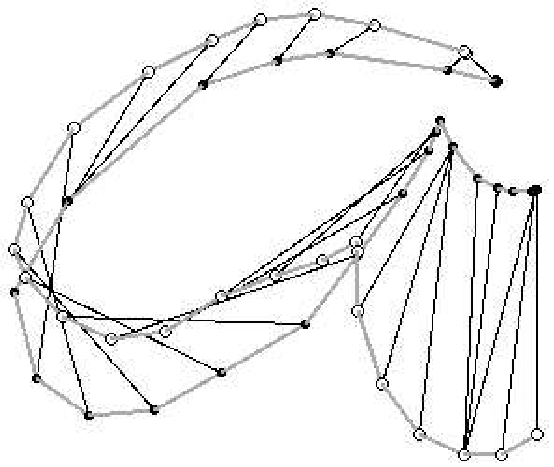
Dynamic Time Warping

An intuitive way of
handwriting recognition?

Ralph Niels
ralph@noviomagum.com
http://dtw.noviomagum.com



MASTER THESIS
Artificial Intelligence / Cognitive Science
November/December 2004

Supervisor
Dr. L. G. Vuurpijl

Radboud University Nijmegen
Faculty of Social Sciences
Department of Artificial Intelligence / Cognitive Science
Nijmegen, The Netherlands

The writings on the wall brothers,
Your life is in your hands.
Its up to you to see the writings on the wall.

George Harrison
Writings on the wall

Abstract

Automatic handwriting recognition has had the interest of researchers for decades. Although there are various applications for which the technique is already used in daily life, a number of problems still has to be solved. At this moment, one of the biggest problems is the low user acceptance: the systems are not accurately enough, and moreover, the mistakes that recognizers make are usually not very understandable to humans, which can frustrate the users of the systems.

This thesis is about the use of the Dynamic Time Warping (DTW) algorithm. We claim that the results of a recognizer based on the DTW-algorithm handwriting recognition are more "intuitive" to humans than the results of other recognizers. Because humans can understand the errors the system makes, this will probably improve the user acceptance of handwriting recognition systems. Furthermore, given that users are aware of what they have to do for the system to understand them, the recognizer is expected to yield better recognition performances. The way the system works also adds a number of new possibilities for applications of automatic handwriting recognizers. A more human-like system, for example, can be of use in the field of forensic document analysis or the teaching of handwriting to young children.

Because the algorithm compares characters in a way that differs from the comparison by other classifiers, we think that it can be useful to add the system to a Multiple Classifier System (MCS): because the system has a different "look" on the data, and therefore is orthogonal to other classifiers, it can give valuable advice about a classification in a MCS.

To test the performance of DTW, our claim about the similarity to human handwriting recognition and the suitability of the system in a Multiple Classifier System, we have implemented the algorithm into a prototype-based classifier. Two different sets of prototypes were also created especially for the classifier.

We have conducted an experiment to test the recognition performance of our classifier, using the different prototype sets. The results show that our classifier correctly classifies between 88.20 and 95.26 percent of the offered samples using one of the prototype sets and between 90.32 and 97.16 percent using the other prototype set. Another experiment was conducted to check the "intuitivity claim". 25 subjects judged the results of our classifier and the results of another classifier, and the results show that our classifier performed significantly better than the other system.

Finally, the classifier was tested in a real MCS, that was used for the cleaning up of a large database (the UNIPEN devset) of handwritten characters, which is known to contain a relatively large number of errors. Using the MCS, we were able to automatically clean up a very large part of the database: only a minimum of human interference was needed. The concept of *rejection*, that allows the classifier to reject a classification if it is not certain enough about it, was implemented for this application.

The performance of our DTW-classifier in the experiments, and in the tested application is very promising, and in the future, the classifier could be applied to other than Latin alphabets or recognizing words. Also, it can be used for other applications than classification or data cleaning.

Contents

1 Introduction **1**

1.1 Research questions . . . 2

1.2 Organization of this thesis . . . 3

2 Handwriting recognition **4**

2.1 What do we need it for? . . . 4

2.1.1 Human computer interaction . . . 5

2.1.2 Process automation . . . 6

2.1.3 Writer identification . . . 6

2.2 Allograph representation . . . 7

2.2.1 Continuous versus discrete . . . 8

2.2.2 Offline versus online . . . 8

2.3 History and techniques . . . 9

2.3.1 Feature space . . . 10

2.3.2 Prototype based systems . . . 11

2.4 Human factors issues . . . 12

2.5 Other domains of handwriting recognition . . . 13

3 The classifier **14**

3.1 Data . . . 14

3.1.1 Cleaning the data . . . 15

3.2 Dynamic Time Warping . . . 18

3.2.1 Matching paths . . . 18

3.2.2 History of DTW . . . 19

3.2.3 The DTW-algorithm for computing a matching path . . . 21

3.2.4 Decision algorithm . . . 26

3.2.5 Options and adjustments . . . 26

3.3 Generating prototypes . . . 30

3.3.1 Dividing data in classes . . . 30

3.3.2 Clustering and selecting . . . 30

3.3.3 Combining samples into prototypes . . . 33

3.3.4 Post processing . . . 37

3.3.5 Prototype sets . . . 38

4 Performance and Applications **40**

4.1 Recognition performance . . . 40

4.1.1 Introduction . . . 40

4.1.2 Method . . . 41

4.1.3 Results . . . 41

4.1.4 Discussion . . . 45

4.2 Rejection . . . 45

4.2.1 Certainty values 46
4.2.2 Usage of the certainty value 48
4.2.3 Results 48
4.3 Data verification 49
4.3.1 Introduction 50
4.3.2 The automated verification procedure 53
4.3.3 The manual labeling process 56
4.3.4 Verifying words 58
4.3.5 Conclusions 59
4.4 Intuitive handwriting recognition 60
4.4.1 Introduction 60
4.4.2 Method 61
4.4.3 Results 63
4.4.4 Discussion 65

5 Discussion **67**
5.1 Conclusions 67
5.2 Future research and usage 68
5.2.1 Forensic Document Analysis 68
5.2.2 Automatic handwriting teaching 69
5.2.3 Other alphabets 69
5.2.4 Word recognition 70
5.2.5 Automatic optimizing 70
5.2.6 Forensic Document Analysis 71

References **73**

Appendices **I**

A UNIPEN r01_v07 trainset specification **I**

B Prototype specification **III**

C Intuitivity Experiment instructions **V**

D SPSS syntax **VI**
D.1 Recognition performance experiment VI
D.2 Intuitivity experiment VII
D.2.1 Hypotheses H_A and H_B VII
D.2.2 Hypothesis H_C VII

Preface and Acknowledgments

When I was looking for a topic for my internship and the accompanying thesis, about a year ago, I was certain that I wanted to do a scientific internship that had “something to do with pattern recognition”. Since I knew that Louis Vuurpijl, whom I met before (when I was a student in his class and when I was an assistant in his Artificial Neural Networks class), was working in this field, I asked him to be my supervisor.

He said yes, and told me that he was looking for someone to implement the Dynamic Time Warping algorithm into a handwriting classifier. A handwriting classifier. Wow, that sounded interesting. I liked the fact that I would be working as a researcher and as a programmer, building something that had both scientific and practical purposes. Because I also wanted to get some experience in the setting up and executing of experiments involving humans, and not only computers, the fact that the Dynamic Time Warping theory claimed to be able to compare characters in a human way, was an additional reason to be enthusiastic about it: after the implementation, I would set up an experiment to find out how right this claim was.

During my internship and the writing of my thesis, I learned a lot. It started with learning how to read large amounts of papers, how to use the Linux computer operating system, how to program in a language I hardly had any experience with, and, not unimportant, how to get used to a steady day schedule, which was not really easy after four years of students’ life. After a couple of months, in which I was busy implementing the system, another learning phase started, in which the execution of experiments and the reporting of the results played a central role. I was a coauthor of a paper for the 9th International Workshop on Frontiers In Handwriting Recognition, about the cleaning up of an important dataset to get it ready for use by scientists all over the world, I conducted my own experiments, I had to find motivated subjects, I analyzed and reported the results of the experiments, and last, but most certainly not least, I wrote a thesis. The thesis you are holding in your hands at this very moment.

A lot of work was done. A lot of work which I never could have done alone. Therefore I would like to thank my supervisor Louis Vuurpijl for introducing me to the practical side of science, the research on handwriting recognition and the wonderful world of the Linux operating system, Egon van den Broek for his help in setting up the *Intuitivity Experiment* described in Section 4.4, Nol Bendermacher of the *ResearchTechnische OnderzoeksGroep* for his help on the statistics in Sections 4.1 and 4.4. For the IWFHR9 paper, which was also the base of Section 4.3, I would like to thank the coauthors Louis Vuurpijl, Merijn van Erp, Lambert Schomaker and Eugene Ratzlaff, and also Jules Ellis for his support on verifying the validity of the statistical tests and Hans Dolfing and Jianying Hu for their contributions to recovering the original devset labels. For the times when my inspiration and motivation were gone, and also for the nice lunches, I would like to thank Eva, Susan, Daphne and Merijn.

Finally, I would like to thank some people that I believe sometimes did not have a clue of what I was doing, but supported me anyway. I want to thank Hanneke, who was my personal psychologist and motivator at the times that I really needed it and my parents without whom I would not even have been here.

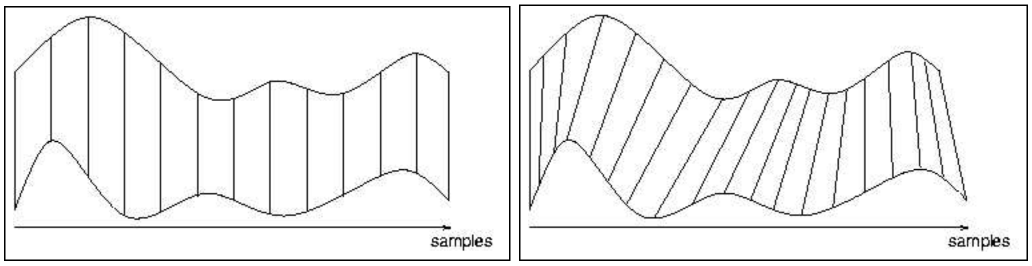
Ralph Niels, November 2004.

Chapter 1

Introduction

This thesis is about handwriting recognition, and a technique called Dynamic Time Warping (DTW) that can be used for handwriting recognition. We believe that the technique can be of importance for the handwriting recognition research: it gives a relatively new view on the data, and thus can be an addition to existing systems (it can be combined with other systems in a so called Multiple Classifier System). It can also be used in a standalone handwriting system that can read human handwriting (these systems are called *handwriting classifiers* or simply *classifiers*).

As can be seen in Figure 1.1, the DTW-algorithm is able to compare two curves in a way that makes sense to humans (we call this sense *intuitivity*) [24]. Because, at a very basic level, handwritten characters are nothing more than special cases of curves, we believe that DTW can compare characters in a way that is similar to the way humans compare characters, or at least generates the same results.



(a) One to one comparison

(b) DTW-comparison

Figure 1.1: Comparison of two curves using one to one comparison and Dynamic Time Warping. As can be seen, the DTW-comparison is more intuitive than the one to one comparison (images retrieved from http://ciir.cs.umass.edu/ trath/prj/hw_retr/wordspot_retr.html).

One reason for the intuitivity of the results of a DTW-comparison is the fact that it maintains the importance of spots in curves that are important for humans when they compare curves (examples of these spots are the beginnings and endings of characters, but also (sudden) changes in direction. See Figure 1.2 for some examples). These spots indicate either positions in the character that are important in human reading [45], or places where characters are divided into smaller parts that are used in human reading [11]. The important spots play an important role in both human reading and in the DTW-comparison: we therefore predict that the DTW-technique has results that are similar to the results of human character comparison.

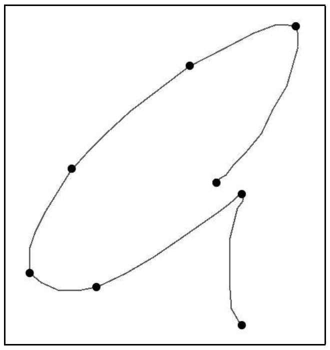


Figure 1.2: Examples of important spots. These spots indicate either positions that are important in human reading, or places where characters are divided into smaller parts that are used in human reading.

1.1 Research questions

Three main research questions will be answered in this thesis (the section number shows in which section the research question is described:

1. How well does DTW-classify handwritten characters, and how does this compare to other systems? (Section 4.1)
2. How can DTW be used in a Multiple Classifier System (MCS)? And does its different view on data improve the results of that MCS? (Section 4.3)
3. Does DTW generate results that are more intuitive to humans than the results generated by other systems? (Section 4.4)

1.2 Organization of this thesis

Chapter 2 introduces the concept of handwriting recognition. Important problems that occur will be described and it will be shown how we believe that Dynamic Time Warping can solve some of these problems.

In Chapter 3, the building, tuning and fine-tuning of the classifier will be described in detail. Every aspect of the classifier, and the theory behind it, will be explained, together with the problems that occurred and how they were solved.

Chapter 4 describes a number of applications that our classifier was used for, together with the performance on those applications. Also, the experiment that was conducted to test the “intuitivity claim” is described in this chapter.

Finally, in Chapter 5, the conclusions will be summarized and the final evaluation will be described. Also, possible topics of future research on our classifier will be mentioned and briefly explained.

Chapter 2

Handwriting recognition

Artificial handwriting recognition, in this thesis called handwriting recognition, is the reading of human handwriting by computers. Symbolic identities are associated with images, or other representations (like trajectories of points), of handwritten characters, words or even sentences: shapes that are meaningless to a computer, are converted to identities that have a meaning to the computer, so they can be processed like items entered using a keyboard or a mouse. This, for example, makes it possible for a computer to alphabetize entered items, search in, or check the spelling of an originally handwritten text, or communicate with a user in, for example, a help dialog or data-entering system.

The subject of handwriting recognition has had the interest of researchers for many years. The first modern attempts to let digital computers read human handwriting were made in the 1940s [31], and today, as will be shown in this chapter, research is still going on.

2.1 What do we need it for?

There are several applications for handwriting recognition. They can be split up in two groups. In the first group, that probably is the biggest of the two, the computer should handle a character written by person X in the same way as it handles that character if it were written by person Y: all that matters is which character is written. Examples of applications in this group are handwriting recognition for postal codes or ZIP codes and amounts on checks. In the other group, the research concentrates on the differences between handwriting of different people. Recognizers are built that use individual characteristics to identify who wrote a particular piece of text. Examples are signature validation and writer identification for forensic document analysis. The most prominent applications of handwriting recognition, of both groups, are reviewed in this section. As will become clear in Chapter 4, our classifier can be used for applications in both groups.

2.1.1 Human computer interaction

The interaction between human and computer traditionally takes place with a keyboard, a mouse and a monitor. The keyboard and mouse allows a person to enter information into the computer, and the monitor provides information back to the user. In the latest decades, several other interaction paradigms, like speech based or pen based interaction have become available. The latter is the topic of research that this thesis is about. As will be shown below, there are a number of reasons why the "electronic pen" is a suitable alternative (or a supplement) for entering information, to using the keyboard and the mouse:

1. Handwriting is a much more natural way to communicate than typing is: people have done it for thousands of years (the most ancient writing known today is Sumerian, which dates back to 4000 B.C.[1]), while the keyboard, in the form of a mechanical typewriter, has been available only since the 1860s[2]. Because it is a more natural way of communication, it could not only be easier to write instead of type, but it could also be a way of decreasing the chance of a computer user to get injuries occurring from repeated physical movements, like Computer Related Disorders (CRD), repetitive strain injury (RSI), and especially the carpal tunnel syndrome [2], which are caused by the frequent use of flat, light-touch keyboards that permit high speed typing [32].

2. Sometimes the keyboard just is not an option to interact with the computer. For example, when using a handheld computer or a personal digital assistant (PDA), a keyboard would add so much to the size of the system that it is no longer usable. In these cases, handwriting turns out to be a good alternative: many, if not most, handheld computers that are produced today have a handwriting recognition module installed.

3. Even if the written text is not meant for computer processing, but for a quick note to oneself, for example, it can be useful to let the computer translate it to a symbolic representation instead of remembering the exact written shape: the symbolic representation of a character needs much less memory than the representation of the shape and thus can save significant amounts of hard disk space, which is still a relevant issue in handheld computers, due to their relatively small hard disks.

4. Applications like ink messaging, graphical design or annotation of digital documents are other examples where the pen is a more natural interaction medium.

Although handwriting recognition has been available to the public for quite some years, it has not become very popular yet.This is due to a number of problems, which are described in Section 2.4. One of our aims is to overcome some of the problems described there.

[1]http://www.wordiq.com/definition/Writing
[2]http://en.wikipedia.org/wiki/Typewriter

2.1.2 Process automation

Another application of handwriting recognition is the automation of processes that are concerned with "reading". A body of literature is available that describes "automatic reading systems", a wide research area that is covered by the communities organized in the International Association for Pattern Recognition[3] technical committees TC-10[4] and TC-11[5]. For example, the reading of postal codes or ZIP codes on envelopes can (partially) be taken over by computers [8]. This can save lots of time and money, since computers can do it much faster and cheaper than humans can [51]. Because reading these codes is a very limited domain (for example, postal codes in The Netherlands always consist of four digits, followed by two capital letters[6] and ZIP codes in the United States consist of 5 (or 9 in the ZIP+4 system) digits[7]), the results are reliable, and thus usable.

Another process that can be automated is the reading of checks. Because often large amounts of money are involved, the recognizers need to be very reliable. If this can be achieved however, a lot of expenses can be saved too [37].

A third process that takes a lot of time if done by humans, and can be done much faster by computers is the labeling of datasets. Items (or "samples") in collections of handwritten characters or words are read by a computer, which decides what character or word each sample represents, and labels the sample with the name of that character or word. If a new collection of handwritten data is created, a handwriting recognizer can be used to label the samples. Also, a collection like, for example, the UNIPEN dataset [17], which is already labeled, but is known to contain a number of errors in the labeling [18], can be relabeled by an automatic handwriting recognizer. In this case labels are available and the recognizers can validate them. This specific application is one of the applications that the classifier described in this thesis was used for. More about this can be read in Section 4.3.

2.1.3 Writer identification

In addition to the recognition of the amounts written on checks as described in Section 2.1.2, the validation of signatures is another application of handwriting recognition in the world of banking: does the signature on the check match the known signature of the client, as stored in some database? Because large amounts of checks have to be processed each day, it is hardly possible for humans to check them all in a reliable way. If this process can be taken over by computers, which can do the validation much faster, fraud can be reduced at a reasonable price. Partly because of its economic value, a lot of research has been done in this field [35, 57, 71].

[3]http://www.iapr.org/
[4]http://www.cvc.uab.es/iapr-tc10/
[5]http://www.ai.rug.nl/iapr/tc-11/
[6]http://nl.wikipedia.org/wiki/Postcode
[7]http://en.wikipedia.org/wiki/Zipcode

Writer verification (e.g. someone claims that this piece of text was written by person X, is that true?) and writer identification (e.g. a piece of handwritten text has been found, who wrote it?) can be of great use in the field of forensic document analysis: if a piece of paper containing handwriting is found at a crime scene, or if a handwritten threatening letter is reported to the police, the handwriting can be used as a kind of fingerprint: if the writers handwriting is known and collected in a database, the identity of the writer can be found using a writer identification system. If a possible suspect is available, writer verification could answer the question whether or not that person wrote the letter. But even if no previous handwriting samples of a person are collected, that persons handwriting can still reveal information about for example his or her origin, writing experience, gender and handedness. Traditionally, this is done by human document examiners [10, 20], but computers can compare the writing of one person to a much larger database in a much shorter time than humans can, and research shows that computers are good at the task, so automated forensic document analysis systems can be used to support human experts [14, 55]. Essential is the way in which handwriting samples are compared to each other. A system that performs the comparison such that its results are similar to those of human experts, may have a chance of being accepted in court stand. When a system performs this way, the decisions can be explained so that they are acceptable and understandable for humans. As will be discussed in Section 4.4, we believe that the handwriting recognizer we built satisfies to this condition.

2.2 Allograph representation

In order for an automated handwriting recognition system to "read" handwritten characters, some kind of internal representation of these characters is required, just like an internal representation in the mind is necessary for a human to read a character. An allograph is a letter of an alphabet in a particular shape[8] (see Figure 2.1 for an example of different allographs of the letter *a*). Every instance of a letter in the world has some form of representation: as ink on paper, as a shape that is in your mind, as pixels on a computer screen, et cetera. This section describes how allographs can be represented in the computer.

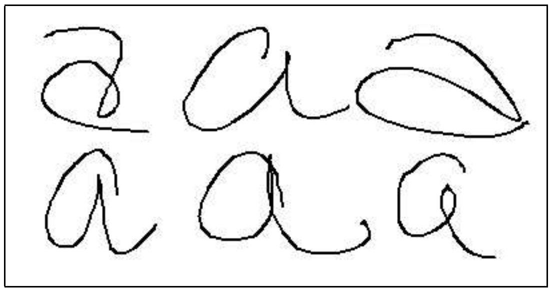


Figure 2.1: Different allographs of lowercase letter *a*.

[8]http://www.merriamwebster.com/cgi-bin/dictionary?va=allograph

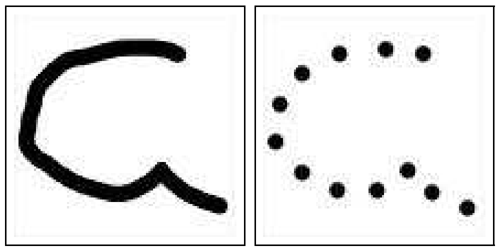
2.2.1 Continuous versus discrete

The real world consists of all kinds of stimuli that are, or at least appear to be, continuous. This means that there are no steps between two values. Examples are the height of a growing flower[9], mass, linear dimensions, time and handwritten characters on a piece of paper.

Opposite to continuous stimuli are discrete ones: successive measurements where there are no values in between. Examples are house numbers, number of correctly answered questions on a test, the pressing of a button (either pressed or not) and a mouse-drawn character that is represented by a limited number of pixels or coordinates.

Because digital computers are discrete systems, any continuous stimulus has to be converted to a discrete one for it to be able to process it.

An example showing the difference between the continuous and the discrete representation of a handwritten character can be seen in Figure 2.2.



(a) Continuous (b) Discrete

Figure 2.2: Two representations of the same allograph: (a) continuous, consisting of a continuous stroke, as it could be represented in the real world and (b) discrete, consisting of a limited number of points, as it could be represented in a digital computer. Note that the discrete representation holds less information than the continuous one. One has to be very careful in the decision of how much information is needed, when converting continuous to discrete data, or when collecting discrete data.

