Institut für Visualisierung und Interaktive Systeme
Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Studienarbeit Nr. 2233

Addressing the Punctuation Marks Problem in Automatic Handwriting Recognition

Thomas Müller

Studiengang:	Informatik
Prüfer:	Prof. Gunther Heidemann
Betreuer:	Dr. Moisés Pastor i Gadea Dipl.-Inf. Sebastian Klenk
begonnen am:	September 1, 2009
beendet am:	April 1,2010
CR-Klassifikation:	I.2.7, I.4.7, I.5.1

Erklärung

Hiermit versichere ich, diese Arbeit
selbständig verfasst und nur die angegebenen
Quellen benutzt zu haben.

_____

(Thomas Müller)

Abstract

In this work the functionality of a modern handwriting recognition system is explained. The need for a good punctuation detection is briefly motivated. Experiments to enhance the performance of the system on the detection of punctuation marks are developed and analysed. Finally, the unsatisfying results of these experiments are discussed and new ideas for future research are sketched.

Contents

1 Introduction

In this work it was attempted to improve the automatic handwriting recognition (HR) by improving the recognition of punctuation marks. In most of the modern HR systems punctuation marks are ignored as they are usually very small and hard to detect. Nevertheless they are very important for the meaning of some sentences and they allow us to separate the text into phrases instead of text lines and therefor provide a better context to detect the actual words of the text. Handwriting Recognition itself is a special case of pattern recognition (PR). On the following pages first a short introduction to pattern recognition and Handwriting Recognition is given. Afterwards the special importance of punctuation marks for the transcription is explained. In chapter 2 the state of the art of HR is explained in detail. The experiments and conclusions can be found in chapter 3 and 4. At last chapter 5 tells about ideas and advices to future research in this field.

1.1 Pattern recognition

A pattern recognition system attempts to understand its environment through physical signals. Common signals in PR tasks are images, audio signals like speech or music and documents. A generic PR system can be described as a sequence of three major steps. Those are preprocessing and filtering, feature extraction and classification.

1.1.1 Preprocessing and filtering

Preprocessing and filtering can be summarised as the task of making the system invariant to every variation of the signals which is not necessary for the classification. Real world signals usually contain noise that has to be removed before a further processing is possible. A standard approach used e.g. in image processing is to consider high frequencies as noise and remove them with a **low-pass filter**. Other preprocessing steps attempt to normalise the signals to make them easier to compare. For example, it is common in image recognition to transform the images in a way that all objects are scaled and rotated in the same way. Furthermore the background of the image might be removed.

1.1.2 Feature extraction

In the feature extraction step the signal is mapped to a more sparse representation. One attempts not to lose information in this mapping. A popular approach are vectors of real numbers as there are many well investigated classifiers for this representation.

1.1.3 Classification

The preprocessing and feature extraction steps have to be adapted to the specific task, but the classification is normally more generic. In the classification step, signals of a common type are labelled with the same name. Emails can be classified as *spam* or *no spam* or images of digits with the corresponding digit. There a two principal approaches to classification, the statistic-geometric and the structural one. The statistic approach

considers the signal x as a random variable. The class c as a function of x is then the most probable class under x. The classification function $c(x)$ is then called optimal Bayes classifier.

$$c(x) = \arg\max_c P(c|x) = \arg\max_c \frac{P(c)\,P(x|c)}{P(x)} = \arg\max_c P(c)\,P(x|c) \tag{1}$$

For a set of tuples of feature vectors and class labels the optimal Bayes classifier can be calculated and then be used to classify new unknown feature vectors. Such a set is called training set and it is the common way to create a classifier. The training set can be used to introduce the analog geometric interpretation. In the -usually high dimensional- space of all the feature vectors a classifier can be described as a plane that separates the points of one class from the points of another class. See figure 1 for a simple example of such a separation line.
In the structural approach a pattern is seen as a composition of primitive patterns. This primitive patterns can be combined using a set of syntactical rules. The recognition problem is then to find the grammar that produced the pattern to be classified.

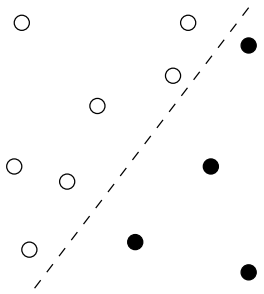


Figure 1: Line that separates points of two classes

1.2 Handwriting recognition

In handwriting recognition a text image is transcripted into a sequence of words. The words themselves are sequences of characters. There are two possible handwriting recognition scenarios called online and offline. Online means that the handwriting is detected while the user is writing. That means the HR system can use the information of the text image but as well the curve the user draws to form the characters. In the offline scenario the HR system can only use the information of the text image. Two possible examples are the detection of numbers of bank checks and the transcription of a whole handwritten book. The vocabulary that we expect in the first scenario are sequences of numbers, in the second example we might be confronted with any word of the particular language and also names and wrong written words. We call the first example a closed vocabulary problem and the second one an open vocabulary problem. Obviously the second one is much harder to solve.

1.3 Punctuation marks

Why is punctuation important for text recognition? First of all a different punctuation can change the meaning of a sentence. A popular example are “eats, shoots and leaves” and “eats shoots and leaves” [16] or “woman, without her man, is nothing,” and “woman: without her, man is nothing,”. If a HR system ignores the punctuation, in both examples it will not be able to add the correct symbols later, at least not with pure syntactic methods.
But there are more reasons why a strong punctuation mark detection is necessary for a good transcription of the text image. Punctuation marks allow us to separate the text into sections with a common context. A usual HR system would only cut the text image into text lines and try to find the sequence of words that maximises the cooccurence probability of neighbouring words. However if a sentence overlaps into the next text line the information of the second part is missing when the first part is detected and vice versa. Consider the phrase “Automakers see two paths to green cars ”. The absence of “automakers” makes it harder to detect the word “car” here.
We can conclude that it is worthwhile and necessary to detect punctuation marks correctly. In the next chapter the state of the art in recognition process is explained.

2 State of the art

In the long history of Handwritten Recognition (HR) many approaches have been proposed. This section introduces some of the most popular ones. HR can be divided into the tasks of isolated character recognition, word recognition and unconstrained text recognition, where unconstrained means that the number of words in the text is a priori unknown [5]. However as it was tried to improve punctuation detection only the latter is described in this article. As HR is considered as a Pattern Recognition problem this section is divided in the from section 1.1 known steps of preprocessing, feature extraction and classification.

