Hidden Markov Models in Text Recognition

J.C. Anigbogu and A. Belaïd

Crin-Cnrs/Inria Lorraine

Bât Loria, B.P. 239

54506 Vandœuvre-lès-Nancy Cedex

France

Abstract

A multi-level multifont character recognition is presented. The system proceeds by first delimiting the context of the characters. As a way or enhancing system performance, typographical information is extracted and used for font identification before actual character recognition is performed. This has the advantage of sure character identification as well as text reproduction in original form. The font identification is based on decision trees where the characters are automatically arranged differently in confusion classes according to the physical characteristics of fonts.

The character recognizers are built around the first and second order hidden Markov models (HMM) as well as Euclidean distance measures. The HMMs use the Viterbi and the Extended Viterbi algorithms to which enhancements were made. Also present is a majority-vote system that polls the other systems for "advice" before deciding on the identity of a character. Among other things, this last system is shown to give better results than each of the other systems applied individually.

The system finally uses combinations of stochastic and dictionary verification methods for word recognition and error-correction. Abstract

Key Words: Multifont Character Recognition, Font Recognition, Hidden Markov Models, Euclidean Distance, Viterbi Algorithm, Majority-Vote.

1 Introduction

Multifont character recognition is a field of machine vision that has a lot of application areas. These are notably in document archiving, mail sorting, automated data-processing, etc. This largely explains why a large number of researchers and industrial giants like IBM, Xerox, Siemens, Toshiba etc. are involved in OCR in general. Multifont recognition is necessary as a large number of typefaces are in use. As such, it is not rare to find a document that contains text in more than one font. Thus, there is necessity for automatic font identification. One of the objectives of first doing this is to reduce the amount of search necessary to identify characters. In document archiving, it is often also necessary to include the font of the document for the eventual reconstruction of the original text. Unfortunately, this important aspect of text recognition is often neglected. A few multifont character recognizers appear in the literatures(1–3). These systems operate without distinguishing between fonts. In this paper we shall present our contributions in the domain of font, character and word recognitions. Various approaches employing the stochastic methods of first and second order hidden Markov models (HMM) and the correlation methods of Euclidean distance measures are presented. The results of applying each of these methods along with their weighted variants are shown. We shall give a rundown of how recognition systems can be built using first and second order HMM. Particularly, we shall show the relationships between recognition results obtained, the model parameters such as the number of states, and the training process. The Viterbi algorithm (VA)(4) was used for the recognition process, with a decision-tree pre-classifier restricting the number of candidate characters. We improved on the results by making decisions based on weighted output probabilities instead of the usual method of using Viterbi scores. Systems based on Euclidean distance measure and its weighted form are also presented. The concept of systems based on majority voting is introduced. This new approach combines the strengths of the above methods. This method, in most cases, yields results superior to each of the individual methods above.

Experiments with datasets in ten fonts, each containing text of approximately 15,000 lower and upper-case characters and digits (i.e. total size $\approx$ 150,000 chars.) yielded scores of between 91.55% and 99.96% for the different methods. (Punctuations were eliminated as they are recognized separately by contextual and template matching methods). In all the methods, the system was forced to make decisions and so there are no rejects. These results were in the absence of any context other than *baseline* and *x-height*. As figure 1 shows, there is a post-processor that picks up the resulting strings for further processing. The typical final error-rate of the complete system was less than 0.45%.

In Section 2 we shall present the overall system architecture as well as the pre-processing aspects of feature extraction, character labeling and font determination.

Three broad recognition methods are presented in section 3. We shall present the first and second order HMM techniques as applied to character recognition. One method of improving the recognition rates is by making decisions on weighted Viterbi scores. The influence of the learning process on the recognition rates for a number states are presented. Theoretically, the second order model uses more information than the first order model and as such should yield better results. We shall show the validity or not of this assertion. Systems based on distance measures are also exposed. Finally, a majority-vote system that bases its response on the outputs of the above systems is presented. This system is shown to give better results in most cases than each individual method.

The output from the first stage of the processing is fed into a contextual processor. This uses a combination of methods, ranging from dictionary verification to second order HMM via trigram frequencies. In section 4 we shall see how all these methods are applied. We shall then conclude in section 5.

2 Image Processing

2.1 System Overview

As can be seen from the system diagram in figure 1, we employ a triple layer structure, the pre-treatment, the character and word recognition levels. As a side note, we shall not be presenting the aspects of the system dealing with skew-correction, having used a technique due to Baird(31) and the segmentation into blocks (Recursive XY-Cuts due to Nagy *et al*(32)) and connected components (due to Capson(33)). At the first level, the dominant font is determined on a paragraph by paragraph basis, the result of which is passed on to the second level where the system has the choice then of using either of three major recognition methods or a consensus of the three, majority-vote. In fact, for each of the methods, there are two sub-methods, the application of the classical recognition algorithm and that of "weighted" versions. As such, there are seven choices for output, the 7th being the result of the combination of the six methods. The output is then fed to a secondary phase that again has multiple methods for deciding on the final output. These methods are : the output of the character recognizer are sent out as strings without post-processing; the strings are validated with an application dictionary that can be updated; first or second order hidden Markov models are used to validate before output or dictionary validation. At the second level, the first and second order HMM rely respectively on tables of digram and trigram frequencies of the language of the document. In the next sections we shall detail more on each of the modules.

Figure 1 should be placed here

2.2 Character Labeling

Generally, feature extraction is the bottle-neck in an OCR system. Therefore, this operation should be reduced to a strict minimum. There is an inherent redundancy in written text as characters repeat themselves several times in the text. This fact can be used to limit recognition only to representatives of each shape for any size. Thus, characters that are accepted as being identical within a given tolerance are given the same labels. Only the first occurrence of each form is subsequently processed. This "code compression" requires the determination of similarity between forms. This calculation is a potential bottle-neck if the comparisons to be made are not heuristically limited. As can be seen later, if this is handled properly, an enormous gain is assured during recognition as features are extracted only for this reduced subset. Goshtasby(5) suggests coding the forms using shape matrices and then using binary operators to compare. Since shape matrices are binary in nature, their comparison would involve only an eXclusive-OR (XOR) operation. The shape discrimination criterion is quite simple.

Let A_1 and A_2 be two such shape matrices of size $\mathbf{m} \times \mathbf{n}$

$$Similarity = 1 - \frac{\sum_{i=1}^{m}\sum_{j=1}^{n} XOR(A_1(i,j), A_2(i,j))}{(m-2)n} \tag{1}$$

This is adjusted so that when two shapes are identical we obtain similarity = 1. A circle and a line are two completely dissimilar shapes. A line corresponds to a shape matrix with two rows of 1's and a circle to one with all elements equal to 1. Their difference is thus by XOR, $(m-2)n$, from where by eqn. 1 a 0 is obtained. Other shapes will produce similarities between 0.0 and 1.0.

Although this has been applied to character recognition[6], the principal fault is that at low point sizes, the shape matrices of characters like **a, e** and **o** do not differ much and produce high similarity scores between them. We opted for a similarity score that works directly on the connected components instead of their representation.

Let I_A and I_B be matrices of two characters of dimensions $H_A \times W_A$ and $H_B \times W_B$. To be comparable, they must obey the size constraints $H_B = \alpha H_A$ and $W_B = \beta W_A$ where α and β are stretch-factors which generally ensure differences of ± 5 pixels.

The correlation coefficient or similarity, S, is then given by

$$S = align(1 - \frac{XOR(I_A, I_B)}{max(I_A, I_B)}) \tag{2}$$

It can be seen that if the two images I_A and I_B are identical, their XOR is 0 and we obtain S = 1. On the other hand, if the images are completely dissimilar their XOR would yield more black pixels than either of the original two images as this operation simply toggles selected bits. This is to say that identical pixels where the two images coincide are cleared while where they differ are turned on. This ensures that

$$\frac{XOR(I_A, I_B)}{max(I_A, I_B)} > 1$$

and we obtain a negative correlation, i.e. complete dissimilarity. In all other cases, similarity values are between 0.0 and 1.0. The operator *align* ensures the best alignment of the two images. Figures 2 and 3 show two examples.

Figures 2 and 3 should be placed here

On large texts, we obtain compression rates of up to 98%, meaning that only 2% of the text is processed during recognition. From now on, we shall assume that this pre-processor has been run.

2.3 Feature Extraction

To use any recognition method, a kind of symbolic representation of the characters is necessary. Several types of features have been proposed and used with varying degrees of success[3,20,22,34–36]. Each of these features always tend to be dictated by the underlying recognition method (structural, stochastic, etc.). In our studies, We have chosen to use a number of geometrical and topological features so as to describe as best as possible the morphology of the characters. Hence, they can be used with different recognition methods, as we shall show. The sequences of these features later form the input to the learning and recognition systems (see section 3).

Figure 4 should be placed here

Transition features

Characters are subdivided into zones and the white-black-white runs are counted in these zones. These are then coded with respect to where they were found. For horizontal transitions, characters are divided into three zones :

[cap-height/ascender - x-height], [x-height - baseline], [baseline - descender]

with *cap-height* being the top of upper-case characters (e.g. **T**); *ascender*, the top of lower-case characters with ascender (e.g. **h**); *x-height*, the top of lower-case characters with neither ascender nor descender with *baseline* as base (e.g. **a**); *descender*, the bottom of lower-case characters with descenders (e.g. **q**).

For the vertical transition, a line is drawn vertically through the centroid. The number of intersections between the character and this line are noted if it is greater than two. If this number is less than three, we code it with respect to the positions of intersections. For examples, figure 4 shows that (**h, n, u**) have single points of intersection, which are situated in the middle, top and bottom respectively. It also shows that (**a, E**) have three, while (**b, p, F**) have two with the first having intersections at the middle and bottom and the others having through the middle and the top.

Cavity features

There are several characters with characteristic holes whose number and positions may vary according to font. For example we have **a**, ***a***, **b**, b, **g**, ***g*** etc. About a third of the characters in the English alphabet have holes. This presence and position of holes distinguish them from others. The holes are extracted and positioned with respect to the external contour of the character. With this characterization, holes are then centered (e.g. **o**), centered-left and top-right (e.g. **œ**), top (e.g. **p**), top and bottom (e.g. **8, B**), etc.

Centroidal Profile

One of the best ways to describe a character is to use its morphology. For such a description, we extract a number of indissociable features. The character is first divided into octants. For each octant, we calculate the average M_1 of the distances from the center of gravity G, of all pixels (N) on the contour of the character. In the same way, we calculate an average M_2, but only considering the imaginary (or real) pixels that belong to the englobing rectangle (equally N in number). A code of 0 to 3 is assigned according to the value of the ratio R of the averages M_1 and M_2. This ratio ensures the stability of codes for each ASCII character irrespective of its size and density.

$$R = \frac{M_1}{M_2} = \frac{\sum_{i=1}^{N} \|P_i - G\|}{\sum_{i=1}^{N} \|C_i - G\|} = \frac{\sum_{i=1}^{N} \sqrt{[(X_{P_i} - X_G)^2 + (Y_{P_i} - Y_G)^2]}}{\sum_{i=1}^{N} \sqrt{[(X_{C_i} - X_G)^2 + (Y_{C_i} - Y_G)^2]}}$$

where P_i and C_i are respectively points on the character's contour and its englobing rectangle.

With this coding, "I" will have the same code in each octant since its external contour coincides with the circumscribing rectangle. The code is thus 33333333 (the value is maximal in each octant). To have a wider range of codes while at the same time taking into account the "geographical" locations where each code was extracted, we added 1,2,3, and 4 to each code in the directions N,E,S and W respectively. We therefore have the following ranges in each of the four zones:

$$[1-4],[2-5],[3-6]\ et\ [4-7]$$

This will transform the code for I to 44556677.

Aspect ratio feature

Certain letters have very high height to width ratio, e.g. **l,i** while others have medium or low ratios e.g. **n,m,w**. This is also a good class indicator for characters.

