Handwritten Recognition

Fabian Alenius, Kjell Winblad and Chongyang Sun

August 1, 2011

Abstract

Recognizing handwritten text means transforming a graphical representation of text into its symbolic representation. Handwritten recognition is used in a wide variety of applications. This paper describes and empirically evaluates a handwritten text recognition system based on Hidden Markov Models. The results show that Hidden Markov Models can successfully be used for handwritten recognition.

1 Introduction

Recognition of handwritten text has been a popular research area for decades because it can be used in many different applications [6]. There are two different approaches to handwritten recognition, *online* and *offline*. In the online approach we know the order in which the strokes and individual points were drawn. This information can easily be captured if the text is recorded by a digital pen or on a touchscreen. In the offline approach we are only given the final image. Online recognition is primarily used for signature verification, author authentication and digital pens. Application areas for offline recognition include postal automation, bank cheque processing and automatic data entry [6]. Formally, handwritten recognition is the task of transforming a language represented in graphical form into its symbolic representation [7].

The ultimate goal in handwritten recognition is to recognize words. However, one way to potentially decompose or simplify the problem is to segment words into its individual characters [3]. Segmentation can either be done *explicitly* or *implicitly*. Explicit segmentation tries to separate the word at character boundaries while implicit segmentation separates the word into equal sized frames. The implicit frames, each represented by a feature vector, are then mapped into characters.

This paper is focused on offline handwritten recognition. We attempt to tackle both character and word recognition. To simplify the word recognition problem, we assume that the images containing the words have already been explicitly segmented into new images containing the separated characters. For word recognition, we also assume that the words come from a finite lexicon. Finally, we assume that the strokes that together compose a character has a width of 1 pixel. These simplifying assumptions were made due to the limited time available for this project. In the following sections we describe how a handwritten text recognition system was developed based on Hidden Markov Models and evaluate its performance.

2 Previous work

Because handwritten recognition is such a well-researched area there is a wealth of literature available.We mention only a few references that we found helpful. Cheriet et al. [1] gives a good review of the development of handwritten recognition. They also go on to give a broad overview of feature extraction and classification using a plethora of different techniques. Rabiner, L. R. [8] gives an excellent review of HMMs and the Baum-Welch training algorithm, as well as how to apply them in speech recognition. El-Yacoubi et al. [3] introduce an approach to recognize text using Hidden Markov Models with explicit word segmentation. Laan et al. [4] evaluate three different initial model selection techniques for the Baum-Welch algorithm, randomized, uniform and count-based initialization. Despite impressive progress over the last couple of decades, performance is still far away from human performance.

3 Method

This section describes the handwritten text recognition system. It starts by giving a top level overview of the system and then it describes the details such as the HMM implementation, feature extraction and the datasets used for training.

3.1 Overview of Hidden Markov Models

For pattern recognition problem, like handwritten image recognition, there will always be some randomness and uncertainty from the source recognition data. Stocastic modeling deals with these problem efficiently by using probabilistic models [2]. Among such stochastic approaches, Hidden Markov Models have been widely used to model dynamic signals. The Hidden Markov Model treats the data as a sequence of observations, while using hidden states that are connected to each other by transition probabilities.

An HMM is characterized by the following [8]:

1. N, the number of states in the model.

2. M, the number of distinct observation symbols.

3. A, the transition probability distribution.

4. B, the observation symbol probability distribution for each state

5. π, the initial state state distribution.

In contrast to a knowledge-based approach, HMMs use statistical algorithms that can automatically extract knowledge from samples. Also, HMMs model patterns implicitly with different paths in the stochastic work. The modeling power can be enhanced by adding more samples [2].

3.2 General Overview of Classifiers

The handwriting recognition system has two levels of classifiers. The first classifier is a function that takes an image as input and outputs a character. The second classifier is a function that takes a string of characters as input and outputs a word. A flowchart that shows the classification process can be seen in figure 1.