2.1 Preprocessing

As said before preprocessing attempts to eliminate variations between the patterns that are not important for the classification step or even constrict the classification. One source of such variation is the exposure of the text image. Image processing techniques are used to reduce noise e.g. inkblots and remove the background e.g. yellowed paper.
A more problem specific source are differences in the handwriting of different authors or even in the handwriting of one author. The HR community developed many procedures to normalise the handwriting of a collection. A widely used approach is *slant* and *slop* correction. In [18] *slope* is defined as the angle between the horizontal direction and the direction of the implicit line on which the word is aligned and *slant* as the angle between the vertical direction and the direction of the strokes supposed to be vertical. Alessandro Vinciarelli claimed in [18] that most desloping and deslanting techniques are inspired by the method proposed in [4]. This slant correction algorithm works as follows. For a given word image all horizontal strokes longer than a certain parameter are removed. This separates the image into an amount of isolated strokes. The slant of the word is now calculated as the average of the slants of all the strokes. For a more detailed description refer to [22]. The correction can then be performed by mapping every foreground pixel (x, y) of the original image to the pixel (x', y) on the slantfree image. Where x' is defined as:

$$x' := x - y \times tan(slant). \tag{2}$$

2.2 Feature extraction

In [17] features based on the distribution of the pixels are proposed. The extraction works on a binarised text line and creates a sequence of feature vectors by moving a window from the left to the right. The window itself is subdivided into 4×4 cells. For every cell the number of foreground pixel f_i is calculated. The resulting feature vector has the from $F = (\frac{f_1}{N}, \frac{f_2}{N}, ..., \frac{f_{16}}{N})^T$ with N as the total number of foreground pixels in the window.
Another approach is proposed in [3, 15]. The text lines are divided into a number of square cells. This is also referred to as *vertical resolution* [15]. For every cell then the

Figure 2: Example of the feature vector for the word “suggested”

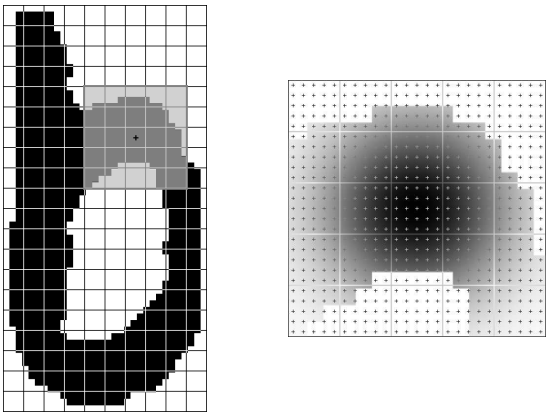


Figure 3: Example of calculating one component using a 5x5 Gaussian bell curve

normalised grey level, *horizontal grey level derivative* and *vertical grey level derivative* are calculated. An example feature vector can be seen in figure 2. In [3] overlapping cells and intensities - percentage of foreground pixels in the cell - are used instead of grey levels. As above, one feature vector is then built out of a column of grid cells. As can be seen in figure 3 the value for one cell is calculated by applying a Gaussian function to the surroundings of the cell. The Gaussian smoothing makes the procedure resistant to noise but, as will be explained later, improper for the detection of punctuation marks. All experiments of this work used the latter features.

2.3 Classification

A classifier finds for a sequence of features the best matching sequence of words. The existing procedures can be divided into segmentation-based and segmentation-free methods. Segmentation-based approaches segment the text into words and the words into characters, later using character recognition techniques to detect the characters. However there are hybrid-techniques which only split the text into words and use other methods to recognise the words.

2.4 Segmentation-based recognition

The intuitive approach tries to segment the sequence in groups, each group corresponding to one letter or word. Now character or word classifiers can be used to label the groups. However, to do the segmentation, the characters have to be detected and to detect the characters the segmentation is to be known. This is called Sayre's Paradox [13]. An approach to overcome this problem searches the space of all possible partitions. To use this approach the sequence of features must be an over-segmentation of the text. In an over-segmentation there is no feature that belongs to two characters. This over-segmentation is normally easier to achieve as the true segmentation into characters. Additionally a character recogniser which returns for a sequence of graphemes a sequence of confidences for every possible character is needed. With the recogniser and a search algorithm [7] the space of possible sequences of characters can be searched for the most feasible sequence. An advantage of this approach is that well investigated Optical Character Recognition (OCR) techniques can be used. On the other hand segmentation and recombination are based on heuristics which have to be adapted manually and can hardly be learnt from training data [6].

2.5 Handwritten text recognition as a statistical problem

As said before the classification process can be described as finding the most plausible text for a feature sequence. This search can be described in terms of statistics [19] as finding for a sequence of observations $O = (o_1, o_2, ..., o_m)$ the most probable explanation $\hat{W}$ of all explanations $W = (w_1, w_2, ..., w_n)$, where the length of the word sequence n is a variable which has to be determined in the recognition process and not a constant.

$$\hat{W} = \arg\max_{W} P(W|O) \tag{3}$$

As seen in section 1.1 this can be rewritten as:

$$\hat{W} = \arg\max_{W} P(O|W)\, P(W) \tag{4}$$

The term $P(W)$ as an a priori probability is independent of the sequence of observations and describes how probable the sequence of words is in the particular language. To estimate this probability statistical Language Models (LMs) are used. While for recent English texts the same modern-grammar-based LM could be used, for antique texts it might be necessary to learn a new LM from the corpus.
The term $P(O|W)$ represents the probability of the observation sequence O being generated by a model of the sentence W. It is also referred to as optical model [23]. This work concentrates on modelling $P(O|W)$ with hidden Markov models. However, other statistical methods could be used as well.

2.6 Language models

As said above a Language Model (LM) estimates the probability of a word sequence. LMs are also used in other Computer Science fields as document classification, information retrieval and machine translation. The probability of a sequence of n words can be calculated as:

$$P(W) = \prod_{i=1}^{n} P(w_i|w_1, ..., w_{i-1}) = \prod_{i=1}^{n} P(w_i|w_1^{i-1}) \tag{5}$$

Where w_1^{i-1} is referred to as history of w_i [19]. This form makes it easy to apply Viterbi decoding as the probability of w_i only depends on the words previously written. There were many different LMs introduced but N-grams are the most used models in recent works on HR. N-Grams use as history only of the $N-1$ last written words.

$$P(W) = \prod_{i=1}^{n} P(w_i|w_{i-N+1}^{i-1}) \tag{6}$$

However, in practise rarely more than unigrams, bigrams and trigrams are used, as the computational cost of calculating all possible histories is rising exponential. Estimating the $P(w_i|w_{i-N+1}^{i-1})$ from training is done by simply building the ratio of how many times the sequence w_{i-N+1}^{i-1} is followed by the word w_i and how many times the sequence can be seen in the training corpus at all.

$$\hat{P}(w_i|w_{i-N+1}^{i-1}) = \frac{\#(w_{i-N+1}^{i})}{\#(w_{i-N+1}^{i-1})} \tag{7}$$

This estimation of the probabilities fulfils the Maximum Likelihood criterion. Hence it chooses the parameters $P(w_i|w_{i-n+1}^{i-1})$ maximising the generation of the training set D.

$$\hat{P} = \arg\max_{P} \prod_{d \in D} \prod_{d_i=0}^{n} P(d_i|d_{i-N+1}^{i-1}) \tag{8}$$

Unfortunately, sequences that are not in the training set receive a probability of zero even if they could be found in the test set. This makes smoothing necessary. Smoothing of LMs can be defined [19] as *redistributing the probability mass estimated using ML across all possible sequences of N words*. One smoothing technique inspired by Zipf's law is called Good-Turing discounting. Zipf's law is an empirical law predicting that the frequency of words in natural languages is falling exponentially. It also can be summarised as in natural languages there are few very frequent words and many very rare ones.

$$f_i \propto \frac{1}{i} \tag{9}$$

With f_i as the frequency of ith-frequent word. Good-Turing discounting maps every estimated probability r to a new probability r' using the equation:

$$r' = \frac{n_{r+1}}{n_r}(r+1) \tag{10}$$

Where n_r describes the number of sequences with frequency r in the text. The amount $r - r'$ summed up over all estimated probability is than distributed over all N-grams with probability zero or a probability lower than a certain threshold. The distribution can be done uniform or using back-off, which practically means that when the probability of an N-gram is too low the probability of a lower order N-gram scaled by a factor is used. Please refer to [14, 19] for a detailed description of this approach.