2.4 Font Recognition

Some character shapes are peculiar to certain fonts while some are common to several fonts. As such, certain combinations of letters can give a clue as to the font. Figure 5 from the Macintosh family of fonts shows some character similarities and differences across some fonts. For example, the characters in Courier and Monaco are identical but differ in sizes and inter-character gaps. It might then be interesting to design the recognizer to be invariant to character size. In this case, the "font" and hence prototypes will be the same for identical characters. Since our features are size and thickness independent, certain cluster centers intersect or coincide (e.g. New York and Times Roman). However, additional contextual information based on average character sizes and inter-character gaps, complete the distinction. Although determining the exact font where only the sizes differ is not important, for the principle of reproducibility of original text, it sometimes is necessary. Any recognizer using these features that are invariant to size would produce identical results using the models for either font. The ideal in this case is to have one set of models but with pointers to names for the eventual reconstruction of the original text.

Figure 5 should be placed here

2.4.1 Font-tree Construction

Given that font information is inherent in the constituent characters$^{(7)}$, constructing prototypes for characters is thus a reasonable approach. During an *a priori* learning phase, and with some of the features earlier enunciated, prototypes are constructed for the characters of each font. These prototypes are simply the cluster centers for each of the characters.

To determine a common base for comparisons, some features such as the presence and position of holes, aspect ratio etc. and contexts like position with respect to *baseline* and *x-height*, are used to create a generic tree (figure 6). By passing all the characters in the database through this tree, different instantiations of this tree appear for each font. This is to be expected because, the tests that differentiate one character from others in one font, do not necessarily do the same in other fonts. This has the simple effect of regrouping different characters in different leaves depending on the font. Even when the characters at the leaves are the same for two fonts, the prototypes of these are not. Figures 7 and 8 for the Courier and Helvetica fonts respectively substantiate this.

Figures 6, 7 and 8 should be placed here

As stated earlier, the nature of fonts are embedded in some or all of their characters. The interest of these trees is therefore to extract different comparison points between fonts by distinguishing between identical and different characters. As can be seen from the above figures, each leaf might contain several different characters (in varying proportions) and the same character can appear in different leaves due to bitmap image defects.

In order to reduce the set of samples (up to 240) of a given character in a leaf-node to a single representative (prototype), we use the K-means algorithm(39). The representative of the samples of each character being a sequence of features, the distance used to make these groupings into classes is the Euclidean distance in $\Re^{13}$ space. For each leaf, we initialize the class centers with the first sample of each character (remember that the learning process is controlled and the database from where the generic tree is being fed is known at any instant). Then, at each iteration k, we distribute the set of features $\{x\}$ are distributed among these classes (K in number, for example) according to the following rule :

$$x \in E_j(k) \;\; if \;\; \|x - z_j(k)\| < \|x - z_i(k)\| \;\; \forall i = 1, 2, \ldots, K, \;\; i \neq j$$

where $E_j(k)$ is the set of samples that have $z_j(k)$ as center at the k^{th} iteration. From the distribution of the samples among the K classes, we calculate new centers $z_j(k + 1), \;\; for \;\; j = 1, 2, \ldots, K$, such that the sum of the distances (D_j) of all the samples of the center is minimum. In summary, we minimize

$$D_j = \sum_{x \in E_j(k)} \|x - z_j(k+1)\|^2, \;\; j = 1, 2, \ldots, K$$

where $z_j(k + 1)$ is simply the mean of the class $E_j(k)$.

$$z_j(k+1) = \frac{1}{N_j} \sum_{x \in E_j(k)} x, \;\; j = 1, 2, \ldots, K,$$

where N_j is the number of samples in class $E_j(k)$. The algorithm ends when the distance between the centers constructed at iterations k and $k + 1$ is zero for all j. This stopping criterion is

$$D = \|z_j(k+1) - z_j(k)\| \;\; \forall j = 1, 2, \ldots, K$$

It is trivial to note that $D = 0$ if $z_j(k + 1)$ and $z_j(k)$ are identical in each axis.

2.4.2 Recognition

Paragraphs are normally homogeneous in terms of the font except where sections are highlighted in *italics*, **bold** or <u>underlined</u>. Characters are passed through the same generic tree that was used to create instantiations for the fonts in the database. This process creates a new tree. To determine the dominant font in the paragraph, all that is needed is to find the tree that most resembles this tree. Since all the trees are static (i.e. they have exactly the same number of leaves), all that is needed is to find the minimal distance between the contents of the corresponding leaves of the new tree and the others.

Since we still do not know the identity of the characters in the paragraph, this involves finding the minimal distance between each form at the terminal node and the prototypes at the same node in the corresponding leaf of a font. This is repeated for all the fonts for which prototypes exist. The correct font is the one that generates a global minimum. We thus have,

$$CorrectFont = Arg \min_{1 \leq j \leq F} [\sum_{i=1}^{N} (Min_{0,a,A \leq \alpha \leq 9,z,Z} \|V_i - P_{\alpha j}\|)] \quad (3)$$

where :

- **N** is the number of characters in the paragraph,
- **F** is the number of fonts for which prototypes exist,
- $\mathbf{V}_i$ is the feature vector for the i^{th} character in the paragraph,
- $\mathbf{P}_{\alpha j}$ is the α^{th} prototype for font j.

It is evident that each $\mathbf{V}_i$ is only compared against the prototypes for the α's that are in the corresponding leaf and not the entire range of possible values of α.

3 Character Recognition

There is an abundance of character recognition methods and systems in the literatures(3,17,20,22,34–38), each with its strengths and weaknesses. We have chosen in this paper to explore two such methods : stochastic and metric, the former based on hidden Markov models and the latter on Euclidean distances. As we shall see later, combining different methods into a voting system generally provides better results and we have attempted to do that in this paper.

3.1 Introduction to Hidden Markov Models

HMM is a doubly stochastic process with an underlying process that can only be observed through another set of stochastic processes. It has a very high capacity of absorbing aberrations in forms. A single model in fact can represent hundreds or thousands of variations of the same form. This is one of the strengths and weaknesses of HMM; strength in the sense that aberrations that occur in a form are absorbed during training if there are enough samples of the correct form, and weakness in that if differing forms are not separated manually or by a clustering algorithm, the resulting models would be too general and thus error-prone. It is thus evident, that to assure the stability and non-overlap of models, homogeneous groups have to be created first.

These models are suited for high speed processing and can easily be hard-wired. HMM has known an increasingly wide-spread usage in the past two decades with a lot of applications in speech processing(8–16). A number of applications have also appeared in word-level recognition(17,21) and character recognition(22,23). The most important aspect of HMM is that it can be applied to practically any type of signal as shown by the diversity of applications cited. It has even been applied to the discrimination of planar shapes(25). The only thing common to these applications is the existence of an ordered list of features. One might then think that, to use this type of recognition method, we only need a vocabulary of stable features arranged optimally to obtain maximum separation of forms in feature-space. But, as can be seen from the varying degrees of success reported, the type of model applied plays a role (there are several, notably: ergodic or completely connected models, constrained left-to-right sequential or Barkis models, left-to-right parallel, continuous-density models, etc.), and so also the assumptions on the underlying model order, the number of states in the models etc. Rabiner(11–13) are recommended reading for treatment of the mathematics of HMM and the application and implication of some types of models. Among other things, it is noted that experience is the best teacher as there are no theoretical methods that guarantee an optimal system.

In this paper, we shall treat two main types of models which we used in our applications : the constrained left-to-right sequential and parallel models. We applied the first type at the character recognition level under two assumptions : that the sequences of features were governed by first-order properties and then that they were governed by second-order properties. The left-to-right parallel methods were applied at the word recognition level, again using both orders, with the further assumption that the models were non-stationary. As Rabiner(12) points out, there are a lot of factors that influence the quality of results obtained. We shall show the influence of the number of states on the result. It is also a well known fact that the training of models plays an important role during the recognition process. With fixed starting conditions we chose to investigate the influence of a stopping criterion : that of the number of iterations made during model training. We shall show the influence or lack of it for some iteration values.

A discrete HMM is described by :

- **N**, the number of states in the model, where states are noted $S = S_1, \ldots, S_N$ and the state at time t as s_t,
- **M**, the distinct number of observable symbols per state, noted as $V = V_1, \ldots, V_M$,
- **A**=$\{a_{ij}\}$, where a_{ij} is the probability of transiting from state i to state j given that the model is in state i,
- **B**=$\{b_j(k)\}$, $b_j(k)$ being the probability of observing the symbol V_k given that the model is in state j.

For the initial transition probabilities $\boldsymbol{\pi}$, we added an additional state designated as the starting state such that we have π_0=1, $\pi_i = 0, i = 1, \ldots, N$.

A complete model is then described by $\lambda = (\mathbf{A},\mathbf{B},\boldsymbol{\pi})$. To use this type of model, we need to resolve two main problems : that of training and of recognition. How do we, from a series of feature vectors $O_1^k \ldots O_T^k$ (T being the feature size) construct an efficient optimal model, $P(O|\lambda)$ for the character k, i.e. λ_k (training). Conversely, given a single feature vector $O_1 \ldots O_T$, how do we optimally and efficiently determine the λ_i (and thus the identity of the character), that best explains this sequence (recognition), from among F models, $\lambda_{j=1,\ldots,F}$.

3.2 First Order HMM

3.2.1 Model Training

Since we chose to process information on a per font basis, we chose to manually constitute the classes. This is straightforward enough as each character is a class. The training samples consisted of 240 samples of each character in point sizes of 10 to 18, scanned at 300 dpi. There are several methods of assigning initial values to the model parameters(11–13). We did not make any hypothesis on the feature distribution among states. Thus we chose to equally weight each feature in each state. As such we have $b_j(k) = \frac{1}{M} \quad \forall j, k$ which satisfies the constraint $\sum_{k=1}^{M} b_j(k) = 1$.

For reestimating the values of the model parameters a_{ij} and $b_j(k)$, we used the iterative Baum-Welch forward-backward algorithm(12). Figure 9 and table 1 show the initial model

(A & B), while figure 10 and table 2 show the resultant models for the characters **a**, ***a*** and **g**. As can be seen, the models are quite distinct although they started from the same initial model.

Figure 9, table 1, figure 10 and table 2 should be placed here

3.2.2 Recognition

We ran a series of *top-k* performance tests up to k=3, to observe the system behavior. We should point out here that the responses given by the system for normal and bold forms of a shape are the same, since the density does not affect the shape and as such the model (This information has already been used to identify the font for restitution purposes and so are not present in the HMMs). As defined by Baird(1), top-k recognition is such that the result is deemed acceptable if the correct character is among the first k responses of the system. This is a good measure of the reliability of a system. If for example, a good top-k performance is obtained with low values of k and with minimal effort and a reliable contextual post-processor exists, an optimal compromise can be reached between the low-level processor (character recognizer) and the high-level processor (word recognizer). Table 3 shows under VS (Viterbi Score), the top-1 performance of the first order HMM using the tree-constrained Viterbi algorithm. VS is calculated as $\delta_T(N)$, which is the score (highest probability) of the best path in an N-state model that accounts for the input sequence $O_1 \ldots O_N$. Thus, a decision-tree classifier first limits the number of possible candidates before recognition. The results range from 93.49% for a 4-state model of the font Courier to 99.83% for an 8-state model for the font Geneva.

It can be seen from this table that there are global improvements with increase in the number of states. However, this evolution is not the same for all fonts. For example, Courier gains about 3.4% between 4 states and 8 states, Princeton gains only 0.5% while Chicago in fact loses 0.24%. This can be explained by the fact that identical ASCII characters differ in shape from font to font (see figure 5). As such, a model that suits the character in one font might not favor it in another. The findings here agree with what obtains in speech analysis, where the optimal number of states varies from speaker to speaker(15,16). One can safely conclude then that the optimal number of states is dependent on the font.

Table 4 shows the performances if we accept the fact that the correct character is either of the first two system responses.

Tables 3 - 5 should be placed here

Table 5 shows that if we accept top-3 performance, going beyond 4 states does not change the results as the correct character is already among the responses. If it is not among the top 3 it is probably because of mutilation or merging and thus misclassification by the decision-tree pre-classifier.

