Writing on Clouds

Vadim Mazalov and Stephen M. Watt

Department of Computer Science
The University of Western Ontario
London Ontario, Canada N6A 5B7
{vmazalov,Stephen.Watt}@uwo.ca

Abstract. 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. This work uses 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.

Keywords: Handwriting Recognition, Mathematical Handwriting Recognition, Cloud Computing, Service Oriented Architecture

1 Introduction

We are interested in online recognition of handwritten mathematics. The widespread use of hand-held mobile devices and tablets has created a ubiquitous environment for two-dimensional math input. Writing mathematics on a digital canvas is similar to traditional pen-on-paper input. It does not require learning any typesetting languages and can be efficient, given a robust and reliable implementation. According to one study [1], pen-based input of mathematics is about three times faster and two times less error-prone than standard keyboard- and mouse-driven techniques. However, recognition of mathematics is a harder problem than recognition of natural language text.

In our classification paradigm, a character is represented by the coefficients of an approximation of trace curves with orthogonal polynomials [4]. Classification is based on computation of the distance to convex hulls of nearest neighbours in the space of coefficients of approximation. Typically, the method 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 the present work 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.

The rest of the paper is organized as follows. Section 2 introduces the character approximation and recognition foundation, as well as some preliminary concepts required by the proposed architecture. Section 3 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 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. The recognition error decrease as a function of the input size of a writer is presented in Section 5. Section 6 concludes the paper.

2 Preliminaries and Related Work

2.1 Recognition Aspects

In an online classification environment, a curve may be given as an ordered set of points in a Euclidean plane. Devices are capable of sampling the coordinates of a stylus as functions of time. The inputs to online classification are typically given as vectors of pen coordinates, represented as real numbers received at a fixed frequency [4]. In addition, some devices can collect other information, such as pressure or pen angle, as well as spatial coordinates when a stylus is not touching the surface. We however do not rely on this additional information so we can maintain hardware independence.

The input traces may be regarded as parametric functions and we may represent these using standard approximation methods as truncated orthogonal polynomial series

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

where $B_i(\lambda)$ are the orthogonal basis polynomials, e.g. Chebyshev, Legendre or Legendre-Sobolev polynomials, and λ is a parameter, e.g. time or arc-length [4]. Multi-stroke characters may be represented by concatenating the coordinate sequences of the strokes.

Having the coefficients of approximation, a character can be represented as the tuple

$$x_0, x_1, ..., x_d, y_0, y_1, ..., y_d.$$

In this vector, coefficients x_0 and y_0 give the initial position of the sample and can be neglected to normalize location of the character. Dividing the rest of the vector by the Euclidean norm will normalize the sample with respect to size. We base classification on a distance (in some norm) to the convex hull of k nearest neighbours in the space of coefficients [4].

2.2 Architectural Aspects

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.

2.3 Related Work

Several related projects have been described that mostly target development of managed experimental repositories and resource sharing in the context of: document analysis [7], astronomical observations [14], or environmental research [2]. 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.

3 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 [3].
- 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.
- *Recognizer* is a character recognition engine, developed according to the principles described in [4].
- *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.

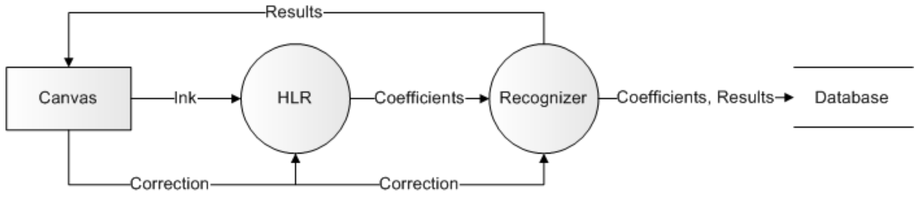


Fig. 1. The data flow diagram for recognition and correction

3.1 Recognition Flow

The overall recognition flow is shown in Figure 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

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

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 [4]

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

The first three elements in (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 [10]. 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

```
coefficients ::= 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>
```

Fig. 2. The format of the SOAP message sent to the cloud

```
...
<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

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 Figure 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