2.7 Hidden Markov models as optical models

After hidden Markov models (HMM) were applied successfully in speech recognition they also became a very common technique in HR. HMMs developed in the 1960s and early 1970s are methods to model stochastic processes and are an enhancement of Markov chains. The foundations of HMMs and Markov chains and their application to speech recognition are described in detail in [12]. Markov chains model systems that change their state at regular steps. Like a traffic light that changes its colour at certain time intervals but completely by chance. So the traffic light might be switching from green directly to red without being yellow. Or it might as well stay green in one step. However the number of possible states N must be finite. The probability of being in state S_t only depends on the preceding k states.

$$P(s_t) = P(s_t|s_{t-1}s_{t-2}...s_{t-k}) \tag{11}$$

However, most applications use first order models with $k = 1$. Also they assume that the transition probability itself does not depend on i. Such a probability is called stationary. The system can then be modelled by one matrix $A = \{a_{ij}\} = \{P(s_t = j|q_{t-1} = i\})$ and a initial state probability distribution π defined as $\pi_i = P(q_1 = i)$. In the case of HMMs the state of the system can only be observed indirectly through an effect or evidence. So parallel to the sequence of states q_i there exist a sequence of observations O_i of a certain finite domain. The emisson of an observation o while being in state i is defined as $B = \{b_{io}\} = \{P(O = o|q = i)\}$. The whole HMM λ is then defined as the tuple (A, B, π). To apply HMMs to the HR problem for every possible word a HMM is created. Now using training data the parameters of the character models are estimated. The recognition of a word given as a sequence of features is then achieved by choosing the word model that generates the highest confidence. For a real world application usually tree problems have to be solved. Firstly, calculating the probability that the HMM produces a sequence of observations $P(O = o_1o_2...o_T|\lambda)$. Secondly, finding the sequence of states $q_1q_2, ...q_3$ that results in the highest probability $\hat{S} = \arg\max_S(P(O = o_1o_2...o_T, S = q1q2...q_T|\lambda))$. And thirdly, given a training set of observations estimating the model parameters $\lambda = (A, B, \pi)$. The three of them will be explained in the following sections.

2.7.1 The forward algorithm

The forward algorithm calculates the probability of producing an observation sequence O by keeping track of all paths of state that could produce O. For that purpose a variable $\alpha_t(i)$ is introduced, which stores the probability of all paths ending in the state i to produce the partial sequence O_1^t. The key point is that $\alpha_{t+1}(j)$ is the sum of the probability of all paths ending in i and producing the sequence O_1^t ($\alpha_t(i)$) times the probability of going from state i to state j (a_{ij}) times the probability of producing the missing observation o_{t+1} in the state j ($b_{jo_{t+1}}$).

$$\alpha_1(i) = \pi_i b_{io_1} \tag{12}$$

$$\alpha_{t+1}(j) = [\sum_i \alpha_t(i) a_{ij}] b_{jo_{t+1}} \tag{13}$$

The forward variable α exploits the fact that all paths leading to a preceding state i can be grouped to one class and handled equally, as the transmission probabilities B only depend on i and the succeeding state j. In this manner all N^T possible paths can be summarised to NT calculations. In the appendix 6.1.1 the algorithm can be found in pseudo code.

2.7.2 The Viterbi algorithm

The Viterbi algorithm -originally described by Andrew Viterbi in [20]- calculates for an observation sequence $O = o_1..o_T$ the most probable sequence of states $q_1q_2...q_T$ or so to say the most probable explanation. To do that, the quantity $\delta_t(i)$ is defined as the probability of the most probable path of t states ending in state i.

$$\delta_t(i) = \max_{q_1,q_2,...,q_{t-1}} P(q_1q_2...q_{t-1} = i, o_1o_2...o_t|\lambda) \tag{14}$$

The Viterbi algorithm is an exponent of dynamic programming, where the final solution is calculated of partial solutions. With a recursive rule for δ, the most probable sequence is been calculated in two phases. Firstly, δ_T is calculated and the maximising state is stored in an array ψ. Secondly, using backtracking the most probable path is extracted. In section 6.1.2 the Viterbi algorithm can be found as described in [12].

2.7.3 The Baum-Welch algorithm

Analog to eq. (8) in the training phase the parameters λ are chosen to maximise the generation of a training set D.

$$\hat{\lambda} = \arg\max_{\lambda} \prod_{d \in D} p(O(d)|\lambda) \tag{15}$$

The Baum-Welch algorithm designed by Leonard Baum and Lloyd Welch [1] incrementally enhances an initial parametrisation to find a **local** maximum of this likelihood.

For this purpose Baum introduced [2] a so called *growth transformation*. Let P be a homogeneous polynomial in the variable $\mathbf{x}$, with non-negative coefficients. Than with

$$\mathbf{x_{ij}} : \sum_j x_{ij} = 1; x_{ij} > 0 \tag{16}$$

The mapping δ is a *growth transformation*, iff

$$\begin{aligned} & \delta(\mathbf{x}) \text{ accomplishes condition (16)} && (17)\\ \wedge \quad & p(\mathbf{x}) \leq p(\delta(\mathbf{x})) && (18)\\ \wedge \quad & p(\mathbf{x}) = p(\delta(\mathbf{x})) \rightarrow \mathbf{x} \text{ is a critical point of P} && (19) \end{aligned}$$

A valid *growth transformation* is given in (20). However, the explicit form for the parameters π, a_i and b_i depends on the algorithm that is used to calculated the probability $p(O|\lambda)$. A typical choice are forward-backward algorithm or Viterbi algorithm (as an approximation).

$$\delta(\mathbf{x})_{ij} = \frac{x_{ij}(\partial p/\partial x_{ij})_x}{\sum_k (\partial p/\partial x_{ik})_x} \tag{20}$$

2.8 Integrating optical and language model

In 2.7.2 a version of the Viterbi algorithm working on HMMs was introduced. Most modern HR systems integrate optical and language model as expressed in eq. (4). Therefor the recursive step (21) is generalised to eq. (22).

$$\delta_t(i) = \max_{1 \leq j \leq N} \delta_{t-1}(j) a_{ji}\, b_{i,o_t} \tag{21}$$

$$\delta_t(w_t) = \max_{w_{t-1} \in V} \delta_{t-1}(w_{t-1})\, p(w_t|w_{t-1})\, p(o_t|w_t) \tag{22}$$