The classifiers contains HMMs for all elements in the set of possible outputs. So if the character classifier is trained to recognize the 26 Latin characters, it will contain 26 HMMs. One HMM for every character. When the classifiers are trained they are given input examples for all possible outputs. If the input I is given to one of the classifiers the following steps are performed to calculate the output:

1. The probability of I is calculated for all HMMs contained in the classifier:

   (a) I is translated to a sequence of observation symbols $\mathbf{O} = O_1, O_2, ..., O_n$. If I is a string of characters and the output of the classifier is a word, the translation is straightforward. Every character in the string is simply translated to the corresponding observation symbol. There are also special observations for the start and end

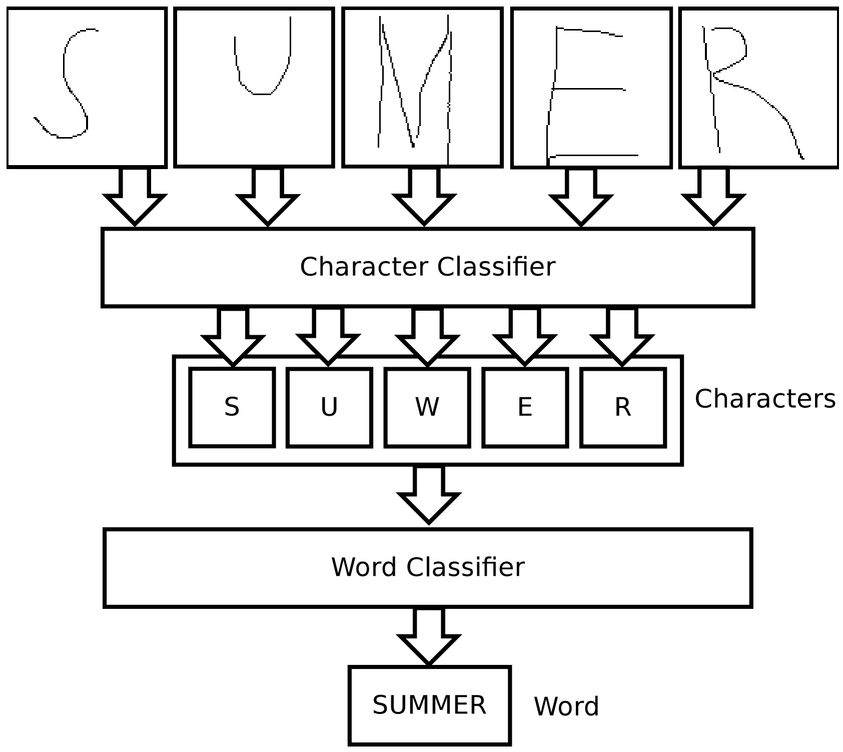


Figure 1: Flowchart of classification process.

states. This is explained in more detail in the following sections. If I is an image, the image is first segmented to a sequence of segments. An observation symbol is then obtained from all segments. See section 3.3 for more information about the image feature extraction.

(b) The forward calculation algorithm [8] is then used to calculate the probability of $\mathbf{O}$ given the HMM.

2. The output symbol with the highest probability in the previous step is returned as output.

The following parameters must be supplied when a classifier is created[1]:

[1] A few more parameters can be given but are not listed here because of lack of importance. See the source code of the system for information about other parameters. How the source code can be obtained is explained in appendix A.

- The set of possible output symbols and corresponding training examples.
- The initialization method that shall be used by the HMMs.
- If the training examples shall be used to train the model with the Baum-Welch [8] training algorithm.

3.3 Image Preprocessing and Feature Extraction

The character classifier classifies images that contain handwritten characters, as described in section 3.2. A sequence of observation symbols is extracted from the supplied training data and used when an image is to be classified. This process is called feature extraction.