2.2.2 Offline versus online

When you see a handwritten sample, usually you cannot see whether it was written slow or fast or whether the writer started at the top or at the bottom (see Figure 2.3). No information about timing is available. This is what we call offline data. On the other hand, there is online data. Here, the time information is available. One can extract the order of the writing (where did the writer start, and where did he or she stop writing, what direction did the pen take) and also the speed of the writing: if points are recorded at a regular frequency, the closer two successive points are, the slower the letter was written. This additional information is hardly ever explicitly used in human reading (although it may be used implicitly), simply

[9]Example taken from http://www.wordiq.com/definition/Continuous_function

because the information cannot easily be extracted from handwriting that is represented by a still image. For computers however, this piece of data can improve the recognition performance. See Figure 2.4 for an example of how online information can help in recognizing a character.

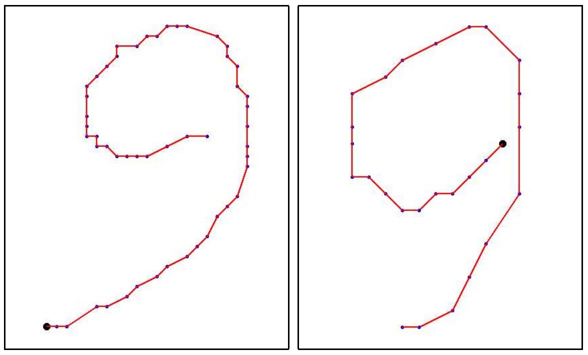


Figure 2.3: Two handwritten 9's with as main difference the writing direction: the big dots mark the starting points. Without the writing-order information one would not have been able to decide the writing direction. This direction can be a distinctive feature of a persons handwriting and can thus be used for identifying the writer.

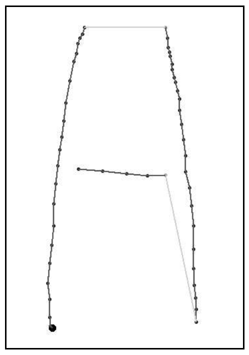


Figure 2.4: A capital A. Without the information about the writing direction (the big dot marks the starting point), a mix-up with the capital H would have been more likely.

2.3 History and techniques

The first serious attempts on character recognition were done in the middle 1940s with the modern version of optical character recognition (OCR). OCR was created to cope with the enormous amounts of paper that had to be processed in all kinds of organizations and companies. The development of OCR started shortly after the creation of the digital computer [31]. The first application of OCR was GISMO, a robot reader and writer created by M. Sheppard [59]. Commercial releases did not become available before 1967, when companies such as IBM started marketing OCR systems. Because they were very expensive, usage was limited to large companies and government agencies [59].

In 1960, Murray Eden, a researcher at the Massachusetts Institute of Technology (MIT) put forward the idea that all characters of the Latin script can be formed by 18 strokes, each of which can be generated from a subset of four strokes, or segments [12]. This idea, known as the “analysis-by-synthesis method”, contributed to a great deal of research and offered the formal proof that all handwritten characters are formed by a finite number of schematic features. This notion was later used in all methods in syntactic pattern recognition applied to character recognition [31].

When in the mid 70s digitizer tablets became available, the pen-tip position could be measured and new forms of handwriting recognition and applications for it, became available. Another technique that added a lot of possibilities was that of Electronic Paper units, produced by the National Physics Lab in England. This was an integrated plasma display and digitizer, connected to a 68000 processor. These techniques have evolved into the so-called Tablet PC [6, 46].

Today, handwriting recognition is widely used (see Section 2.1). Since users are not satisfied with the recognition rates [48] and because new fields of application are invented, research continues and new techniques are still being developed[10]. The next subsections will discuss two available techniques.

2.3.1 Feature space

One way to find out which symbol some unknown shape represents, is to find characteristic properties or features of the shape and compare them to a known list of such properties. One feature could for example be the height-width ratio. Usually, for the letter *d*, this ratio is higher than it is for the letter *e*. For different representations of characters, different feature extraction methods are designed to optimize the features extracted from the data [54]. Most recognitions systems have a list of distinguishing features for each symbol. Recognition of a handwritten character, amounts to computing the relevant features and matching them to the list. The system then decides which of the symbols is most similar to the input character, based on the differences between the features, and outputs the name of that symbol.

The quality and combination of the chosen features, the feature extraction method and the comparison method are what make or break this system. Different feature extraction methods are designed for different representations of the characters, such as solid binary characters, character contours or gray level sub images of each individual character [54]. Also a combination of different feature extractions can improve the performance of a system. Feature comparison can be done by simply deciding which features are the same and which differ, or calculate a rating based on how much features differ, but more complex ways also exist. Using an artificial neural network for example makes it possible to compare features without having to declare explicit comparison rules [33].

[10]There is actually a group of researchers that do the opposite: they try to find ways to make recognition of handwritten letters as hard as possible for computers, so that a distinction can be made between human readers and computers. These techniques can be used on websites where user registration is needed and where the problem of automatic registration flooding (i.e. computer abusers letting their computers create new accounts over and over again to overload the systems) needs to be prevented [43].

In handwriting recognition, classification is not the only application of features. They, for example, also can be used for preprocessing. They can be used to decide whether a piece of handwriting is a character or a non-character (rubbish or noise) [60], so the classifier gets an easier job.

2.3.2 Prototype based systems

A prototype of any category is an example (not necessarily a member) of that category that represents one or more members of that category, or even the category as a whole [30]. It is a standard or typical example[11]. In the field of handwriting recognition, a prototype usually represents (a part of) all possible ways to write a certain character. Prototypes can be real examples of characters ("in this set of handwritten *a*'s, this particular *a* represents the set best"), averaged samples (see Figure 2.5), or even a prototypical description:

```
   PROTOTYPE_H:
- has one straight vertical line at the far left
- has one straight vertical line at the far right, with the same
length as the other vertical line
- has one horizontal line which starts at the middle of one of
the vertical lines and ends at the middle of the other vertical
line, it is about half as long as the vertical lines are
```

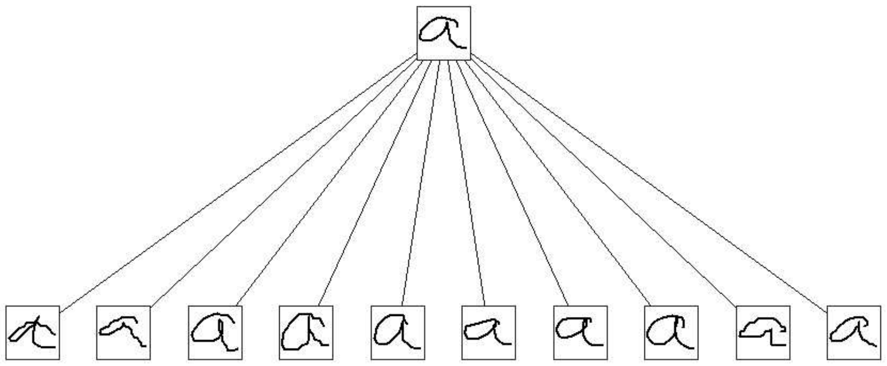


Figure 2.5: An example of prototyping: the character in the top row is an averaged version of the ones in the bottom row. The selection of the samples which are used for the prototype and the averaging algorithm decide the shape of the resulting prototype.

In our classifier, which is prototype based, prototypes are averaged shapes of real handwritten characters (this will be explained in Section 3.3). For a prototype-based classifier to be able to compare a sample to a prototype, and to decide which of the prototypes is most similar to the query, some kind of distance measure is needed. The prototype that is the closest to the sample (has the smallest distance to it) is called the closest match. In a simple prototype-based systems, the label of the closest matching prototype is the classifier decision. In a little more sophisticated versions, a list of the best x matching prototypes

[11] http://www.merriamwebster.com/cgi-bin/dictionary?va=prototype

could be created and some algorithm could decide what the classification would be (for example, it could be decided that the label that appears the most times in the top x prototypes is returned as the final classification). More about the usage of prototypes in our systems will be discussed in Section 3.3.

In a handwriting recognition system using prototypes, it is possible to let the system optimize itself to a specific user. This can be useful if the system is implemented in for example a PDA computer. This can be done by starting with a prototype set that is suitable for a large group of users and, during the use of the system, adapting each prototype to the handwriting of the user: each character that the user writes gives some information about his or her average handwriting style. If the prototypes are reshaped each time new information comes in, after some time, the prototypes will be optimized for recognizing the handwriting of the specific user. An adaptive system like this was built by Vuori et al. [64]. Two of the used techniques (*Dynamic Time Warping* and the *Learning Vector Quantization algorithm*) were used in our recognizer too.

2.4 Human factors issues

The previously discussed techniques all try to make the computer understand human writing by adjusting the system to the writer. To improve recognition results, it is also possible to let the writer adjust to the system. One famous example is the creation of a new script font by Palm Computing, called Graffiti (see Figure 2.6), which was later followed by the Jot system[12]. Another alternative alphabet is implemented in the T-Cube system [58], which also holds a sophisticated user interface.

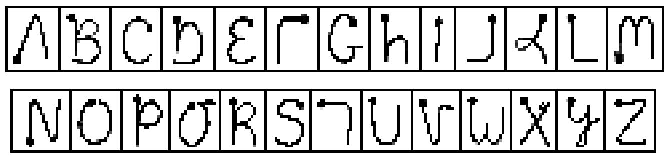


Figure 2.6: Graffiti, an alternative writing system (dots indicate the starting point), was developed for usage on handheld computers. All characters are represented by one stroke, which makes it very easy for the system to segment different characters (to find the place where one character ends and the next one starts).

The adjustment of the user to the system, however, undoes the argument of using handwriting as a more natural way for humans to communicate with computers and thus is not the ideal way to improve automatic handwriting recognition.

The performance of a handwriting recognizer is very important for humans to accept it as an alternative for traditional input methods, like using a device like the computer keyboard [48]. If a user uses handwriting to enter information into a computer and is faced with a system that has a relatively

[12]http://www.cic.com/products/pdf/Jot.pdf

low accuracy and has to recover errors made by the system too often, the natural interaction argument is undone, and the user probably will go back to using a keyboard. This effect can be reduced if the systems makes errors that users can understand. If a user writes the letter *i*, he or she will probably be less annoyed if the system classifies it as being a *j* than when it classifies it as being an *s* or a *p*.

This is where our handwriting recognizer enters the picture. The classifier we built is based on the Dynamic Time Warping algorithm. We claim that this algorithm compares handwritten samples in a way that has results similar to the human handwriting comparing system. This results in a system that has a relatively high recognition performance, but in the case it makes mistakes, does it in a way that is intuitively believable to humans.

2.5 Other domains of handwriting recognition

The domains of handwriting recognition that were described in this chapter, all are based on the 62 Latin characters (uppercase letters [A-Z], lowercase letters [a-z] and digits [0-9]). However, character recognition is not limited to this alphabet: a lot of research has been done in the area of recognizing characters written in, among others, the Japanese [19, 50], Chinese [29], Korean [4, 23] and Arabic [1, 34] alphabet. Also the recognition of non-alphabetical characters or other handwritten shapes has been in the interest of researchers: music notes [3, 15], mathematical formula [13, 22, 27], bathroom designs [65] and the previously mentioned signatures (see Section 2.1.3) are just a number of examples. Our classifier has only been tested with the 62 Latin characters however.

Chapter 3

The classifier

As discussed in the previous chapter, we aimed to create a handwriting recognition system that compares handwritten samples in a way so that the results are similar to the results of a human comparing handwritten samples. Because we believe that the Dynamic Time Warping algorithm, as described by Vuori [64], offers this possibility, we have developed a recognizer based on the algorithm.

This chapter describes the building, tuning and fine-tuning of the classifier. Section 3.1 describes the data that was used for training, testing and validating the system, Section 3.2 describes the used DTW-algorithm, Section 3.3 describes the selection and calculation of two sets of prototypes.

3.1 Data

For the creation, fine-tuning and testing of the two prototype sets that were created (see Section 3.3), the UNIPEN r01_v07-trainset [17] was used. This UNIPEN database, and the accompanying standard, were generated in a large collaborative effort of a wide number of research institutes and industry. The UNIPEN trainset consists of 9 different sets, containing isolated characters, words and texts, which are represented discrete and online (see Sections 2.2.1 and 2.2.2). See Table 4.5 for a specification. Because we decided to limit the classifier to the recognition of only the 62 Latin characters, we only used the isolated digits (set *1a*), lowercase letters (set *1b*) and uppercase letters (set *1c*). As will be described in this section, some preprocessing had to be done to make the dataset useful for the creation of our classifier, and the accompanying prototypes.

Each of these sets was cleaned (see next section) and randomly divided over three subsets of equal size. These subsets were called *trainset*, *testset* and *validationset*, based on the function they were to have in the creation of our classifier (see Figure 3.1 for a schematic view on the data arrangement). In the process of building, testing and using the classifier, the subsets were held separately at all times: no two subsets were ever used in the same process. For example, a lowercase letter from the *trainset*, was neither compared to any uppercase letter, nor to a lowercase letter from the *validationset*, simply because the subsets were strictly separated (note, however, that the prototypes that were created from the *trainset* at a later stage, were not limited to this separation).

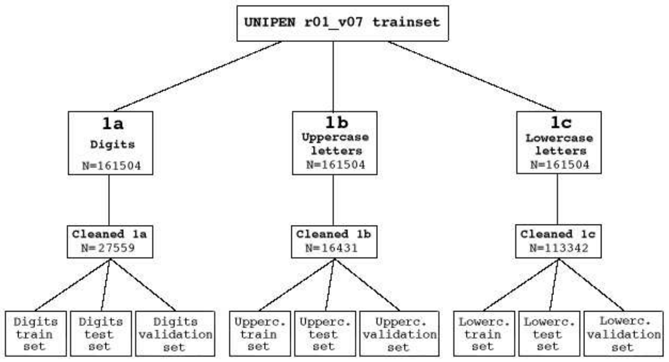


Figure 3.1: Schematic view on the data arrangement. The samples in each class that was taken from the UNIPEN r01_v07 dataset (digits, uppercase letters, lowercase letters) were cleaned and randomly divided over three subsets of equal size.

3.1.1 Cleaning the data

As is specified by the UNIPEN standard, every sample in the UNIPEN set is labeled with the character, word, or text it represents (e.g., a sample labeled “a” is supposed to represent a handwritten lowercase letter 'a'). Because the UNIPEN r01_v07-trainset is known to contain a relatively high number of errors (Hu [18] estimated the number of errors in the set to be about 4%), it was necessary to clean up the data. Three kinds of errors could be distinguished: labeling errors, case errors (a special kind of labeling errors, where lowercase and uppercase labels are mixed up) and segmentation errors. The errors are illustrated in Figure 3.2. The cleaning up was done by temporarily dividing the sets of each of the classes in either twenty-six (in case of the letters) or ten (in case of the digits) smaller sets: one for each character. Using the UNIPEN displayer `upview` (a program contained in the `uptools` distribution [70] for fast visualization of large amounts of UNIPEN data), visual representations of the samples in every resulting smaller set were offered to a human expert, who decided for each label whether or not it was correct (see Figure 3.3 for an example). The only, and very simple, criterion was: would a human reader possibly classify the sample in agreement with the label? If the answer to this question was “yes”, the sample was accepted, if it was “no”, the sample was rejected (note that no information about the kind of error was used: all that mattered was if there was an error or not). After the cleaning up, the data for each of the classes was randomly divided into three subsets of equal size, which were called *trainset*, *testset* and *validationset*. After dividing, we noticed that there were some duplicate samples in each of the sets. They were removed and the resulting nine datasets (for each of the three classes, there was a *trainset*, a *testset* and a *validationset*) were accepted as the final datasets. These datasets were divided into either 26 (in the case of letters) or 10 (in the case of digits) subsets. The resulting data arrangement is visualized in Figure 3.1.

Tables A.1, A.2 and A.3 show detailed specifications of how many samples remained in each of the subsets, and how many were removed in the cleaning and duplication removal. Note that the duplicate samples were removed after the division in the three subsets. This explains the differences in the sizes of the subsets.

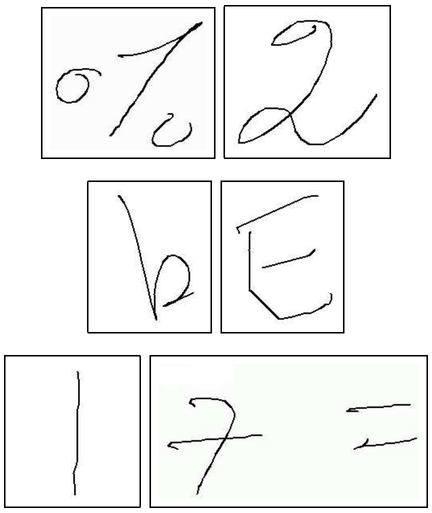


Figure 3.2: Examples of samples that are erroneous in some way. Samples like these were removed from the three data subsets before they were used for the creation, fine-tuning and testing of the classifier. The samples in the top row are examples of labeling errors. The left character was labeled an *8*, while it is a *%* (note that this character should not even be in the dataset!). The right character was labeled a *Q*, while it is a *2* or maybe a *z*. The samples in the middle row are examples of case errors (lowercase letters are labeled as being uppercase, or vice versa). The left character was labeled a *B*, while it is a *b*. The right character was labeled an *e*, while it is an *E*. The samples in the bottom row are examples of segmentation errors. The left character was labeled a *T*, but possibly is a *1*, *I* or a part of any character that has a vertical line. The right character was labeled a *7*, which it is, but with noise (possibly a part of the next character) on the right side of it.

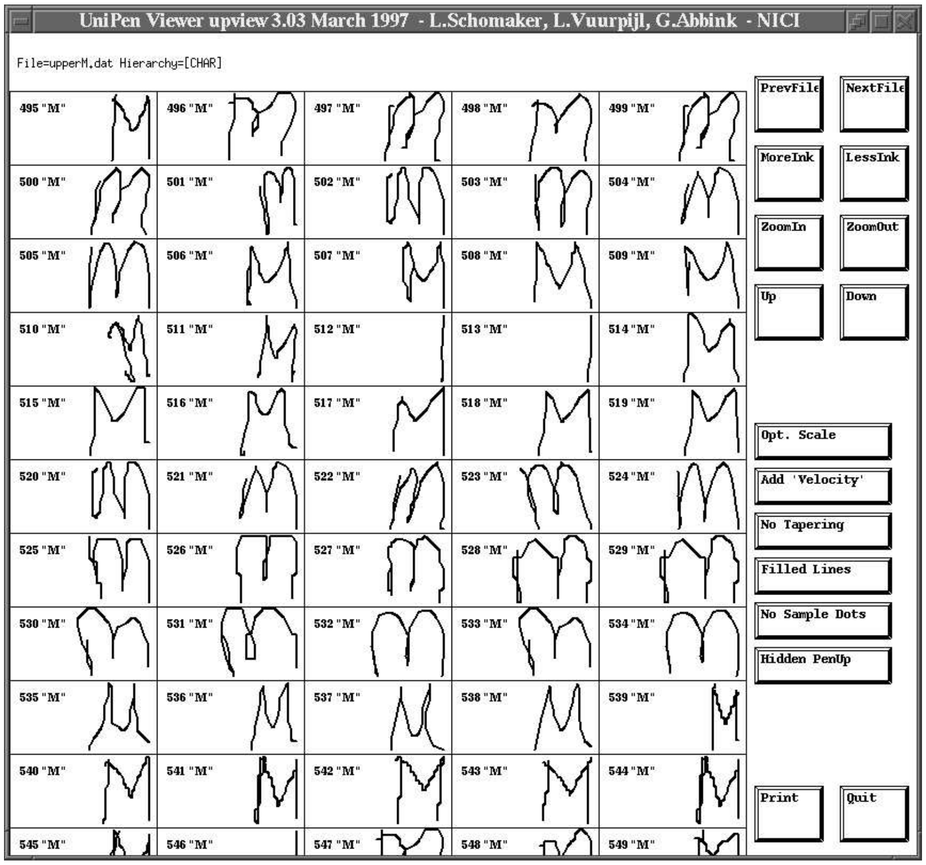


Figure 3.3: Screen shot of `upview`, the program that was used for cleaning up the data. Visual representations of all samples were offered to a human expert who had the task to inspect the representations and remove incorrectly labeled samples from the dataset. Here, samples 512 and 513 are examples of either labeling (samples represent an *I*, *1* or *i*) or segmentation errors (only a part of an *M* is shown). These samples were removed from the dataset.

3.2 Dynamic Time Warping

The DTW-classifier developed in this thesis is a prototype-based classifier. This means that an unknown sample, or *query*, offered to the system is compared to all prototypes in the system. The classification is based on the ranking of the prototypes by the distance to their query. It could for example be decided that the query is classified as having the same label as the nearest prototype. Other possible classifications based on the nearest prototype exist and some will be reviewed later in this section.

Essential for a prototype-based system is the used distance measure. One (very basic) example of a distance measure is the difference between the number of points of the two curves. The nearest prototype would then be the prototype that has about the same number of points as the query has. This difference measure is very trivial and probably useless in practice. A lot of more complex methods to calculate the distance between two curves exist, and *Dynamic Time Warping* (or DTW) is one of them.

This section will begin with an introduction to matching paths and a short introduction to the development of DTW. This will be followed by an extensive description of the algorithm, a comparison to two related, but less complex distance measures and a description of the various available options of the system and it will conclude with an overview of the chosen options and adjustments that were made to the algorithm to optimize its performance.

3.2.1 Matching paths

For Dynamic Time Warping to be able to compare two curves, they need to be represented discrete and online (see Sections 2.2.1 and 2.2.2). The DTW-implementation described here uses a special file format, called HWR. A curve is represented by an ID, of which the first character is the label (hence a curve with the ID "A002608" (which means that the sample is the 2608th sample of the uppercase A's) is labeled an uppercase *A*), followed by a number specifying the number of points. Then the points (each consisting of an X, Y and Z-coordinate) are listed, in the order they were produced: the first listed point is also the starting point of the character. Multiple characters can be represented in one file. Because the used data was represented in the UNIPEN format, a conversion was necessary. This was done using the `upread2` tool from the `uptools` distribution [70]

A matching path technique basically follows the next steps to calculate the distance between two curves:

1. A matching path needs to be created. A matching path is a list of combinations of the points of the first curve and the points of the second curve. The technique used for the creation of this list is what distinguishes different matching-path-using methods, of which DTW is one. The trick is to find a matching method that creates a matching path that is useful for classifying allographs. A detailed description of the creation of the matching path by the DTW-algorithm will follow below.

Table 3.1: The matching path shown in Figure 3.4.