A problem with HMM recognizers is that the training algorithm, though optimal, only guarantees reaching a critical point for any model. This point might be a point of inflexion or a local maximum which may or may not be the global maximum. We chose to study the performance variations for each n-state model as a function of the number of iterations. We constructed models after each of 5 to 28 iterations. Fig. 11 shows the results averaged over all fonts for each of the 5 n-state models. Apparently, the number of iterations does not greatly affect the recognition rate. This is confirmed by table 6 for the fonts Helvetica and Times Roman. We can conclude that the number of iterations does not appreciably modify the recognition rate.

Figure 11 and Table 6 should be placed here

As a way of improving the reliability of the OCR, we chose to base the recognition decision on the weighted probability $\delta_T(N) * weight[L][C]$, where $weight[L][C]$ is a weight factor determined experimentally. It is defined as the probability of the character C belonging to the class L as determined by the decision-tree classifier. These probabilities were estimated by passing all characters in our database through pre-classifiers constructed for each font and noting the proportions of each character that end up in each class. These values were then used to reinforce or reduce the scores determined by the Viterbi classifier. This simple heuristic increased the reliability of the recognition algorithm without increasing its complexity. The results are shown in tables 3, 4 and 5 under WVS (Weighted Viterbi Score). It can be seen that very good results can be obtained without necessarily going to higher-order models or large number of states. Fig. 12 shows the average recognition rates of this method as a function of the number of iterations.

Figure 12 should be placed here

Figures 11 and 12 also show that the two recognition methods (VS and WVS) exhibit the same behavior except for the 4-state model under WVS whose recognition rate increased with each increase in the number of iterations. From our results, we can state that there is no good theoretical way to choose the number of states as they are not physically related to any observable symbol and also with respect to the font.

3.3 Second Order HMM

Kundu *et al*(17,18) and Liu(16) contend, and rightly so (see section 4.3.1), that second order models use more information than first order models, and can thus outperform the latter. This assertion might be true at the word level, where the succession of characters in a language follow certain rules. But can the same thing be said of the succession of features? To answer this question, we modeled the sequences of features as second order Markovian functions.

3.3.1 Model Training

For training, we used the extended version of the Baum-Welch forward-backward algorithm as presented by Kriouile(14). These extensions for the forward variable α and backward variable β are given below.

1. Initialization

   $\alpha_2(i,j) = \pi_i b_i(O_1) a_{ijk} b_j(O_2),\ 1 \le i,j \le N$

2. Induction for $2 \le t \le T-1$

   $\alpha_{t+1}(j,k) = [\sum_{i=1}^{N} \alpha_t(i,j) a_{ijk}] b_k(O_{t+1}),\ 1 \le j,k \le N$

3. Termination

   $P(O|\lambda) = \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_T(i,j)$

1. Initialization

   $\beta_T(i,j) = 1,\ 1 \le i,j \le N$

2. Induction for $T-1 \ge t \ge 2$

   $\beta_t(i,j) = \sum_{k=1}^{N} a_{ijk} b_k(O_{t+1}) \beta_{t+1}(j,k)$

We only need to rewrite the re-estimation formula for a_{ijk} as the definitions for a_{ij} and $b_j(k)$ remain valid from the first order model. $\overline{a}_{ijk}$ is a straightforward extension of the first-order re-estimation of $\overline{a}_{ij}$. Thus we have

$$\begin{aligned}\overline{a}_{ijk} &= \frac{\mathit{Average\ number\ of\ the\ double\ transitions}\ s_i\ \mathit{to}\ s_j,\ s_j\ \mathit{to}\ s_k}{\mathit{Average\ number\ of\ transitions\ from}\ s_i\ \mathit{to}\ s_j} \\ &= \frac{\sum_{t=1}^{T-2} \eta_{t+1}(i,j,k)}{\sum_{t=1}^{T-2}\sum_{k=1}^{N} \eta_{t+1}(i,j,k)}\end{aligned}$$

where

$$\eta_{t+1}(i,j,k) = \frac{\alpha_{t+1}(i,j)a_{ijk}b_k(O_{t+2})\beta_{t+2}(j,k)}{P(O|\lambda)}$$

3.3.2 Recognition

For recognition, we applied the extended Viterbi algorithm as found in He[(24)]. As with first order models, we skipped the tracking of the sequence of states. The results under the two recognition modes are shown in tables 7 through 9. On comparing tables 3 and 7, we can see that there are no remarkable improvements in the recognition rates. As such, and considering the increased complexity, we can conclude that the first order Markov process adequately models our data.

Tables 7 - 9 should be placed here

3.4 Euclidean Distance Measures

Typically, the similarity between two forms x and z is given by

$$S_{xz} = \sum_{i=1}^{N}(x_i - z_i)^2, (x_i, z_i, i = 1, \ldots, N) \tag{4}$$

Since all the features are points in Euclidean space, we decided to transform eqn. 4 into a probability measure such that :

$$P(x = p) = (1 - \frac{\sqrt{S_{xp}}}{\sqrt{S_{po}}}) \tag{5}$$

where $\sqrt{S_{xp}}$ is the Euclidean distance between the character x and the prototype p, and $\sqrt{S_{po}}$ is the distance between p and o, a fixed origin in space where all feature values are null. Thus

$$S_{xp} = \sum_{i=1}^{N}(x_i - p_i)^2$$

and

$$S_{po} = \sum_{i=1}^{N}(p_i)^2$$

If the characters x and p are identical, $\sum_{i=1}^{N}(x_i - p_i)^2 = 0$ and we obtain P(x=p)=1. For all other values, we obtain values less than 1. To be true probabilities, all values less than zero are considered 0.

3.4.1 Training

As already explained for HMM, prototypes were constructed on a per character and per font basis. The prototype vectors represent the centers formed by the 240 samples available for each character. A simple frequency count was used to generate the representative of each feature value. This process is equivalent to applying the K-means algorithm and creating only one class.

3.4.2 Recognition

For the recognition process, we applied eqn. 5. The output character is the prototype that yields the highest score with respect to the input feature vector. The results are shown in fig. 13 and table 10 under ED. We can see that the results again vary with font and range from 91.55% for the font Athens to 99.20% for the font Princeton.

The drawback with eqn. 5 is that it assumes, as is normal in Euclidean distance measures, that two characters x and y that are equidistant in hyper-space from a prototype p have equal chances of being the character represented by p. As example, consider a prototype $p = 11122$ and two characters $x = 00021$ and $y = 11120$. Both yield similarity scores of 0.5 according to eqn. 5, but it is evident from mere looking that y is closer to p in a certain sense than is x. y and p differ only in the 5th feature value f_5, while x and p differ in $f_1f_2f_3f_5$. To correct for this phenomenon, it was necessary to introduce weights so as to favor resemblance while accentuating differences. This yields a simpler formula

$$P(x = p) = \frac{\sum_{\forall x_i = p_i = \sigma_i, i=1}^{N} wt_p(\sigma_i)}{N} \tag{6}$$

Equation 6 is very simple but resolves the problems associated with distance measures in general. $wt_p(\sigma_i)$ is the probability of the i^{th} feature for the prototype p, having the value

σ. In other words, it is the proportion of samples of the character represented by p that have the value σ as the i^{th} feature value. It is trivial to note that the numerator in eqn. 6 equals N if $x = p$. For all other cases, $0 \leq P(x = p) < 1$. The results obtained with this transformation are shown in fig. 13 and table 10 under WED (Weighted Euclidean Distance).

We remark the very good performance of this method in spite of its simplicity. The increase in recognition rates is general across all fonts. It in fact outperforms all other methods for the font Toronto. The results range from 95.91% for Courier to 99.72% for Toronto.

Figure 13 should be placed here

3.5 Majority-Vote from Multiple Recognizers

From observations, we remarked that the errors committed by the different recognizers were not always on the same character. This has already been noted in Ho(40) and Suen *et al*(41) where the former applied multiple classifiers to degraded printed text and the latter to hand written numerals, both with success. If it does improve the recognition rate of degraded text, it should logically do the same for normal text. We decided to construct a system based on a majority-vote strategy. For each character, the system polls all the six methods for their opinions, collates the results and the character that has the highest votes is the output. Although this strategy is not always guaranteed to give the best results, one would expect it to make less mistakes than each of the voters. In most cases, this is true as shown in table 10 for 8-state models. The reasons for certain failures may be character blurring, merging or breaks that can cause the pre-classifier or the voters to make mistakes. Also the majority is not always right as all the voters use exactly the same set of features as opposed to Ho's system that used different features for each of her six voting methods (with the heavy penalty of several passes over bitmap data to extract the features). The system response-time is still reasonable as recognition is typically performed on less than 5% of the original text. If we add a feedback loop so as to handle situations where the majority is only relative, we might recover from some errors. We chose, however, to leave the error correction for the word recognizer as there is a limit to what can generally be done at the character recognition level.

Table 10 should be placed here

Table 10 shows that the direction of the recognition curves of all the methods are identical. For simple fonts like Geneva and Princeton, they all have very high recognition rates that differ by less than 1%. But, for exceptionally thick fonts like Athens (possible blurring) and light fonts like Courier (possible breaks) they all show reduced recognition rates that again differ by less than 1% except for the unweighted Euclidean distance method that degrades considerably.

3.6 Conclusion

It is often not enough recognizing characters. Due to constraints, a balance between the percentage recognition rate and the recognition speed is sometimes necessary. For example, if two algorithms A and B yield the same results, but A takes half as much time as B, generally algorithm A would be preferred. For each given number of states, we timed the recognition speed over the same datasets, for the first and second order models. We also timed the recognition speed of the Euclidean distance methods which averaged 3040cps. HMM methods account for 4 of the voters in the majority-vote method and as such, the number of states used affects its recognition rate and speed. We thus timed it for each choice of a given number of states. These results are depicted in table 11. These speeds, from a SUN IPX are averaged over 10 fonts.

Table 11 should be placed here

4 Word Recognition

Character recognition has its limits and as such at some point post-processing is necessary. This generally involves word-level recognition where we employ the rules of the underlying language to disambiguate certain confusions generated by the character recognizer. For word recognition, depending on choice, our system either furnishes a sequence of the first response for each character position as words (within clear word boundaries), or this output sequence is passed to another module for matching against a dictionary. The sequence of the first three possibilities as furnished by the character recognizer can also be used to form a Viterbi net. The output from this system is the "most probable" word. This is obtained in two ways: either by treating the information on character-pair (digram) or on triplet (trigram) basis. The result obtained can again be verified with a lexicon as the output is never guaranteed to be a valid word or this is sent out as output.

4.1 Dictionary Verification

The dictionary is partitioned into classes using a technique similar to the envelope method(30). The words are first classified by length and then regrouped according to the positions of the constituent characters with respect to the *x-height* and the *baseline*. Characters that have ascenders are given the code 1, those that have descenders are given the code 3 and those with neither descender nor ascender are assigned the code 2. For successive identical codes, only one is conserved. This coding method preserves the typographical envelope of the word. Of the remaining codes, (after suppression), only five are kept and where there are less, zeros are used to pad. The length of the original word, in two digits, is then appended to this code for a total of seven digits. This ensures that all the codes have the same length for ease of organization and access. As examples

e x a m p l e

2 2 2 2 3 1 2

will translate to 2 3 1 2 and then to 2 3 1 2 0 which in turn gives the final code of 2 3 1 2 0 0 7.

a l g o r i t h m s

2 1 3 2 2 2 1 1 2 2

which translates into 2 1 3 2 1 2 and then to 2 1 3 2 1 and finally to 2 1 3 2 1 1 0.

To take into account the fact that some characters can merge and others broken, we use an edit distance of one on either side of the observed length of the word. This takes care of the deletion or the addition of one letter. To limit the explosive nature of the possibilities from a standard dictionary, the OCR output is used to constrain the result.

The matching between the input and dictionary words is achieved using the Ratcliff-Obershelp(27) pattern matching algorithm. As fig. 1 shows, the user can update a working dictionary as the recognition proceeds.