```
...
<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

```
...
<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

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 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 can be achieved by considering the most popular expressions in the subject and their empirical or grammatical properties, see for example [8].

3.2 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

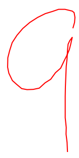


Fig. 3. A sample that belongs to classes "q" and "9"

that belongs to both classes can look as the one shown in Figure 3. A naïve approach to compute such 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 $S_q \setminus S_9$ that will result in points that can not be confused with 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.

4 Implementation

From a high level viewpoint, the system contains the following parts, as shown in Figure 4.

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

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.

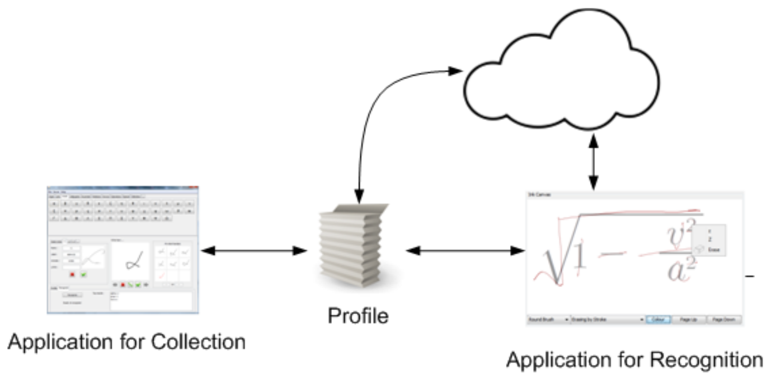


Fig. 4. Interaction of user interfaces for collection and recognition with the cloud

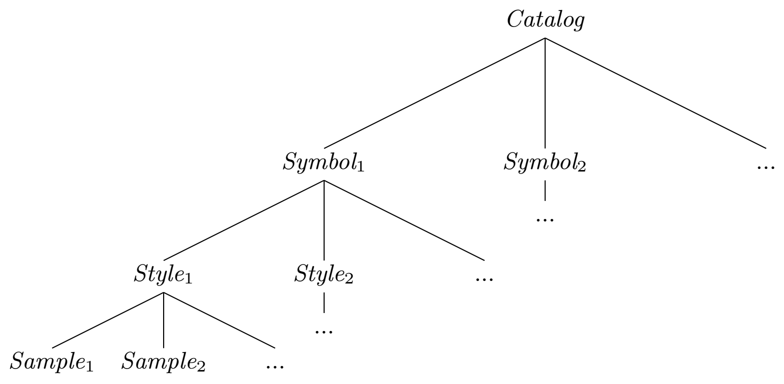
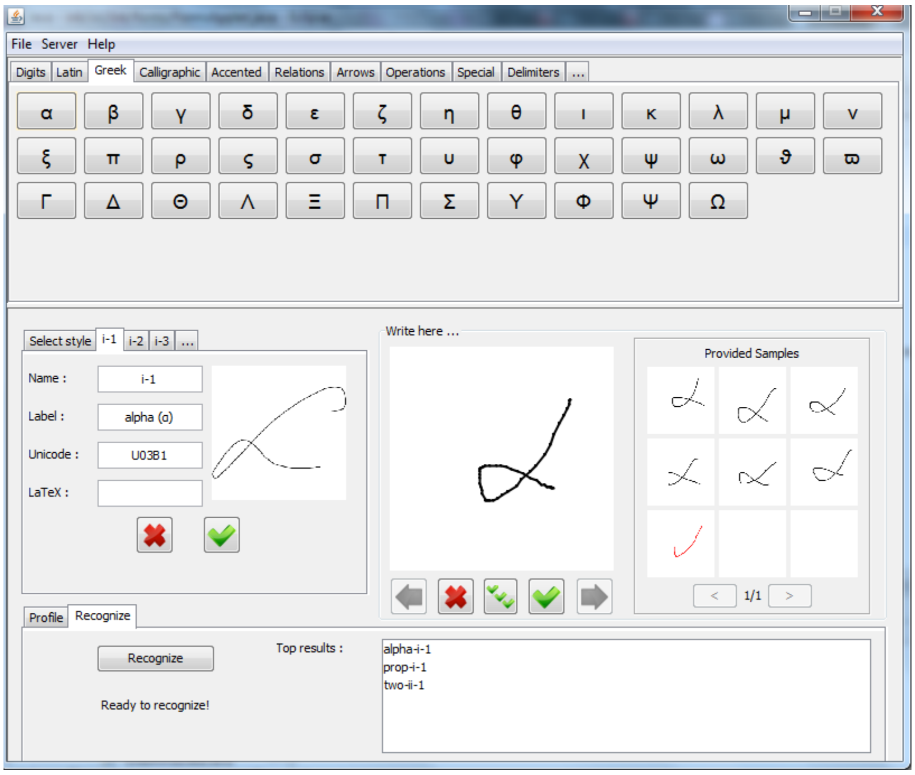


Fig. 5. The structure of a catalog

The Structure of a User Profile 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 5 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.

Each user can have several profiles used together or independently, representing, for example, different areas of mathematics, chemistry or music. *System* profiles


File Server Help
Digits Latin Greek Calligraphic Accented Relations Arrows Operations Special Delimiters ...
α β γ δ ε ζ η θ ι κ λ μ ν
ξ π ρ ς σ τ υ φ χ ψ ω ϑ ϖ
Γ Δ Θ Λ Ξ Π Σ Υ Φ Ψ Ω
Select style i-1 i-2 i-3 ...
Write here ...
Provided Samples
1/1
Profile Recognize
Recognize
Top results :

Name : i-1
Label : alpha (α)
Unicode : U03B1
LaTeX :

Ready to recognize!