Curve 1	Curve 2
0	0
1	2
2	3
3	5
4	7
4	8
5	8
5	9
6	11
6	12

2. For each of the combinations of points i, j in the matching path, the distance D(i,j) between them is calculated. Various methods for calculating the distance between two points exist. This DTW-implementation uses Euclidean distance (see Equation 3.1).

3. The distances calculated in the previous step are summed and this total distance is normalized by divided it by the number of combinations in the matching path. The resulting value is the distance between the curves.

The Euclidean distance between two points $P = (x_1, y_1, z_1)$ and $Q = (x_2, y_2, z_2)$ is calculated as[1]:

$$|PQ| = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \tag{3.1}$$

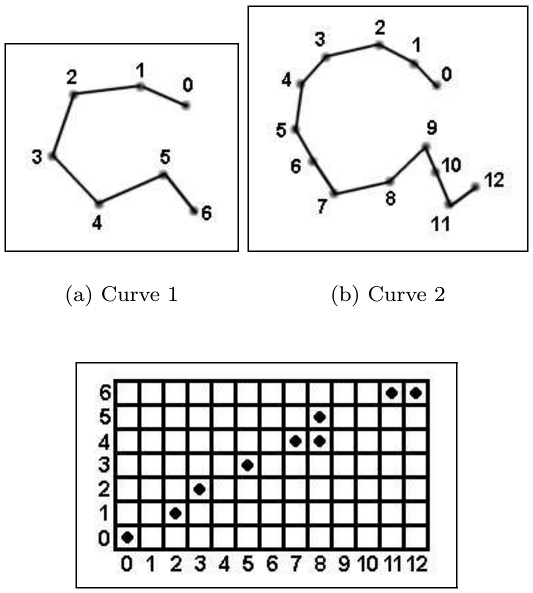
As can be seen, the z coordinates are not used for distance calculation. As will become clear later, the so-called *penup/pendown condition* of the Dynamic Time Warping algorithm (see Section 3.2.3) does use the z coordinates.

Graphical representations of matching paths will be used in the rest of this section. An example of such a graphical representation can be found in Figure 3.4 which shows the graphical representation of the matching path shown in Table 3.1.

3.2.2 History of DTW

In the year 1983, Joseph Kruskal and Mark Liberman [24] introduced a new technique to compare two curves (calculate the distance between them). The technique, that they called *time warping*, made it possible to make a matching between two curves that are “subject not only to alteration by the usual additive random error but also to variations in speed from one portion to another”. DTW can distort (or, as Kruskal and Liberman call it, “warp”) the time axis: it compresses it at some places and expands

[1]http://en.wikipedia.org/wiki/Euclidean_geometry



(a) Curve 1

(b) Curve 2

(c) Matching path

Figure 3.4: Example of a matching path. (c) shows a possible matching path of the curves shown in (a) and (b). Curve 1 is on the vertical axis, curve 2 is on the horizontal axis of the matching path. This is a visual representation of the matching path of Table 3.1.

it at others. This means that variation in writing speed, which can be seen as noise, are deleted, or at least decreased. A feature of DTW that is useful for the field of handwriting recognition is it being able to handle curves of unequal length [40]: curves that consist of a different number of points. This allows comparison without resampling. Because resampling usually deletes information or adds “false” information (i.e., a curve consisting of more points appears to contain more information than a curve consisting of less points, while in the case of a resampled curve, this is not necessarily the case), it is better not to use it.

Kruskal and Liberman recommended time warping for use in speech recognition, since speech is very sensitive for variations in time, but the technique has proved to be useful for many other applications like gesture recognition [16], robotics [44], data mining [21] and, as will be shown in this thesis, handwriting recognition [41, 64].

3.2.3 The DTW-algorithm for computing a matching path

As described in the previous section, the creation of a matching path between two curves is the most important part of the comparison of two curves: it determines which points match and are to be used to calculate the distance between the two curves.

One way of creating a list of matching points is called *linear matching.* Plain linear matching can only be used if the two curves are of equal length (have the same number of points): every ith point of the first curve matches with the ith point of the second curve (see Figure 3.5). A little more sophisticated version of this matching technique makes it possible to match curves with a different length: If N_1 and N_2 are the number of points in the two curves, the ith point of curve 1 and the jth point of curve 2 match if

$$\frac{i-1}{N_1} * N_2 \leq j \leq \frac{i}{N_1} * N_2 \tag{3.2}$$

In both the plain as the more sophisticated version of this technique, each point can match with no more than one point of the other curve.

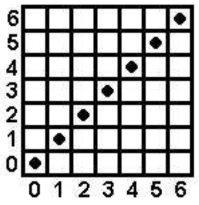
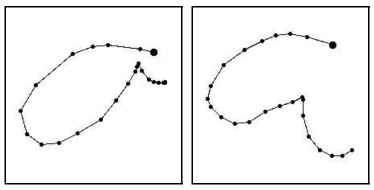


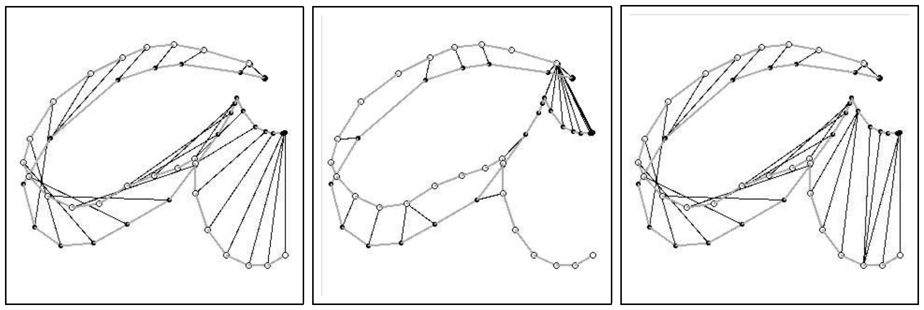
Figure 3.5: Matching path between two curves of equal length, created by plain linear matching.

Another technique is *complete matching.* This simple, but computationally expensive technique calculates the distance between every point of the first curve and every point of the second curve. For every point the smallest distance to the other curve is decided. These distances are summed and divided by the number of points. Each point can match with no more than one point of the other curve.

As can be seen in Figure 3.6, and more generally in Figure 1.1, the results of DTW-matching can generate results that are more intuitive than that of either linear or complete matching. Linear matching is very straightforward and cannot cope with speed variations, which can be seen at the end of the curves: the DTW-match, which does use spatial and time information, shows a more intuitive match (i.e. points that are more obviously belonging together are matched by DTW, but not by linear matching). Complete matching shows a result that is not very intuitive either: the end of curve 1 is matched with the beginning of curve 2, simply because this is closer to it than the end of curve 2. Like linear matching, the complete matching technique also cannot cope with speed variations: it ignores information about the order in which points are created.



(a) Sample 1 (b) Sample 2



(c) Linear (d) Complete (e) DTW

Figure 3.6: Examples of different matching techniques. The curves shown in (a) and (b) are matched. (c) Linear matching (the ith point of curve 1 matches with the ith point of curve 2). (d) Complete matching (the ith point of curve 1 matches with the closest point of curve 2). (e) DTW-matching (the ith point of curve 1 matches with the closest point of curve 2 that DTW allows it to match with).

Dynamic Time Warping constructs a matching path in a more complex way. It is based on linear matching, but has four conditions or constraints (of which three are optional) that need to be satisfied. These conditions are called the *continuity condition*, the *boundary condition*, the *monotonicity condition* and the *penup/pendown condition*. As said, three of the conditions are optional and they all have a number of options that can be changed. In Section 3.2.5, a list of the used conditions and values of the different available options that were used for the final implementation will be given.

Continuity condition

The *continuity condition* decides how much the matching is allowed to differ from linear matching. This condition is the core of the Dynamic Time Warping and thus is not optional. The condition is formulated by Vuori et al. [64] as: If N_1 and N_2 are the number of points in the first and second curve, respectively, the ith points of the first curve and the jth points of the second curve can be matched if (note that the other conditions can bypass this if-statement)

$$\frac{N_2}{N_1}i - cN_2 \leq j \leq \frac{N_2}{N_1}i + cN_2 \tag{3.3}$$

The c parameter determines the amount that the matching is allowed to differ from linear matching. If c=1, the continuity condition has no effect (every points of the first curve matches with every points of the second curve), if c=0, the resulting match is exactly the same as linear matching. Figure 3.7 shows how this condition puts a constraint on the matching path.

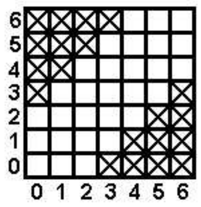


Figure 3.7: Continuity condition. In this matching path, the matches corresponding to the crossed squares in the path are prohibited by this condition (when $c = 0.4$).

Boundary condition

The *boundary condition*, if turned on, forces a match between the first points of the curves and a match between the last points of the curves. If this condition is turned on, the named points are matched, whether the *continuity condition* agrees or not and whether the points are nearest or not. Figure 3.8 shows how this condition puts a constraint on the matching path.

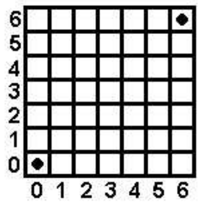


Figure 3.8: Boundary condition. The first points of the two curves and the last points of the two curves are forced to match.

Monotonicity condition

The *monotonicity condition* is a condition that prevents the matching from "going back in time". If at some point in the matching process it is decided that the ith point of the first curve matches with the jth point of the second curve, it is not possible for any point of the first curve with index $> i$ to match with a point of the second curve with index $< j$th and for any point on the first curve with index $< i$th to match with any point on the second curve with index $> j$. Figure 3.9 illustrates this condition.

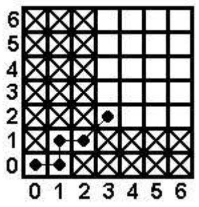


Figure 3.9: Monotonicity condition. After the matching of the 5 shown point combinations, the matches corresponding to the crossed squares in the path are prohibited by this condition.

Penup/pendown condition

This condition is an addition to the DTW-algorithm that our classifier is based on [64]. We have implemented it because, as will be shown later, we believe that the condition may improve both the recognition performance and the "intuitivity" of the classifier. The *penup/pendown condition* ensures that pen up points of the first curve (points of which the z coordinate, which represents the pen pressure > 0) can only match with pen up points of the second curve, and that pen down points of the first curve (points of which the z coordinate $<= 0$) can only match with pen down points of the second curve. Figure 3.10 illustrates this condition.

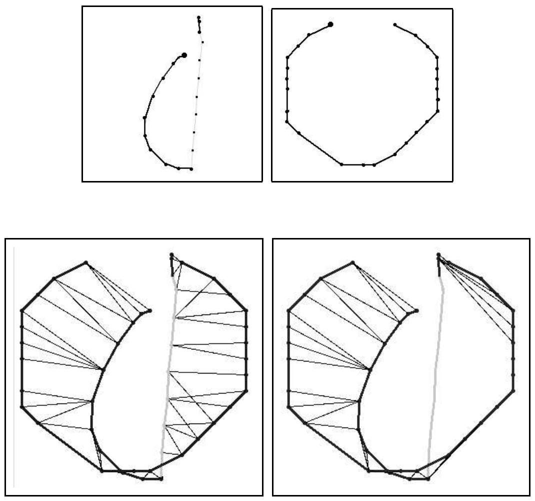


Figure 3.10: Penup/pendown condition. Figures c and d show the matching of the allographs in figures a and b, with the condition turned off and on, respectively. As can be seen, the matching with the condition turned on is more natural than the matching with the condition turned off. Using the condition, the classifier can make a better distinction between allographs that have or do not have pen up points.

The algorithm in pseudo code

Now, when our Dynamic Time Warping algorithm is applied to a combination of two curves (called $curve_1$ and $curve_2$), the following heuristic (in pseudo code) is applied to calculate the DTW-distance between them:

```
create empty matching path;
total distance = 0;
for every i (N1) {
  smallest distance = +infinite;
  for every j (N2) {
    if (boundary condition = on) AND
           ((i=0 AND j=0) OR (i=N1 AND j=N2)) {
      add i-j combination to matching path;
      add distance between i and j to total distance;
    }
    else {
      if (i matches j /*satisfying the combination of all used conditions*/) {
        if (distance i-j < smallest distance) {
          smallest distance = distance between i and j;
        }
      }
    }
  }
  if (i-j combination with smallest distance is not in path yet) {
    add i-j combination with smallest distance to matching path;
    add smallest distance to total distance;
  }
}
/*The next part of the heuristic guarantees that the DTW-distance
  is symmetrical*/
for every j (N2) {
  smallest distance = +infinite;
  for every i (N1) {
    /*if the boundary condition is on, it already is
      satisfied by the first part of the heuristic*/
    if (j matches i /*satisfying the combination of all used conditions*/) {
        if (distance j-i < smallest distance) {
          smallest distance = distance between j and i;
        }
      }
    }
  }
  if (j-i combination with smallest distance is not in path yet) {
    add j-i combination with smallest distance to matching path;
    add smallest distance to total distance;
  }
}
divide total distance by number of matches in matching path;
```

3.2.4 Decision algorithm

When a query is offered to the classifier (which uses a prototype set called *protos*), the following heuristic (in pseudo code) is applied (*top* is a user specified value that decides how many of the prototypes (counting from the one with the smallest distance to the query) are used by the algorithm):

```
for each p (prototypes) {
  calculate and remember DTW-distance between query and p;
}
sort all prototypes by their distance to query;
make list of characters occurring in top number of prototypes;
for each q (prototypes in top) {
  decide the character of the prototype;
  calculate weight of prototype using the decision algorithm;
  add weight to total weight of character;
}
return character with highest weight;
```

As can be seen here, the decision algorithm is important for the last phase of the classification: it is applied to the sorted list of prototypes. We implemented five different algorithms, which are described in Table 3.2. Combination of algorithms is not possible. Each of the decision algorithms makes a list of the characters occurring in the *top* number of prototypes and decides a weight for each of those characters. The character with the highest weight is returned as the classification. Table 3.2 shows the formula's that every decision algorithm uses to calculate the weights of a character. For every prototype in the *top*, a value is calculated that is added to the weight of the corresponding character.

The five implemented algorithms can be split into three groups: the first group does not use the order in which the prototypes occur in the top list, the second group uses that order, and the third group indirectly uses the order, by using the distances between the query and the prototypes. For the second and third groups, the closer the prototype is to the query, the heavier the weight: in the case of the third group, the distance to the query plays a direct role in the weight of a prototype, in the second groups, the order of the prototypes is used, and the distance only plays an indirect role in the weight calculation.

For groups 2 and 3, two different decrease methods are implemented: a linear one (the weight decreases in a linear fashion) and a sinusoidal one (the weight decreases slowly at the beginning and ending, and less slowly in the middle). Figure 3.11 shows the two decrease methods.

3.2.5 Options and adjustments

As described in the previous section, a number of different options was implemented in the DTW-classifier, to be able to try different combinations of settings to construct the optimal classifier. The "ultimate combination of options" had to be clear before the creation of the prototypes, because a working classifier was needed for that. For this, an artificial, manually created set of lowercase letter

Table 3.2: The five implemented decision algorithms. *top* is the number of used prototypes, x is the ranking number ($x \in [1, top]$), *dist* is the distance between the prototype and the query, MAXdist is the distance between the query and the furthest prototype. The underlined letters explain the abbreviations used in Table 3.3.

Group	Name	Implementation
1 (No order)	Number of occurrences	1
2 (Order)	Linear occurrences	$(-0.9/top) * x + 1$
	Sinusoid occurrences	$.5 + .5 * (cos(\pi/(top + 1) * x))$
3 (Distance)	Linear distance	$MAXdist - dist$
	Sinus distance	$(.5 + .5 * (cos(\pi/(top + 1) * x))) * 1/dist)$

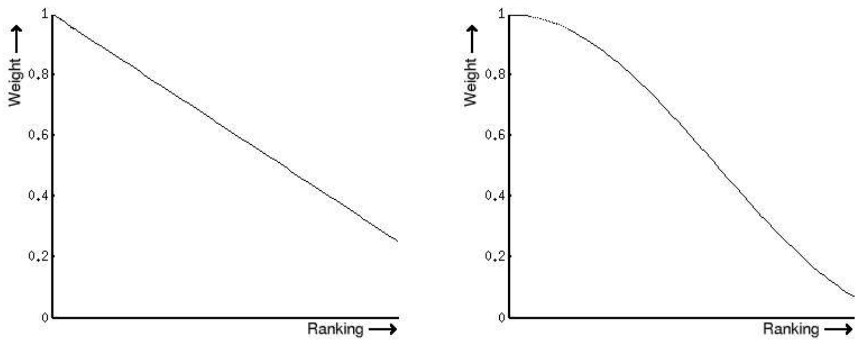


Figure 3.11: The two decrease methods used for the decision algorithms. The left figure shows the linear decrease and the right figure shows the sinusoidal decrease.

prototypes, extracted from the *trainset*, was used for classifying a random selection of lowercase letter samples from that same *trainset*. Several different combinations of options were tried, and the results are shown in Table 3.3. As this table shows, the best *top value* was 1. This means that the choice of the decision algorithm has no effect: they all generate the same result. Also, it shows that it does not matter whether the boundary, monotonicity and penup/pendown condition are turned on or off. We chose not to use the monotonicity condition, because it substantially slows down the recognition process and because we do not think it adds any "intuitivity" to the recognition: the condition, that was developed for speech recognition, makes sure that no "going back in time" is possible. We believe that in contrast with speech recognition (in which the signal can only go one way), in handwriting recognition (in which time plays a different role) the prevention of going back in time is not very intuitive. The boundary condition was turned on because we believed that the chance that it would have a positive effect on the recognition performance would be bigger than the chance that it would have a negative effect (although, as shown, this is not proved by the test). We decide to turn the penup/pendown condition on because we believe that humans make a distinction between penup and pendown information for comparing handwritten shapes too (e.g. whether some piece of the written shape is visible or not, see Figure 3.10 for an example).

As Table 3.3 shows, one combination generated the best results (however, as said before, it did not matter whether the boundary, monotonicity and penup/pendown condition were turned on or off). For all applications described in this thesis, including the further development of the classifier, the next combination of options was used:

```
c value = 0.13
top value = 1
Decision algorithm = not applicable
Boundary condition = on
Monotonicity condition = off
Penup/Pendown condition = on
```

Table 3.3: Classifier options. Many more combinations were tried, but only the relevant ones are shown here. The combinations that are not shown, generated performances that are comparable to, or much worse than, other shown combinations. The abbreviations used in the *Decision algorithm* column are explained in Table 3.2. The shaded line shows the combination of options which resulted in the best recognition performance.

c value	top val.	Dec. alg.	Bound. cond.	Mono. cond.	Penup/dwn cond.	Recog. perf. (%)
0.11	1	-	off	off	off	70.17
			off	off	on	70.17
			off	on	off	70.17
			off	on	on	70.17
			on	off	off	70.17
			on	off	on	70.17
			on	on	off	70.17
			on	on	on	70.17
	5	n	on	off	on	69.11
		l	on	off	on	68.42
		s	on	off	on	65.79
		i	on	off	on	65.79
		u	on	off	on	65.79
	10	u	on	off	on	68.42
0.13	1	-	on	off	on	75.19
	5	u	on	off	on	72.54
	10	n	on	off	on	70.17
0.15	1	-	on	off	on	68.42
	5	l	on	off	on	66.67
	10	s	on	off	on	64.91

3.3 Generating prototypes

As said, our DTW-classifier is a prototype-based classifier, which means that every to-be-classified sample is compared to a number of prototypes, on which the classification is based. A prototype represents a group of samples. The prototypes we generated are in some way the average of a number of similar samples. The decision of which shapes are similar and of how the average of a set of samples is calculated are crucial for the to-be-created prototype set. For this thesis, one selection of similar shapes was created, and two different averaging algorithms were applied on this selection, resulting in two different prototype sets (called *Resample and Average* and *Mergesamples*). This section describes how these sets were generated. Section 3.3.1 describes how the data was prepared by dividing it into subsets. Section 3.3.2 describes how the subsets were clustered, and how a manual selection of to-be-generated prototypes was taken from those clusterings. Finally, Section 3.3.3 describes the two algorithms that were used to create the prototypes from the manual selections.

3.3.1 Dividing data in classes

The *trainset*, described in Section 3.1, was used for the prototype generation. The *trainset* was divided into 62 subsets: one for each character (26 lowercase letters, 26 uppercase letters and 10 digits). Tables A.1, A.2 and A.3 show how many samples were in each of the subsets. The procedure for clustering, selecting and creating prototypes was executed for each class separately, so that 6 sets of prototypes would be the result: two different algorithms applied to each of the three classes. This separation of the three classes was necessary because we wanted our classifier to be able to work on each of them separately (e.g. when classifying a lowercase letter, only the lowercase letters prototype sets should be used).

3.3.2 Clustering and selecting

One selection of groups of similar allographs was made. This selection was used as a basis of both created prototype sets. The *trainset* was used for this process. This set was divided into three subsets, one for each allograph category. These three subsets were divided into 26 (in the case of letters) or 10 (in the case of digits) smaller subsets. For quickly finding similar shapes, a hierarchical clustering, based on the DTW-distance, was created. To be able to do this, for each letter and each digit, a matrix containing the DTW-distances between all the combinations of samples of that letter or digit was created.

These (symmetrical, since the distance between sample A and sample B always equals the distance between sample B and sample A) matrices were used for the clustering. By basing the clustering on the DTW-distance, we were certain that each of the resulting prototypes was based on shapes that are similar according to the DTW-distance. Visual inspection of the clustering showed us that the results were also visually believable (i.e., a human handwriting expert would judge the prototypes to be a good representation of the samples it was based on): a small hint in the direction of the validation of the claim that DTW can generate results that are close to human judgment. As said, for each letter or digit, a clustering was created (see Figure 3.12 for a representation of a sample classification).

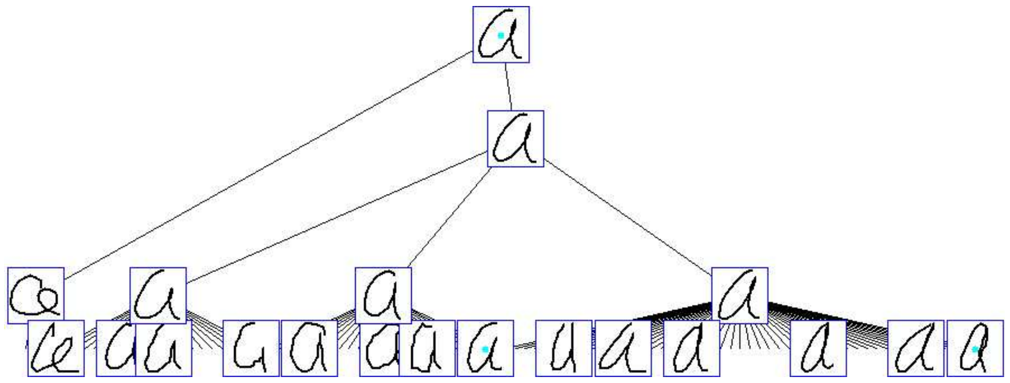


Figure 3.12: Part of the clustering of the lowercase letter *a* testset. Every node represents an average of the nodes below it. The nodes on the bottom line represent real handwritten allographs. Note that the shown averages are not based on one of the two used averaging algorithms, but are based on an internal method of the visualization application, as described in [67].