4.2 First Order HMM

4.2.1 Model Construction

First order HMM has limited correcting capability as Hull *et al*(26) points out. Kundu(18) has proved that for the English language, the optimal model is at least of order two. However, it is a useful method if the parameters are well chosen and if it is also complemented with a dictionary, especially if the OCR has limited reliability. For our French applications, we constructed constrained parallel Markov models (or Viterbi Nets) for words using a 190,000 word dictionary while for English we used a 48,000 word dictionary. From the dictionary, we calculated the matrix **A** which in this case, is the digram frequencies of letter pairs, while the traditional matrix **B** was replaced with character position dependent probabilities. Similar techniques have been applied to English text by Sinha and Prasada(29), and by Kundu *et al*(17). Shinghal and Toussaint(28) have also studied this phenomenon with regards to the source statistics. There were 22 models for the English language (2-22, 28. No words of lengths 23-27 were available) and 23 models for French (2-23, 25. No 24-letter words were found). We showed in section 3 that the number of states has an influence on the recognition rate and hence on the distributions of

A and B. Hence, as opposed to the method employed in Kundu[17] to determine A and B (eqn. 7), we preferred using length constraints (eqn. 8). The word-length is simply the equivalent of N, the number of states in an HMM. Using the same notations as in section 3, this is to say that $A = a_{ij}$ is the probability of the letter V_i being followed by the letter V_j for a given word-length. Also, B(t,k) is the probability that the letter V_k is observed at position t in the word.

$$b_j(k) = \frac{\gamma_j(k)}{\gamma_j} \tag{7}$$

where γ_j is the total number of times the letter depicting the state j appear in the sample and $\gamma_j(k)$, the total number of times the symbol represented by V_k is observed while the model is in state j.

Modifying eqn. 7 to take into account the inter-dependence between the number of states and letter distribution yields

$$b_t(k) = \frac{\gamma_t(k)}{\eta_T} \tag{8}$$

where $b_t(k)$ is the probability of the letter or symbol V_k appearing in state t, $\gamma_t(k)$ is the total number of times the symbol V_k is observed in state or letter-position t and η_T is the total number of symbols observed in state t summed over all samples of size T. That is,

$$\eta_T \equiv \sum_{\forall V_i} \gamma_t(i)$$

In the same manner, the calculation of $A = a_{ij}$ is constrained by $N = T$. That is,

$$a_{Tij} = \frac{\xi(i,j)}{\sum_{\forall V_k} \xi(i,k)} \tag{9}$$

where $\xi(i,j)$ is the number of transitions from letter V_i to letter V_j and $\sum_{\forall V_k} \xi(i,k)$ is the total number of transitions from letter V_i.

We assume that letter-pair frequencies or digrams depend on word-length (eqn. 8). As an example, the probability of having the pair "ab" in a two-letter word is generally not the same as having the same letter-pair in a seven-letter word. This assertion is more evident for triple-letter frequencies.

Figure 14 shows an example of a 7-letter word for the following OCR output : **(m,w), a, (l,t,f), a, i, (s,z), e**. The character position dependent probabilities are in **bold** while the transition frequencies are on the arcs.

Figure 14 should be placed here

The path followed by the recognition algorithm is shown in **bold** with the maximum probability of 1.498e-15 being for the word "mataise". This is of course false, as the real word is "malaise" which was well recognized by the OCR. (The figure shows that the correct letters were the first choice of the recognizer). The reason for this anomaly is not far fetched. Figure 14 shows that the probabilities of choosing "al", "at" and "af" having chosen "ma" are such that

$$pr(af) < pr(al) < pr(at)$$

and hence the choice of "at". As shown later in section 4.3, this problem is resolved by taking triple-letter pairs. However, a simpler way of resolving this without increasing the complexity of the recognition algorithm can be designed. Intuitively, weighting the choices using the output probabilities as emitted by the OCR should modify the behavior. This heuristic should have the net effect of reinforcing the choice of certain paths while reducing the importance of others. This situation is depicted in fig. 15 where the correct path is now followed although the absolute recognition probabilities are much lower, here 5.582e-30, because of the additional multiplication involved. This method requires less calculation than is necessary for second order models but uses much more memory as the output probabilities from the character recognizer are carried along with the choices made. The OCR Viterbi scores are shown in *italics*. We can see that the OCR confidence in the 3^{rd} character being l is several orders of magnitude that of it being either t or f (0.0461 compared to 2.616e-16 and 1.193e-23).

Figure 15 should be placed here

In the classical Viterbi algorithm there are 26 possible alternatives for each character position. We know that in any language, not all letter successions are possible. Besides, the OCR is there to limit the number of choices possible at any time. Thus, since the text is modeled as a first order Markov process, we apply the Modified Viterbi Algorithm(28) with the maximum depth of search set to $N = 3$ (3 alternatives).

4.2.2 Viterbi algorithm for first order models

Suppose that the array $p(T, N)$ holds the output triplets given by the OCR, with T being the word length. $p_t(i)$ is to be read as the i^{th} alternative for character position t. Optimizing the calculation by using the trellis structure of $p(T, N)$ and calculating $\delta_t(j)$ only for the non-empty $p_t(j)$, that is $p_t(j) \neq \phi$.

1. Initialization

   $\delta_1(i) = log[\pi_i] + log[B(1, o(1))], \quad 1 \le i \le N, \quad \ni \pi_i \neq \phi$

   where π_i is equivalent to $A(0, p_1(i))$; that is, the probability that the alternative i is the first character of the word; o(t) is the t^{th} observation, which is also equivalent to $p_t(1)$; (if taken directly from the input matrix $p(T, N)$).

$\delta_t(i)$ in general, is the maximum probability that accounts for the first t characters ending in state i where $t \leq T$ and i is any of the letters a to z.

ψ which holds the traversed optimal states (i.e. characters) is not initialized. $\psi_1(i)$ is the delimiter or word boundary. From 4 below, we can see that the optimal choice of the first character $q_1^* = \psi_2(q_2^*)$ and ψ_2 is determined during recursion. See 2 below.

Remark: This initialization is often found in the literatures.

2. Recursion

$$\delta_t(j) = \max_{1 \leq i \leq N}[\delta_{t-1}(i) + log[A(p_{t-1}(i), p_t(j))]] + log[B(t, o(t))], \quad 2 \leq t \leq T$$

$$1 \leq j \leq N$$

$$\ni p_t(k) \neq \phi$$

$$k = i, j$$

$$\psi_t(j) = arg \max_{1 \leq i \leq N}[\delta_{t-1}(i) + log[A(p_{t-1}(i), p_t(j))]]$$

3. Termination

$P^* = \max_{1 \leq i \leq N}[\delta_T(i)]$ for left-to-right parallel models.

$q_T^* = arg \max_{1 \leq i \leq N}[\delta_T(i)]$

4. Optimal state sequence backtracking.

$q_t^* = \psi_{t+1}(q_{t+1}^*), \quad$ t = T-1,...,1

The output sequence of this model is used to index back into the original output sequence to retrieve the most probable sequence of characters.

4.3 Second Order HMM

4.3.1 Model Construction

Second order HMM uses more information than a first order model and thus can potentially perform better. With the same assumption of the non-stationary nature of the data, it is then necessary to reformulate only the calculation of **A**. Character position dependent probabilities that constitute **B** are unchanged. Thus, for triple-letter transitions we have,

$$a_{Tijk} = \frac{\xi(i,j,k)}{\sum_{\forall V_k} \xi(i,j,k)} \tag{10}$$

where $\xi(i,j,k)$ is the number of transitions from letter V_i at $t-2$ to letter V_j at $t-1$ and to state k at t and $\sum_{\forall V_k} \xi(i,j,k)$ is the total number of transitions from letter V_i.

Figure 16 shows the new Viterbi net and the path taken by the recognition algorithm. The correct word "malaise" is chosen as it has the highest probability (3.957e-15). The null (0.0) transition values are assigned low values instead of being cut-off so as to cater for valid sequences found by the OCR but are not authorized in the language, according to the dictionary. Acronyms are common examples of this type of problem.

Figure 16 should be placed here

The trigram frequencies are demarcated in figure 16. For example, the probability of having "maf" is 0.0031 and that of having "tai" is 0.196.

4.3.2 Viterbi algorithm for second order models

As with first order, the array $p(T,N)$ holds the output triplets given by the OCR for each position $t = 1,\ldots,T$; The optimization quantity $(p_t(j) \neq \phi)$ still remains valid.

The algorithm for traversing a second order net is directly derivable from that of first order. Since the number of states is still fixed (i.e. the word length and hence **B** is static.), we only need to compute $N \times (N \times N)$ transitions.

1. Initialization

   $$\delta_1(i) = log[\pi_i] + log[B(1, o(1))], \quad 1 \leq i \leq N \quad \ni \pi_i \neq \phi$$

   where π_i remains equivalent to $A(0, p_1(i))$; the delimiter "-" shown in figure 16 has position 0. That is, $A(0, p_1(i)) \equiv A(p_0(-), p_1(i))$.

   $$\delta_2(i,j) = \delta_1(i) + log[A(p_1(i), p_2(j))] + log[B(2, o(2))], \quad \begin{array}{l} 1 \leq i,j \leq N \\ \ni p_t(l) \neq \phi \\ l = i,j \end{array}$$

   For the same reasons as in first order models, ψ_1 and ψ_2 are not initialized.

2. Recursion

   $$\delta_t(j,k) = \max_{1 \leq i \leq N} [\delta_{t-1}(i,j) + log[A(p_{t-2}(i), p_{t-1}(j), p_t(k))]] + log[B(t, o(t))],$$

   $$\psi_t(j,k) = arg_i \max_{1 \leq i \leq N} [\delta_{t-1}(i,j) + log[A(p_{t-2}(i), p_{t-1}(j), p_t(k))]], \quad \begin{array}{l} 3 \leq t \leq T \\ 1 \leq j,k \leq N \\ \ni p_t(l) \neq \phi \\ l = i,j,k \end{array}$$

3. Termination

$P^* = \max\limits_{1 \le i,j \le N} [\delta_T(i)]$ for our left-to-right parallel models.

$q^*_{T-1} = arg_i \max\limits_{1 \le i,j \le N} [\delta_T(i,j)]$

$q^*_T = arg_j \max\limits_{1 \le i,j \le N} [\delta_T(i,j)]$

4. Optimal state sequence backtracking.

   $q^*_t = \psi_{t+2}(q^*_{t+1}, q^*_{t+2}), \quad$ t = T-2,...,1

As before, the values in q^*_t are used to index back into p_t to retrieve the most probable character sequence.

4.4 Conclusion

We ran several tests to study the performance of the HMM based correction methods. Table 12 shows a table of some words as determined by the OCR and the decisions taken by each of the different methods.

Figure 12 should be placed here

Our conclusion is that it is not necessary to use second order models. The weighted first order HMM with much less computation performed as well as second order models. However, both require a lot of memory; the first to carry along OCR recognition scores for all alternatives besides digram frequencies, while the second is for memorizing trigram frequencies. Hull *et al*(26) attempted error-correction using the first order model and concluded that this method gives mediocre results. They did not have the benefit of OCR context having worked on simulated data.

5 Conclusions

In this paper we have presented a multi-layered, multifont character recognition system. The character recognizer incorporates the stochastic methods of first and second order hidden Markov models and the correlation method of Euclidean distance measures. These methods serve as either stand-alone systems or as voters in a majority-vote scheme. Improvements were made on the recognition rates by augmenting each of the basic algorithms by weight functions. This raised the recognition rate to a maximum of 99.96% for 10 fonts.

To benefit from the inherent character redundancy in running text, a similarity filter is used to determine representatives of character classes. This reduced number of classes is used to automatically determine the dominant font on per paragraph basis so as to be able to reconstruct the original text. This font determination also reduces the classes on which the above recognition algorithms are applied. This process speeds up the character recognition and word reconstruction processes.

The second level of the global recognition process uses several methods for contextual post-processing. First and second Markov processes are used to attempt correction of errors made by the OCR at the first level. The first order model results are considerably improved upon by using OCR scores to constrain transition choices. This dramatic increase in recognition results without increased complexity shows that a good first order hidden Markov process can be used to model languages instead of a second model process. It in fact outperforms the second order model when certain character transitions are rare and thus have very low probabilities. The system presented allows the results obtained here to be filtered by an application dictionary which can also work directly on the OCR output.