As we are now searching the most probable word sequence, the states q were replaced by words w. Hence the possible state values are all words in the vocabulary V. Furthermore, the probability to change from one state to an other is now estimated with the LM and the probability to generate an observation (or feature vector) is calculated with the OM. The mindful reader might see that the eq. (22) already is specialised to bigram models. As the true probabilities of LM and OM are only estimated by e.g. n-Grams and HMMs, two parameters are introduced to compensate the resulting error. For computational reasons also the monotone function *log* is applied to the equation.

$$\delta_t(w_t) = \max_{w_{t-1} \in V} \delta_{t-1}(w_{t-1}) + \alpha\, log\, p(w_t|w_{t-1}) + log\, p(o_t|w_t) + \beta \tag{23}$$

In the literature α is called *Grammar Scale Factor* [21] as it determines the importance of the LM. Along the lines β is called *Word Insertion Penalty* as it helps to control the length of the decoded sequences. It is easy to see that a parametrisation of $(0, 0)$ leads to

eq. (21). The optimal value for GSF and WPI are usually estimated using a validation set.

3 Experiments

In this work it was attempted to discover the influences of certain parameters and procedures on the performance of the punctuation sign detection. Several experiments were performed to investigate the ideas developed in 3.4. All experiments were lunched with the recognition tool iAtros and the IAMDB corpus and evaluated with the word error rate (WER), which are explained in the following sections.

3.1 iAtros

The *improved Automatically Trainable Recognizer of Speech* (iAtros) [9] is a set of tools for preprocessing, feature extraction and recognition of speech and text. The detection module searches with a Viterbi algorithm in the integrated network of morphological, lexical and language model. The morphological model contains a hidden Markov model for every character that can be used by the lexical model to build words. The lexical model is a simple dictionary that explains how the known words are spelled. A typical entry would be: *hello - h e l l o*. The integrated network of morphological and lexical model was introduced as optical model in section 2.5. As language model N-grams or finite state machines can be used. Besides the parametrisation of the model, which is usually learned from training data, *iAtros* introduces three important parameters, usually estimated with a validation set. These parameters are the already known grammar scale factor (GSF) and word insertion penalty (WIP), the histogram pruning value (HP) and the number of states of the HMMs. The HP value determines the number of paths, followed in the Viterbi decoding. Contrary to GSF and WIP higher HP values always lead to better results. However, the used memory and needed time rise seriously. For all the performed experiments a HP value of 10000 was used. The iAtros state transition matrices for the state i only allow transitions to the states i and $i+1$. For this reason the state count limits the minimal number of feature vectors to be read to detect a letter.

3.2 Corpus

The experiments were performed with the Lancaster-Oslo/Bergen corpus (LOB) [8] of the University of Oslo and the IAMDB corpus of the institute of Computer Science and Applied Mathematics (IAM) of the University of Bern [11]. The databases contained full English sentences. The IAMDB corpus is a partial transcription of the LOB corpus. The filled forms (figure 6) have been segmented into text lines shown in figure 5. Every text line is available as an image in the *tif* format and as transcribed ASCII text.
The LOB corpus consists of about one million words written by about 400 writers. The whole corpus was used for the training of the language model and the lexical model. The used dictionary contained about $20,000$ words.The IAMDB corpus was divided into a training and a test partition (figure 4). The first was used for the training of the morphological model. Primarily it was planed to use the latter for validation and training, but it turned out, that the experiments took more time than expected. Thus, on an Intel(R) Core(TM)2 Quad with 2.4 GHZ and 4 GB of RAM, decoding one line

of text needed about 4 minutes. Finally the values for grammar scale factor and word insertion penalty were calculated using a small validation data set of 50 lines. All tests were executed on a test set of 500 text lines. Of the 4621 words in the test set about 16.5% were punctuation marks or words that contained punctuation (like for example you've).

	training	test	total
writers	448	100	548
sentences	2,124	200	2,324
words	42,832	3,957	46,789
characters	216,774	20,726	237,500