The matrices created in the previous step were offered to an implementation of a clustering technique that is a variant of agglomerative hierarchical clustering techniques. This technique (described by Vuurpijl and Schomaker [67]) that was designed especially for clustering handwritten allographs, makes it possible to generate a clustering that is hierarchical and non-binary. This means that it is possible that one node in the clustering tree contains more than two members. This, as it is called, n-ary clustering produces results that give a better reflection of the real distribution of samples than a standard binary clustering method does. A visualization of a small part of the clustering of lowercase letter *a* can be seen in Figure 3.12. This visualization of the clustering was created using the application `xnary_tree`, a component of the HCLUS-software[2], that was also used in the next step: the selection of relevant clusters.

As said, the clusterings that were produced, were used to make a manual selection of clusters that, in the opinion of the human handwriting expert that did the task, could make good prototypes. In an `xnary_tree` visualization of a clustering, it is possible to click a node and add it to the list of to-be-generated prototypes. This list contains the name of the node, and the names of the real handwritten allographs that are the members of that node. The program also has the option to remove specific samples from the list. This can be of use in cases where a certain cluster contains one or more samples that are not acceptable for the human expert. This could for example happen if a bad sample slipped through the cleaning up described in Section 3.1.1, or if for some reason the DTW-algorithm has made a decision that is not accepted by the human expert. Because the clustering was based on DTW, and very visually believable, the possibility to delete samples from clusters was hardly ever used.

[2]http://hwr.nici.ru.nl/∼vuurpijl/hclus/index.html

Because the used clustering algorithm has a number of options which can produce very different results (see Figure 3.13), several setups were tried. For each of the letters and digits, different clusterings were created and one of each was used for prototype selection. Which of the clusterings was used, was based on the judgment of a human expert: all to-be-used clusterings had roughly the same shape and the prototypes that were to be generated were good representations of the set. Because the decision about which clustering to use was taken before the selection of the prototypes, it was very hard to make an objective decision.

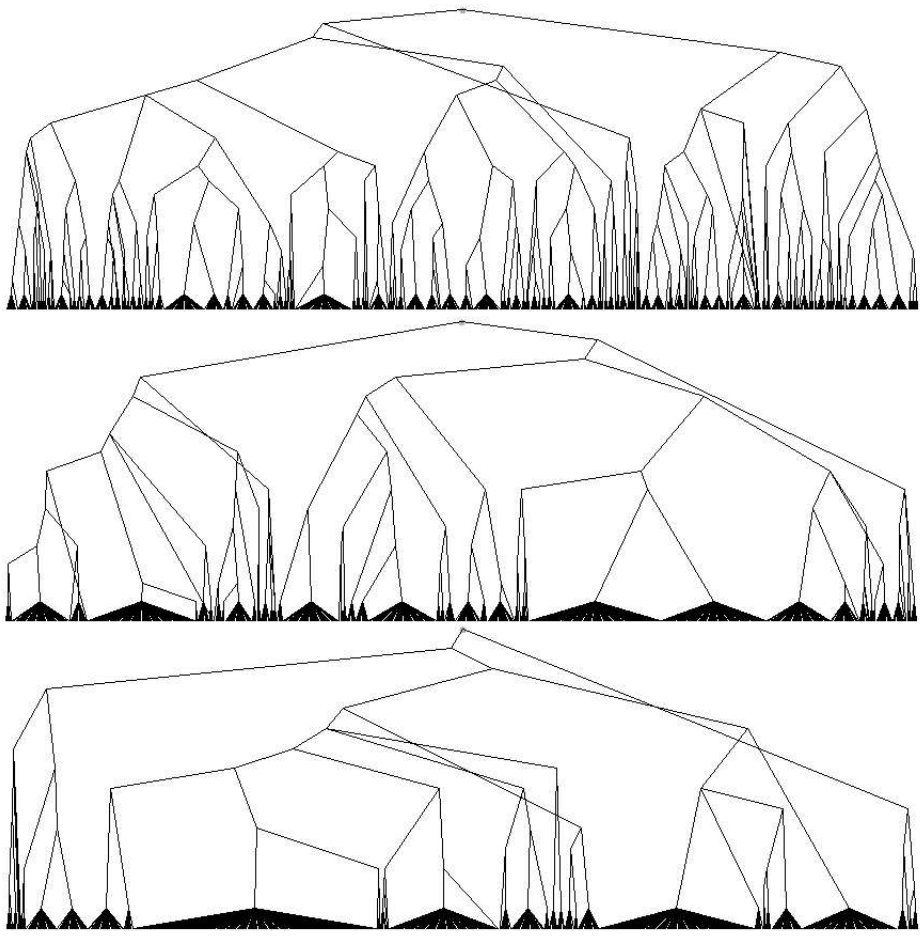


Figure 3.13: Clusterings of the same data subset, but with three different setups. It can be seen here that the rough shapes and the sizes of the clusters at the bottom level are very different. This does not necessarily mean that the clusters differ concerning content (for example, it is possible that all high level clusters in the top figure are present at a lower level in the other figures), but because prototypes are generated from the lowest level, these different clustering results guarantee that different prototypes are generated.

After a clustering was made for each letter and digit, the selection of prototypes started. Manually, each bottom level cluster was visually inspected. The human decided whether the combination of samples would turn into a useful prototype. The criteria were: the member samples look very similar and their average could turn into a good representation of real handwritten shapes. Because judgment was done by human inspection, the decision measure was not formal and therefore not very objective. However, because the inspector was acquainted with human writing, the results were expected to be both reliable and useful. The reason for taking into account only bottom level clusters (and occasionally one level higher) was to get a large set of prototypes, each of which is based on a relatively small amount of shapes. We believed this would produce a better representation of all possible handwritten allographs.

The final results of this step were 62 (one for each letter or digit) lists containing prototype names, each containing the names of their member allographs. These lists were used to calculate averages, or prototypes, as will be described in the next section. Tables B.1, B.2 and B.3 show how many prototypes of each of the letters and digits were selected (and thus created).

3.3.3 Combining samples into prototypes

Every prototype is an average of a set of real written samples that were selected as described in the previous section. Both algorithms described in this section use the same selection. Still, because the two algorithms are very different, the two resulting prototype sets each have their own qualities. The two used averaging techniques are described in this section.

Technique 1: Resample and Average

This technique for calculating an average sample from a number of samples is relatively straightforward. Each sample consists of a series of points (each of which has an X, Y and Z-coordinate). Every prototype-member sample was resampled (using the resampling technique described in [67]) to the same number of points (we will call that number *npoints*). After the resampling, for every coordinate, the average X, Y and Z-coordinate[3] was calculated. The resulting series of *npoints* points is the average sample: the newly generated prototype.

The most important factor for the result of this technique is *npoints*. Three different numbers, of which two were variable (different for each prototype) and one was not, were tried. To test the quality of each tried *npoints* calculation, a small random selection of samples was taken from the *trainset*. For every value of *npoints*, a prototype set was created and a the selection of samples was used for computing the recognition performance of the prototype set.

These are the three different calculations of *npoints* were tried:

1. For each to-be-generated prototype, the average number of points of the handwritten member allographs was calculated. This number was used as *npoints*. Since the member allographs had different numbers of points, this resulted in different values for each prototype.

[3]Note that in all samples $Z - coordinate \in [0, 1]$. Because only points with $Z - coordinate \leq 0$ are interpreted as penup, for a point in a generated prototypes to be interpreted as penup, it needs to be based only on penup coordinates. This results in prototypes that have relatively little penup coordinates (see Figure 3.16 for an example).

2. For each to-be-generated prototype, the average number of points of the handwritten member allographs, and the standard deviation were calculated. *npoints* was calculated by dividing the standard deviation by two, and subtracting the result from the calculated average. Since the member allographs had different numbers of points, this resulted in different values for each prototype.

3. In this setting, *npoints* was always 30, independent of the number of points that were in the member samples.

As the results in Table 3.4 show, the third setting (where *npoints* is equal to 30), although it appears the least sophisticated, on the average gave the best recognition results. This number corresponds to the findings of Schomaker [47] which say that the majority of character shapes contains 5 (or less) strokes and that each stroke can be described by 6 points. Although we do not make use of strokes, this finding justifies the choice for *npoints* = 30. Although the recognition rates do not differ much, the set created with *npoints* = 30 had the best overall recognition results on this small test and we decided to use this set as the *Resample and Average*-prototype set.

Table 3.4: Different values of *npoints*. The datasets used for testing were random selections from the *testset*. The shaded cells shows the best performance for each category.

npoints	Lowercase letters (N=1498)	Uppercase letters (N=1617)	Digits (N=1848)
1	83.91	91.09	96.16
2	84.38	91.28	▨ 96.21
30	▨ 84.71	▨ 91.53	96.10

Technique 2: Mergesamples

This technique for calculating an average sample from a number of samples is more sophisticated than *Resample and Average*. It is a variation of the Learning Vector Quantization algorithm (LVQ), as described in [25, 26]. One of the prototype-member samples was taken and in some order, the other members of that prototype were offered to the algorithm. The algorithm, moved the first sample toward the second one. The resulting sample was moved towards the third, et cetera. This moving is what we call *merging*. The order in which the samples are offered to the algorithm is very important for the result: the first sample is the base sample, which not only has a large influence on the resulting shape, but also decides the number of points of the resulting prototype, and the penup or pendown status of these points (the penup or pendown status of a points in the first sample, independently of the status of that points in the other samples, is preserved in the generated prototype). Also, the amount of the movement of one curve towards the other is an important factor.

For the merging of two samples, first a matching path between the two curves has to be calculated. Because the prototypes will be used for a DTW based classifier, the DTW-matching path was calculated, according to the decisions described in Section 3.2.5. After the calculation of the path, each of the points of the first curve was moved towards the matching points. The direction was equal to that of the vectors between the first curve coordinate and the matching points. See Figure 3.14 for an example of LVQ merging. The amount of the movement was calculated as can be seen in Equation 3.4. The *learningrate* was calculated in one of the three ways that are described below.

$$Movement = \frac{Vector\ length}{Number\ of\ matching\ points} * learningrate \tag{3.4}$$

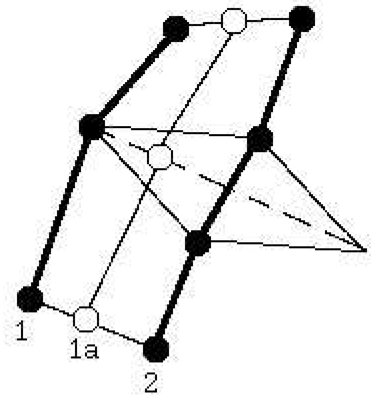


Figure 3.14: Example of LVQ merging. Curve 2 is added to curve 1 by creating a DTW-matching, and moving the points of curve 1 towards the matching points on curve 2. The direction of the movement is decided by the direction of vector (or summed vector in the case where one point of curve 1 matches with more than one points in curve 2). The amount of the movement is decided by the length of the (summed) vector, the number of matching points and the learningrate (see Equation 3.4). Curve 1a is one of the possible resulting curves.

Three different methods were used to decide the learningrate. In all three of them, the learningrate did not change within a sample: for every coordinate in the curve, the learningrate was the same. In one of the methods, the learningrate between samples did change however: the learningrate when merging the first with the second curve was not equal to the learningrate when merging the results of the first two curves with the third curve. It gradually decreased. These are the three methods that were used (the *startingrate* is a user specified value, on which, as will be shown, the *learningrate*s are based, for each of the three methods, three different *startingrate* values were tried):

1. The learningrate did not change between samples. It was equal to the *startingrate* during the complete process.

2. The learningrate did not change between samples. It was equal to the *startingrate*, divided by the number of to-be-processed samples.

3. The learningrate decreased between samples: the later a sample was added, the smaller its influence on the resulting prototype. For every added sample, the *learningrate* was calculated using Equation 3.5 (see Figure 3.15 for a visual representation). Note that the *learningrate* for the first added sample always was equal to $1 * startingrate$.

$$learningrate = \frac{0.9}{n-2} * (1-i) + 1 \tag{3.5}$$

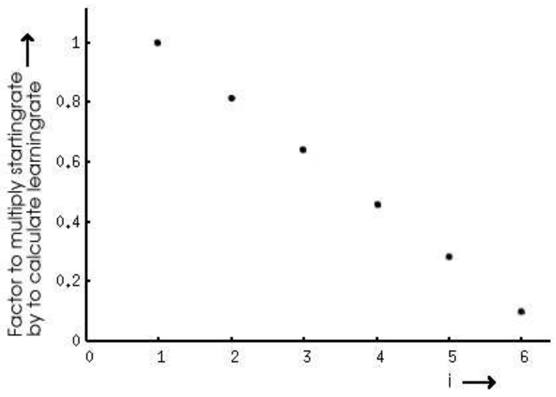


Figure 3.15: A visual representation of the factor that the startingrate is multiplied by to calculate the learningrate, as can be found in Equation 3.5. The learningrate of the first added sample always is $1 * startingrate$, the learningrate of the last added sample always is $0.10 * startingrate$. In this example $nsamples = 7$, and the first sample (with index $i = 0$ is not taken into account, because the first sample is the base sample, to which the other samples ($i = [1, nsamples - 1]$) are added.

As said, the order in which the samples were offered to the systems was important for the shape of the resulting prototype. Three different orders were tried on one random selection of to-be-generated prototypes. These orders, called *Number of points*, *Reversed number of points* and *Difference to average number of points* are listed in Table 3.5. Each of these methods was based on the number of points of the samples[4]. A human expert judged the quality of the resulting prototypes (based on their shape and on how good they represented the samples they were based on) and it was decided that on average, the method called *Difference to average number of points* generated the best results. Therefore, this method was used.

For every calculation variation, three different *startingrate*s were tried. For every combination, prototype sets were created. The results after testing the classifier with each of the created prototype

[4]In future research, one could decide to try other, more sophisticated, methods.

Table 3.5: The different orders in which samples were offered to the algorithm that were tried. *Difference to average npoints* was found to be the most successful.

Name	Description
Number of points	The samples were ordered on the number of points, starting with the sample with the lowest number of points
Reversed number of points	The samples were ordered on the number of points, starting with the sample with the highest number of points
Difference to average number of points	The average number of points of the samples was calculated and the samples were ordered on their distance to that average (e.g. if the average number of points was 50, a sample with 51 points would be offered before a sample with 48 points)

Table 3.6: Recognition performances of different learning rates and different calculation methods. For every class (digits, lowercase letters and uppercase letters), a random selection of about 1500 samples was taken from the *testset* and offered to the classifier using the prototypes created by the different calculation methods. *srate* is the *startingrate*, *nsamples* is the number of to-be-processed samples. The shaded cells shows the best performance for each category.

Calculation method	Starting rate	Lowercase letters	Uppercase letters	Digits
learningrate equals *startingrate*	0.01	86.65	93.14	97.08
	0.05	88.79	94.12	97.40
	0.1	89.19	94.43	97.24
learningrate equals *srate/nsamples*	0.01	83.44	91.71	96.97
	0.05	83.85	91.90	97.02
	0.1	84.11	92.15	96.86
learningrate as in Eq. 3.5	0.01	86.32	92.46	97.02
	0.05	88.05	93.69	97.02
	0.1	88.72	94.37	97.51

sets can be found in table 3.6. Although the performances did not differ a lot, it was decided to use the prototype sets where the *learningrate* was equal to the *startingrate*, and *startingrate* = 0.1 for both letter categories, and the prototype set where the learningrate decreased, and *startingrate* = 0.1 for the digits. In future research, automatic optimizing (see Section 5.2.5) could be used for trying more *startingrate*s or other orders.

3.3.4 Post processing

After the creation of the two prototype sets, as described in the previous section, we removed bad and unused prototypes. This section describes how this post processing was done.

To test the quality of each of the prototypes in the two sets, we offered the *testset* to the classifier and registered for every prototype how many times it was responsible for a correct, and how many times for an incorrect classification. The samples in the *testset* were previously unseen by the classifier: they were not used for the creation of either the classifier or the prototype sets. This means that they can be used as real life examples of handwriting, so that the results of the post processing were optimized for real life applications.

The cleaning up was done by taking the next steps:

1. The complete *testset* was offered to the classifier using either one of the prototype sets. This happened separately for lowercase letters, uppercase letters and digits.

2. For every sample of the *testset* it was decided which prototype was the nearest, according to the DTW-classifier.

3. It was decided whether or not the label of the prototype was equal to that of the sample. For each of the prototypes, a list was created, saying how many times it was responsible for a correct classification and how many times it was responsible for an incorrect classification.

4. After the processing of all the samples, the lists created in the previous step were evaluated. If the proportion T_{proto} of Equation 3.6 was more than 0.5[5] (i.e. if a prototype was responsible for more incorrect than correct classifications), it was removed. It was also removed if it was not responsible for any classification at all. Removal of unused prototypes was done to speed up the recognition process.

$$T_{proto} = \frac{\#\ Incorrect\ classifications_{proto}}{Total\ \#\ classifications_{proto}} \tag{3.6}$$

After applying this removal of "bad" prototypes, the two prototype sets (each consisting of three subsets) were finished. These two sets were used for the applications described in Section 4. A specification of the size of the prototype sets, among with some examples, is given in the next section.

3.3.5 Prototype sets

As said, two different prototype sets were generated. They are named after the technique used to generate them. The first set is called the *Resample and Average* set, the second is called the *Mergesamples* set. Both sets are based on the same manual selection of sets of similar samples, as described in Section 3.3.2.

[5] In future research, it could be interesting to try to either decrease T_{proto} to decrease the number of prototypes in the set (which would increase the speed of classification) without decreasing the recognition performance, or to increase T_{proto} to increase the number of prototypes in the set, and find out if this would increase the recognition performance

After the creation of the sets, they were post processed: bad prototypes were removed (see the previous section). Although the prototype sets were based on the same manual selection, the results were different. Not in the last place because all prototypes in the *Resample and Average* set existed of 30 points, while the number of points of the prototypes in the *Mergesamples* set depended on the average number of points of the samples that were used to generate the prototype. Figure 3.16 shows an example of two prototypes that were based on the same manual selection but generated using different techniques.

Tables B.1, B.2 and B.3 show how many prototypes were manually selected, and how many remained in either of the prototype sets after the post processing had taken place.

It was predicted that the *Mergesamples* set would perform better on various tasks, because no resampling had taken place. Resampling usually deletes information or adds "false" information (i.e., a curve consisting of more points appears to contain more information than a curve consisting of less points, while in the case of a resampled curve, this is not necessarily the case).

Different experiments in the next chapter show whether or not this prediction was correct.

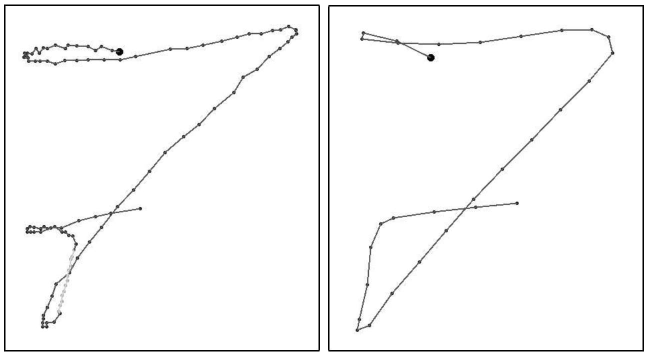


Figure 3.16: These prototypes of the digit *7* were based on the same set of samples, but were generated with two different techniques. The left prototype was created using the *Mergesamples* technique, the right one was created using the*Resample and Average* technique. As can be seen in this example, not only the shape and number of coordinates of the prototypes differ, but there is also a difference in the presence of penup coordinates in the prototypes: in the *Resample and Average* technique a point is regarded as penup only if all points it is based on are penup too, while in the *Mergesamples* technique, a point is regarded as penup if it is penup in the first sample offered to the system.

Chapter 4

Performance and Applications

To test the performance of our DTW-classifier in real applications, we set up two experimental situations and used it in a real application. The results of these tests are described in this section. First, the recognition performance of the classifier was tested. The two created prototype sets (see Section 3.3) were compared to see which one had the best results. Second, an added feature of the classifier is described, followed by a real application that this feature (called *rejection*) was used for. The chapter ends with the description of an experiment in which the claim that DTW produces an allograph comparison that is more intuitively than that of other classifiers is tested. As will be shown, the results of the different experiments and the real life application are promising.

4.1 Recognition performance

4.1.1 Introduction

As described in Section 3.1, the used dataset was split into three subsets. Until this point, only two of them were used: the *trainset* was used for creating the prototypes, the *testset* was used for the post processing of the prototypes (see Section 3.3). The remaining set, called *validationset* was used for testing the performance of the classifier. Important is that the set was kept "secret" or "unseen" to the classifier until this point. This means that the classifier has not been able to adjust itself to this specific set of samples. This makes sure that the found performance is representative for other new samples, and thus that the results of the performance test can be generalized to real-world handwriting classification.

The idea behind this experiment was not only to find out how well the classifier would perform in recognizing real-world handwriting, but also to find out which of the two created prototype sets, *Resample and Average* or *Mergesamples* would give the best results.

Our hypothesis is that both sets generate good results, but that the results of the *Mergesamples* set are better than that of the *Resample and Average* set, because of the next reasons:

- When using the *Resample and Average* set, the classifier is less able to make use of penup/pendown information (due to the averaging of Z-coordinates, a lot of penup information is thrown away in prototype creation).
- Resampling can cause loss of information. This can cause the prototypes to be relatively bad in representing the samples they are based on. Resampling does not take place in *Mergesamples.*
- The *Mergesamples* technique has proved to be useful before [25, 26], while the *Resample and Average* set has not.

4.1.2 Method

The whole *validationset* was offered to both the classifier using the *Resample and Average* set and the classifier using the *Mergesamples* set. 37683 lowercase letters, 16451 uppercase letters and 9195 digits were offered to both systems. The exact number of samples for each letter or digit can be found in Tables 4.1, 4.2 and 4.3. The UNIPEN set is known to be a set of samples that is a good representation of real life (Latin) handwriting.