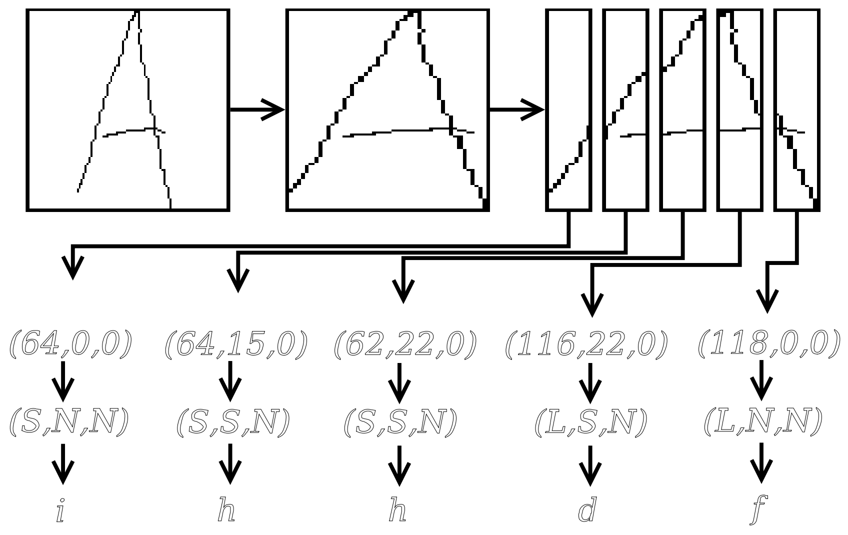


Figure 2: An illustration of the feature extraction process.

As mentioned in section **??**, our system assumes that there is one image per character, that the lines in the characters consists of a single color and that the lines are one pixel wide. The feature extraction process can be divided into the following steps:

1. The scaling step makes sure that the character fills the whole image. Because of this step, it does not matter where in the given image the character is painted. The scaling may cause the problem that the lines

are getting thicker than one pixel after scaling if the original painted character just fills a small part of the image[2]. This problem will be avoided if the training examples contain images where the character fills a small part of the image. The following algorithm is used to do the scaling:

   (a) The minimum rectangle R in the image that contains the whole character is found.

   (b) The rectangle R is scaled to fill the size of the original image. The scaled version of R is returned as the scaled image.

2. The new scaled image is sliced into N vertical segments of the same size.

3. An observation symbol is extracted from every segment in the following way:

   (a) The number of pixels in the three largest components in the segment are found and put into a triple (s_1, s_2, s_3) which is sorted so that the largest number is first. A component is defined as a set of colored pixels that are connected and that contains all pixels that are connected to one of the pixels in the set. Two colored pixels are connected if they are neighbors or if it is possible to create a path of colored connected pixels between them. All pixels except the border pixels have 8 neighbors. If the segment contains less than three components, the triple is filled with zeros.

   So for example if a segment contains two lines. One line containing 10 pixels and another line containing 5 pixels. Then the resulting triple will be $(10, 5, 0)$.

   (b) The elements in the triple is classified as *Large*, *Small* or *None*. The classification function c is defined as in Equation 1. The constant d in the equation is given as a parameter to the feature extraction.

$$c(s) = None \text{ if } s = 0, Large \text{ if } s > d \text{ and } Small \text{ otherwise.} \quad (1)$$

   The triple (s_1, s_2, s_3) is translated to a triple of classes (c_1, c_2, c_3) by applying the function c to all elements in the triple.

[2]The standard Java image scaling algorithm is used for the scaling.

(c) The triple of classes is mapped to an observation symbol. In total there are 10 different triples of classes and there is one observation symbol per triple of classes. So in total there are 10 different observation symbols.

The classification constant c and the number of segments N are parameters to the feature extractor. An example of feature extraction for an image can be seen in figure 2.

3.4 Dataset

We tried to find a dataset with handwritten text at the beginning of the project, but it turns out there are not that many available. The datasets that do exist, like following image example. See Figure figure: wordsexamples, they would have needed a lot of preprocessing before we could use them in our project. We would have had to implement baseline slant normalization, skew correction, skeleton and so on.