REFERENCES

1. H. S. Baird and R. Fossey, "A 100-font Classifier," *Proc. 1st Int. Conf. Doc. Anal. Recognition*, Saint-Malo, France, 332-340 (October 1991).
2. S. Shlien, "Multifont Character Recognition for Typeset Documents," *Int. J. Pattern Recognition Art. Intell.* **2,** 608-620 (1988).
3. S. Kahan, T. Pavlidis and H. S. Baird, "On the Recognition of Printed Characters of Any Font and Size," *IEEE Trans. Pattern Anal. Mach. Intell.* **PAMI-9,** 274-288 (1987).
4. G. D. Forney Jr., "The Viterbi Algorithm," *Proc. IEEE* **61,** 268-278 (1973).
5. A. Goshtasby, "Description and Discrimination of Planar Shapes Using Shape Matrices," *IEEE Trans. Pattern Anal. Mach. Intell.* **PAMI-7,** 738-743 (1985).
6. A. Goshtasby and R. W. Ehrich, "Contextual Word Recognition Using Probabilistic Relaxation Labeling," *Pattern Recognition* **21,** 455-462 (1988).
7. R. Rubinstein, "Digital Typography: An Introduction to Type and Composition for Computer System Design," Addison-Wesley (1988).
8. J. K. Baker, "Stochastic Modeling For Automatic Speech Understanding," *Speech Recognition*, Academic Press, New York, 521-542 (1974).
9. J. F. Mari and S. Roucos, "Speaker Independent Connected Digit Recognition Using Hidden Markov Models," *Speech Technology*, Academic Press, New York, 22-24 (1985).
10. L. R. Rabiner, S. E. Levinson and M. M. Sondhi, "On the Application of Vector Quantization and Hidden Markov Models to Speaker-Independent, Isolated Word Recognition," *B.S.T.J.* **62,** 1075-1105 (1983).
11. L. R. Rabiner and R. H. Juang, "An Introduction to Hidden Markov Models," *IEEE ASSP Magazine* **3,** 4-16 (1986).
12. L. R. Rabiner, "Mathematical Foundations of Hidden Markov Models," *NATO ASI Series* **F46,** Recent Adv. Speech Understanding Dialog Sys., 183-205 (1988).
13. L. R. Rabiner, "A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition," *Proc. IEEE* **77,** 257-286 (1989).
14. A. Kriouile, J. F. Mari and J. P. Haton, "Speech Recognition Based on Second Order HMM," *Proc. 5th Int. Symp. Applied Stoch. Models Data Anal.*, Granada, Spain, World Scientific, 360-370 (April 1991).

15. L. R. Rabiner, J. G. Wilpon and F. K. Soong, "High Performance Connected Digit Recognition Using Hidden Markov Models," *IEEE Trans. Acoustics Speech Signal Process.* **37,** 1214-1225 (1989).

16. L. C. Liu, D. Chiou and H. C. Wang, "A Speech Recognition Method Based on Feature Distributions," *Pattern Recognition* **24,** 717-722 (1991).

17. A. Kundu, Y. He and P. Bahl, "Recognition of Handwritten Word: First and Second Order Hidden Markov Model Based Approach," *Pattern Recognition* **22,** 283-297 (1989).

18. A. Kundu and Y. He, "On Optimal Order in Modeling Sequence of Letters in Words of Common Language as a Markov Chain," *Pattern Recognition* **24,** 603-608 (1991).

19. D. L. Neuhoff, "The Viterbi Algorithm as an Aid in Text Recognition," *IEEE Trans. Inform. Theory*, 222-225 (1975).

20. R. F. H. Farag, "Word-Level Recognition of Cursive Script," *IEEE Trans. Comput.*, **C-28,** 172-175 (1979).

21. J. C. Anigbogu and A. Belaïd, "Application of Hidden Markov Models to Multifont Text Recognition," *Proc. 1st Int. Conf. Doc. Anal. Recognition*, Saint-Malo, France, 785-793 (October 1991).

22. K. Kordi, C. Xydeas and M. Holt, "Printed Character Recognition Using Markov Models," *Onzième Colloque GRETSI*, Nice, France, 507-509 (June 1987).

23. J. C. Anigbogu and A. Belaïd, "Recognition of Multifont Text Using Markov Models," *Proc. 7th Scandinavian Conf. Image Anal.*, Aalborg, Denmark, 469-476 (August 1991).

24. Y. He, "Extended Viterbi Algorithm for Second Order Markov Process," *Proc. 9th IEEE Conf. Pattern Recognition*, Rome, Italy, 718-720 (1988).

25. Y. He and A. Kundu, "2-D Shape Classification Using Hidden Markov Model," *IEEE Trans. Pattern Anal. Mach. Intell.* **PAMI-13,** 1172-1184 (1991).

26. J. J. Hull, S. N. Srihari and R. Choudhari, "An Integrated Algorithm for Text Recognition: Comparison with a Cascaded Algorithm," *IEEE Trans. Pattern Anal. Mach. Intell.* **PAMI-5,** 384-395 (1983).

27. J. Ratcliff, "Pattern Matching by Gestalt," *D.D.J* **181,** (1988).

28. R. Shinghal and G. T. Toussaint, "The Sensitivity of the Modified Viterbi Algorithm to the Source Statistics," *IEEE Trans. Pattern Anal. Mach. Intell.* **PAMI-2,** 181-185 (1980).

29. R. M. K. Sinha and B. Prasada, "Visual Text Recognition Through Contextual Processing," *Pattern Recognition* **21,** 463-479 (1988).

30. R. M. K. Sinha, "On Partitioning a Dictionary for Visual Text Recognition," *Pattern Recognition* **23,** 497-500 (1990).

31. H. S. Baird, "The Skew Angle of Printed Documents," *Proc. SPSE 40th Conf. Symp. Hybr. Imag. Sys.*, Rochester, NY, 21-24 (1987).

32. G. Nagy, S. C. Seth and S. D. Stoddard, "Document Analysis with an Expert System," *Pattern Recognition Pract.* **II,** 149-159 (1986).

33. D. W. Capson, "An Improved Algorithm for the Sequential Extraction of Boundaries from a Raster Scan," *Comp. Vis., Graph. Imag. Process.* **28,** 109-125 (1984).

34. H. S. Baird, "Applications of Multi-dimensional Search to Structural Feature Identification," *Work. Syn. Stru. Pat. Rec.*, Sitges, 1-13 (1986).

35. H. S. Baird, "Feature Identification for Hybrid Structural/Statistical Pattern Classification," *Comp. Vis., Graph. Imag. Process.* **42,** 318-333 (1988).

36. G. Baptista and K. M. Kulkarni, "A High Accuracy Algorithm for Recognition of Handwritten Numerals," *Pattern Recognition* **21,** 287-291 (1988).

37. P. W. Frey and D. J. Slate, "Letter Recognition Using Holland-Style Adaptive Classifiers," *Mach. Learn.* **6,** 161-182 (1991).

38. S. Impedovo, L. Ottaviano and S. Occhinegro, "Optical Character Recognition - A Survey," *Int. J. Pattern Recognition Art. Intell.* **5,** 1-24 (1991).

39. J. T. Tou and R. C. Gonzalez, "Pattern Recognition Principles," Addison-Wesley (1974).

40. T. K. Ho, "A Theory of Multiple Classifier Systems and Its Application to Visual Word Recognition," Ph.D Thesis, SUNY Buffalo, 160pp. (1992).

41. C. Y. Suen, C. Nadal, R. Legault, T. A. Mai and L. Lam, "Computer Recognition of Unconstrained Handwritten Numerals," *Proc. IEEE,* 1162-1180 (1992).

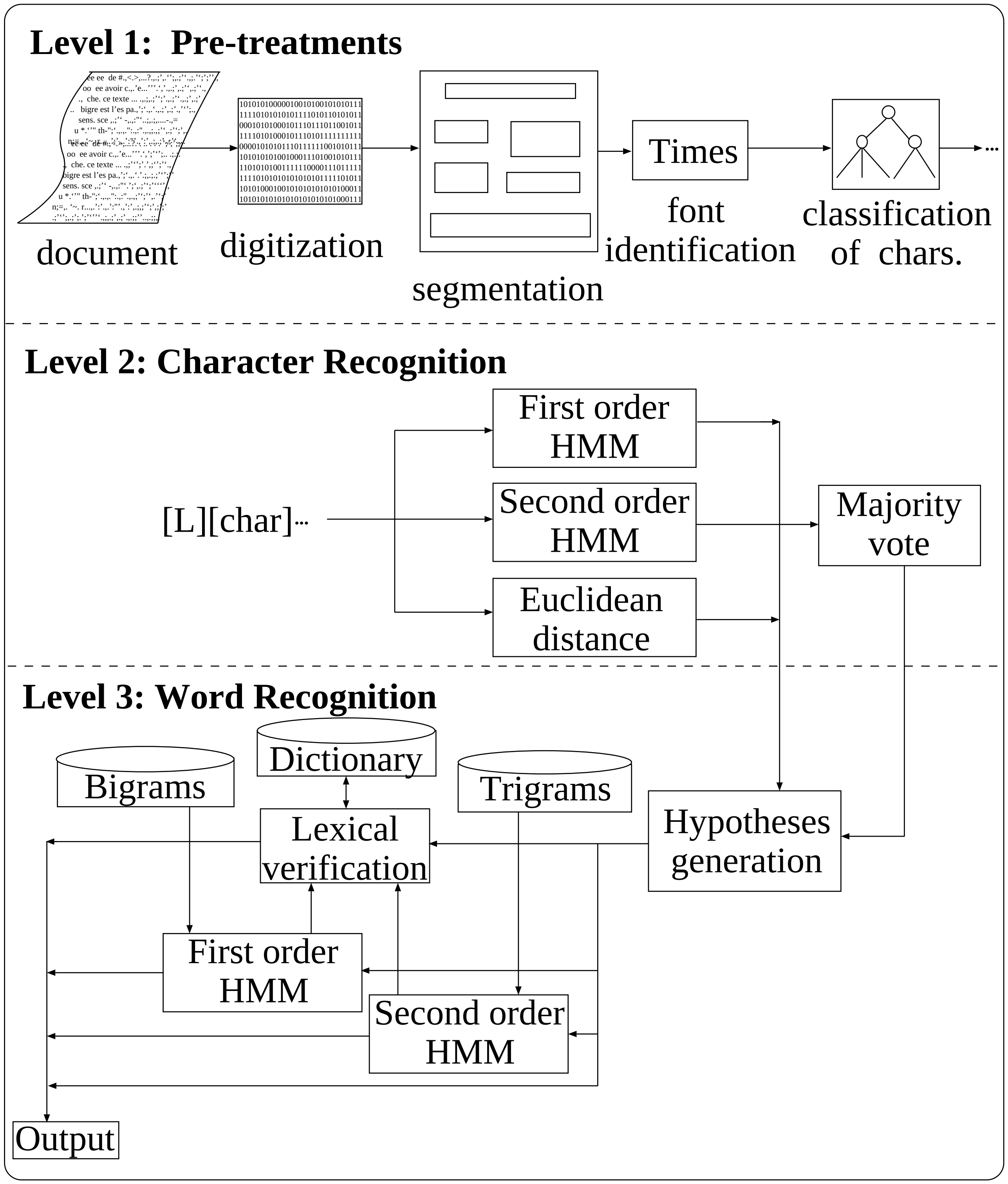
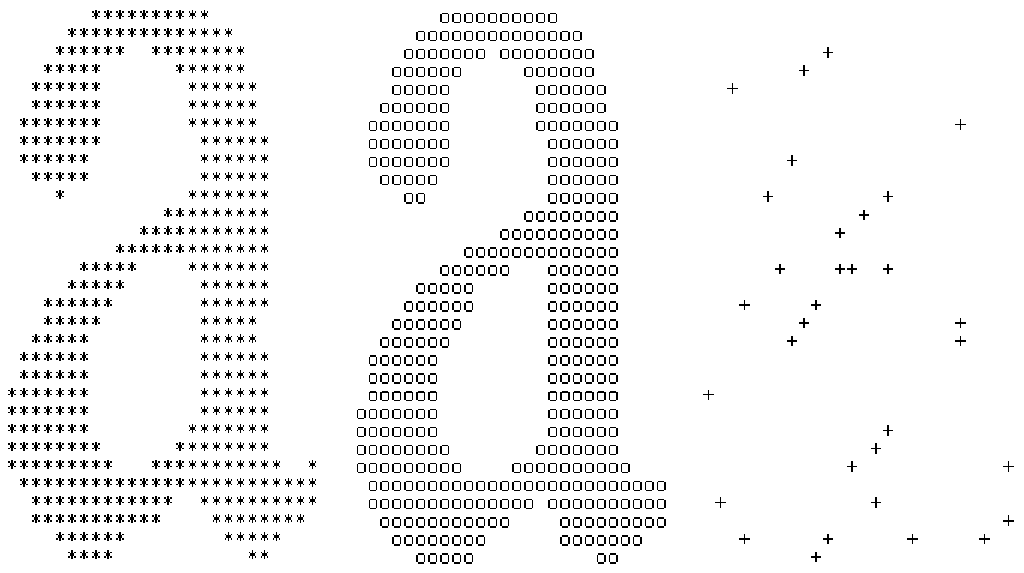