As we have done every time we used the prototype set, the three categories were tested separately: the lowercase samples in the *validationset* were offered to the system using the lowercase subset of the prototype sets, and the same goes for the uppercase letters and digits. For every letter and digit, and also for the total of each of the three categories, the number of correctly classified samples was counted.

4.1.3 Results

To find out whether or not the performances of the two classifier variations were significantly different, Fisher's Exact Test was used to analyze the data (see Appendix D.1 for the SPSS-syntax). The results of the performance test can be seen in Tables 4.1, 4.2 and 4.3. The tables do not only show the percentage of correctly classified samples for every letter or digit, and for the three categories in total, but also the prototype set that generated the best performance is marked, and the p-value (and whether or not this is significant) is reported.

Table 4.1: Results of the performance test on lowercase letters. The shaded cells in the *Resample and Average* and *Mergesamples* columns show which classifier performed best on a certain character. Whether or not *Mergesamples* performed significantly better than *Resample and Average* can be seen in the *p-value* column. If $p < 0.05$, the difference is significant. This is emphasized by the shading of the cell. In the case where *Resample and Average* performed better than *Mergesamples*, an asterisk (*) is added to the p-value. The *Total* row indicates the summed performance (and accompanying p-value) of all characters.

Character	Number of samples	Resample and average-performance	Mergesamples performance	p-value
a	3088	93.01	93.07	.480
b	776	89.95	90.08	.500
c	1237	93.05	95.23	.013
d	1220	97.70	96.89	.131*
e	3536	90.75	91.97	.038
f	938	85.18	87.63	.069
g	965	83.73	85.28	.189
h	1497	88.98	92.52	.001
i	2361	69.97	78.44	.000
j	578	82.18	88.93	.001
k	720	79.58	82.78	.069
l	1983	91.38	93.09	.025
m	919	91.84	92.82	.242
n	2441	88.90	90.41	.045
o	2304	93.10	94.31	.051
p	1092	95.97	96.25	.413
q	646	88.39	91.95	.020
r	2140	87.62	89.02	.084
s	1999	93.50	94.00	.278
t	2271	85.38	86.31	.197
u	1440	85.62	86.81	.194
v	688	76.31	80.38	.039
w	909	93.40	94.72	.138
x	477	87.63	90.57	.088
y	875	79.66	88.34	.000
z	583	82.16	89.19	.000
Total	37683	88.20	90.32	.000

Table 4.2: Results of the performance test on uppercase letters. The shaded cells in the *Resample and Average* and *Mergesamples* columns show which classifier performed best on a certain character. Whether or not *Mergesamples* performed significantly better than *Resample and Average* can be seen in the *p-value* column. If $p < 0.05$, the difference is significant. This is emphasized by the shading of the cell. The *Total* row indicates the summed performance (and accompanying p-value) of all characters.

Character	Number of samples	Resample and average-performance	Mergesamples performance	p-value
A	774	94.83	97.67	.002
B	629	96.18	96.34	.500
C	654	93.12	96.48	.004
D	581	85.71	92.94	.000
E	1096	91.06	94.25	.003
F	507	64.69	87.77	.000
G	572	90.03	95.10	.001
H	488	79.92	86.48	.004
I	896	93.64	94.20	.346
J	513	81.48	88.30	.001
K	446	83.86	92.60	.000
L	555	93.33	96.58	.009
M	568	94.19	95.95	.109
N	649	91.68	93.68	.100
O	1011	95.25	95.35	.500
P	642	95.79	96.26	.388
Q	438	93.15	96.58	.016
R	741	93.12	94.60	.043
S	747	97.32	98.13	.193
T	841	91.68	94.41	.017
U	625	80.00	87.52	.000
V	577	90.12	91.85	.177
W	507	90.14	96.25	.000
X	433	94.23	96.77	.050
Y	459	93.25	94.77	.202
Z	502	93.23	96.02	.034
Total	16451	90.64	94.28	.000

Table 4.3: Results of the performance test on digits. The shaded cells in the *Resample and Average* and *Mergesamples* columns show which classifier performed best on a certain character. Whether or not *Mergesamples* performed significantly better than *Resample and Average* can be seen in the *p-value* column. If $p < 0.05$, the difference is significant. This is emphasized by the shading of the cell. In the case where *Resample and Average* performed better than *Mergesamples*, an asterisk (*) is added to the p-value. The *Total* row indicates the summed performance (and accompanying p-value) of all characters.

Character	Number of samples	Resample and average-performance	Mergesamples performance	p-value
0	772	95.34	96.37	.061
1	983	93.59	94.61	.195
2	1055	96.49	98.29	.007
3	988	95.85	98.58	.000
4	853	94.72	96.48	.049
5	941	91.82	95.64	.000
6	762	96.72	98.82	.004
7	916	92.47	97.38	.000
8	941	98.72	97.87	.016
9	984	96.85	97.56	.206
Total	9195	95.26	97.16	.000

4.1.4 Discussion

As can be seen in Tables 4.1, 4.2 and 4.3, the *Mergesamples* set performed significantly better than the *Resample and Average* set on the total categories. On a number of specific characters however, the *Mergesamples* set did not significantly perform better than the *Resample and Average* set. In two cases (*d* and *8*), it did even perform worse.

The difference between the characters on which the results differ significantly on the one hand, and the characters on which the results do not differ significantly on the other hand, must be in either the technique that created the prototypes (see Section 3.3.5) or in the post processing (see Section 3.3.4). Since both are very nontransparent, further research has to be done to explain the differences. For now, the fact that for each category, the total difference is significant is enough: it shows that the overall performance of the *Mergesamples* set is better than the *Resample and Average* set. It is also shown that the overall recognition performance of both sets is near or over 90 percent.

It is hard to compare the performance of our classifier to that of other classifiers. Not only because the dataset that was used for testing the performance was randomly taken from the UNIPEN set, and therefore cannot really be compared to datasets that were used for testing other systems (a solution to this problem is proposed in by Ratzlaff [42]), but also because other techniques often use approaches that differ so much from our approach, that a comparison would not be fair: for example, sometimes top down information is used (e.g. in the dScript program[1], a list of all possible words is available. This lexicon is used for providing feedback to the character recognizers, which improves their performance [68]), sometimes only offline data is available [59, 61], and sometimes research is about other than Latin alphabets [1, 4, 19, 23, 29, 34, 50]. Still, because most of the reported recognition performances of different classifiers vary between 85 and 95 percent, we consider the performance of our classifier comparable to that of other systems.

4.2 Rejection

To decrease the number of errors made by the classifier, we introduced a concept to the system called *rejection*. We gave the classifier the possibility to reject a sample after trying to classify it. This rejection was done by letting the classifier decide how certain it was of a classification and letting it reject the classification if this certainty value was not over a specified threshold. This results in a classifier that, on the one hand, is more trustworthy (if it says something about a sample, the chance that it is correct is higher), but, on the other hand, does not accept all offered samples (and thus does not return a final classification), and probably rejects samples that were classified correctly after all (these are called *false rejects*). A classifier using the rejection concept could possibly improve the user acceptance of handwriting

[1]http://hwr.nici.kun.nl/recog/dscript/

recognition systems: a user will probably be less irritated by a system that asks him or her to rewrite a character than by a system that, although it could have known it to be incorrect, makes an error. False rejects (rejected samples that would have been correctly classified) however, increase the total number of samples that have to be rewritten. To optimize the effect of rejection, both the number of false rejects and the number of false accepts (samples that were not rejected, but were classified incorrectly) have to be minimized. As will be shown in Section 4.2.3, the ideal situation has not been reached (if the portion of correctly classified samples in the accepted set increases (the number of false accepts decreases), the size of the accepted set decreases (the number of false rejects increases)), but the practical usage of rejection (in another form, but using the same concept) that is described in Section 4.3, shows that rejection can be a great help in the field of semi-automatic labeling.

4.2.1 Certainty values

Two certainty values, called *agreeness* and *rejectionlist* were implemented in the classifier. Both methods will be described in this section.

Agreeness

This certainty value is calculated using a list of the five[2] prototypes that are nearest to the sample (i.e. have the lowest DTW-distance to the sample). First, the label of the nearest prototype is decided. The certainty value is the number of remaining prototypes that agree with this label. If this certainty is high, it means that the chance that the label of the nearest prototype is correct is high, and that the chance that an incorrect prototype has ended up as being the nearest prototype is small. For example, if the labels of the top five of nearest prototypes to a certain sample are:

```
1. a
2. a
3. o
4. a
5. u
```

Then the corresponding certainty value is 2, since the labels of two prototypes from the numbers 2-5 agree with the label of prototype 1. As can be seen, the certainty value always is an integer between 0 and 4 (inclusive), where a value of 0 represents the lowest certainty, and the value of 4 represents the highest certainty.

[2]A small test was performed to find the optimal number of top prototypes to use. The number 5 generated the best results. In future research, a more extensive test could be performed to find out if this number really is optimal.

Rejectionlist

This second certainty value is slightly more complex than the previously described one. In contrast with *agreeness*, *rejectionlist* is prototype dependent. This means that for each of the prototypes, some information is available that makes it possible to decide the certainty of a classification. This piece of information that is available, is the smallest DTW-distance between the prototype and the samples that have previously been incorrectly classified, due to that very prototype:

1. For each class (digits, uppercase letters, lowercase letters), the complete *testset*, was offered to the classifier
2. For each sample, the nearest prototype was decided by the DTW-classifier
3. If the label of that prototype did not match the label of the sample, the name of the prototype, and the DTW-distance between the prototype and the sample were recorded
4. At the end of the classification process, for each prototype that occurred in the list, the smallest accompanying DTW-distance was decided (see Figure 4.1). In the cases in which a prototype occurred only once in the list, that accompanying DTW-distance was used. The rejectionlist holds every prototype at least once responsible for an incorrect classification, and its smallest found distance
5. Prototypes that were not at least once responsible for an incorrect classification were not listed

Note that the list of samples offered to the classifier had a big influence on the created list. The *testset* (see Section 3.1), that previously had been used for prototype optimization was used.

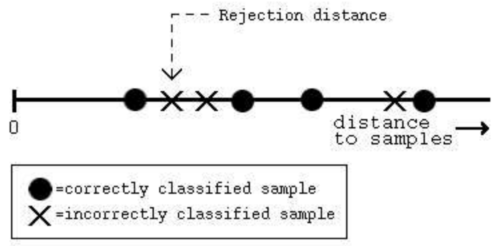


Figure 4.1: Rejection distance. The recorded distance for each prototype is equal to distance between the prototype and the nearest sample that it caused to be incorrectly classified. In this example, that visualizes the distances between the prototype and the classifications it was responsible for, the prototype was responsible for 4 correct, and 3 incorrect classifications. The recorded distance (or "rejection distance") is equal to the leftmost (therefore nearest) incorrectly classified sample that the prototype was responsible for.

The certainty value of a classification is either 0 or 1: if the DTW-distance between the sample and the nearest prototype is bigger than or equal to the distance in the rejectionlist, the certainty value is 0.

If it is smaller, the certainty value is 1. In the cases that the nearest prototype is not in the rejectionlist (it never made a false accept), the certainty value always is 1. Like with the *agreeness* value, a higher value (1) is better than a lower value (0). An added feature is the possibility to multiply the distances in the rejectionlist with a certain factor. If this factor is between 0 and 1, the method gets stricter, if it is higher than 1, the method gets loosener.

4.2.2 Usage of the certainty value

After the two certainty values are calculated, it has to be decided whether or not to let the classifier reject the classification. For this, a measure has to be decided. In the case of the *agreeness* method, one has to decide a number between 0 and 4. If the found value is lower than the selected number, the classification is rejected. If it is higher or equal to the number, the classification is accepted. In the case of the *rejectionlist* method, the classification is rejected if the found value is 0, and the classification is accepted if the found value is 1. As said, variation is possible using the multiplying factor.

One can decide to use one of the two certainty values at a time, but a combination is possible too. Both an AND and an OR variation were implemented. AND means that both conditions have to be fulfilled to let the classifier accept a classification, and OR means that at least one of the conditions has to be fulfilled to let the classifier accept a classification. Note that using both values together with AND is stricter than using only one value, and that using both values together with OR is loosener than using only one value.

4.2.3 Results

The rejection concept makes it possible to increase the certainty of a classifier: if a classifier makes a statement, the chance that it is correct is higher than when it is not using rejection (the proportion of correct accepts to false accepts is bigger when the rejection option is used than when it is not used). Due to false rejects however, the total recognition performance decreases (the proportion of correct accepts to all samples that are offered to the classifier is smaller when the rejection option is used than when it is not used).

To show this, we let our classifier process an unseen set of data (the *validationset* described in Section 3.1) with different rejection thresholds. The results are depicted in Table 4.4. As can be seen, the amounts in which the certainty increases and the performance decreases depends on the strictness of the rejection. For each application, one has to decide how to set the rejection parameters: the size of the accepted set is negatively correlated with the portion of correctly classified samples in that set. One thus has to choose whether to maximize either the size of the set, or the portion of correctly classified samples in that set. As said, in our rejection implementation, there are two thresholds that can be set. Using the two thresholds, one can achieve a very close approach of the desired certainty performance rate.

Table 4.4: Results of different combinations of rejection parameters. *Accepted* is the amount of samples that was accepted and classified (not rejected), *corr. class. of accepted* is the amount of accepted samples that was correctly classified, *corr. class. of rejected* is the amount of rejected samples that would have been classified correctly if they would have been accepted (false rejects), *acc. and corr. classified* is the amount of the total set of samples that were both accepted and correctly classified (this is the real performance grade). *** means that no rejection was set. This allows one to compare the results of the classifier using rejection with the same classifier not using rejection. Note that dependent of the nature of the application, either the size of the accepted set (*% accepted*) or the portion of correctly classified samples in that set (*% corr. class. of accepted*) should be maximized. Therefore, there is no objective measure to decide which parameter combination is the best. One has to decide for each application which combination to use.

Category	Prototype set	t	m	% accepted	% corr. class. of accepted	% corr. class. of rejected	% acc. and corr. classified
Lower-case letters	Resample and Average	*	*	100.00	88.20	-	88.20
		0	3.0	95.16	89.95	53.81	85.59
		2	1.5	71.98	96.00	68.14	69.11
		4	0.5	33.29	99.49	82.56	33.12
	Merge-samples	*	*	100.00	90.32	-	90.32
		0	3.0	97.47	91.45	46.43	89.14
		2	1.5	76.12	96.54	70.47	73.49
		4	0.5	34.64	99.61	85.39	34.51
Upper-case letters	Resample and Average	*	*	100.00	90.64	-	90.64
		0	3.0	97.58	91.41	59.55	89.20
		2	1.5	79.86	96.50	67.41	77.06
		4	0.5	38.39	99.79	84.93	38.31
	Merge-samples	*	*	100.00	94.28	-	94.28
		0	3.0	98.60	94.71	64.07	93.38
		2	1.5	83.96	98.02	74.69	82.30
		4	0.5	47.72	99.85	89.20	47.64
Digits	Resample and Average	*	*	100.00	95.26	-	95.26
		0	3.0	98.84	95.64	62.62	94.53
		2	1.5	91.40	97.78	68.77	89.34
		4	0.5	68.53	99.73	85.52	68.34
	Merge-samples	*	*	100.00	97.16	-	97.16
		0	3.0	99.62	97.31	57.14	96.94
		2	1.5	76.12	96.54	70.47	73.49
		4	0.5	73.40	99.81	89.86	73.26

4.3 Data verification

One application of the classifier is the verification of data. Because the classifier is able to calculate the certainty of a decision and can either report this or decide by itself to reject a classification (see the *Certainty condition* in Section 3.2.3), it can be used to verify a certain dataset. If some dataset is known to contain erroneous samples (samples that are mislabeled, incorrectly segmented, ambiguous, of bad quality, or have other problems), the classifier can be used to mark these samples for further inspection, or can even delete them.

This application of our classifier was used in the cleaning of the UNIPEN r01_v07-devset [17]. This section describes how Vuurpijl et al. [66] performed this cleaning using a combination between the DTW-classifier, three other classifiers (a multi-layered perceptron, a knn classifier, and the allograph matcher HCLUS [67]) and some human expertise. As will be shown, this combination allowed Vuurpijl et al. to clean up the database with little human intervention.

4.3.1 Introduction

In a large collaborative effort, a wide number of research institutes and industry have generated the UNIPEN standard and database [17]. Originally hosted by the US National Institute of Standards and Technologies (NIST), the data was divided into two distributions, dubbed the r01_v07-trainset and r01_-v07-devset. Since 1999, the International UNIPEN Foundation (iUF) hosts the data, with the goal to safeguard the distribution of the trainset and to promote the use of on line handwriting in research and applications. In the last years, dozens of researchers have used the trainset and described experimental performance results. Many researchers have reported well established research with proper recognition rates, but all applied some particular configuration of the data. Just like in the use of the dataset for the building of our DTW-classifier (see Section 3.1), in most cases the data were decomposed, using some specific procedure, into three subsets: one for training, one for testing and one for validation. Therefore, although the same source of data was used, recognition results can not really be compared as different decomposition techniques were employed. Furthermore, like in the building of our DTW-classifier (see Section 3.1), in most reported cases, a particular set of badly segmented or wrongly labeled data was removed or changed, which makes the comparison of results even more difficult.

For some time now, it has been the goal of the iUF to organize a benchmark on the remaining data set, the devset. Although the devset is available to some of the original contributors to UNIPEN, it has not officially been released to a broad audience yet. This section is meant to describe the procedure for verifying the devset, i.e. validating and correcting the data. This procedure should ensure the quality of a proper new benchmark data set, to be made available to the global handwriting recognition community. The original UNIPEN devset is organized in 9 sets (Table 4.5).

It is known that labeling and segmentation errors are present in both the trainset and the devset. An estimate of the number of errors in the trainset is given in [18]. It was reported that approximately 4% of the samples are errors. Other errors occurring in both sets are described in, e.g., [5, 63], reporting about segmentation errors, and in [38] and [42], reporting about segments that were obviously too wide or too narrow. In a recent effort made by Ratzlaff [42], scripts were generated that divide the (trainset) data into configurable train and evaluation sets and which can be used to generate uniform subsets for UNIPEN benchmarks that are comparable between researchers. However, a number of segmentation

Table 4.5: UNIPEN benchmark overview and the number of files and samples in the devset.

Set	Number of files	Number of samples	Description
1a	508	8598	isolated digits
1b	1087	16414	isolated upper case
1c	1672	37506	isolated lower case
1d	973	9898	isolated symbols
2	2144	72416	characters (mixed)
3	1267	44416	characters (word context)
4	0	0	words
5	0	0	words
6	2072	46353	words (cursive or mixed)
7	2143	52700	words (any style)
8	3592	11059	text ($\geq$ two words)
Total	15458	299360	

errors still remains in the data and moreover, the scripts do not check on labeling errors.

The focus of this section is to report on the quality of the UNIPEN data by examining the observed and detected errors in detail. To this end, a semi-automated procedure is described that distinguishes between a number of sample categories. The first step of this process is automated. A number of classifiers are combined to increase the confidence in cases where samples may be accepted. In the second step, the rejected samples are manually verified. As a result of this procedure, the following classes of samples are produced, where all but the first category require human supervision:

1. *Correct segments*, containing samples that are accepted with sufficient confidence by the procedure. This category is not further inspected. In the next section it is explained how it is ensured that the amount of errors that slip through the automated selection process can be minimized.

2. *Segmentation errors*, containing samples that are badly segmented. In UNIPEN, segmentations are specified through a so-called delineation, which marks the beginning and end of a sample (also called *segment*). Segmentation errors are caused by wrong delineations. In some cases these errors can be recovered, which is explained in Section 4.3.3.

3. *Labeling errors*, containing samples with wrong labels. Such errors may be caused by the writer producing the samples or by the human labeler, who may have interpreted the handwriting incorrectly. There is a fuzzy line between obvious labeling errors and cases where the label cannot be determined because of sloppy handwriting, or because the shape of a sample is ambiguous.

4. *Ambiguous samples*, containing shapes that can be interpreted in at least two ways. Most often, such shapes cannot be interpreted without context.

5. *Unfamiliar samples*, containing allographs that are unfamiliar to a classifier or human expert. Such samples typically are encountered in multi-lingual databases or databases with writers from different origin, as is the case in UNIPEN.

Figure 4.2 displays some examples from the latter four categories. The first row depicts samples with a proper label, but that have a poor quality, because of sloppy handwriting. In UNIPEN, such samples would be labeled as having a BAD quality. Rows 2,3,4 in Figure 4.2 depict problems of actual mislabeling, erroneous segmentation and interpretation ambiguity, respectively.

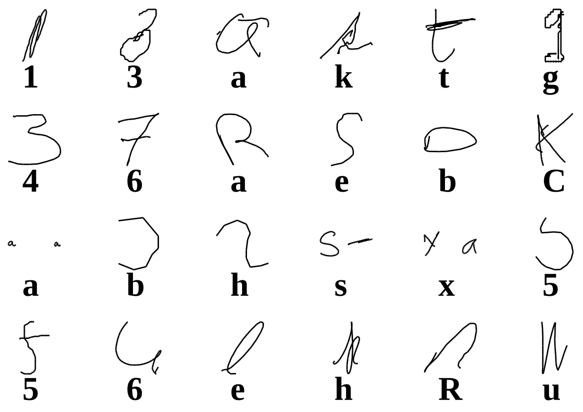


Figure 4.2: Problematic cases in UNIPEN data.

There is a particular note to be made on the first and last categories, containing samples with inferior quality or which are ambiguous. There are examples of other, well-segmented and labeled data sets that are used for training and testing handwriting recognizers, yielding high recognition rates. Although it is valid to report such results in literature, it also leads to systems that fail in real-life conditions. Rather than removing such bad samples, we opt for leaving them in the database and label the quality as BAD, or as a more suitable category like INFERIOR or AMBIGUOUS. The latter two qualifications are not contained in the current UNIPEN definition, however.

The verification procedure that is described in this section has been completed for the first three character sets (1a,1b,1c) and is currently being applied on the remaining sets. Results for these character sets are presented here and the preliminary set up of the verification procedure for word and text segments is discussed as well. In Section 4.3.2, the automated verification procedure for accepting or rejecting samples is discussed. Any samples that are rejected must be visually inspected. This process is described in Section 4.3.3. The procedure for verifying word and text segments is described in Section 4.3.4.

4.3.2 The automated verification procedure

The requirements for the verification procedure are straightforward: The first requirement is that the number of accepted samples (the *yield*) should be maximized. This reduces the laborious amount of manual verification and correction. The second requirement is that the amount of errors should be kept to a minimum. It is our goal to reduce the amount of errors in the data significantly below 1%. In the verification procedure, two parameters rule the yield and error quantities: (i) the quality and amount of the classifiers and (ii) the way in which the output hypotheses from multiple classifiers are combined for accepting or rejecting a sample.