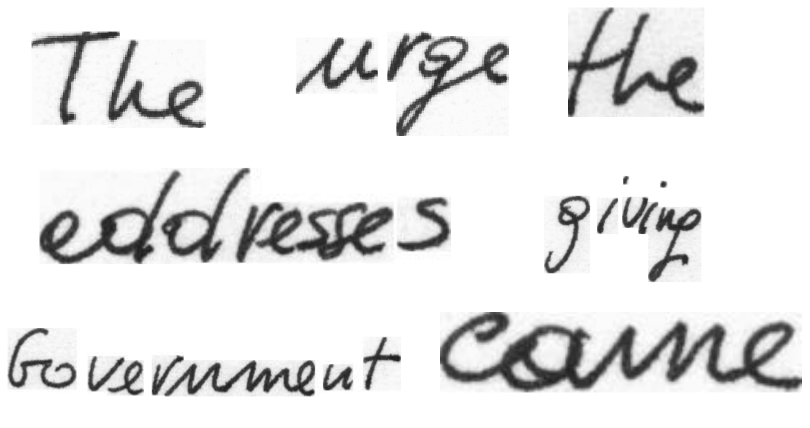


Figure 3: Word image examples

Therefore, instead of spending a lot of time preprocessing the datasets, we implemented a Graphic User Interface to create our own dataset. The biggest advantages of this solution is that our solution records one pixel wide letters and the characters are already separated. The most important part of the work, image processing, was thus reduced significantly.

Furthermore, if the vocabulary is relatively large, we found that it became easier for us to test the HMM. This is because our word training data is made up of randomly chosen samples.

4 Implementation Issues

While implementing HMM for both character recognition and word recognition, we encounter some practical issues.

4.1 Initial Model Selection

Hidden Markov models can be efficiently trained by Baum Welch algorithm, which is an iterative process for estimating parameters for HMMs. As an iterative algorithm, BW starts from an initial model and estimates transition and emission probability parameters by computing expectations via Forward and Backward, which sums over all paths containing a given event until convergence is reached.

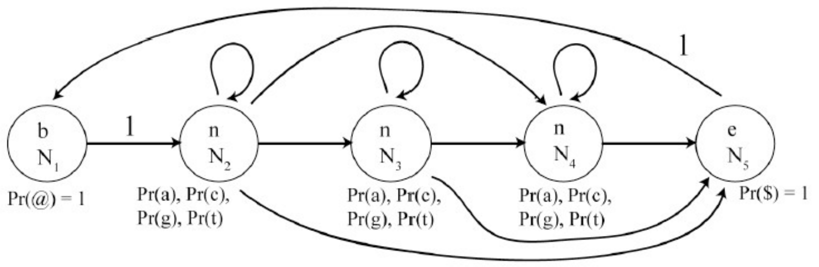
Since the Baum-Welch algorithm is a local iterative method, the resulting HMM and the number of required iterations depend heavily on the initial model. Of the many ways to generate an initial model, some techniques consider the training data while others do not [5].

According to the paper [5], we tried the three initialization strategies as well, namely, count-based, random and uniform tested on the training data for character HMM model.

4.2 Topology of HMM

Since EM method assumes that the model is correct, it is important to devise a suitable topology before training starts. The topology of the model is usually built by using a prior knowledge on the data [9]. Generally, for machine learning or handwriting signals, a left-to-right HMM is often carried out, which no back transitions from right to left are allowed.

1. Character classifier: A model is created for each class in the training phase.



b
N1
n
N2
n
N3
n
N4
e
N5
1
1
Pr(@) = 1
Pr(a), Pr(c), Pr(g), Pr(t)
Pr(a), Pr(c), Pr(g), Pr(t)
Pr(a), Pr(c), Pr(g), Pr(t)
Pr($) = 1

Figure 4: HMM topology for a DNA sequence of length three.

Since alphabet in English is limited, up to 26, we can take the cost that each separated model is trained for each character using segmented word images. Furthermore, We specialized our beginning and ending states denoted by b and e respectively according to one suggested model for training [5]. Special beginning and ending states are included generally because then the multiple observation training sequences are concatenated together to form one observation sequence for input into the Baum-Welch algorithm[5].

In the concatenated string, the original segments of character images sequences are connected by the special observation symbols "@" and $. Also the special beginning state always transitions to the first normal state, and the special ending state always transitions to the special beginning state. Then forward and backward algorithm is used to choose the model that has the most probability of the observation sequence. See Figure 4[5].