```
alpha-i-1
prop-i-1
two-ii-1
```

Fig. 6. The main window of the training application

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 [4].

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. A tab contains a list of symbols of the catalog. Once the user selects a symbol, the panel with styles becomes 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 [5], 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 [6], a pen-based system for interactive mathematics, or MathInk [13], a mathematical pen-based plug-in that can run inside computer algebra systems, such as Maple [9], 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 7. 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.

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.

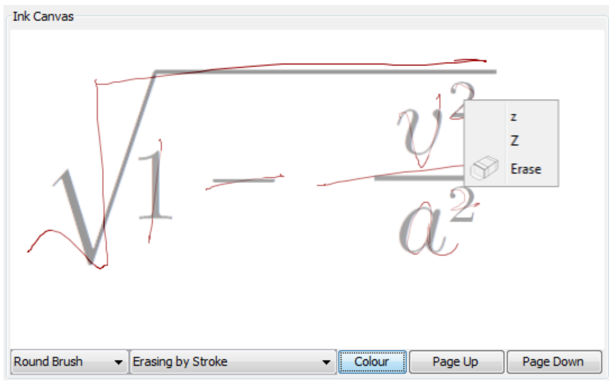


Fig. 7. Client interface for recognition

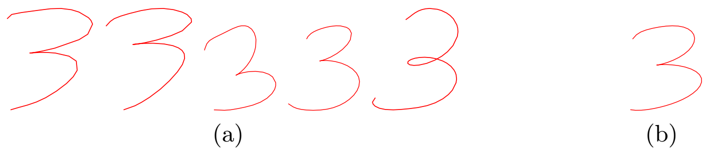


Fig. 8. (a) A set of provided samples, and (b) the average sample

4.3 Attractive Display of Recognized Characters

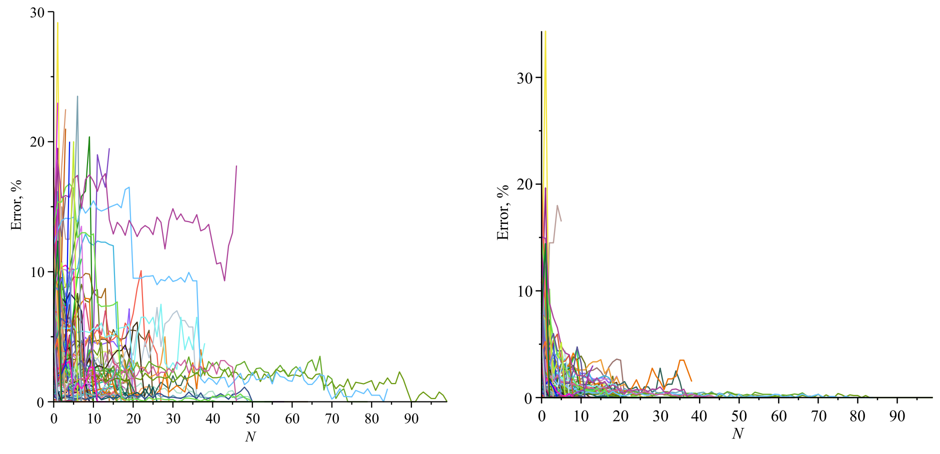
Some research has shown that averaging can be used to make faces look attractive [12]. We adopt a similar approach to generate visually appealing output. The shape of each output stroke is obtained by taking the average of coefficients of approximation of corresponding strokes of samples in the style

$$\bar{c}_i = \frac{\sum_{j=1}^{n} c_{ij}}{n}$$

where $\bar{c}_i$ 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 8.

5 Experimental Evaluation

We describe results of an experiment that shows performance of adaptive author-centered recognition that can be implemented with 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

Fig. 9. 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.

5.1 Setting

The experimental dataset is identical to the one described in [4]. 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 [11], 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.

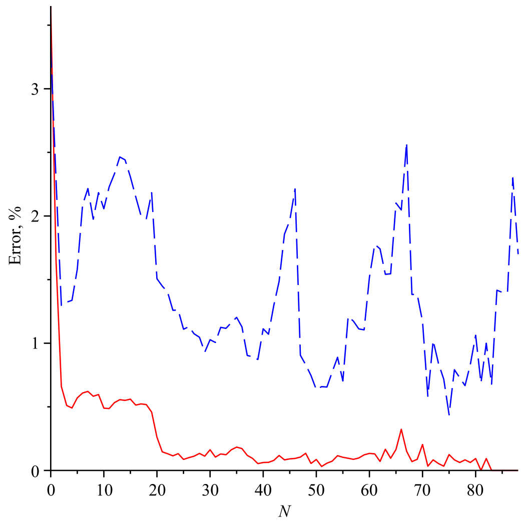


Fig. 10. The average recognition error among all authors of the $(N+1)$-th sample in a class for the Basic strategy (solid) and the Null strategy (dash).

5.2 Results

Figures 9(a) and 9(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 10 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 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.

References