Quality of the accepted samples

Given a division of samples in two categories: the accepted and rejected samples, a test can be performed to assess (with a certain significance α) whether the number of errors that may be expected in the accepted samples is below a certain fraction ϵ. The test says that if no errors are observed in a randomly drawn subset of N accepted cases, it is valid to assume that the error fraction is below ϵ with confidence $1 - \alpha$. Let the probability of drawing an erroneous sample i from this pool be ϵ_i, which equals ϵ for all samples if samples are drawn with replacement. In this case, the total probability of detecting no errors in the subset is defined as:

$$\alpha = \quad \prod_i^N (1 - \epsilon_i) \quad = (1 - \epsilon)^N \tag{4.1}$$

So, in order to be certain with a probability α that only a fraction of ϵ errors occur in the data, it has to be verified that N randomly drawn samples contain no errors, with:

$$N \quad = \quad {}^{(1-\epsilon)}log(\alpha) \tag{4.2}$$

An event is usually considered as statistically significant, if the probability of the event occurring randomly is smaller than 5% (or a stricter 1%). Here, the event is “no errors in the subset of N samples”. It is our goal to ensure with significance $\alpha = 0.01$ that maximally 1% of the accepted samples are errors and therefore, we must visually verify that no errors occur in $N = 459$ samples. Note that neither α nor ϵ can reach 0% with this test. Also note that this test does not use any information on how the accepted set was constructed. This would require an intricate knowledge of the behavior of the classifiers and their interdependency, which is beyond the scope of this research.

Quality and amount of classifiers employed

In this subsection, the quality and yield of the employed individual classifiers for character verification are discussed. Four different classifiers are used, trained on data from the UNIPEN trainset. The trainset and devset comprise distinct samples, but do not distinguish between writers. This makes classifiers trained on the first set very suitable for recognizing the latter set, as it may be expected that samples from the training set do not differ to a large extent. Different feature schemes are being employed, describing spatial-temporal (trajectory, running angle, angular velocity) characteristics and spatial (bitmap) characteristics. All four classifiers were trained on the 1a, 1b and 1c sets of the UNIPEN trainset, from which 36130 digits, 65483 isolated upper case, and 157264 isolated lower case segments where extracted. A multi-layered perceptron (MLP) using these features, a knn classifier (k=5) with the same features, our DTW-classifier, and the allograph matcher HCLUS as described in [67] were used for classifying the three sets 1a, 1b and 1c from the devset. The knn classifier matches each of the devset samples to the training samples from the corresponding set. Although it uses the same feature vector as the MLP, the completely different knn, DTW, and MLP algorithms ensure a distinct view on the data.

All four classifiers can use an individual threshold for deciding to accept or reject samples (for the DTW-classifier, the Agreeness certainty value, described in Section 4.2.1 was used). Each classifier only accepts samples if two conditions hold: (i) the threshold is passed and (ii) the output of each classifier corresponds to the label of the original devset. All other cases are rejected. Table 4.6 depicts the typical yield for the DTW-classifier and the MLP, given a certain threshold, for the devset. In case of the multi-layered perceptron, $MLP(t)$ corresponds to the percentage of accepted samples where the activation of the best output unit passes t. In case of the latter three classifiers, respectively $KNN(k)$, $DTW(k)$ and $HCLUS(k)$ represent the percentage of accepted samples for which the k closest neighbors are correct. For each individual classifier, a randomly drawn set of N reference samples was selected to be visually inspected. The column "errors" indicates the number of errors detected in the accepted samples from a particular classifier, for a given threshold.

Increasing the yield while passing the test

As can be observed in Table 4.6, in very strict settings each classifier is able to pass the test. However, this is at the cost of rejecting a large amount of samples. Therefore, a number of different combination schemes were evaluated that increase the yield, while still passing the test. The assumption is that when a particular number of classifiers accept a sample (i.e. mutually agree on the outcome, which equals the label), this can be considered as a success. All classifiers are treated equal in this procedure. In Table 4.7, different yields for respectively one, two, three, or four classifiers that agree on each sample in the devset are listed. Numbers that are marked with a '(y)' did pass the test.

Table 4.6: Yield (percentage) and errors for the MLP and DTW-algorithms. Similar results are produced by the other two classifiers for the thresholds 1. . . 5.

Classifier	set 1a		set 1b		set 1c	
	Y	#E	Y	#E	Y	#E
MLP(0)	94.3	3	87.5	2	87.1	1
MLP(.7)	90.6	2	86.4	1	74.7	1
MLP(.8)	84.4	1	76.4	1	66.4	1
MLP(.9)	60.3	none	55.1	none	51.0	none
DTW(1)	85.8	3	92.9	none	81.9	none
DTW(2)	91.6	1	83.5	1	78.5	1
DTW(3)	88.3	1	77.2	2	71.5	1
DTW(4)	83.1	none	66.6	1	61.3	none
DTW(5)	72.9	none	49.4	none	45.1	none

Table 4.7: Yield and results on passing the tests for cases where *nc* classifiers must agree. Numbers marked with a '(y)' passed the test. For each of the subsets of the devset, the highest yield that passed the test is shaded.

nc	yield 1a	yield 1b	yield 1c
4	90.5(y)	59.3(y)	57.5(y)
3	▨ 91.4(y)	83.4(y)	81.1(y)
2	95.6	▨ 91.1(y)	▨ 89.8(y)
1	98.2	96.1	95.2

When comparing the results in Table 4.7 to that in Table 4.6, it can be observed that the yield is much higher in the case of combining classifiers than when using individual (strict) threshold values. At the same time, even with combinations of only two out of four classifiers, all tests (except in the case of digits) are passed. This is an excellent example of using multiple classifiers for increasing the robustness of pattern recognition. Rather than increasing the decision threshold, the different views on the data ensure that only samples are accepted when two or more distinct observations agree.

The 1a, 1b and 1c sets from the devset were automatically divided in two categories (accepted and rejected) by using the marked combinations from Table 4.7. After this first step of the procedure, respectively 7858 (1a), 14952 (1b) and 33671 (1c) samples are accepted, where it is assumed that these samples contain less than 1% errors, i.e. samples that should have been rejected.

Verification of the procedure

In order to verify the correctness of the procedure, a major effort was performed by manually verifying all processed segments from the 1a, 1b and 1c sets. For each data set, the original data were split into multiple files, each file containing the data for only a single label type. So, for example, the digits set

(1a) was split into ten files, one for each of the digits 0-9. The data were then displayed 100 at a time in a 10x10 grid. This allows for rapid review of the data for verification purposes. It also provides a context for reviewing a single writers work as a group, and for viewing and comparing several writing styles at the same time. This sorted and multiple view context is especially helpful to discern between labeling errors, sloppy instances of a particular allograph, and for discovering unusual allographs or writing styles that might otherwise be evaluated as bad or mislabeled. The data are then evaluated and appropriately assigned.

This manual verification process was performed independently of the manual labeling process described in the next section. Based on the completely verified data set, it is possible to assess the correctness of the assumption made in Equation 4.2. This assessment was performed by comparing the set of samples that were judged as erroneous by the manual verification process, to the set of samples that were automatically accepted by the procedure described above. As a result of this comparison, no samples that were accepted for 1a appeared to have errors. Only 0.061% falsely accepted samples from the 1b set appeared to have slipped through and for the 1c set, this number was less than 0.064%. These numbers indicate that although statistically, the number of automatically accepted samples contain less than 1% errors, the real (validated) estimates are much better.

4.3.3 The manual labeling process

All samples that were rejected in the previous process are candidates for errors. Although the majority of these samples are probably correct (as only 4% errors were expected [18] and about 10% of the samples are rejected), they must be verified through human supervision. Here, three main categories of interactive operations have to be performed: (i) marking false rejects, i.e. samples that were rejected by the ensemble but that were judged as correctly labeled after visual inspection, (ii) correcting wrong labels, i.e. samples that were correctly rejected and should definitely be labeled differently, and (iii) correcting wrong segmentations, i.e. samples that could not be accepted because they were badly segmented. Please note that as indicated in the introduction, labels and segmentations in any of these categories may be distinguished in various levels of quality (sloppiness) and confidence (depending on ambiguity or familiarity of the allographs).

For each collection (1a, 1b, and 1c) of the UNIPEN devset, the appropriate ensemble of classifiers was used to filter out samples that could not be recognized with sufficient confidence. These samples were alphabetically sorted and displayed via the UNIPEN displayer `upview`. Upview is a program, contained in the `uptools` distribution [70] for fast visualization of large amounts of UNIPEN data. Similar to the viewer described in Section 4.3.2, upview depicts segments in a matrix organization. Specific routines for processing particular samples can be engaged by clicking on the corresponding segment. If one of the three kinds of interactive operations mentioned above should be applied to a segment, the human verifier can click on the segment by using either the left, middle or right button of his mouse. Correcting falsely rejected samples (the majority of cases) can be performed very efficiently in this manner. As samples were depicted in alphabetical order, anomalies can be detected fast.

Correcting false rejects Upon manually overriding the rejected samples, two options were made available to the supervisor. The first option marks the segment as correctly labeled, but with a proper quality. In the second option, the segment is still marked as correctly labeled, but the quality is labeled bad. The latter option is typically in place for the samples depicted in the first row of Figure 4.2.

Correcting wrong labels Similar to the cases where rejected samples had to be accepted, labeling errors can be distinguished in two categories: wrong labels with bad quality and wrong labels with good quality.

Marking segmentation errors Segmentation errors that can be visually detected, are marked as recoverable and stored for a later correction process using an interactive segmentation tool (see Figure 4.3).

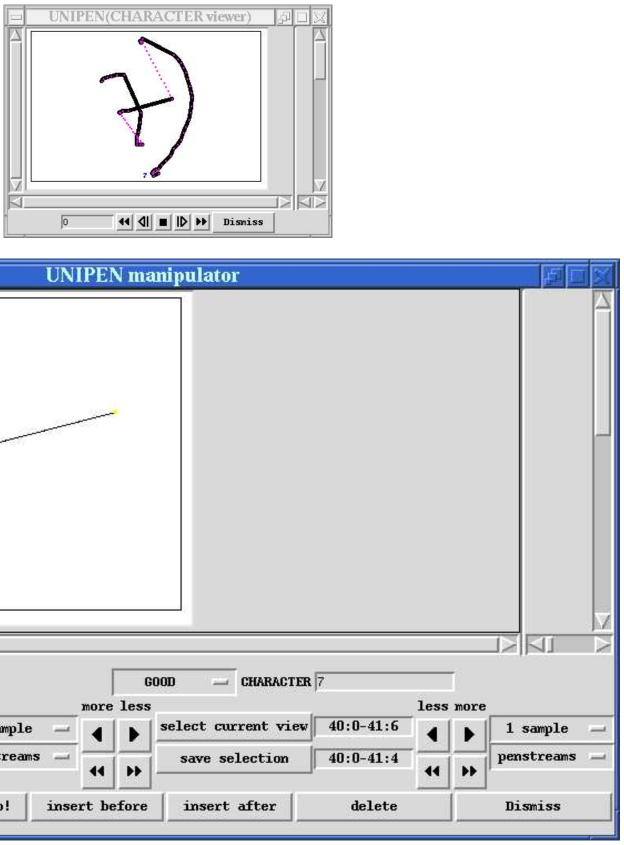


Figure 4.3: Wrongly segmented character and UNIPEN manipulator for recovering the correct segmentation.

Handling undecidable errors The former three cases can be identified through careful examination of the depicted segments using Upview. However, as depicted in Figure 4.2, some cases cannot be determined as they either contain segmentation errors or ambiguous shapes. These cases are marked as

Table 4.8: Manual categorization of rejected samples by two handwriting experts 'A' and 'B'. Judgments were made between correct (L=OK) and wrong (L=W) labels, good (Q=OK) and bad (Q=BAD) quality, segmentation errors and undecidable cases. The columns labeled "A" and "B" indicate judgments made by a single expert. Columns marked "both" list the number of cases in which both experts agree.

	Lowercase letters			Uppercase letters			Digits		
Category	Both	A	B	Both	A	B	Both	A	B
L=OK; Q=OK	1962	664	266	507	163	67	348	27	160
L=OK; Q=BAD	291	240	680	42	60	180	46	204	18
L=W; Q=OK	78	296	23	20	36	15	6	4	7
L=W; Q=BAD	11	12	160	7	14	11	0	5	5
Seg. error	258	8	90	554	48	49	95	3	43
Undecidable	4	11	12	8	3	2	0	2	12

undecidable and are stored for further processing, for which the `upworks` tools contained in the `uptools` distribution [70] are employed.

Two human handwriting experts performed the manual labeling process described above. As the results depicted in Table 4.8 show, it appears that many samples provide causes for uncertainty. The main reason for this uncertainty is that judging the quality of a handwriting sample is a subjective process. Judgments on quality (i.e., when is a sample too sloppy or not), allograph familiarity (is the label correct, is the shape ambiguous?), and even segmentation errors are examples of subjective decisions.

Estimating a lower bound on the number of correctly accepted samples in the original 1a, 1b and 1c sets can be performed by adding the number of overruled samples on which both experts agree (categories 1 and 2 in Table 4.8) to the number of samples accepted in the automated verification step described in Section 4.3.2. As the latter is guaranteed to have maximally 1% errors, it can be deduced that the maximum percentage of errors in the original sets is respectively 3.0 (1a), 4.5 (1b) and 3.2 (1c).

The sixth category (undecidable) as well as all cases where both experts do not agree are stored for subsequent processing, either using more context (e.g. considering the surrounding coordinate trajectories) or discussing these cases with further experts. However, as may be concluded at this point, there is a considerable amount of samples for which judging between labels or quality is ambiguous. It will be examined whether a more elaborate distinction in, e.g. INFERIOR (shape, segmentation) or AMBIGUOUS (shape, label) is required.

4.3.4 Verifying words

The procedure described above was tested on three character sets. There are plans for using the same procedure for verifying the other character sets 1d, 2 and 3. In order to semi automatically verify the word and text categories 6, 7 and 8, a more elaborate procedure is required. Although we have not completed the word and text verification procedure, the approach that is currently being implemented is described briefly below.

In word recognition of unknown trajectories containing (X,Y,Z) coordinates, an approach that is often followed is to find proper character segmentation points and to generate a character hypothesis lattice of possible word outcomes. The hypothesis space can become very large, but is restricted by the size of the lexicon, which is used to prune irrelevant character paths from the lattice. In the case of word *verification*, the lexicon contains only one word. In the word verification procedure, the word recognizer will first try to find the character paths that match the word label. If this does not succeed, the word will be rejected. If it does succeed, one or more character paths will be contained in the lattice. In the second stage, the trajectories of subsequent characters in a path will be verified as described in the character verification procedure. If all characters in a path are accepted, the word can be accepted. If one of the characters in a path cannot be accepted with sufficient confidence, the word must be rejected.

We have performed experiments with this approach, by using the velocity-based segmentation algorithm described in [53]. The method works quite well, but is restricted to fluent handwriting. Unfortunately, not all words in the UNIPEN database confirm to the required temporal characteristics that are needed to perform segmentation on points of minimal velocity. In particular, some data are acquired through mice or tablets with a low temporal resolution. Therefore, our current efforts are targeted on implementing other segmentation schemes, like those based on points of maximum curvature or Y extrema. The current word verifier has a yield of 91% with very low error rates. However, these numbers have to be sustained in further experiments.

4.3.5 Conclusions

This section presents a procedure for detecting and solving errors present in the UNIPEN devset. The procedure is currently being applied to all character sets from the database. The goal of this work is (i) to remove unrecoverable labeling and segmentation errors, (ii) to correct labels and segmentations for cases where this is possible, and (iii) to review the quality of UNIPEN segments.

It is shown that by using classifier combination schemes, a large portion of the data samples can be automatically checked, while keeping the remaining error margin well within a respectable 1% range. The samples that are rejected by the classifier combination have been checked manually, resulting in a number of ambiguous cases that need to be further investigated.

It has been debated that in particular the ambiguous cases or cases with bad quality present problems for handwriting classifiers and that rather than removing these samples from the database, a more elaborate qualification scheme is required.

Our current efforts are targeted on finalizing the verification process for the remaining categories and further processing samples that have not been decided upon.

4.4 Intuitive handwriting recognition

4.4.1 Introduction

The results of the way in which the Dynamic Time Warping algorithm compares curves, are claimed to be more intuitive than the results of other methods: in some way, the results of DTW-comparison are more plausible to humans than the results of other systems [62]. This "more human-like" comparison could be of use in the field of forensic document analysis: Today, in order to assess the correspondence between handwritten documents, pieces of handwriting are compared by human experts. A particular comparison that is made focuses on the comparison of allographic shapes present in the handwriting. In order for a computer to perform this task, it should be able to yield allographic prototypes that are in some way similar to the prototypes that a human expert would retrieve if given the same task. This kind of "visually perceptive and plausible" pattern matching is believed to improve the user acceptance of handwriting recognition systems in for example PDA-systems, given that the errors made by the system can be understood by the user [48].

To test the claim that the Dynamic Time Warping algorithm, and especially our implementation of that algorithm, compares allographs in a way that is "intuitively believable" to humans, we set up an experiment. In this experiment, we compared three different systems: the DTW-classifier with the *Resample and Average* prototype set (Section 3.3.3), the DTW-classifier with the *Mergesamples* prototype set (Section 3.3.3) and the HCLUS-classifier[3], that is based on the hierarchical clustering method described in [49, 67, 69]. In the experiment, a selection of samples from the *testset* was offered to each of the classifiers, and the results were judged by human subjects.

Both DTW-classifiers were trained on the *trainset* (as is described in Section 3.3). The HCLUS-classifier was trained on selections of the UNIPEN v07_r01 trainset (as is described by Vuurpijl [67]). Table 4.9 depicts the recognition performances of the three classifiers on the *testset* and on the complete UNIPEN devset for lowercase and uppercase characters.

Table 4.9: Summarized recognition performance of the three classifiers. The full performance of the DTW-classifiers on the *trainset* can be found in Tables 4.1 and 4.2.

Classifier	*trainset* lowercase	*trainset* uppercase	devset lowercase	devset uppercase
Res. and Avg.	88.20%	90.64%	80.96%	83.76%
Mergesamples	90.32%	94.28%	82.61%	97.29%
HCLUS	70.89%	92.55%	68.94%	85.89%

As can be observed, in general the DTW-algorithm has a better performance than the HCLUS-system. It should be noted that the HCLUS prototypes for uppercase letters have evolved the last years and that

[3]http://hwr.nici.ru.nl/~vuurpijl/hclus/index.html

the relatively high recognition performance of 92.55% on the *trainset* is most probably the result of over-fitting on that dataset. Since the goal of the experiment was to assess whether or not the matching results of any of the three systems were more intuitive than the others, differences in recognition performance were removed as follows: For each sample query with a certain label, the five best matching prototypes with the same label produced by each system were collected. This excluded the possibility that, for example, a prototype of the letter *a* would come up with a query *e*.

Based on the assumption that the results of the DTW-classifier are more intuitive than the results of the HCLUS-classifier, our first main hypothesis was that we would find that humans judge the results of the two DTW-variations more believable than the results of the HCLUS-classifier.

Our second main hypothesis was that the *Mergesamples* prototypes would be judged as more believable than the *Resample and Average* prototypes. We expected this because the *Mergesamples* prototypes proved to generate better results in recognition performance (see Section 4.1). On the other hand we expected that, because the *Mergesamples* prototypes are more "bumpy" and "coarse" than the *Resample and Average* prototypes (according to a human handwriting expert, see Section 3.3.5), subjects might be less tend to select them (the little variations and lack of smoothness make them look a little less like real handwritten shapes, see Figure 4.4 for an example). We however believed that subjects were able to (and were going to) ignore this and judge the "likeness" of a sample to the query by the rough shape and the deviation of the important spots that are mentioned in Section 1, and not by the smoothness of the curve.

Our hypotheses in this experiment were: H_A: The *Mergesamples* prototypes would be judged to be more intuitive than the *Resample and Average* prototypes. H_B: The *Mergesamples* prototypes would be judged to be more intuitive than the *HCLUS* prototypes. H_C: The *Resample and Average* prototypes would be judged to be more intuitive than the *HCLUS* prototypes.

4.4.2 Method

25 subjects, both men and women in the age of 20-55, participated in the experiment. A variation of the experiment described in [56] was conducted. Each subject was given 133 trials, of which the first 3 were practice trials. In each trial, the subject was shown a so-called "query" allograph and a $5 * 3$ matrix of different "result" allographs. The subjects were told that the query was offered to a handwriting recognizer and that the allographs in the matrix were the 15 characters that were found to be the nearest to the query allograph. In each trial the subject were asked to select a self chosen number of allographs that they found to be resembling the query "enough". Because we wanted to test for a comparison between the systems and the average person, we did not give the subjects any instructions on what criteria to use or how many allographs to select. In each trial, the subject was allowed to select any number, between zero and fifteen, of allographs. An example of an offered trial can be seen in Figure 4.5.

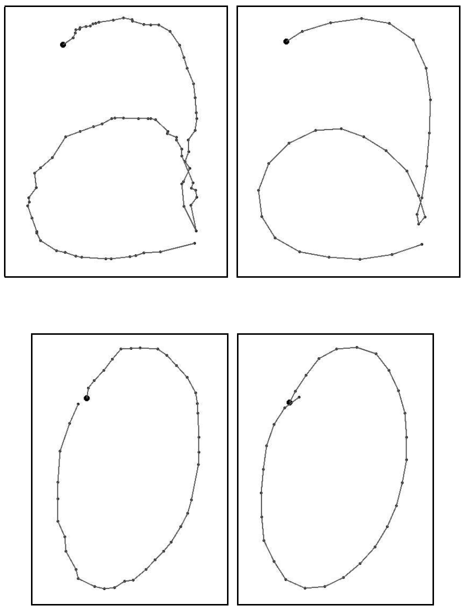


Figure 4.4: Prototype examples. The *Mergesamples* prototypes (on the left) are less smooth, and more "coarse" and "bumpy" than the *Resample and Average* prototypes.

The instructions that were offered to the subjects can be found in Appendix C. Subjects could select allographs by clicking them (the selected allographs were emphasized by a green border). Deselecting an allograph could also be done by clicking it. The results of each trial were stored when the subject clicked on a submit button, which also loaded the next trial.

In reality, the subjects were not shown the results of one, but of three different character recognizers (HCLUS and the two DTW-variations). A random selection of 133 samples (3 for the practice trials, 130 for the experiment) was taken from the *testset*. Each of the selected samples was offered to the three recognizers, and each of the classifiers returned the five prototypes that were found to be nearest to the query[4].