2. Word classifier: A single model is constructed for the whole vocabulary.

   Generally it's natural to implement Hidden Markov Models for each of the word when vocabulary is limited and small. And similar topologies can be found among those HMMs. Then we can also use forward and backward algorithm to choose the most likely model to tell which word it is as what we did for character classifier above. It works really well when we classify the test data to get the results within 20 words as following.

   ["dog","cat","pig","love","hate","scala","python","summer","winter", "night",daydream","nightmare","animal","happiness","sadness","tennis", "feminism","fascism","socialism","capitalism"]

However, it is time consuming and not realistic to generate HMMs for every word when the vocabulary is relatively large. Therefore, later on we came up with another HMM topology to ideally describe unconstrained words of mixed style.

To be specific, a Hmm with a topology that is a complete directed graph is modeled. It has 28 hidden states(26 for 26 letters and 2 for @ and $). See Fiture 5.

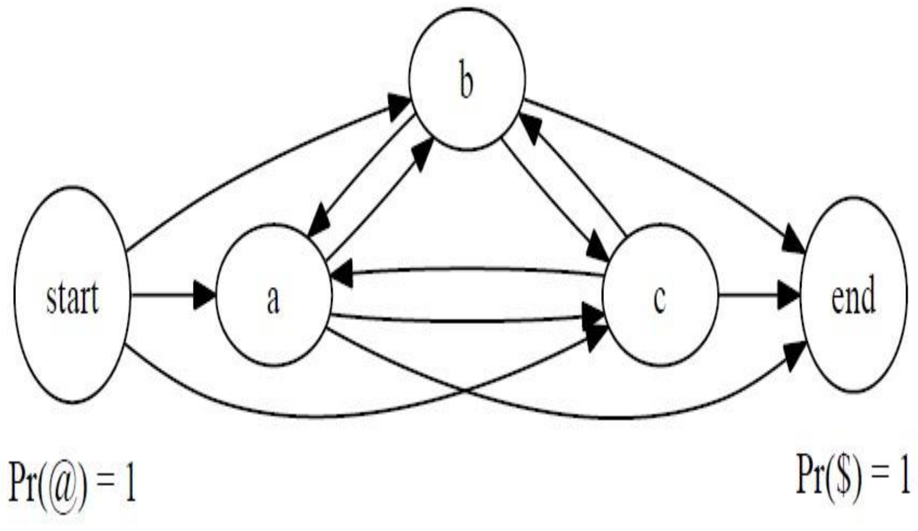


Figure 5: Word HMM topology for a three-letter alphabet.

Transition matrix is a 28 * 28 matrix, which is estimated from the lexicon analysis. For example if there's only three words in our vocabulary: dog cat cap for "A to T is set to 0.5 and A to P is set to 0.5" and "A to other letters is set to 0".

Each state's observation is the letter we observed from the character classifier in the previous step. For setting the observation probability matrix, it's reasonable to set them to the probabilities from the test results from word classifier. For example, if we have 10 test examples , for A and 5 of them are classified to be A, 3 to B and 2 to C by the character classifier, then P(observation A)=0.5, P(observation B)=0.3 and P(observation C)=0.2. Then we will assign the row for A as"0.5 0.3 0.2 0 0 ......0 0".

4.3 prevention of underflow

For sufficiently large t states, forward variable $\mathbf{a}_t(i)$ and backward variable $\mathbf{b}_t(i)$ head exponentially to zero and exceed the precision range of computer. The only reasonable way of performing the computation is by incorporating a scaling procedure[rabineer]. For each t, first we compute $\mathbf{a}_t(i)$ according to the induction fomula(20) in rabiner's paper [8],and then multiply it by a scaling factor $\mathbf{c}_t$, where c is equal to $\frac{1}{\displaystyle\sum_{i=1}^{N} \mathbf{a}_t(i)}$.

To escape the underflow situation in viterbi algorithm, in order to give the maximum likelihood state sequence, we can simply adopt adding log probability rather than multiplying probabilities. Then no scaling is required.