Figure 4: Resume of the IAMDB corpus

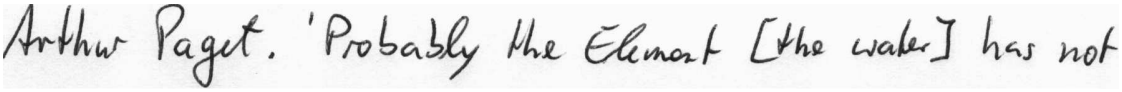


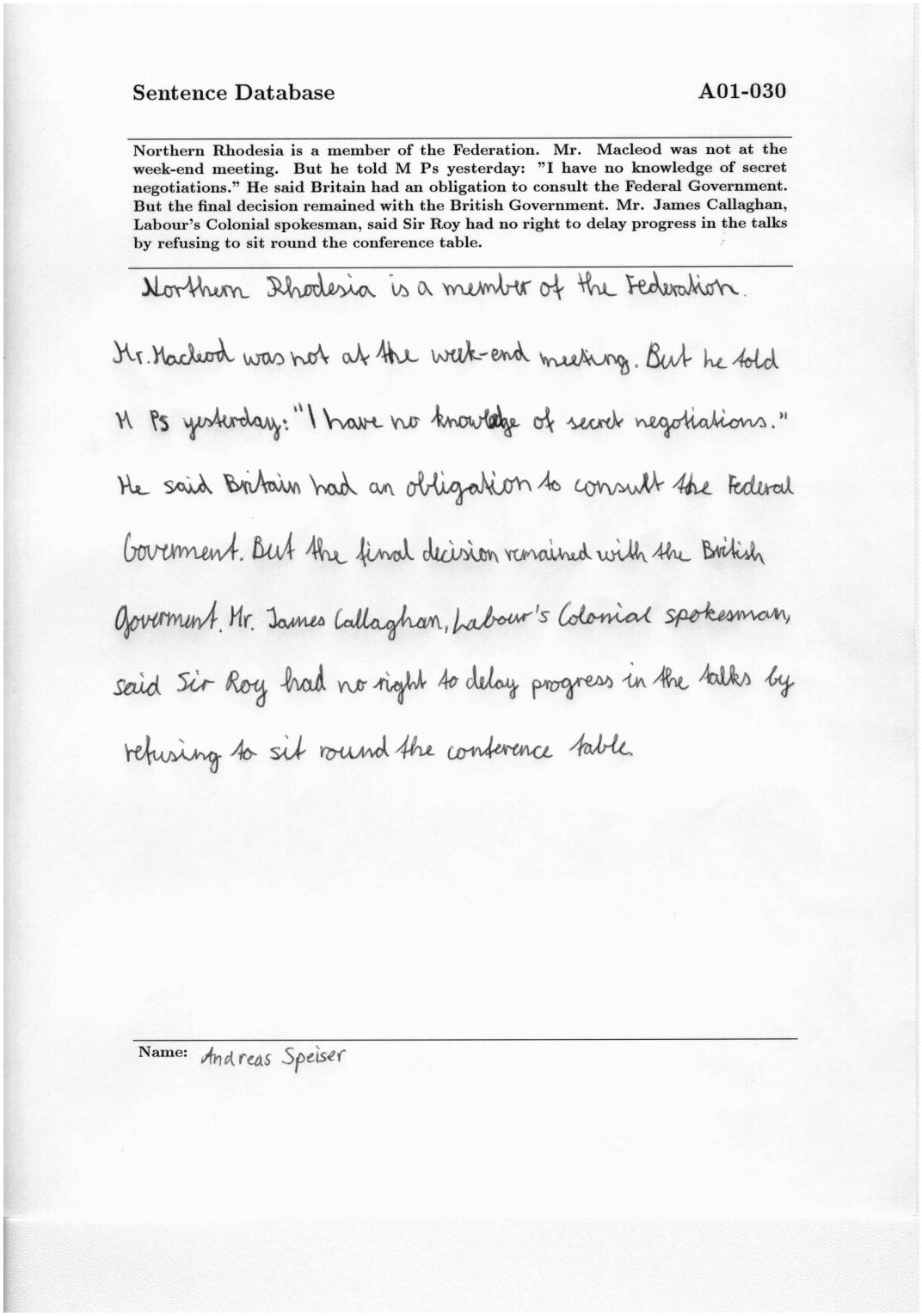
Figure 5: Example of a IAMDB text line

3.3 Evaluation

For calculating the classification error it is necessary to compare the text lines of the corpus and the lines recognised by the classifier. Hence a method to measure the difference/similarity of two phrases is needed. The working horse of the HR community is the *word error rate* (WER). The WER is a normalised *minimal edit distance* (MED) at word level. The idea behind the MED is to define basic operation to transform one sentence x into an other sentence y. This basic operations are *insert*, *delete*, *replace* and *copy*. In the space of all possible paths of operations that transform x into y the path with the minimal cost is searched. Its cost is the MED of x and y. The cost of a path of operations is defined as the sum of the costs of all basic operations. This costs are usually defined as $cost(insert) = cost(delete) = cost(replace) = 1$ and $cost(copy) = 0$. However, as an *insert* operation can be achieved by deleting the character at hand and inserting the new character sometimes also $cost(replace) = 2$ is used. To calculate the WER the MED is divided by the length of the sentence x.

$$\text{WER} = \frac{i + d + r}{n} \tag{24}$$

The MED is calculated by the use of dynamic programming and the fact that the MED to transform the prefix of length i of x into the prefix of length j of y can be expressed as [10]:



Sentence Database **A01-030**

Northern Rhodesia is a member of the Federation. Mr. Macleod was not at the week-end meeting. But he told M Ps yesterday: "I have no knowledge of secret negotiations." He said Britain had an obligation to consult the Federal Government. But the final decision remained with the British Government. Mr. James Callaghan, Labour's Colonial spokesman, said Sir Roy had no right to delay progress in the talks by refusing to sit round the conference table.

Northern Rhodesia is a member of the Federation.

Mr. Macleod was not at the week-end meeting. But he told M Ps yesterday: "I have no knowledge of secret negotiations." He said Britain had an obligation to consult the Federal Goverment. But the final decision remained with the British Goverment. Mr. James Callaghan, Labour's Colonial spokesman, said Sir Roy had no right to delay progress in the talks by refusing to sit round the conference table.

Name: Andreas Speiser

Figure 6: Example of a IAMDB form page

$$\mathrm{MED}(x_1^i, y_1^j) = \min\{ \quad \mathrm{MED}(x_1^{i-1}, y_1^j) + cost(delete),$$
$$\mathrm{MED}(x_1^i, y_1^{j-1}) + cost(insert),$$
$$\mathrm{MED}(x_1^{i-1}, y_1^{j-1}) + \begin{cases} cost(copy) & \text{if } x[i] = y[j] \\ cost(replace) & \text{else} \end{cases} \}$$

Additionally, a confusion matrix can be calculated. The confusion matrix M stores how often the word v was replaced by the word w. With ϵ as the empty string, entries of the form $M(v, \epsilon)$ ($M(\epsilon, w)$) correspond to *delete* (*insert*) operations. The diagonal entries $M(w, w)$ are the copy operations. All other entries are true replace operations. With the confusion matrix a more specific WER can be calculated. In this work -only by counting the entries where v or w contain punctuation marks- the punctuation error of the performed experiments was calculated. An example of a confusion matrix can be found in figure 3.3. The matrix should have a diagonal dominance, as diagonal entries relate to correct classified words. Huge values in the first row or the first column might be handled with a variation of parameters like the *Word Insertion Penalty* (2.8).

#	Word	0	1	2	3	4	5	6	7	8	...
0	ϵ	0	89	0	460	0	0	4	0	0	...
1	“	24	30	0	14	0	0	0	0	0	...
2	Impressive	0	0	0	0	0	0	0	0	0	...
3	,	97	4	0	1009	0	0	0	0	0	...
4	Lord	0	0	0	0	0	0	0	0	0	...
5	Undertone	0	0	0	0	0	0	0	0	0	...
6	said	5	0	0	2	0	0	52	0	0	...
7	gazing	0	0	0	0	0	0	0	1	0	...
8	reverently	0	0	0	1	0	0	0	0	0	...
...	...	...	...	...	...	...	...	...	...	...	...

Figure 7: Extract of a confusion matrix. As Zipf's Law (2.6) predicts most words only appear rarly.

In the analysis of the experiments three kinds of errors where calculated. All three are slight modifications of the WER as they do not use n as normalisation constant. The total error index or word error index estimates the performance of the whole recognition system. The relative punctuation error index measures how the system performed on words containing punctuation marks (e.g.: .,!,- and you've). The absolute punctuation error index measures the fraction of the total error that is caused by punctuation errors. If c,d,i and r are the edit operations of an experiment and c_p,d_p,i_p and r_p are the operations on words that contain punctuation, the three errors can be calculated as:

$$\mathrm{EI}_{total} = \frac{d+i+r}{c+d+r} \tag{25}$$

$$\mathrm{EI}_{p,absolute} = \frac{d_p+i_p+r_p}{c+d+r} \tag{26}$$

$$\mathrm{EI}_{p,relative} = \frac{d_p+i_p+r_p}{c_p+d_p+r_p} \tag{27}$$

$$\tag{28}$$

3.4 Tests

For the comparison of the tests a reference value was calculated. Every character was represented by a HMM with a state count of 6. Bigrams were used as language models. As explained in the preceding sections the test corpus consisted of 500 text lines. The parameters of the Viterbi algorithm were estimated to a GSF of 45 and a WIP of 30.
To investigate the performance of the morphological model the state count of all punctuation marks was set to 2. As punctuation marks are smaller than letters it was expected to achieve a lower error than with the initial test.
In a second experiment the influence of the language model was explored. The transcription of the used corpuses are not forceful with the notation of a word preceded or followed by a punctuation mark. As an example the word sequence *, when* can be found transcripted as one word: *,when* and as two words: *,* and *when*. It was tried to reduce the complexity of the lexical and the language model, by eliminating the written as one word form. This should simplify the Viterbi decoding and lead to a better transcription. The script written to manipulate the iAtros model files can be found in the appendix.