Every subject was shown the same query and accompanying classification results. To rule out any influence of time, like the losing of interest in the task toward the end of the experiment, the order in which they were presented was randomized. Also, to make sure that the location of the classification results on the screen did not have an effect on the results, the order in which the results were presented were also randomized. In theory, it was possible that a certain trial was shown to only one subject.

[4]All queries and results of the three classifiers can be found at http://dtw.noviomagum.com

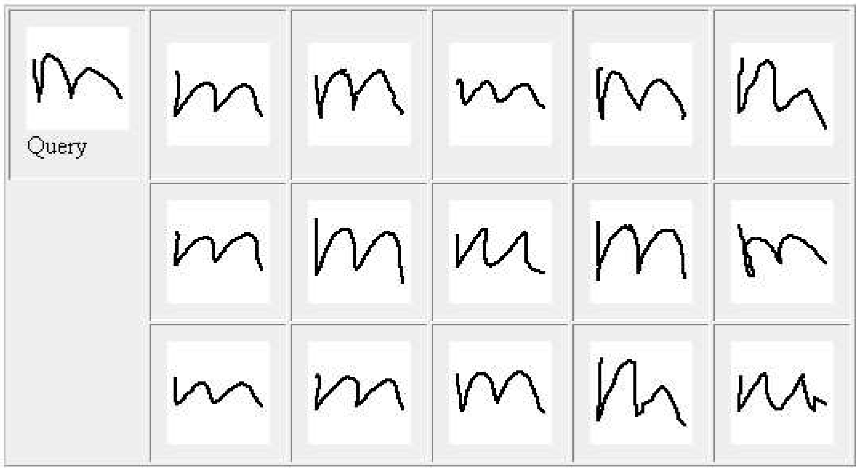


Figure 4.5: Example of a trial of the experiment. The subjects were asked to select the allograph(s) that they found to be resembling the query "enough". Each subject could decide for himself or herself what criteria to use and how many allographs to select.

4.4.3 Results

To test hypotheses H_A and H_B, a General Linear Model (GLM) was created on the data (see Appendix D.2.1 for the SPSS-syntax). The number of clicked samples from the *Mergesamples* classifier was compared to the number of clicked samples from the other two classifiers. This was done for every single letter, and for the total of all letters, resulting in 27 p-values for every comparison, each indicating whether or not the *Mergesamples* results were judged significantly better.

To test hypothesis H_C, another GLM was created (see Appendix D.2.2 for the SPSS-syntax). Here, only the relation between the *Resample and Average* and the HCLUS-classifier was taken into account. Like for hypotheses H_A and H_B, 27 p-values were calculated.

Table 4.10 shows both the number of clicked items for each classifier, and p-value belonging to each of the hypotheses. In the cases where $p < 0.05$, the accompanying hypothesis is significantly validated.

Table 4.10: Results of the Intuitivity Experiment. The *p-value* columns show whether or not *Merge-samples* had a significant higher score than either *HCLUS* or *Resample and Average*. If $p < 0.05$, the difference is significant. This is emphasized by the shading of the cell. In the cases where the results are opposite to the hypothesis, an asterisk (*) is added to the p-value.

Char.	Merge-samples	Res. and Average	HCLUS	p-value H_A	p-value H_B	p-value H_C
a	127	120	67	.430	.000	.000
b	120	144	29	.085*	.000	.000
c	172	214	83	.002*	.000	.000
d	137	126	56	.262	.000	.000
e	137	139	91	.811*	.000	.000
f	157	31	17	.000	.000	.080
g	73	105	40	.009*	.002	.000
h	143	161	116	.089*	.071	.002
i	92	17	46	.000	.003	.030*
j	186	148	32	.004	.000	.000
k	153	43	46	.000	.000	.538*
l	201	254	93	.000*	.000	.000
m	74	125	37	.000*	.001	.000
n	136	121	50	.066	.000	.000
o	89	69	63	.084	.059	.574
p	96	101	111	.564*	.162*	.355*
q	130	127	116	.722	.375	.514
r	107	145	77	.001*	.014	.000
s	67	74	93	.337*	.014*	.062*
t	113	23	39	.000	.000	.039*
u	149	167	55	.068*	.000	.000
v	194	184	63	.284	.000	.000
w	132	143	19	.191*	.000	.000
x	135	51	25	.000	.000	.000
y	71	98	71	.016	1.000*	.000
z	206	13	18	.000	.000	.435
Total	3397	2942	1553	.000	.000	.000

4.4.4 Discussion

The results of this experiment are shown in Table 4.10. As can be seen there, the overall results support all hypotheses (H_A, H_B and H_C). On a number of specific characters however, the hypotheses were not confirmed. In some cases the results are even opposite to the hypotheses.

The difference between the characters which support, and the characters that do not support the hypotheses, is not easy to explain. We tried to find an explanation considering the overall shapes of the characters, but this was not successful (e.g. the *b* is a mirrored version of the *d*, and they both support hypotheses H_B and H_C, but the *d* is a rotation of the *p*, and the former does, and the latter does not confirm hypotheses H_B and H_C. About the same can be said about the *z* and the *s*). It could be in the way the classifiers work (which is so nontransparent that it is hardly possible to conclude anything from it), and it could be the set of prototypes that each classifier uses. To find a detailed answer to the question how the difference in techniques leads to better or worse results per character, more transparency is needed, and the used features should be better explainable.

It is possible that the cases in which hypotheses H_B and H_C were not confirmed, can be explained by the results of the performance test that is described in Section 4.1. Tables 4.11 and 4.12 show that there is an overlap between the cases in which, in the performance test, the *Mergesamples* set did not significantly perform better than the *Mergesamples* set. It is possible that because the *Mergesamples* set did not significantly perform better, the results were of lower "intuitive" quality too. The relative "intuitive" quality of the results of the other two classifiers would improve. As is shown in the mentioned tables, for hypothesis H_B, this explanation could account for 3 of the 6 cases, and for hypothesis H_C it could account for 6 of the 9 cases.

Table 4.11: Characters that do not confirm hypothesis H_B. The column *Significant* shows the characters on which the *Mergesamples* set significantly performed better than the *Resample and Average* set in the performance test. The column *Not significant* shows the characters on which this was not the case.

Significant	Not significant
o	h
p	q
s	y

Table 4.12: Characters that do not confirm hypothesis H_C. The column *Significant* shows the characters on which the *Mergesamples* set significantly performed better than the *Resample and Average* set in the performance test. The column *Not significant* shows the characters on which this was not the case.

Significant	Not significant
f	i
k	q
o	z
p	
s	
t	

The question we were trying to answer was: does DTW generate results that are more intuitive than the results of other classifiers. In this experiment, we compared DTW to only one other classifier. This means that generalization to all classifiers is hardly possible. Also, only lowercase letters were taken into account. This makes generalization to all characters difficult.

To be able to make a better generalization, more experiments should be done, in which more classifiers are taken into account, and in which not only lowercase letters, but also uppercase letters and digits are part of the experiment.

To explain why some characters do, and others are do not differ significantly, more research is needed as well. Subjects, for example, can be asked why they think a certain allograph does, or does not resemble the query enough. Perhaps more information about how humans compare handwritten characters can be found and used in the setting up of another experiment. Also, in our experiment, the classifiers only had access to their own sets of prototypes (prototypes that were created for optimal compatibility with the system). A better comparison between the systems would be possible if they all used the same prototypes. The problem would occur however, that the used set of prototypes simply is more suitable for usage with one than with another classifier. This problem would have to be solved first.

Although a number of remarks can be made about the conducted experiment and the results that were found, we still believe that DTW is able to generate results that are more intuitive than other classifiers can, and the overall results of this experiment support this believe. The experiment therefore supports a positive answer to one of the main research questions of this thesis: does DTW generate results that are more intuitive than the results of other classifiers?

Chapter 5

Discussion

5.1 Conclusions

In the previous chapter, the research questions that were stated in Section 1.1 were answered. The conclusions for each of the questions are:

1. *How well does DTW-classify handwritten characters, and how does this compare to other systems?*
   Using the *Resample and Average* prototype set, the classifier correctly classified 88.20 percent of the lowercase letters, 90.64 percent of the uppercase letters, and 95.26 percent of the digits. Using the *Mergesamples* prototype set, 90.32 percent of the lowercase letters, 94.28 percent of the uppercase letters, and 97.16 percent of the digits were correctly classified. Although serious problems exist in comparing performance results of different classification systems, we believe that the recognition performance of our classifier, using either of the prototype systems, is similar to that of other classifiers.

2. *How can DTW be used in a Multiple Classifier System (MCS)? And does its different view on data improve the results of that MCS?*
   A practical example of the use of our classifier in an MCS has shown that it can really add some new information to the information provided by the other classifiers in the system. Due to the fact that DTW matches in a very different way than other feature based or statistical classifiers do, makes the view on the data in some way orthogonal to that of other classifiers. Therefore the system can be very useful in an MCS containing one or more of these other classifiers. In the MCS that was created, our system played a role in the decision of whether or not a sample in a dataset was correctly labeled. Using the judgments of different classifiers, the MCS decided whether or not to report the sample as being a possible problem. As was shown in Section 4.3, less than 1 percent of the samples that the MCS reported to be correctly labeled, was incorrectly labeled after

all. Furthermore, only a small portion of the samples that were reported to be incorrectly labeled, turned out to be correctly labeled after all. This positive achievement shows that our system can very well be used in an MCS.

3. *Does DTW generate results that are more intuitive to humans than the results generated by other systems?*
The conducted experiment shows that the results of our classifier, using the *Mergesamples* prototype set are judged to be similar to the queries than the results of the HCLUS-classifier. Although this only gives information about the DTW-classifier compared to only one other classifier, the results are promising. For a final answer to this question, our classifier should be compared to a much larger set of other classifiers, but until then, we believe we can answer the question positively: DTW generates results that are intuitive.

5.2 Future research and usage

In Chapter 4 the used applications and performed research of our DTW-classifier were discussed. There are more possible applications and possible research, of which some will be discussed in this section.

5.2.1 Forensic Document Analysis

The experiment described in Section 4.4, about the intuitiveness of the handwriting comparison of the DTW-classifier, showed that the results of the classifier are in some way comparable to the way humans compare different handwritten samples. This ability of the classifier can be used in the field of forensic document analysis. In this practical field of handwriting recognition, handwriting is used as a kind of fingerprint: if a piece of paper, containing handwriting, has been found at a crime scene, or has been sent as a threatening letter, the handwriting can give information about the identity of the writer. If that persons handwriting is already available in some database, our DTW-classifier could help in searching the database. Also, with knowledge about the influence of origin, writing experience, gender and handedness, some information about the identity of the writer can be extracted from the handwriting.

Because humans are known to be good at the task of comparing handwriting, and because it is known that the results of the comparison of our DTW-classifier are in some way equal to that of humans, the classifier could be of help in the process of finding the identity of the writer of some piece of text.

Another way to use the DTW-algorithm for writer verification is described in a recent paper by Lei, Palla and Govindaraju [28]. They suggest a method called *Extended R-squared* that can calculate a similarity measure that is not relative, like in our system (e.g. the distance itself does not give information about the similarity between two curves without comparing it to other distances), but absolute. This enables one to answer the question “how similar are these samples?” with a percentage. Lei, Palla and

Govindaraju suggest the technique for application in signature verification, but we believe that it might be useful for verification of handwritten text as well.

Consistency

For the DTW-classifier to be useful for the field of forensic document analysis, it needs not only to be able to compare two handwritten characters in a way that has the same results as the comparison by humans, it also needs to be proved that it is consistent: two different handwritten characters of the same writer, should be identified as written by the same writer, or at least as belonging to the same category (in origin, writing experience, gender or handedness). This property could for example be tested in an automated experiment in which sets of prototypes (each set based on the writing of a single person) are used, and in which a number of previously unseen samples of handwriting are offered to the classifier. If a significant amount of the queries are classified as belonging to the prototype set based on the handwriting of the person who also wrote the query, the consistency of the classifier can be statistically proved.

5.2.2 Automatic handwriting teaching

Because the DTW-classifier can compare curves in a way that is similar to human comparison, and because it can compare allographs with prototype allographs, we believe that the algorithm can be of use as an automatic handwriting teacher. Most children in Western countries learn to write at school at the age of 5 to 7. If (a portion of) the writing exercises are done on a writing tablet instead of on paper, an automatic corrector could judge the written characters and help the child to improve his or her writing. This application could be of interest for researchers in the area of e-learning and Intelligent Tutoring Systems (a research field that has had the interest for many years, the first attempts reported as early as in 1926 [39]).

5.2.3 Other alphabets

We tested our DTW-classifier only with the basic Latin uppercase ([A-Z]) and lowercase ([a-z]) letters and the Arabic digits ([0-9])[1], but as noted in Section 3, characters from other alphabets have had the attention of researchers and developers of practical applications of handwriting recognition as well.

In future research, it could be interesting to see how well our classifier performs with other alphabets, like the Japanese, Arabic, Laotian and Sanskrit. Also, it could be interesting to use the DTW-classifier for classifying alphabets that are based on the basic Latin alphabet, but extended with other characters, like the Icelandic, German, French, Dutch and Esperanto alphabet, which, among others, use the additional characters ð, ß, ç, ë and ĉ respectively. If the system performs good enough, it could be used for

[1]In spite of the name, most historians believe that the Arabic numeral system was developed in India (http://encyclopedia.thefreedictionary.com/arabic%20numerals).

applications in those languages (e.g. PDA systems in Japan, forensic document analysis in Germany, etc) or even in multilingual applications (e.g. word processors that understand input in different alphabets).

5.2.4 Word recognition

As can be seen in Table 4.5, the UNIPEN dataset does not only contain isolated characters. There is also a large number of isolated handwritten words in the collection. This is an example of the interest of researchers in the recognition of handwritten words. In a lot of real life applications, people do not write isolated characters, but embed them in words. This can be solved by decomposing (either online or offline) representations of words into characters. This is called segmentation [7, 36][2]. After the decomposition, the different characters can be offered to a character recognizer. Additionally, the resulting classifications can be combined and checked with a lexicon to find out if the found word is a real world. The DTW-classifier can be used, just like it is used for character recognition, for this application. Since segmentation is a very difficult process, the word recognition is sometimes done without the segmentation: whole words are offered to a classifier, and classified as a whole [9, 52]. This is also a possibility of our DTW-classifier. For practical reasons however, it is only possible to use the classifier for whole word recognition if the lexicon is known and (very) limited, since prototypes are necessary for every word in the lexicon. It would for example be possible to use the classifier as a word recognition in the previously mentioned application dScript, in which a user is asked to write down the name of any town in the Netherlands. This list is limited to about 4800 words[3].

5.2.5 Automatic optimizing

Optimizing the options

During the implementation, testing, tuning and fine-tuning of our classifier, and also during the generation of the two prototype sets, a lot of different options were available. Most of those were tried and evaluated manually. This limited us in the number of options and combinations of options we could try. To further optimize the classifier and the prototype sets, the testing and evaluation could be automated. This could increase the number of possible (combinations of) options that can be tried, and this increases the chance that if better (combinations of) options exist, they will be found. Using a Monte Carlo simulation, one does not have to try every possible combination (which can sometimes take a lot of time), while a good estimation of the best combination of options can still be found.

[2]Information about dScript, a working example of word segmentation, can be found at http://www.ai.rug.nl/ lambert/recog/dscript/

[3]http://home.wxs.nl/ pagklein/almanak.html

Automatic generation of optimal prototypes

As described in Section 3.3, the selection of samples to generate prototypes was done manually. Since the prototypes were based on these selections, and, as shown in Sections 4.1 and 4.4, the (quality of the) prototypes is an important factor in the recognition performance and the judged intuitivity, it might be a good idea to automate this selection to optimize the selection and creation of the prototypes. To automate the generation of the prototypes, one can let a computer try (a lot of) different selections and creation methods, and evaluate every generated prototype by calculating the average distance between the prototype and the samples that is was based on. This average distance should be minimized, so that every prototype is a good average of the samples it represents. One should maintain the number of prototypes within a certain limit to avoid prototypes being based on a number of samples that is either too small (a prototype based on only one sample would minimize the average distance between the prototype and the samples it is based on, but would probably not be a good representation of a larger set of allographs) or too big (the prototype would possibly become odd shaped and not a good representation of the set it represents). After the creation of the prototypes, the cleaning up, as described in Section 3.3.4 could also be automated. As described in that section, prototypes that were responsible for more incorrect than correct classifications were removed, but this factor could be changed to find the best selection of the generated prototypes. With the automation of the generation and cleaning up of the prototypes, one could find a set of prototypes that could enable the DTW-classifier to optimize its recognition performance. Whether or not the intuitivity would also improve is hard to say: one would need to examine that with an experiment, possibly similar to that described in Section 4.4.

5.2.6 Forensic Document Analysis

Our DTW-implementation showed that Dynamic Time Warping is well able to compare handwritten allographs in a way that the results are similar to the results of human allograph comparison. This can be very useful in the field of forensic document analysis, in which human handwriting experts are manually comparing samples of handwriting to large databases. With the help of a program that can do the same thing, the number of documents that has to be compared to the sample can be reduced significantly. Because handwriting samples available in forensic documents are offline (they are written on paper), a conversion to online data has to be made in order to employ the DTW-algorithm described in this thesis. The automated conversion from scanned handwritten material to online trajectories is still unsolved, although recently, some promising results have been published in the literature [72].

The combination of offline and online handwriting recognition techniques opens up the possibility to search for allographic material in scanned documents. A first step toward this approach is implemented in the WANDA workbench[4]. In WANDA, the forensic expert is provided with the option to manually

[4]http://pentel.ipk.fhg.de/wanda/

trace (i.e. copy drawing) the trajectory of allographs in still images. Given the dynamic equivalent of an image-based allograph, the technique presented here can be used.

In the near future, the challenge of automatically extracting online allographic information and combining this with DTW-matching techniques, will be pursued in the TRIGRAPH[5] project. TRIGRAPH is a new research program concerning the use of automatic online plus offline handwriting recognition techniques in the field of forensic document analysis. This project will be carried out by the University of Groningen, the NFI (Netherlands Forensic Institute) and the NICI (Nijmegen Institute for Cognition and Information).

[5]TRIGRAPH will be executed within the Token2000 (http://www.token2000.nl) program.

Bibliography

[1] Adnan Amin. Off line arabic character recognition - a survey. In *Proceedings of the 4th International Conference Document Analysis and Recognition (ICDAR '97)*, pages 596–599, Ulm, Germany, August 1997. IEEE Computer Society.

[2] Johan Hviid Andersen, Jane Froelund Thomsen, Erik Overgaard, Christina Funch Lassen, Lars Peter Andreas Brandt, Imogen Vilstrup, Ann Isabel Kryger, and Sigurd Mikkelsen. Computer Use and Carpal Tunnel Syndrome: A 1-Year Follow-up Study. *JAMA*, 289(22):2963–2969, 2003.

[3] I. Stephan B. Coasnon, P. Brisset. A music notation construction engine for optical music recognition. In *Proceedings of the International Conference on the Practical Application of Prolog*, pages 115–134, April 1995.

[4] C. S. Kim W. S. Kim B. H. Jun, M. Y. Kim and J. H. Kim. On-line cursive korean character recognition by using curvature models. In *Proceedings of IEEE International Conference on Document Analysis and Recognition*, volume 2, pages 1051 –1054, Atlanta, GA, United States, 1995.

[5] C. Bahlmann, B. Haasdonk, and H. Burkhardt. On-line handwriting recognition with support vector machines — a kernel approach. In *Proceedings of The 6th International Workshop on Frontiers in Handwriting Recognition*, pages 49–54, Taejon City, Korea, August 2002.

[6] L. Belenkaia, J. Lienhard, K. A. Mohamed, and Th. Ottmann. Creation, presentation, capture and replay of freehand writing in e-lecture scenarios. `http://ad.informatik.uni-freiburg.de/-mmgroup/Projects/f-moll/fmoll-Dateien/bericht.pdf`.

[7] T. Breuel. Segmentation of handprinted letter strings using a dynamic programming algorithm. In *Proceedings of Sixth International Conference on Document Analysis and Recognition*, pages 821–826, Seattle, Washington, USA, September 2001.

[8] E. Cohen, J. J. Hull, and S. N. Srihari. Understanding handwritten text in a structured environment: Determining zip codes from addresses. *International Journal of Pattern Recognition and Artificial Intelligence*, 5:221–264, 1991.

[9] S. Connell. *Online Handwriting Recognition Using Multiple Pattern Class Models.* PhD thesis, Dept. of Computer Science and Engineering, Michigan State University, 2000.

[10] Tom Davis. Forensic handwriting analysis in the uk. `http://www.birmingham.ac.uk/english/-bibliography/handwriting/new_web_pages/UKhw.htm`.

[11] Claudio de Stefano, Marco Garruto, and Angelo Marcelli. A saliency-based multiscale method for on-line cursive handwriting shape description. In *Proceedings of the 9th International Workshop on Frontiers In Handwriting Recognition (IWFHR9)*, pages 124–129, Tokyo, Japan, October 2004.

[12] M. Eden. On the formalization of handwriting. *Proc. Symposia in Appl. Math.*, 12:83–88, 1961.

[13] Richard J. Fateman and Taku Tokuyasu. Progress in recognizing typeset mathematics. *Proceedings of SPIE — The International Society for Optical Engineering*, 2660:37–50, 1996.

[14] Katrin Franke, Lambert R.B. Schomaker, Christian Veenhuis, Christian Taubenheim, Isabelle Guyon, Louis G. Vuurpijl, Merijn van Erp, and Geertje Zwarts. WANDA: A generic framework applied in forensic handwriting analysis and writer identification. In M. Koppen A. Abraham and K. Franke, editors, *Design and Application of Hybrid Intelligent Systems. Proceedings 3rd International Conference on Hybrid Intelligent Systems (HIS03)*, page 927. 2003.

[15] Michael Droettboom Ichiro Fujinaga Brian Harrington Karl MacMillan G. Sayeed Choudhury, TimDiLauro. Optical music recognition system within a large-scale digitization project. In *Read at the First International Symposium on Music Information Retrieval*, Plymouth, Massachusetts, United States, October 2000.

[16] D.M. Gavrila and L.S. Davis. Towards 3-D Model-based Tracking and Recognition of Human Movement. In Martin Bichsel, editor, *Int. Workshop on Face and Gesture Recognition*, pages 272–277, June 1995.

[17] I. Guyon, L. Schomaker, R. Plamondon, M. Liberman, and S Janet. UNIPEN project of on-line data exchange and recognizer benchmarks. In *Proceedings of the 12th International Conference on Pattern Recognition, ICPR'94*, pages 29–33, Jerusalem, Israel, October 1994.