4.4 Handling null transitions

Another problem may occur when training HMM parameters is that transition probability is very likely to be null. In our implementation, we initialze them to very small number, such as 10^{-10}.

5 Result

In this section we present the results from evaluating the system described in Section 3. We discuss the results in more detail in Section 6. Finally, in Section 7 we present some future changes to the system that we believe will improve the results.

5.1 Word Classifier and Different Initialization Methods

In this subsection the classification results for the words in Table 1 will be presented. See Section 3 for a description of the implementation of the classifiers. The training and test example words were randomly generated with the generator having the properties in Table 2. See Section **??**, for more information about the word example generator.

We used a total of 100 test examples to test the accuracy of the created classifiers. Five test examples each for the 20 words. The test examples were generated using the same properties as the training examples. Two initialization methods, count based initialization and random initialization, were tested with 100, 200, 400, 800 and 1600 training examples. The results of the test is presented in Table 3. It contains the test scores for the two initialization methods before and after training with the Baum-Welch algorithm. The test score is defined as the percentage of correctly classified test examples.

dog	cat	pig	love	hate
scala	python	summer	winter	night
daydream	nightmare	animal	happiness	sadness
tennis	feminism	fascism	socialism	capitalism

Table 1: Words supported by the resulting classifier.

Probability of extra letter at position	0.03
Probability of extra letter equal neighbor	0.7
Probability of wrong letter at position	0.1
Probability of letter missing at position	0.03

Table 2: Properties obeyed by the word training example generator.

5.2 Character Classification with Different Parameters

As described in Section 3.3 the image feature extraction step in the character classifier takes two parameters. The first parameter is the number of segments that shall be created. The second parameter is the size classification factor which is used in equation 1. For the experiment we only had 100 examples for each of the 26 characters. How the examples are produced is described in Section **??**. An initial experiment was performed to test count based initialization and random initialization before and after training with the Baum-Welch algorithm. The initial experiment shows that there is probably too few training examples for the training to have any positive effect for count based initialization. This could possible be fixed to some extend with some kind of smoothening of the model produced by the training. 10 test examples and 90 training examples for every character were selected

NOE	RIBF	CBIBT	RIAT	CBIAT	RITT	CBITT
100	3	99	1	0	6	3
200	2	100	13	36	12	5
400	2	100	90	95	23	7
800	1	100	100	100	46	13
1600	5	100	100	100	96	26

Table 3: Test with different number of training examples and different initialization methods. NOE="number of training examples for every word", RIBF="random initialization score before training", CBIBT="count based initialization score before training", RIAT="random initialization score after training", CBIAT="count based initialization score after training", RITT="random initialization training time (minutes)", CBITT="count based initialization training time (minutes)"

randomly from the example set for the experiments. The results from the initial experiment can be found in table 4. In the initial experiment 1.3 was used as the size classification factor and the number of segments was set to 7.

NOE	RIBF	CBIBT	RIAT	CBIAT
90	4	53	16	16

Table 4: Test of the character classifier with different initialization methods and before and after training. NOE="number of training examples for every word", RIBF="random initialization score before training", CBIBT="count based initialization score before training", RIAT="random initialization score after training", CBIAT="count based initialization score after training"

Only count based initialization is looked upon in the experiment of different parameters, because the initial experiment showed that the best result seems to be produced when just using count based initialization and no training. When testing the parameters, 5 models were created in the same way as in the initial experiment. The average accuracy for these 5 models when testing them with their own test example set was recorded as the accuracy for the configuration. For all 5 models that were created, different training example sets and test example sets were randomly selected. 90 training examples and 10 test examples were used for every character as in the initial experiment. The results of the experiment can be seen in table 5.