4 Analysis and conclusions

4.1 Validation

As explained before, the main goal of the validation is to estimate good values for GSF and WIP. The comparison of the error functions of both values (fig. 8,9) shows that the influence of the GSF on the error is bigger than the influence of the WIP. Also it can be seen that the total error index curve always is above the relative punctuation error curve. That is not a surprise, it was expected that the recognition system would perform worse on punctuation marks than on letters. The curves of relative and absolute punctuation error progress similar as both quantities are basically the same quantity differently normalised. Tough, it is interesting to see that the total error has a slightly different progression. Hence, the best parametrisation in terms of the absolute error is not forcefully the best in terms of the punctuation error.

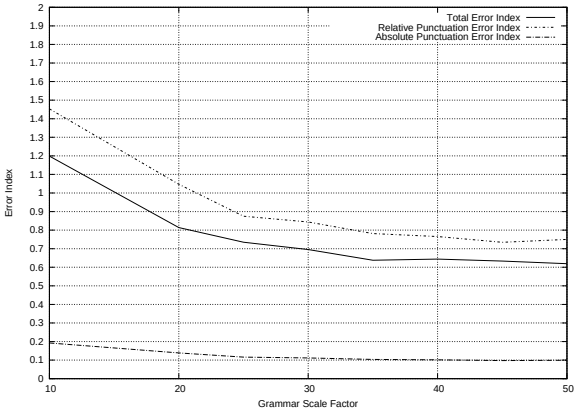


Figure 8: Error Index as a function of the GSF at a WIP of 0

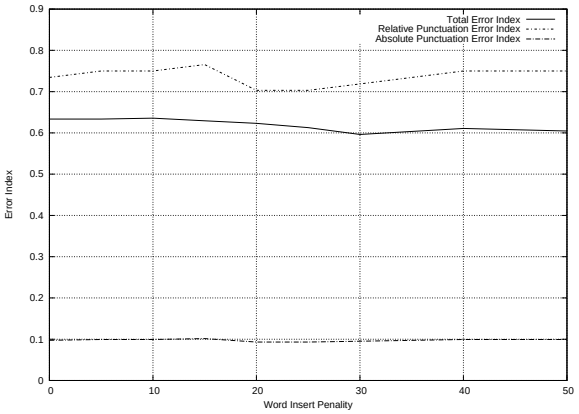


Figure 9: Error Index as a function of the WIP at a GSF of 45

4.2 Tests

Up front the error of all three tests is pretty high. Although the test set contained much punctuation and was therefor difficult to classify, a better result was expected. Especially, the high number of insert operations manifest that the found parametrisation could be optimised. Possibly, validation set and test set are not distributed equally. With respect to the tests, it is concluded that the generally high error covers the differences of the individual errors.
Table 10 shows that the difference of the initial test and the test with the changed language model is only marginal. The effect of the LM reduction seems to be much smaller than expected. Even if the error of the reduced LM is slightly worse than the original error, in other experiments (at a GSF of 35 and a WPI of 5) it was fractionally better.
The changes performed to the optical model were quite more grave than the changes to the LM. For this reason it was expected that the error indices would change noticeably. However, the effect to the result is minor again. It is assumed that the small effect is caused by a poor performance of the morphological model. The performance of the morphological model depends on its parametrisation (like the number of states and the number of Gaussian per state) and primarily on the features. The features used, as already mentioned in section 2.2, use a Gaussian bell function to reduce noise. It is concluded that the smoothing performed in the feature extraction was to intense.

Test number	total EI	absolute punct. EI	relative punct. EI
initial	0.609	0.709	0.0956
language model	0.613	0.711	0.0958
morphological model	0.618	0.760	0.1025

Figure 10: Experiment results

5 Future work

Future work should try to improve the feature extraction step. The used features were designed to be resistant to noise and small variations. However, it seems that the Gaussian smoothing deletes too much of the information of the punctuation marks. It is proposed to test the effect of the variance on the different punctuation signs to find a parametrisation that conserves more information by not being prone to noise. Alternatively, other features that are known to be more sensitive to small patterns could be tested. Such are the distribution-based method, explained in 2.2 or the method proposed in [19].
Furthermore, it could be tried to detect punctuation marks in a preprocess. Punctuation marks like dots and commas or quotes are known to appear only in certain parts of a text line. A special classifier, like a neural network or a radial basis function, could be trained to search for the punctuation in the respective area. This might improve the detection of punctuation and allow to separate the text lines into segments of a common context before the actual detection is done.

6 Appendix

6.1 Algorithms

6.1.1 The forward algorithm

Initialisation:
for $i = 1$ to N **do**
  $\alpha_1(i) \leftarrow \pi_i b_{io_1}$
end for

Recursion:
for $j = 1$ to N **do**
  for $t = 1$ to $T - 1$ **do**
    $\alpha_{t+1}(j) = [\sum_{i=1}^{N} \alpha_t(i) a_{ij}] b_{jo_{t+1}}$
  end for
end for

Termination:
$p(O|\lambda) = \sum_{i=1}^{N} \alpha_T(i)$

6.1.2 The Viterbi algorithm for hidden Markov models

Initialisation:
for $i = 1$ to N **do**
  $\delta_1(i) \leftarrow \pi_i b_{i,O_1}$
  $\psi_1(i) \leftarrow 0$
end for

Recursion:
for $t = 2$ to T **do**
  for $i = 1$ to N **do**
    $\delta_t(i) \leftarrow \max_{1 \le j \le N} (\delta_{t-1}(j) a_{ji})\, b_{i,o_t}$
    $\psi_t(i) \leftarrow \arg\max_{1 \le j \le N} (\delta_{t-1}(j) a_{ji})\, b_{i,o_t}$
  end for
end for

Backtracking:
$p(\hat{S}) \leftarrow \max_{1 \le i \le N} \delta_T(i)$
$\hat{q}_T \leftarrow \arg\max_{1 \le i \le N} \delta_T(i)$
for $t = T - 1$ to 1 **do**
  $\hat{q}_t \leftarrow \psi_{t+1}(\hat{q}_{t+1})$
end for

6.2 Experiment results

6.2.1 Error indices as a function of the GSF at a WIP of 0

GSF	total EI	absolute punct. EI	relative punct. EI
10	1.199	1.45	0.193
20	0.815	1.05	0.139
25	0.735	0.88	0.116
30	0.696	0.84	0.112
35	0.638	0.78	0.103
40	0.644	0.77	0.101
45	0.634	0.73	0.097
50	0.619	0.75	0.099

6.2.2 Error indices as a function of the WIP at a GSF of 45

WIP	total EI	absolute punct. EI	relative punct. EI
0	0.6335	0.734	0.0973
5	0.6335	0.750	0.0994
10	0.6356	0.750	0.0994
15	0.6294	0.766	0.1014
20	0.6232	0.703	0.0932
25	0.6128	0.703	0.0932
30	0.5963	0.719	0.0952
40	0.6108	0.750	0.0994
50	0.6046	0.750	0.0994