[18] Jianying Hu, Sok Gek Lim, and Michael K. Brown. HMM based writer independent on-line handwritten character and word recognition. In *Proceedings of The 6th International Workshop on Frontiers in Handwriting Recognition*, pages 481–485, Taejon City, Korea, August 2001.

[19] Stefan Jaeger and Masaki Nakagawa. Two on-line japanese character databases in unipen format. In *Sixth International Conference on Document Analysis and Recognition (ICDAR)*, pages 566–570, Seatle, United States, 2001.

[20] Moshe Kam, Gabriel Fielding, and Robert Conn. Writer identification by professional document examiners. *The Journal of Forensic Sciences*, 42, 1997.

[21] Eamonn Keogh and M. Pazzani. Scaling up dynamic time warping to massive datasets. In J. M. Zytkow and J. Rauch, editors, *3rd European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD'99)*, volume 1704, pages 1–11, Prague, Czech Republic, 1999. Springer.

[22] Andreas Kosmala and Gerhard Rigoll. On-Line Handwritten Formula Recognition Using Statistical Methods. In *Int. Conference on Pattern Recognition (ICPR)*, pages 1306–1308, Brisbane, 1998.

[23] Nils Krahnstoever and Bart Paulhamus. Development of a korean ocr system. `http://vision.cse-.psu.edu/krahnsto/Projects/Korean/cse581.pdf`.

[24] Joseph B. Kruskal and Mark Liberman. The symmetric time-warping problem: from continuous to discrete. In David Sankoff and Joseph B. Kruskal, editors, *Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparisons.* Addison-Wesley, Reading, Massachusetts, 1983.

[25] J. Laaksonen, J. Hurri, E. Oja, and J. Kangas. Comparison of adaptive strategies for online character recognition. In L. Niklasson, M. Bodén, and T. Ziemke, editors, *ICANN 98. Proceedings of the 8th International Conference on Artificial Neural Networks*, volume 1, pages 245–50, London, UK, 1998. Springer-Verlag London.

[26] J. Laaksonen, V. Vuori, E. Oja, and J. Kangas. Adaptation of prototype sets in on-line recognition of isolated handwritten latin characters. In Seong-Whan Lee, editor, *Advances in Handwriting Recognition*, pages 489–497. World Scientific Publishing, 1999.

[27] S. Lavirotte and L. Pottier. Mathematical formula recognition using graph grammar. In *Proceedings of the SPIE*, volume 3305, pages 44–52, San Jose, CA, 1998.

[28] Hansheng Lei, Srinivas Palla, and Venu Govindaraju. ER^2: An intuitive similarity measure for on-line signature verification. In *Proceedings of the 9th International Workshop on Frontiers In Handwriting Recognition (IWFHR9)*, pages 191–195, Tokyo, Japan, October 2004.

[29] Cheng-Lin Liu, Stefan Jaeger, and Masaki Nakagawa. Online recognition of chinese characters: the state-of-the-art. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(2):198–213, February 2004.

[30] Eugene E. Loos, Susan Anderson, Dwight H. Day Jr., Paul C. Jordan, and J. Douglas Wingate. Glossary of linguistic terms. `http://www.sil.org/linguistics/GlossaryOfLinguisticTerms/-Index.htm`.

[31] J. Mantas. An overview of character recognition methodologies. *Pattern Recognition*, 19(6):425–430, 1986.

[32] Paul Marxhausen. Computer related repetitive strain injury. `http://eeshop.unl.edu/rsi.html`.

[33] Panka Mehra and Benjamin W. Wah. *Artificial Neural Networks: Concepts and Theory.* Ieee Computer Society Press, Los Alamitos, California, US, 1992.

[34] Neila Mezghani, Mohamed Cheriet, and Amar Mitiche. Combination of pruned kohonen maps for on-line arabic characters recognition. In *Proceedings of the 7th International Conference Document Analysis and Recognition*, pages 900–904, Edinburgh, Scotland, August 2003.

[35] Vishvjit S. Nalwa. Automatic on-line signature verification. In *Proceedings of the Third Asian Conference on Computer Vision*, volume 1351. Springer, 1998.

[36] G. Nicchiotti, S. Rimassa, and C. Scagliola. A simple and effective cursive word segmentation method. In *Proceedings of the Seventh International Workshop on Frontiers in Handwriting Recognition*, pages 11–13, Amsterdam, Netherlands, September 2000.

[37] Rafael Palacios and Amar Gupta. Training neural networks for reading handwritten amounts on checks. MIT sloan working paper no. 4365-02., May 2002. `http://ssrn.com/abstract=314779`.

[38] M. Parizeau, A. Lemieux, and C. Gagné. Character recognition experiments using UNIPEN data. In *Proceedings of the 6th International Conference on Document Analysis and Recognition (ICDAR)*, pages 481–485, Seattle, 2001.

[39] Stephen Petrina. Luella cole, sidney pressey, educational psychoanalysis and the establishment of educational psychology, 1921-1931. *History of Education Quarterly*, pages 1–43. Under review.

[40] Toni M. Rath and R. Manmatha. Lower-bounding of dynamic time warping distances for multivariate time series. Technical Report MM-40, Center for Intelligent Information Retrieval Technical Report, 2003. `ciir.cs.umass.edu/pubfiles/mm-40.pdf` .

[41] Toni M. Rath and R. Manmatha. Word image matching using dynamic time warping. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, volume 2, pages 521–527, Madison, WI, USA, 2003.

[42] Eugene H. Ratzlaff. Methods, report and survey for the comparison of diverse isolated character recognition results on the UNIPEN database. In *Proceedings of the 7th International Conference on Document Analysis and Recognition (ICDAR)*, pages 623–628, Edinburgh, Scotland, August 2003.

[43] Amalia Rusu and Venu Govindaraju. Handwritten CAPTCHA: Using the difference in the abilities of humans and machines in reading handwritten words. In *Proceedings of the 9th International Workshop on Frontiers In Handwriting Recognition (IWFHR9)*, pages 226–231, Tokyo, Japan, October 2004.

[44] M. Schmill, T. Oates, and P. Cohen. Learned models for continuous planning. In *Proceedings of Uncertainty 99: The Seventh International Workshop on Artificial Intelligence and Statistics*, pages 278–282, Fort Lauderdale, Florida, USA, January 1999.

[45] Lambert Schomaker and Eliane Segers. A method for the determination of features used in human reading of cursive handwriting. In *Proceedings of the 6th International Workshop on Frontiers in Handwriting Recognition (IWFHR6)*, pages 157–168, Taejon, Korea, August 1998.

[46] Lambert R. B. Schomaker. The NICI stroke-based recognizer of on-line handwriting. `http://hwr.nici.kun.nl/recog/nici-stroke-based-recognizer.html`.

[47] Lambert R. B. Schomaker. Using stroke- or character-based self-organizing maps in the recognition of on-line, connected cursive script. *Pattern Recognition*, 26(3):443–450, 1993.

[48] Lambert R. B. Schomaker. User-interface aspects in recognizing connected-cursive handwriting. In *Proceedings of the IEE Colloquium on Handwriting and Pen-based input*, London, UK, March 1994.

[49] L.R.B. Schomaker, D. Mangalaiu, L.G. Vuurpijl, and M. Weinfeld. Two tree-formation methods and fast pattern search using nearest neighbor and nearest-centroid matching. In *Proceedings of the 7th International Workshop on Frontiers in Handwriting Recognition (IWFHR7)*, pages 261–270, Amsterdam, The Netherlands, September 2000.

[50] S. Srihari, T. Hong, and Z. Shi. Cherry blossom: A system for japanese character recognition. *Symposium on Document Image Understanding Technologies*, pages 127–141, 1997.

[51] S.N. Srihari, V. Govindaraju, and A. Shekhawat. Interpreting handwritten addresses in US mail-stream. In *Proceedings of the Second International Conference on Document Analysis and Recognition*, October 1993.

[52] Tal Steinherz, Ehud Rivlin, and Nathan Intrator. Offline cursive script word recognition - a survey. *International Journal of Document Analysis and Recognition (IJDAR)*, 2(2-3):90–110, 1999.

[53] H.-L. Teulings and L.R.B. Schomaker. Un-supervised learning of prototype allographs in cursive script recognition using invariant handwriting features. In J.-C. Simon and S. Impedovo, editors, *From Pixels to Features III*, pages 61–73. North-Holland, Amsterdam, 1992.

[54] Oivind Due Trier, Anil K. Jain, and Torfinn Taxt. Feature extraction methods for character recognition - a survey. *Pattern Recognition*, 29(4):641–662, 1996.

[55] Katsuhiko Ueda, Kenichi Matsuo, and Yoshikazu Nakamura. A computer-based tool for forensic document analysis - OCR-aided preparation of japanese handwriting comparison charts. In *Proceedings of the first international conference on information technology and applications (ICITA)*, Bathurst, Australia, 2002.

[56] Egon van den Broek, Peter Kisters, and Louis G. Vuurpijl. The utilization of human color categorization for content-based image retrieval. In *Human Vision and Electronic Imaging IX, submitted*, 2004.

[57] J.F. Véle, A. Sánchea, and A.B. Moreno. Robust off-line signature verification using compression neural networks and positional cuttings. In *Proceedings of the 2003 IEEE Signal Processsing Society Workshop*, Toulouse (France), September 2003.

[58] Dan Venolia and Forrest Neiberg. T-cube: a fast, self-disclosing pen-based alphabet. In *Proceedings of the SIGCHI conference on Human factors in computing systems: celebrating interdependence*, pages 265–270, Boston, Massachusetts, United States, 1994.

[59] B. Verma, M. Blumenstein, and S. Kulkarni. Recent achievements in off-line handwriting recognition systems. In *Proceedings of the International Conference on Computational Intelligence and Multimedia Applications*, pages 27–33, Melbourne, Australia, Februari 1998.

[60] Brijesh Verma. A contour code feature based segmentation for handwriting recognition. In *Proceedings of the Seventh International Conference on Document Analysis and Recognition*, Edinburgh, Scotland, August 2003.

[61] Alessandro Vinciarelli. A survey on off-line cursive word recognition. *Pattern recognition*, 35:1433–1446, 2002.

[62] Vuokko Vuori. *Adaptive Methods for On-Line Recognition of Isolated Handwritten Characters*, volume 119 of *Acta Polytechnica Scandinavica, Mathematics and Computing Series*. Finnish Academies of Technology, 2002.

[63] Vuokko Vuori. Clustering writing styles with a self-organizing map. In *Proceedings of The 8th International Workshop on Frontiers in Handwriting Recognition*, pages 345–350, Niagara-on-the-Lake, 2002.

[64] Vuokko Vuori, Matti Aksela, Jorma Laaksonen, and Erkki Oja. Adaptive character recognizer for a hand-held device: Implementation and evaluation setup. *Proceedings of the 7th International Workshop on Frontiers in Handwriting Recognition (IWFHR7)*.

[65] L. Vuurpijl, L. ten Bosch, S. Rossignol, A. Neumann, N. Pfleger, and R. Engel. Evaluation of multimodal dialog systems. In *Proceedings, LREC 2004 Workshop on Multimodal Corpora*, 2004.

[66] Louis Vuurpijl, Ralph Niels, Merijn van Erp, Lambert Schomaker, and Eugene Ratzlaff. Verifying the UNIPEN devset. In *Proceedings of the 9th International Workshop on Frontiers In Handwriting Recognition (IWFHR9)*, pages 586–591, Tokyo, Japan, October 2004.

[67] Louis G. Vuurpijl and Lambert R.B. Schomaker. Finding structure in diversity: A hierarchical clustering method for the categorization of allographs in handwriting. In *Proceedings of the Fourth International Conference on Document Analysis and Recognition*, pages 387–393, Piscataway, NJ, USA, August 1997. IEEE Computer Society.

[68] Louis G. Vuurpijl and Lambert R.B. Schomaker. Multiple-agent architectures for the classification of handwritten text. In *Proceedings of the 6th International Workshop on Frontiers in Handwriting Recognition (IWFHR6)*, pages 335–346, Taejon, Korea, August 1998.

[69] Louis G. Vuurpijl and Lambert R.B. Schomaker. Two-stage character classification: A combined approach of clustering and support vector classifiers. In *Proceedings of the 7th International Workshop on Frontiers in Handwriting Recognition (IWFHR7)*, pages 423–432, Amsterdam, The Netherlands, September 2000.

[70] Louis G. Vuurpijl, Lambert R.B. Schomaker, and Gerben H. Abbink. Uptools 3: A set of programs for viewing, editing, and transforming UNIPEN files. `http://www.unipen.org/uptools3`.

[71] Brendt Wohlberg and John R. Greene. Automatic on-line signature verification using dynamic time-warping. In *Proceedings of the Third South African Workshop on Pattern Recognition*, pages 185–192, Pretoria, South Africa, November 1992.

[72] Qiao Yu and Makato Yasuhara. Recovering dynamic information from static handwritten images. In *Proceedings of the 9th International Workshop on Frontiers In Handwriting Recognition (IWFHR9)*, pages 118–123, Tokyo, Japan, October 2004.

Appendix A

UNIPEN r01_v07 trainset specification

The next three tables show detailed information about the numbers of samples in the various datasets that were used for the creation of the DTW classifier and the prototypesets. For each of the three classes (lowercase letters, uppercase letters and digits), data was extracted from the UNIPEN r01_v07 trainset. The number in the column with the name *Complete set* represents the amount of samples that was taken from this set, minus the number of samples that was removed in the cleaning up (this number of errors is represented in the columns named *Errors*). The amount of samples in each of the three subsets, *trainset*, *testset* and *validationset* are represented in the columns of the same name. The number in the columns called *Duplicates* represents the number of duplicates that were removed from this subset.

Table A.1: Lowercase-letters.

Letter	Complete set	Errors	*train set*	Dupli-cates	*test set*	Dupli-cates	*validation set*	Dupli-cates
a	11113	181	3083	622	3086	618	3088	616
b	3251	87	776	308	778	306	776	307
c	4574	98	1525	0	1239	286	1237	287
d	5676	208	1217	675	1216	676	1220	672
e	16333	281	3539	1906	3535	1909	3536	1908
f	3553	230	941	244	939	245	938	246
g	4133	171	956	422	960	418	965	412
h	6293	228	1492	606	1493	605	1497	600
i	10081	237	2359	1002	2363	997	2361	999
j	2122	61	574	134	577	130	578	129
k	3122	261	719	322	720	321	720	320
l	7846	49	1983	633	1980	635	1983	632
m	4166	177	918	471	914	475	919	469
n	9799	417	2451	816	2436	830	2441	825
o	10718	134	2306	1267	2307	1266	2304	1268
p	4459	158	1093	394	1094	392	1092	394
q	2464	70	649	173	643	178	646	175
r	8730	308	2139	771	2146	764	2140	770
s	8614	50	1998	874	2002	869	1999	872
t	9878	458	2278	1015	2266	1027	2271	1021
u	6059	56	1444	576	1439	581	1440	579
v	2568	26	688	168	685	171	688	168
w	3665	57	912	310	914	308	909	312
x	1987	82	479	184	475	187	477	185
y	3636	109	877	335	879	333	875	337
z	2424	41	588	220	589	219	583	225
Total	157264	4235	37984	14448	37675	14746	37683	14728

Table A.2: Uppercase-letters.

Letter	Complete set	Errors	*train set*	Dupli-cates	*test set*	Dupli-cates	*validation set*	Dupli-cates
A	3527	249	769	407	768	408	774	401
B	2222	51	619	122	626	115	629	111
C	2492	32	654	177	653	178	654	176
D	2303	67	576	192	576	192	581	186
E	4347	107	1093	356	1102	347	1096	353
F	1986	105	506	156	510	152	507	155
G	2035	106	573	106	571	107	572	106
H	2267	78	489	267	490	266	488	267
I	3627	111	892	317	893	316	896	313
J	1801	37	515	86	517	83	513	87
K	2051	71	444	240	449	235	446	237
L	2768	1	556	367	556	367	555	367
M	2307	30	565	204	567	202	568	201
N	2566	156	650	206	648	207	649	206
O	4128	19	1009	367	1018	358	1011	365
P	2414	74	644	161	642	163	642	162
Q	1802	124	438	163	441	160	438	162
R	2810	153	743	194	750	187	741	195
S	2974	292	745	247	755	236	747	244
T	3057	203	838	181	843	176	841	178
U	2722	21	630	278	624	283	625	282
V	1857	18	574	45	575	44	577	42
W	2044	44	506	176	511	170	507	174
X	1643	75	435	113	432	116	433	114
Y	1819	294	466	141	463	143	459	147
Z	1914	82	502	136	502	136	502	136
Total	65483	2600	16431	5405	16482	5347	16451	5367

Table A.3: Digits.

Digit	Complete set	Errors	*train set*	Dupli-cates	*test set*	Dupli-cates	*validation set*	Dupli-cates
0	3726	41	768	474	773	469	772	470
1	4055	60	985	367	982	370	983	368
2	3807	56	1051	218	1053	216	1055	214
3	3658	23	990	230	982	237	988	231
4	3668	100	853	370	851	372	853	369
5	3475	40	937	222	928	230	941	217
6	3218	209	768	305	761	312	762	310
7	3515	24	912	260	915	257	916	255
8	3514	39	943	229	939	232	941	230
9	3494	3	990	175	983	182	984	180
Total	36130	595	9197	2850	9167	2877	9195	2844

Appendix B

Prototype specification

Table B.1: Specification of the lowercase letters prototypes.

Letter	Protos before postprocessing	Resample and Average	Mergesamples
a	99	96	98
b	43	38	39
c	110	92	97
d	80	77	78
e	61	59	60
f	59	51	46
g	69	56	55
h	53	50	51
i	43	29	33
j	42	30	35
k	46	33	43
l	76	58	57
m	58	54	49
n	96	89	86
o	62	57	58
p	57	51	52
q	50	41	48
r	64	58	61
s	58	57	58
t	53	48	49
u	79	64	65
v	38	29	30
w	57	53	53
x	33	30	31
y	54	44	52
z	43	40	39
Total	1583	1384	1423

Table B.2: Specification of the uppercase letters prototypes.

Letter	Protos before postprocessing	Resample and Average	Mergesamples
A	59	52	57
B	57	53	55
C	49	48	48
D	43	39	41
E	50	47	49
F	31	25	26
G	45	43	42
H	25	23	25
I	39	38	37
J	30	23	27
K	35	34	34
L	39	36	36
M	52	46	47
N	35	32	34
O	44	43	43
P	47	40	43
Q	41	40	40
R	54	50	51
S	41	40	39
T	39	37	39
U	33	27	32
V	33	30	30
W	40	38	39
X	33	32	33
Y	34	31	31
Z	28	27	27
Total	1056	974	1005

Table B.3: Specification of the digits prototypes.

Digit	Protos before postprocessing	Resample and Average	Mergesamples
0	59	59	59
1	51	48	48
2	67	66	65
3	44	43	44
4	47	45	45
5	47	43	46
6	37	36	36
7	46	43	45
8	46	44	44
9	46	43	45
Total	490	470	477

Appendix C

Intuitivity Experiment instructions

In every trial, a 1+5x3 matrix filled with characters is shown (see Figure C.1 for an example). The character at the top left is the so-called *query*. This *query* was offered to a handwriting recognizer and the 15 characters on the right show the results.

For each trial, you are requested to select the letters that resemble the *query* enough (do this by clicking on them). You may decide what criteria to use and what "enough" is. You also may decide how many letters you select. You can even decide not to select any letters. The order in which you click is not relevant: it is not necessary to make a ranking.

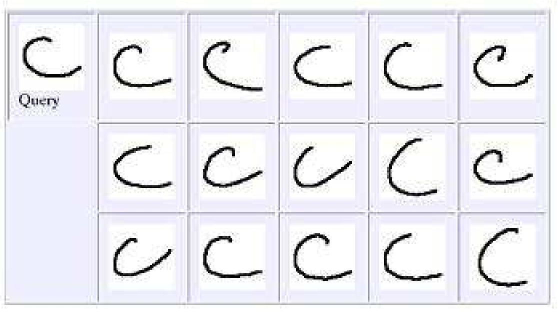


Figure C.1: Example of a trial.

The background of a selected letter turns green. If you want to undo a selection, click on the letter again. If you want to deselect all clicked letters, click "Reset" at the bottom of the screen.

As soon as you have finished one trial, click on "Submit" at the bottom of the screen. The next trial will automatically appear. As soon as you have clicked "Submit", it is not longer possible to make any changes.

Please note: **there are no correct or incorrect responses**, try not to think too long!

The experiment starts with three practice-trials in which you can explore the possibilities. After the three practice-trials, the real experiment will automatically begin. At the top of the screen you can see how many trials you have finished. At the end of the experiment you can close your browser.

Appendix D

SPSS syntax

D.1 Recognition performance experiment

```
DATA LIST Free /Letter(A) Recognize Classify Weight.
* Colomn 1: Character.
* Colomn 2: Classifier (1=Mergesamples, 2=Resample and Average).
* Colomn 3: Correct or incorrect (0=Incorrect, 1=Correct).
* Colomn 4: Frequency.
VAR LABELS Recognize ’Recognizer’ Classify ’Classification’.
VALUE LABELS Classify 0 ’Fout’ 1 ’Goed’.
BEGIN DATA.
a  1 0  216
a  1 1  2872
a  2 0  214
a  2 1  2874
END DATA.
WEIGHT BY Weight.
SORT CASES BY Character .
SPLIT FILE SEPARATE BY Character .
* Compare performance of two recognizers.
CROSSTABS
  /TABLES=herken  BY Classify
  /FORMAT= AVALUE TABLES
  /STATISTIC=CHISQ
  /CELLS= COUNT
  /METHOD=EXACT TIMER(5).
SPLIT FILE OFF.
```

D.2 Intuitivity experiment

D.2.1 Hypotheses H_A and H_B

```
GLM
  a_mergesamples a_resampleandaverage a_hclus
  /WSFACTOR = rater 3 Simple(1)
  /METHOD = SSTYPE(3)
  /PLOT = PROFILE( rater )
  /EMMEANS = TABLES(OVERALL)
  /EMMEANS = TABLES(rater)
  /PRINT = DESCRIPTIVE ETASQ
  /CRITERIA = ALPHA(.05)
  /WSDESIGN = rater .
```

D.2.2 Hypothesis H_C

```
GLM
  a_resampleandaverage a_hclus a_mergesamples
  /WSFACTOR = rater 3 Simple(1)
  /METHOD = SSTYPE(3)
  /PLOT = PROFILE( rater )
  /EMMEANS = TABLES(OVERALL)
  /EMMEANS = TABLES(rater)
  /PRINT = DESCRIPTIVE ETASQ
  /CRITERIA = ALPHA(.05)
  /WSDESIGN = rater .
```