CCF\NOS	4	5	6	7	8	9	10	11	12
0.7	32	38	43	42	44	44	45	48	48
1.0	35	39	42	41	44	49	48	47	48
1.3	43	47	53	53	53	56	57	56	59
1.6	47	49	53	55	57	57	58	57	57
1.9	50	52	56	57	58	57	58	59	59
2.2	55	57	60	60	62	62	62	65	63
2.5	57	59	60	61	62	62	65	65	63
2.8	55	60	61	62	64	65	66	64	65
3.1	55	59	64	63	65	66	65	67	**68**
3.4	56	60	62	64	64	65	65	67	65
3.7	55	62	62	64	63	64	67	67	67
4.0	56	61	61	65	64	64	66	66	65
4.3	56	61	63	65	65	65	66	67	66
4.6	55	60	62	65	65	65	67	**68**	66
4.9	55	60	63	67	67	66	67	67	66
5.2	54	60	63	65	64	67	66	65	65
5.5	53	59	63	63	66	66	67	67	66
5.8	51	60	63	65	63	67	68	66	66

Table 5: Results for character classification test with different parameters. The scores are percentage of correctly classified characters. NOS="number of segments", CCF="component classification factor"

6 Discussion

The results in Section 5.1 clearly show the importance of having enough training data. When using the count based initialization method, the accuracy actually becomes worse after training the model with Baum-Welch when using less than 800 training examples. The effects of not having enough training data when using the Baum-Welch algorithm is further discussed in [8]. One way to potentially solve this problem is to use some kind of smoothening of the probability matrices after training. The smoothening could be done by for example setting all transitions with probability zero to a small value greater than zero. The count based initialization method gives almost perfect accuracy on the test set without the Baum-Welch training.

The vocabulary used by the classifier is quite small as it only contains 20 words. The accuracy would probably be worse with a larger vocabulary with many words that are similar to each other. The classifier implementation would also have performance problems for many applications with large vocabularies. This is because the time complexity of classifying an example grows linearly with the size of the vocabulary. This is easy to see if one consider that the classifier contains one HMM for every word in the vocabulary and that the forward calculation algorithm needs to run for all HMMs when an example is to be classified. When the training examples are generated as previously described it's probably not very useful in real applications. For most applications it would be better to use a spell-checking algorithm that can find words similar to a string in an effective way. However, if the training data instead is the result of a handwritten recognition system it could be more useful, because then the model could learn to correct mistakes that the handwritten recognition system does.

We believe that the amount of available training data is a liming factor for the character classifier. Because when we train the system, after it has been initialized with the count based method, with the Baum-Welch algorithm the performance becomes worse. If the classifier is to be accurate for a random person's handwriting it would be beneficial to let more people paint training examples to get a more generalized solution.

Our approach will always have problem with characters that look similar to other characters when turned upside down. For example "M" and "W" look exactly alike if they are turned upside down for some handwriting styles. Why this problem occur is obvious if one looks at the feature extraction process.

7 Future Work

While the results show that using HMMs for building a handwritten recognition system is a viable alternative, there is a lot of room for improvement. One way to potentially improve performance is by extending the feature extraction to consider segments from top to bottom as well as from left to right. This means that the observation sequence would become twice as long, given a square image.

During scaling, the lines can become wider than one pixel. Because we are categorizing the strokes based on how many pixels they contain, this is a problem. This could be fixed by adding a thinning phase after the scaling is completed to make the strokes one pixel wide again.

Another way to improve performance is by using a different feature representation. For example, vector quantization can be used to map vectors into a smaller space which can then be used as observations in the HMM. Using this method we would not be reliant on arbitrary constants to map features into observations. It would also make it easier to the extend the system to use additional features.

We saw in our results that adding more training data improves performance so this is an easy way to enhance the system. Finally, to make the system more general we should allow for words that are not pre-segmented into characters.

References

[1] Mohamed Cheriet, Nawwaf Kharma, Cheng-Lin Liu, and Ching Y. Suen. *Character Recognition Systems. A Guide for Students and Practioners.* Wiley-Interscience. Wiley, 2007.

[2] Wongyu Cho, Seong-Whan Lee, and Jin H. Kim. Modeling and recognition of cursive words with hidden Markov models. *Pattern Recognition*, 28(12):1941–1953, December 1995.