Figure 1: System Architecture.



```
cor = 0.921951
```

Figure 2: Comparison of two **a**s.

```
                 *           ooooooooo  oo          +++++++++  ++
        **    ******       ooooooooooooooo        +++++++++++++ +
     ******  ********     ooooooo oooooooo       +++++   +++     +
    ******* *********     ooooo      ooooo       ++   +++  +     ++
*********************    ooooo        oooo      ++   ++++ +++    ++
  *******************   ooooo         oooo    +     +++++++++    ++
    *********  ******   ooooo          ooo     +    ++++++++++   ++
    ********            ooooo          ooo     +++  +++++++  +   ++
    *******             oooooo         ooo     +++   +++++    +++
     ******             ooooooo                +++    +++
     ******             oooooooo               ++++    ++
     *****               oooooooo               +++     +
     *****               oooooooooo             +++     ++
     *****                ooooooooooo            ++     ++++
     ******                 ooooooooooo                 ++++++
     ******                  ooooooooooo           +     ++++++
     *****                    oooooooooooo         ++    ++++++++
     *****                      oooooooooo         ++++ +++++++++
     *****                       oooooooooo        +++++++++++++++
    ******                        ooooooooo        +++++ +++++++++
    ******                           oooooo       ++++++    ++++++
     ******             ooo          oooooo    +++++++++    ++++++
     ******             ooo           ooooo    +++ ++++++    +++++
    *******             ooo           ooooo    +++ ++++++    +++++
    *******             oooo           oooo    +++ ++++++     ++++
    *******             oooo          ooooo    +++ ++++++    +++++
    *******             ooooo         ooooo    +++  +++++    +++++
    ********            oooooo       ooooo     +++   ++++   +++++
  ***********           ooooooooo  ooooooo     +++      +++++++++
***************         oooooooooooooooo       +           ++++
                              ooooooooo       +++++++        +

                                cor = -0.048232
```

Figure 3: Comparison of **r** and **s**.

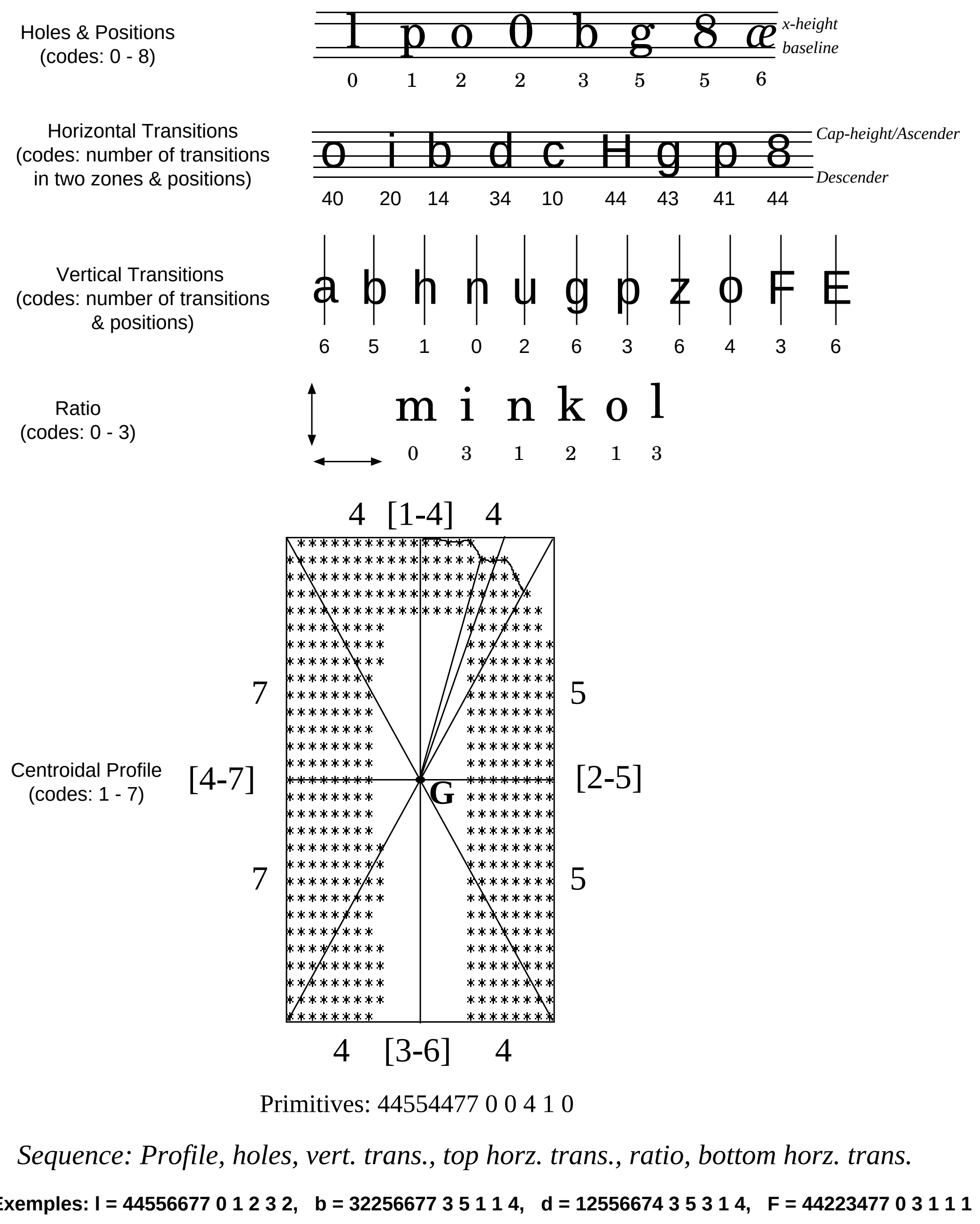
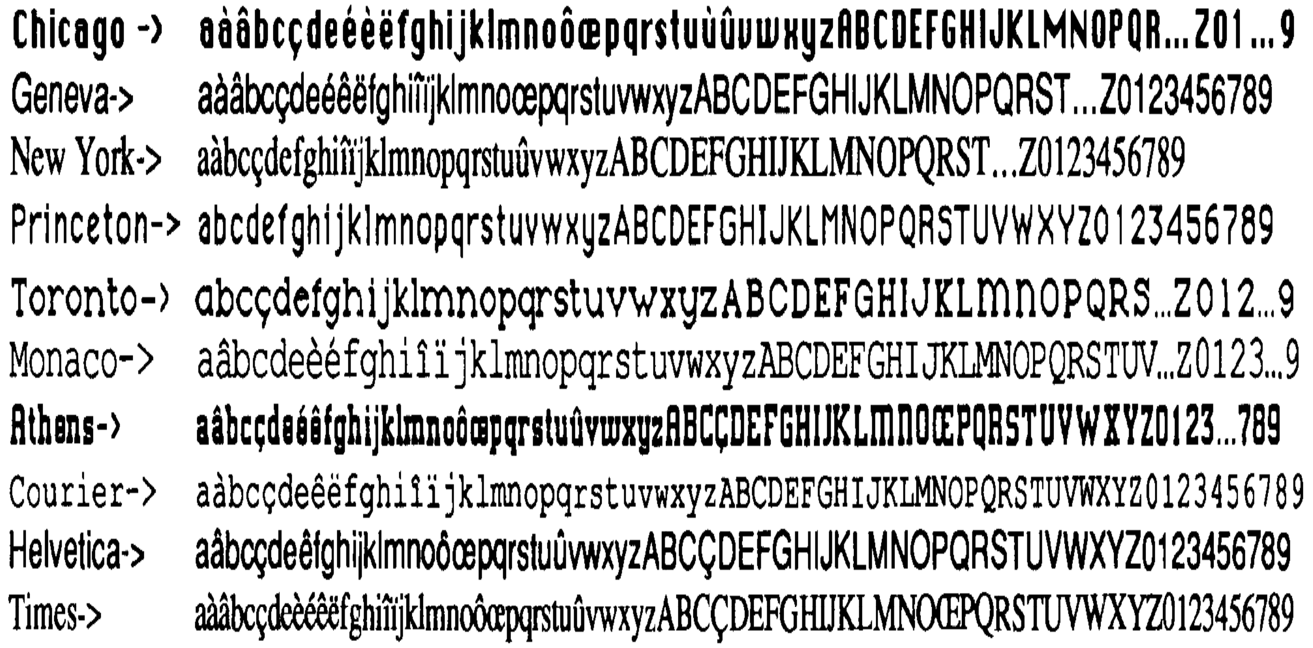


Figure 4: Features and coding methods.



Chicago -> aàâbcçdeéèëfghijklmnoôœpqrstuùûvwxyzABCDEFGHIJKLMNOPQR...Z01...9
Geneva-> aàâbcçdeéêëfghiîïjklmnoœpqrstuvwxyzABCDEFGHIJKLMNOPQRST...Z0123456789
New York-> aàbcçdefghiîïjklmnopqrstuûvwxyzABCDEFGHIJKLMNOPQRST...Z0123456789
Princeton-> abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789
Toronto-> abcçdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRS...Z012...9
Monaco-> aâbcdeèéfghiîïjklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUV...Z0123...9
Athens-> aâbcçdeéêfghijklmnoôœpqrstuûvwxyzABCÇDEFGHIJKLMNOŒPQRSTUVWXYZ0123...789
Courier-> aàbcçdeêëfghiîïjklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789
Helvetica-> aâbcçdeêfghiïjklmnoôœpqrstuûvwxyzABCÇDEFGHIJKLMNOPQRSTUVWXYZ0123456789
Times-> aàâbcçdeèéêëfghiîïjklmnoôœpqrstuûvwxyzABCÇDEFGHIJKLMNOŒPQRSTUVWXYZ0123456789

Figure 5: Characters in selected fonts.

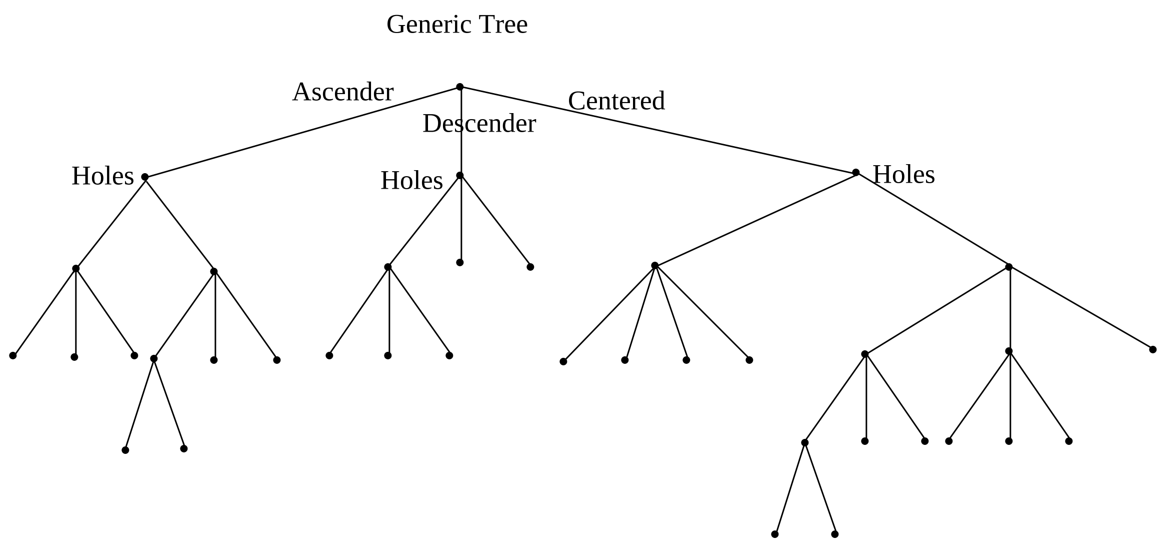


Figure 6: Generic tree.