6.3 Scripts to manipulate the iAtros model files

6.3.1 calc_mat.py

```
#!/usr/bin/env python
#
# file name: calcerror_mat.py
# author: Thomas Müller
#
# the WER error calculation tool tasas can be configured to drop the
# confusion matrix. This script will calculated the total error index,
# relative punctuation error index  and absolute punctuation error index,
# out of this file.
#
# Use "> tasas -C mymatrix.mat [other options]" to generate the matrix
# Use "> calcerror_mat.py mymatrix.mat " to calculate the mentioned error
# values.

import sys
```

```
# names of the punctuations marks in "mat"
PUNCTUATION_MARKS = (['\'',',','.','"','-','\\','?',';','(',')',':','!'])

def contains(sign,symbol_list):

  for symbol in symbol_list:

    if sign.find(symbol) != -1:
      return True

  return False

# read the numbers as floats in a 2-dimensional array
# and the symbols in a 1-dimensional one
def read_matrix(mat):

  f = open(sys.argv[1],'r')
  matrix = []

  # the first line of the matrix file is useless
  f.readline()

  # the first matrix row/column relates to the empty word epsilon
  # unfortunately it is represented as a space character
  # therefore the first line is handled apart.

  first_line = f.readline()
  clean_list = []
  character_list = first_line.split(' ')
  for symbol in character_list:
    if symbol != '':
      clean_list.append(symbol.strip())
  number_list = map(int,clean_list[1:])
  matrix.append(number_list)
  symbols = ['EPSILON']

  for line in f.readlines():

    character_list = line.split(' ')
    clean_list = []

    for symbol in character_list:
```

```
      if symbol != '':

        clean_list.append(symbol.strip())

    number_list = map(int,clean_list[2:])
    symbols.append(clean_list[1])
    matrix.append(number_list)

  f.close()

  return matrix,symbols

matrix,symbols = read_matrix(sys.argv[1])

# The word operations
copy = 0
replace = 0
delete = 0
insert = 0

# The operation for words, that contain punctuation
p_copy = 0
p_replace = 0
p_delete = 0
p_insert = 0

for i in range(len(matrix)):

  for j in range(len(matrix)):

    value = matrix[i][j]

    if value != 0.:

      if i == 0:
        insert += value
        if contains(symbols[j],PUNCTUATION_MARKS):
          p_insert += value
      elif j == 0:
        delete += value
        if contains(symbols[i],PUNCTUATION_MARKS):
          p_delete += value
      elif i == j:
        copy += value
```

```
        if contains(symbols[i],PUNCTUATION_MARKS):
          p_copy += value
      else:
        replace += value
        if contains(symbols[i],PUNCTUATION_MARKS):
          p_replace += value

ei = (insert+replace+delete)/float(copy+delete+replace)
ei_p_relative = (p_insert+p_replace+p_delete)/float(p_copy+p_delete+p_replace)
ei_p_absolute = (p_insert+p_replace+p_delete)/float(copy+delete+replace)

print 'total error index:',ei
print 'relative punctuation error index:',ei_p_relative
print 'absolute punctuation error index:',ei_p_absolute
```

6.3.2 reduce_arpa.py

```
#! /usr/bin/env python
#
# file name: reduce_arpa.py
# author: Thomas Müller
#
# This script turns a iAtros language model file into a language model
# file, without words that start or end with punctuation marks.
# For examples words like 'yesterday!' or '.What' will be deleted, however
# words which actually are single characters wont be deleted.
#
# Syntax: reduce_apra iamdb.arpa.old iamdb.arpa.new

import re,sys,math

def contain_punct(x):

  if len(x) == 1:
    return False

  if x == '\'\'':
    return False

  for a in [x[0],x[-1]]:
    if re.match('\w',a) == None:
      return True

  return False
```

```
def read_empty_line(f):

  empty_line = f.readline()
  assert (empty_line == '\n')

def read_prefix(f):

  read_empty_line(f)

  data = f.readline()
  assert (data == '\\data\\\n')

  n_gram_1 = f.readline()
  assert (n_gram_1.startswith('ngram 1='))
  n_gram_1 = int(n_gram_1[len('ngram 1='):])

  n_gram_2 = f.readline()
  assert (n_gram_2.startswith('ngram 2='))
  n_gram_2 = int(n_gram_2[len('ngram 2='):])

  return n_gram_1,n_gram_2

def read_1_gram(f,count):

  read_empty_line(f)

  n_gram_1_content = []

  n_gram_1 = f.readline()
  assert (n_gram_1 == ('\\1-grams:\n'))

  for i in range(count):

    line = f.readline()

    if line == '':

      sys.stderr.write('ERROR: not enough lines\n')
      sys.exit(1)

    values = map(lambda x:x.strip(),line.split('\t'))
```

```
    if len(values) == 3:

      prob,word,foo = values

    else:

      prob,word = values
      foo = '0'

      sys.stderr.write('WARNING: no foo in line: %s\n'%line.strip())

    if not contain_punct(word):
      n_gram_1_content.append(line)
    else:
      count -= 1

  return n_gram_1_content,count

def read_2_gram(f,count):

  read_empty_line(f)

  n_gram_2_content = []

  n_gram_2 = f.readline()
  assert (n_gram_2 == ('\\2-grams:\n'))

  for i in range(count):

    line = f.readline()

    if line == '':

      sys.stderr.write('ERROR: not enough lines\n')
      sys.exit(1)

    prob,word1,word2 = map(lambda x:x.strip(),line.replace('\t',' ').split(' '))

    if not (contain_punct(word1) or contain_punct(word2)):
      n_gram_2_content.append(line)
    else:
      count -= 1
```

```
  return n_gram_2_content,count

def write_all(f,n_gram_1_content,n_gram_1,n_gram_2_content,n_gram_2):

  # prefix:
  f.write('\n')
  f.write('\\data\\\n')
  f.write('ngram 1=%s\n'%str(n_gram_1))
  f.write('ngram 2=%s\n'%str(n_gram_2))

  # 1-gram:
  f.write('\n')
  f.write('\\1-grams:\n')
  for line in n_gram_1_content:
    f.write(line)

  # 2-gram:
  f.write('\n')
  f.write('\\2-grams:\n')
  for line in n_gram_2_content:
    f.write(line)

  # end:
  f.write('\n')
  f.write('\\end\\\n')

f = open(sys.argv[1],'r')

n_gram_1,n_gram_2 = read_prefix(f)

n_gram_1_content,n_gram_1 = read_1_gram(f,n_gram_1)
n_gram_2_content,n_gram_2 = read_2_gram(f,n_gram_2)

f.close()

f = open(sys.argv[2],'w')
write_all(f,n_gram_1_content,n_gram_1,n_gram_2_content,n_gram_2)
f.close()
```