[3] A. El-Yacoubi, R. Sabourin, C. Y. Suen, and M. Gilloux. An hmm-based approach for off-line unconstrained handwritten word modeling and recognition. *IEEE Trans. Pattern Anal. Mach. Intell.*, 21:752–760, August 1999.

[4] Nicholas C. Laan, Danielle F. Pace, and Hagit Shatkay. Initial model selection for the baum-welch algorithm as applied to hmms of dna sequences.

[5] Nicholas C Laan, Danielle F Pace, and Hagit Shatkay. Initial model selection for the Baum-Welch algorithm as applied to HMMs of DNA sequences . *DNA Sequence.*

[6] Umapada Pal, Tetsushi Wakabayashi, and Fumitaka Kimura. Comparative Study of Devnagari Handwritten Character Recognition Using Different Feature and Classifiers. pages 1111–1115, 2009.

[7] Réjean Plamondon and Sargur N. Srihari. On-line and off-line handwriting recognition: A comprehensive survey. *IEEE Trans. Pattern Anal. Mach. Intell.*, 22(1):63–84, 2000.

[8] L. R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. 77(2):257–286, 1989.

[9] Ching Y Suen. CHARACTER RECOGNITION SYSTEMS A Guide for Students and Practioners.

A Source Code

The source code for the system described in this report can be found at the following address:

`http://github.com/kjellwinblad/HandReco`

The source code can be downloaded as a ZIP-archive or cloned using git[3].

B Result Reproduction

The test results presented in section 5 are produced by scripts written in the Jython[4] programming language. The following steps will run the test scripts:

1. Follow the instructions in appendix A to download the source code for the system.
2. Follow the instructions in the `README.md` file in the root of the source code directory. These instructions will help you to set up your environment for running the test scripts.
3. Open a system terminal and execute the following commands (Notice that the path may look different in your system):
   (a) `cd /path/to/HandReco/src/test`
   (b) To run tests for word classifier:
   (c) `jython word_classifer_tester.py`
   (d) To run tests for character classifier:
   (e) `jython character_classifier_tester.py`

C Testing Handwriting Recognition in Graphical User Interface

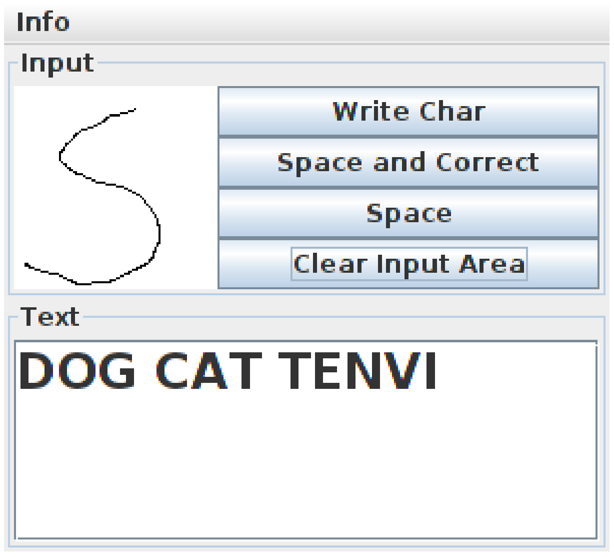
A graphical user interface (GUI) has been created in order to test the handwriting recognition system in practice. See figure 6 for a screenshot of the graphical user interface. The following steps can be used to run the GUI:

[3]http://git-scm.com/

[4]Jython is a version of Python for the Java Virtual Machine (http://www.jython.org/)

1. Follow the instructions in appendix B to step 2.
2. Open a system terminal and execute the following commands (Notice that the path may look different in your system):
   (a) `cd /path/to/HandReco/src/gui}`
   (b) `jython hand_reco_writer.py`

The GUI can only recognize capital Latin letters. To see which words are available for word corrections click on the **Info⇒Available Words...** menu item. To input a character, first paint the character in the paint area and then press the **Write Character** button. To do a space, press the **Space** button. To do a space and let the word classifier correct the last word, press the **Space and Correct** button.



DOG CAT TENVI

Figure 6: Screenshot of HandReco Writer.