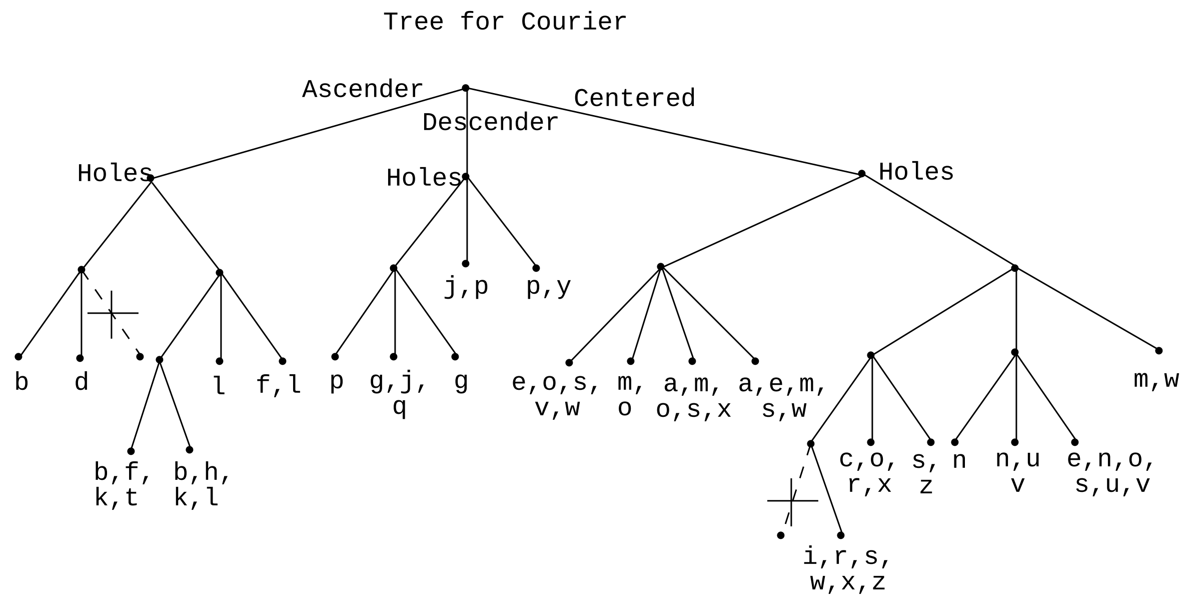


Figure 7: Instantiation for Courier.

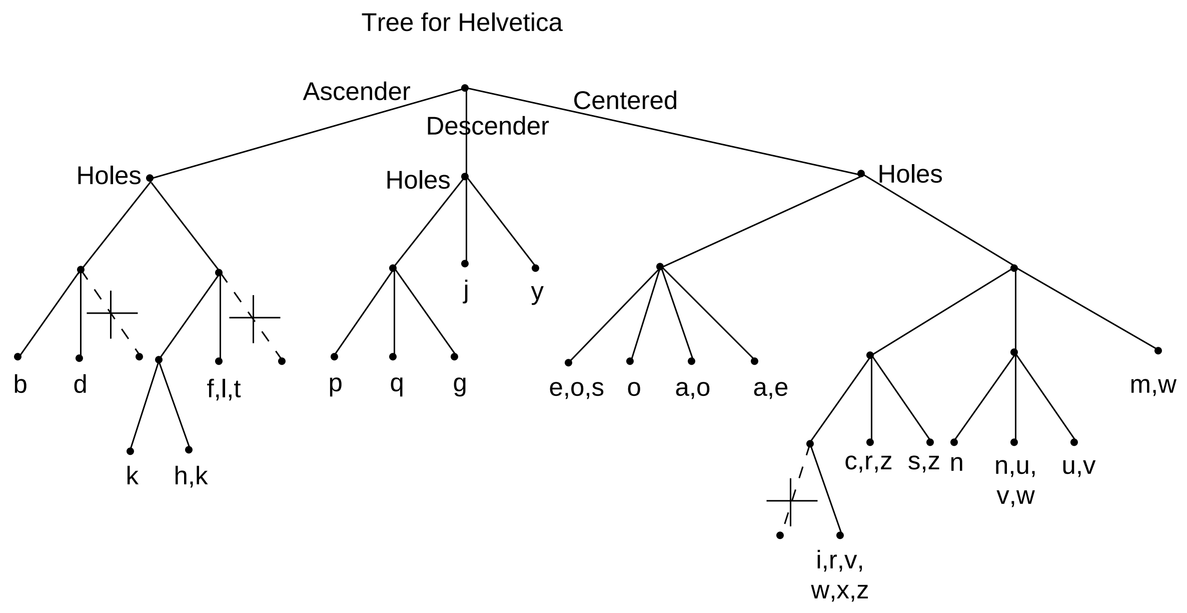


Figure 8: Instantiation for Helvetica.

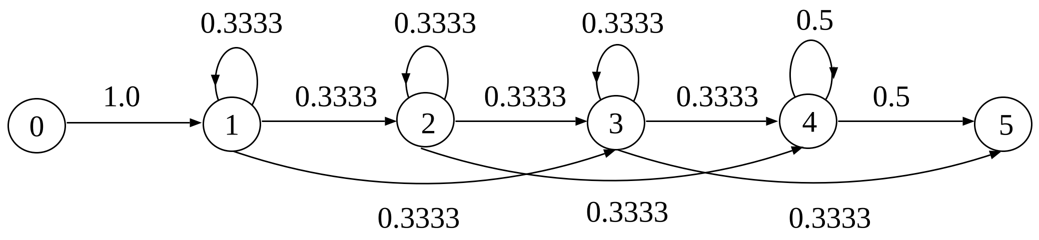
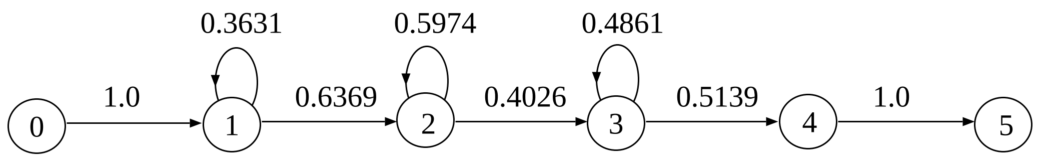


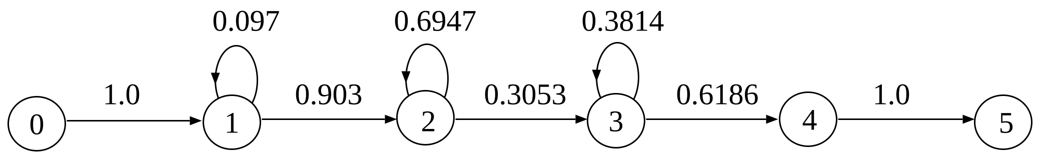
Figure 9: Initial transitions.

$$B = \begin{pmatrix} 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 \\ 0.111 & 0.111 & 0.111 & 0.111 & 0.111 & 0.111 & 0.111 & 0.111 & 0.111 \\ 0.111 & 0.111 & 0.111 & 0.111 & 0.111 & 0.111 & 0.111 & 0.111 & 0.111 \\ 0.111 & 0.111 & 0.111 & 0.111 & 0.111 & 0.111 & 0.111 & 0.111 & 0.111 \\ 0.111 & 0.111 & 0.111 & 0.111 & 0.111 & 0.111 & 0.111 & 0.111 & 0.111 \\ 0.111 & 0.111 & 0.111 & 0.111 & 0.111 & 0.111 & 0.111 & 0.111 & 0.111 \end{pmatrix}$$

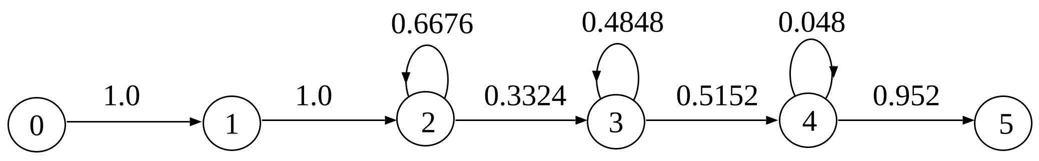
Table 1: Initial symbol distributions.



(a)



(a)



(g)

Figure 10: Final transitions for **a**, ***a*** and **g**.

$$B_{\mathbf{a}} = \begin{pmatrix} 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 \\ 0.001 & 0.001 & 0.200 & 0.783 & 0.011 & 0.001 & 0.001 & 0.001 & 0.001 \\ 0.001 & 0.001 & 0.159 & 0.001 & 0.588 & 0.247 & 0.001 & 0.001 & 0.001 \\ 0.001 & 0.001 & 0.001 & 0.001 & 0.001 & 0.026 & 0.967 & 0.001 & 0.001 \\ 0.001 & 0.001 & 0.001 & 0.001 & 0.001 & 0.001 & 0.037 & 0.956 & 0.001 \\ 0.001 & 0.001 & 0.001 & 0.001 & 0.001 & 0.513 & 0.480 & 0.001 & 0.001 \end{pmatrix}$$

$$B_{a} = \begin{pmatrix} 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 \\ 0.001 & 0.001 & 0.792 & 0.201 & 0.001 & 0.001 & 0.001 & 0.001 & 0.001 \\ 0.001 & 0.001 & 0.001 & 0.008 & 0.501 & 0.485 & 0.001 & 0.001 & 0.001 \\ 0.001 & 0.001 & 0.001 & 0.001 & 0.001 & 0.001 & 0.992 & 0.001 & 0.001 \\ 0.001 & 0.001 & 0.001 & 0.001 & 0.001 & 0.001 & 0.017 & 0.976 & 0.001 \\ 0.001 & 0.001 & 0.001 & 0.001 & 0.001 & 0.414 & 0.579 & 0.001 & 0.001 \end{pmatrix}$$

$$B_{\mathbf{g}} = \begin{pmatrix} 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 \\ 0.001 & 0.001 & 0.054 & 0.895 & 0.046 & 0.001 & 0.001 & 0.001 & 0.001 \\ 0.001 & 0.001 & 0.001 & 0.001 & 0.984 & 0.009 & 0.001 & 0.001 & 0.001 \\ 0.001 & 0.001 & 0.001 & 0.003 & 0.001 & 0.270 & 0.721 & 0.001 & 0.001 \\ 0.001 & 0.001 & 0.001 & 0.001 & 0.001 & 0.001 & 0.213 & 0.780 & 0.001 \\ 0.001 & 0.001 & 0.001 & 0.001 & 0.001 & 0.001 & 0.939 & 0.054 & 0.001 \end{pmatrix}$$

Table 2: Final symbol distributions for **a**, ***a*** and **g**.