6.3.3 delete_states.py

```
#! /usr/bin/env python
#
# file name: delete_states.py
```

```
# author: Thomas Müller
#
# This script turns a iAtros hidden Markov model file into a new HMM file.
# It reduces the number of states of the HMMs of punctuations Marks to 2.
# WARNING: The new HMM file is printed to the standard out put.
#
# Syntax: delete_states Macro_hmm > Macro_hmm.new

import re,sys

f = open(sys.argv[1],'r')
content = f.readlines()
f.close()

punct = ['.','CD',',',':','\'','!',';',"(",")","-"]

exp = re.compile('~h "(.|..)"')

editmode = False

i=0

while i<len(content):

  line = content[i]

  if line.startswith('~h'):
    match = re.match(exp,line)

    if match.groups()[0] in punct:
      editmode = True
    else:
      editmode = False

    print line.strip()

  elif line.startswith('<NUMSTATES>'):
    if editmode:print '<NUMSTATES> 4'
    else: print line.strip()

  elif line.startswith('<STATE>'):

    state = int(line.strip().split(' ')[1])
```

```
  if editmode:

    if state == 4 or state == 5:
      print '<STATE> %i'%(state-2)
    else:

      #sys.stderr.write(line)

      i += 1
      line = content[i]
      #sys.stderr.write(line)
      assert(line.startswith('<NUMMIXES>'))
      nummixes = int(line.strip().split(' ')[1])

      for mixture in range(nummixes):

        if content[i+1].startswith('<MIXTURE>'):

          assert(content[i+2].startswith('<MEAN>'))
          assert(content[i+4].startswith('<VARIANCE>'))
          assert(content[i+6].startswith('<GCONST>'))
          i += 6

          #sys.stderr.write(''.join(content[i+2:i+7]))

        elif content[i+1].startswith('<STATE>'):

          sys.stderr.write('Missing MIXTURE\n')

        else:
          assert False
  else:
    print line.strip()

elif line.startswith('<TRANSP>'):

  if editmode:

    print '<TRANSP> 4'

    trans_matrix = map(lambda x:x.strip().split(' '),content[i+1:i+9])

    print ('0.000000e+00 1.000000e+00 0.000000e+00 0.000000e+00')
    print ''.join(map(lambda x:x+' ',trans_matrix[3][2:6]))
```

```
    print ''.join(map(lambda x:x+' ',trans_matrix[4][2:6]))
    print ('0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00')

    i += 8

  else:

    print line.strip()

else:

  print line.strip()

i += 1
```

References

[1] Leonard E. Baum, Ted Petrie, George Soules, and Norman Weiss. A maximization technique occurring in the statistical analysis of probabilistic functions of markov chains. *The Annals of Mathematical Statistics*, 41(1):164–171, 1970.

[2] Leornand E. Baum and Georg R. Sell. Growth transformations for functions on manifolds. *Pacific Journal of Mathematic*, 27:211–227, 1968.

[3] Issam Bazzi, Richard Schwartz, and John Makhoul. An omnifont open-vocabulary ocr system for english and arabic. *IEEE Trans. Pattern Anal. Mach. Intell.*, 21(6):495–504, 1999.

[4] R.M. Bozinovic and S.N. Srihari. Off-line cursive script word recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(1):68–83, 1989.

[5] Horst Bunke. Recognition of cursive roman handwriting - past, present and future. In *ICDAR '03: Proceedings of the Seventh International Conference on Document Analysis and Recognition*, page 448, Washington, DC, USA, 2003. IEEE Computer Society.

[6] Richard G. Casey and Eric Lecolinet. A survey of methods and strategies in character segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 18(7):690–706, 1996.

[7] I. Guyon, L. Schomaker, R. Plamondon, M. Liberman, and S. Janet. Unipen project of on-line data exchange and recognizer benchmarks. In *Pattern Recognition, 1994. Vol. 2 - Conference B: Computer Vision & Image Processing., Proceedings of the 12th IAPR International. Conference on*, volume 2, pages 29–33 vol.2, 1994.

[8] Stig Johansson. Manual of information to to accompany the lancaster-oslo/bergen corpus of britisch enlish, for use with digital computers. 1978.

[9] Míriam Luján-Mares, Vicent Tamarit, Vicent Alabau, Carlos-D. Martínez-Hinarejos, Moisés Pastor, Alberto Sanchis, and Alejandro Toselli. iatros: A speech and handwritting recognition system. In *V Jornadas en Tecnologías del Habla (VJTH'2008)*, pages 75–78, Bilbao (SPAIN), Nov 2008.

[10] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schtze. *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA, 2008.

[11] U.-V. Marti and H. Bunke. A full english sentence database for off-line handwriting recognition. In *ICDAR '99: Proceedings of the Fifth International Conference on Document Analysis and Recognition*, page 705, Washington, DC, USA, 1999. IEEE Computer Society.

[12] Lawrence Rabiner. A tutorial on hmm and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, February 1989.

[13] Kenneth M. Sayre. Machine recognition of handwritten words: A project report. *Pattern Recognition*, 5(3):213 – 228, 1973.

[14] Katz SM, J. L. Gauvain, L. F. Lamel, G. Adda, and J. Mariani. Estimation of probabilities from sparse data for the language model component of a speech recognizer. *Int. J. Pattern Recognition and A.I*, 8, 1987.

[15] A. H. Toselli, A. Juan, J. González, I. Salvador, E. Vidal, F. Casacuberta, D. Keysers, and H. Ney. Integrated handwriting recognition and interpretation using finite-state models. *Int. J. Patt. Recognition and Artifial Intelligence*, 2004.

[16] Lynne Truss. *Eats, Shoots and Leaves.* Profile Books, 2003.

[17] A. Vinciarelli and J. Luettin. Off-line cursive script recognition based on continuous density hmm, 2000.

[18] Alessandro Vinciarelli. A survey on off-line cursive script recognition, 2001.

[19] Alessandro Vinciarelli, Samy Bengio, and Horst Bunke. Offline recognition of unconstrained handwritten texts using hmms and statistical language models. *IEEE Trans. Pattern Anal. Mach. Intell.*, 26(6):709–720, 2004.

[20] A. J. Viterbi. Error bounds for convolutional codes and an asymptotically optimal decoding algorithm. *IEEE Transactions on Information Theory*, 13, 1967.

[21] Markus Wuthrich, Marcus Liwicki, Andreas Fischer, Emanuel Indermuhle, Horst Bunke, Gabriel Viehhauser, and Michael Stolz. Language model integration for the recognition of handwritten medieval documents. In *ICDAR '09: Proceedings of the 2009 10th International Conference on Document Analysis and Recognition*, pages 211–215, Washington, DC, USA, 2009. IEEE Computer Society.

[22] G. Zachopoulos. Slant estimation and correction for off-line cursive script., 1984.

[23] Matthias Zimmermann and Horst Bunke. Optimizing the integration of a statistical language model in hmm based o .ine handwritten text recognition. In *ICPR '04: Proceedings of the Pattern Recognition, 17th International Conference on (ICPR'04) Volume 2*, pages 541–544, Washington, DC, USA, 2004. IEEE Computer Society.

List of Figures