1. Anthony, L., Yang, J., Koedinger, K.R.: Evaluation of multimodal input for entering mathematical equations on the computer. In: CHI '05 extended abstracts on Human factors in computing systems. pp. 1184–1187. CHI EA '05, ACM, New York, NY, USA (2005), `http://doi.acm.org/10.1145/1056808.1056872`
2. Beran, B., van Ingen, C., Fatland, D.R.: Sciscope: a participatory geoscientific web application. Concurrency and Computation: Practice and Experience 22(17), 2300–2312 (2010)
3. Chan, K.F., Yeung, D.Y.: Mathematical expression recognition: a survey. IJDAR 3(1), 3–15 (2000)
4. Golubitsky, O., Watt, S.M.: Distance-based classification of handwritten symbols. International J. Document Analysis and Recognition 13(2), 133–146 (2010)
5. Hu, R.: Portable implementation of digital ink: collaboration and calligraphy. Master's thesis, University of Western Ontario, London, Canada (2009)
6. Labahn, G., Maclean, S., Marzouk, M., Rutherford, I., Tausky, D.: A preliminary report on the MathBrush pen-math system. In: Maple 2006 Conference. pp. 162–178 (2006)
7. Lamiroy, B., Lopresti, D., Korth, H., Heflin, J.: How carefully designed open resource sharing can help and expand document analysis research. In: Document Recognition and Retrieval XVIII - DRR 2011. vol. 7874. SPIE, San Francisco, United States (Jan 2011)
8. MacLean, S., Labahn, G., Lank, E., Marzouk, M., Tausky, D.: Grammar-based techniques for creating ground-truthed sketch corpora. Int. J. Doc. Anal. Recognit. 14, 65–74 (March 2011), `http://dx.doi.org/10.1007/s10032-010-0118-4`
9. Maplesoft: Maple 13 user manual (2009)
10. Mazalov, V., Watt, S.M.: Digital ink compression via functional approximation. Proc. of International Conference on Frontiers in Handwriting Recognition. pp. 688–694 (2010)
11. Mazalov, V., Watt, S.M.: A structure for adaptive handwriting recognition. In: Proc. of the International Conference on Frontiers in Handwriting Recognition (submitted) (2012)
12. Perrett, D., May, K., Yoshikawa, S.: Facial shape and judgments of female attractiveness. Nature 368, 239–242 (March 1994)
13. Smirnova, E., Watt, S.M.: Communicating mathematics via pen-based computer interfaces. In: Proc. 10th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, (SYNASC 2008). pp. 9–18. IEEE Computer Society (Sept 2008)
14. Szalay, A.S.: The sloan digital sky survey and beyond. SIGMOD Rec. 37, 61–66 (Jun 2008), `http://doi.acm.org/10.1145/1379387.1379407`