	96.0	96.1	95.9	96.0		
y_3	94.9	95.0	94.0	94.4	96.1	96.0	95.8	95.9	95.0	95.1	94.5	94.8	96.1	96.1	95.9	96.1		
y_4	94.6	95.0	93.9	94.5	96.0	95.9	95.7	95.8	95.0	95.1	94.6	94.8	96.1	96.1	96.0	96.0		
y_5	94.7	94.8	94.3	94.5	96.2	96.3	96.0	96.1	94.8	94.9	94.4	94.6	96.2	96.3	96.1	96.2		
y_{mean}	94.7	94.9	94.0	94.4	96.1	96.1	95.8	96.0	94.9	94.9	94.4	94.7	96.1	96.2	96.0	96.1	95.4	

Curriculum Vitae

Name	Vadim Mazalov
Post-secondary Education	The University of Western Ontario London, Ontario, Canada 2010 - 2013 PhD(Computer Science) *Supervisor*: Pr. Stephen M. Watt 2009 - 2010 MSc(Computer Science) *Supervisor*: Pr. Stephen M. Watt Kuban State University Krasnodar, Russia 2003 - 2009 BS/MSc (Applied Mathematics and Computer Science) Roanoke College Salem, Virginia, USA 2005 - 2006 Exchange Student
Honours and Awards	University Students' Council Teaching Honour Roll for teaching CS1027a in 2011 Ontario Graduate Scholarship for 2012-2013 GTA Union Scholarship for Academic Achievements, UWO, 2012 GTA Union Scholarship for Academic Achievements, UWO, 2011. Graduate Thesis Research Award, UWO, 2011.

Prasanna Mohan Scholarship, UWO GTA Union, 2010.

Russian Government Scholarship for scientific research and academic excellence, KubSU, 2007-08 and 08-09.

Krasnodar Region Government Scholarship for scientific research and academic excellence, KubSU, 2007-08 and 08-09.

Scholarship of Philip Morris Int. for academic achievements and leadership, KubSU, 2007-08 and 08-09.

Academic Scholarship, KubSU, 2003-09.

Work Experience

Lecturer, Research Assistant, Teaching Assistant
The University of Western Ontario
2009 - 2013

Research and Development Specialist
Cyborg Trading Systems
London, Ontario, Canada
2010 - 2013

Software Development Engineer in Test, Intern
Microsoft, Inc.
Redmond, WA, USA
2012-2012

Software Engineer
Peter-Service, CJSC
Krasnodar, Russia
2007 - 2009

Publications

1. Vadim Mazalov and Stephen M. Watt. A structure for adaptive handwriting recognition. In *ICFHR*, pages 692–697, 2012.

2. Vadim Mazalov and Stephen M. Watt. Recognition of relatively small handwritten characters or “size matters”. In *Frontiers in Handwriting Recognition (ICFHR), 2012 International Conference on*, pages 319–324, Sept.

3. Rui Hu, Vadim Mazalov, and Stephen M. Watt. A Streaming Digital Ink Framework for Multi-Party Collaboration, *Proc. 2012 Conferences on Intelligent Computer Mathematics, (CICM 2012)*, pp. 81-95, July 9-14 2012, Bremen, Germany, Springer Verlang.

4. Vadim Mazalov and Stephen M. Watt. Writing on clouds. In *Proceedings of the 11th*

international conference on Intelligent Computer Mathematics, CICM'12, pages 402–416, Berlin, Heidelberg, 2012. Springer-Verlag.

5. Vadim Mazalov and Stephen M. Watt. Linear compression of digital ink via point selection. In *Proceedings of the 2012 10th IAPR International Workshop on Document Analysis Systems*, DAS '12, pages 429–434, Washington, DC, USA, 2012. IEEE Computer Society.

6. Lu Xiao and Vadim Mazalov. Message Visualizer: a Visualization Tool for Chat Messages, pp. 426-428, *Proc. 2012 iConference*, February 07-10, 2012, Toronto, Canada.

7. Vadim Mazalov and Stephen M. Watt. Improving isolated and in-context classication of handwritten characters. pages 82970B–82970B–8, 2012.

8. Vadim Mazalov, Dmitry Mazalov, and Anna Pauer. Pen-Based Computing in Medicine: Factorial Analysis of the Rotation-Invariant Recognition Algorithm, *Proc. Canadian Student Conference on Biomedical Computing and Engineering 2011*, pp. 85-89.

9. Vadim Mazalov and Stephen M. Watt. Digital ink compression via functional approximation. *Proc. of International Conference on Frontiers in Handwriting Recognition.*, pages 688–694, 2010.

10. Oleg Golubitsky, Vadim Mazalov, and Stephen M. Watt. Toward affine recognition of handwritten mathematical characters. In *Proc. International Workshop on Document Analysis Systems, (DAS 2010)*, pages 35–42, Boston, USA, June 9-11 2010. ACM Press.

11. Oleg Golubitsky, Vadim Mazalov, and Stephen M. Watt. Orientation-independent recognition of handwritten characters with integral invariants. In *Proc. Joint Conference of ASCM 2009 and MACIS 2009: Asian Symposium of Computer Mathematics and Mathematical Aspects of Computer and Information Sciences, (ASCM 2009)*, pages 252–261, Fukuoka, Japan, December 2009. COE Lecture Note Vol. 22, Kyushu University, ISSN 1881-4042.