Font	Number of states in each model									
	4		5		6		7		8	
	VS	WVS	VS	WVS	VS	WVS	VS	WVS	VS	WVS
Chicago	98.01	98.15	97.94	97.97	98.50	98.49	97.57	97.57	97.77	97.77
Geneva	98.35	98.04	99.12	99.12	99.52	99.64	99.83	99.96	99.83	99.96
New York	97.03	98.10	97.90	98.17	97.83	98.70	98.62	99.31	98.77	99.24
Princeton	99.11	99.60	99.50	99.62	99.61	99.69	99.62	99.62	99.62	99.65
Toronto	98.86	98.93	99.10	99.48	99.49	99.52	99.53	99.52	99.54	99.54
Monaco	94.76	95.54	97.22	97.35	97.48	97.53	97.65	97.83	97.57	97.98
Athens	96.30	95.79	96.01	95.90	96.14	96.43	96.47	96.69	96.85	97.02
Courier	93.49	93.54	95.13	96.19	96.12	96.67	96.44	96.66	96.92	97.18
Helvetica	97.58	97.90	98.18	98.37	99.59	99.60	99.57	99.60	99.50	99.57
Times	95.84	97.53	97.01	97.84	97.41	97.95	97.94	98.78	98.27	98.74
Average	96.93	97.31	97.71	98.00	98.17	98.42	98.32	98.55	98.46	98.67

Table 3: Top-1 performance.

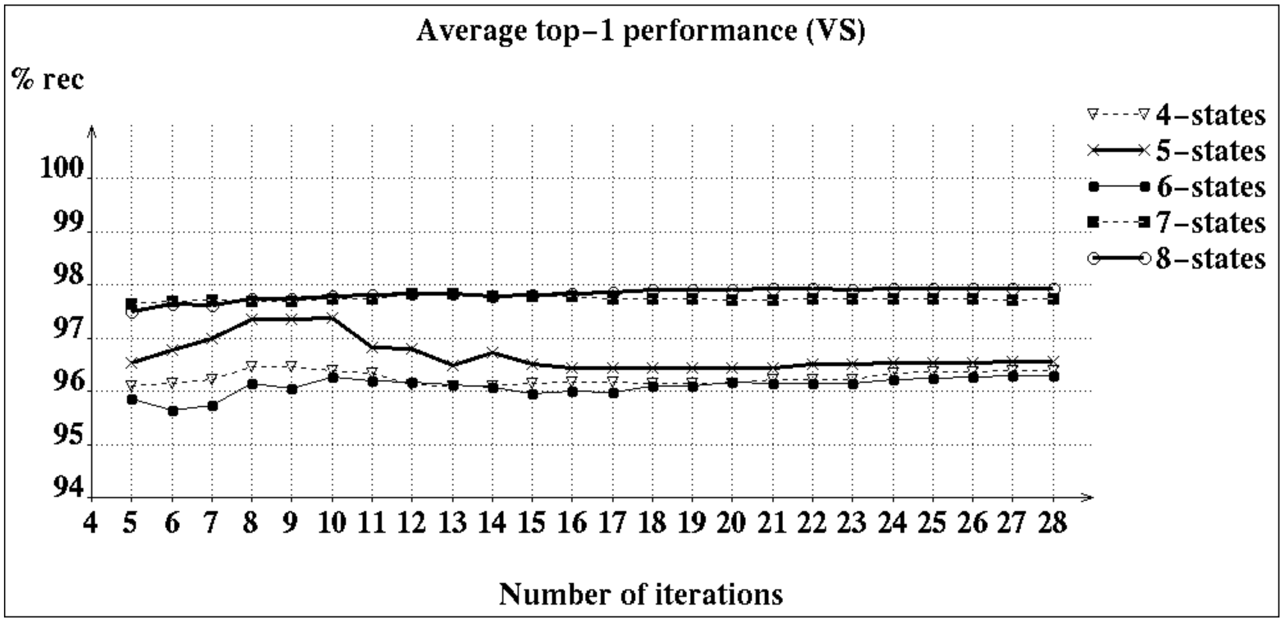
Font	Number of states in each model									
	4		5		6		7		8	
	VS	WVS	VS	WVS	VS	WVS	VS	WVS	VS	WVS
Chicago	99.69	99.69	99.65	99.65	100.00	99.98	99.71	99.71	99.99	99.99
Geneva	99.91	99.91	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
New York	99.29	99.35	99.68	99.71	99.80	99.81	99.98	99.99	99.99	99.99
Princeton	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
Toronto	99.76	100.00	99.73	100.00	99.99	100.00	100.00	100.00	100.00	100.00
Monaco	98.31	98.62	99.50	99.54	99.67	99.68	99.77	99.76	99.93	99.90
Athens	99.64	99.70	99.49	99.38	99.86	99.83	99.95	99.95	99.95	99.93
Courier	98.27	98.64	98.73	98.95	99.05	99.29	99.33	99.44	99.37	99.52
Helvetica	99.50	99.54	99.98	99.99	100.00	99.99	99.99	99.99	99.99	99.99
Times	99.48	99.73	99.77	99.92	99.95	99.95	99.98	99.97	99.99	99.99
Average	99.39	99.52	99.65	99.71	99.83	99.85	99.87	99.88	99.92	99.93

Table 4: Top-2 performance.

Font	Number of states in each model									
	4		5		6		7		8	
	VS	WVS	VS	WVS	VS	WVS	VS	WVS	VS	WVS
Chicago	100.00	99.98	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
Geneva	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
New York	99.99	99.99	99.99	100.00	100.00	100.00	100.00	100.00	100.00	100.00
Princeton	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
Toronto	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
Monaco	99.86	99.83	99.85	99.83	99.83	99.88	100.00	100.00	100.00	100.00
Athens	99.95	99.99	99.98	99.93	99.99	99.95	100.00	100.00	100.00	100.00
Courier	99.22	99.67	99.46	99.64	99.51	99.84	99.92	99.89	99.95	99.94
Helvetica	100.00	99.99	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
Times	99.87	99.95	99.99	99.99	100.00	99.99	100.00	100.00	100.00	100.00
Average	99.89	99.94	99.93	99.94	99.93	99.97	99.99	99.99	100.00	99.99

Table 5: Top-3 performance.

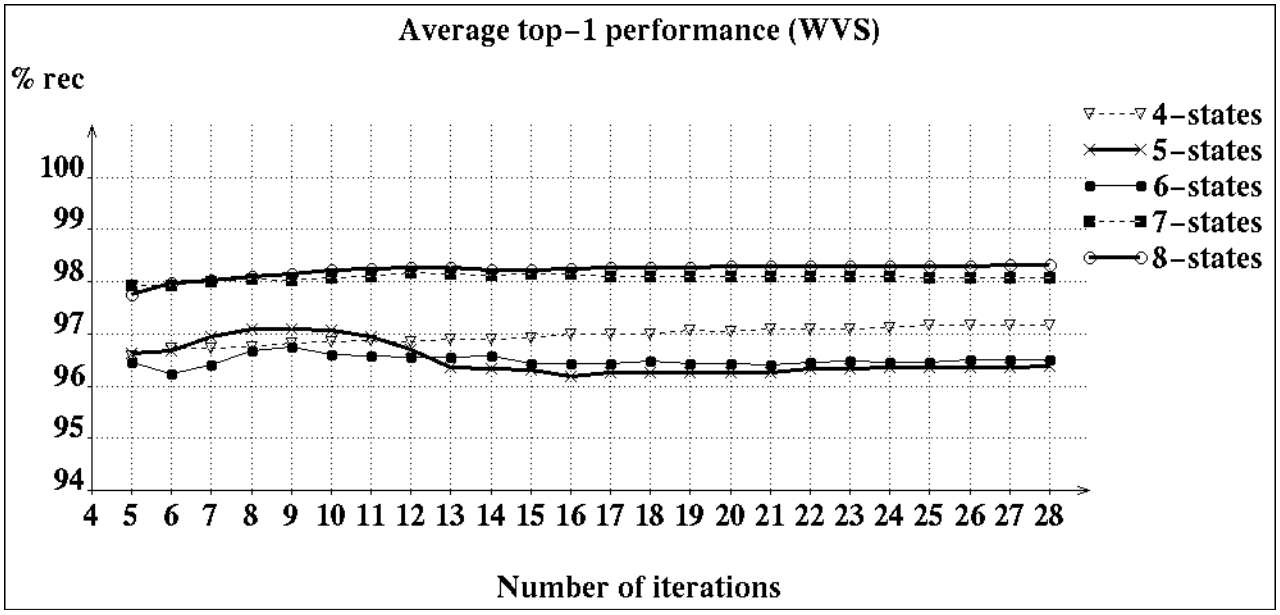
Figure 11: Recognition rate as a function of number of iterations.



Iterations	Number of states in each model									
	Helvetica					Times				
	4	5	6	7	8	4	5	6	7	8
5	97.67	97.45	99.25	99.56	99.39	95.34	96.30	96.87	97.63	98.05
6	97.67	97.74	99.36	99.58	99.52	95.84	96.50	97.07	97.69	97.65
7	97.67	97.92	99.45	99.61	99.52	95.86	96.55	97.18	97.80	98.18
8	97.40	98.11	99.52	99.56	99.54	95.62	96.85	97.26	97.86	98.24
9	97.57	98.07	99.58	99.58	99.50	95.53	96.83	97.10	97.96	98.23
10	97.58	98.18	99.59	99.57	99.50	95.84	97.01	97.41	97.94	98.27
11	97.55	98.18	99.59	99.58	99.50	95.86	96.98	97.76	97.89	98.27
12	97.55	98.21	99.58	99.58	99.50	95.84	96.99	97.78	97.90	98.28
13	97.59	98.21	99.46	99.54	99.50	95.93	97.39	97.72	97.30	98.28
14	97.63	97.58	99.49	99.48	99.50	96.16	97.41	97.72	97.25	98.28
15	97.64	97.59	99.49	99.48	99.50	96.16	97.35	97.64	97.26	98.28
16	97.64	97.87	99.49	99.48	99.50	96.16	97.25	97.62	97.26	98.28
17	97.47	98.20	99.49	99.48	99.50	96.01	97.34	97.62	97.33	98.28
18	97.48	98.30	99.49	99.48	99.50	95.98	97.34	97.63	97.33	98.28
19	97.47	98.38	99.49	99.48	99.50	95.95	97.33	97.62	97.34	98.28
20	97.43	98.51	99.49	99.48	99.50	95.91	97.33	97.72	97.37	98.28
21	97.43	98.52	99.49	99.48	99.50	95.95	97.35	97.72	97.37	98.28
22	97.39	98.55	99.49	99.48	99.50	95.94	97.35	97.72	97.36	98.28
23	97.39	98.51	99.49	99.48	99.50	95.95	97.37	97.71	97.36	98.28
24	97.40	98.51	99.49	99.48	99.50	95.99	97.37	97.69	97.36	98.28
25	97.40	98.51	99.49	99.48	99.50	95.96	97.38	97.69	97.36	98.28
26	97.40	98.51	99.49	99.48	99.50	95.96	97.38	97.69	97.36	98.28
27	97.39	98.56	99.49	99.48	99.50	95.96	97.39	97.69	97.36	98.28
28	97.39	98.56	99.49	99.48	99.50	95.96	97.43	97.69	97.36	98.28

Table 6: Recognition as a function of number of iterations.

Figure 12: Recognition rate as a function of number of iterations.



Font	Number of states in each model									
	4		5		6		7		8	
	VS	WVS	VS	WVS	VS	WVS	VS	WVS	VS	WVS
Chicago	98.16	98.16	95.93	95.93	97.41	97.49	97.49	97.49	98.23	98.23
Geneva	99.71	99.61	99.76	99.89	98.24	98.24	99.78	99.91	99.83	99.95
New York	97.86	98.34	97.43	98.51	98.29	98.80	98.90	99.11	98.34	98.90
Princeton	99.58	99.60	99.61	99.64	99.59	99.64	99.62	99.64	99.62	99.65
Toronto	99.27	99.34	98.91	98.93	99.48	99.48	99.52	99.52	99.52	99.52
Monaco	95.83	96.67	96.81	97.66	97.49	97.57	97.33	97.83	97.86	98.05
Athens	95.34	95.27	96.20	95.77	96.08	96.51	96.29	96.23	96.28	96.38
Courier	94.01	95.00	95.70	95.99	96.21	96.13	96.49	96.67	96.10	96.95
Helvetica	98.85	99.06	99.33	99.54	97.43	97.72	99.37	99.51	99.42	99.58
Times	96.60	97.75	97.47	97.93	97.76	98.46	97.94	98.70	97.74	98.63
Average	97.52	97.88	97.72	97.98	97.80	98.00	98.27	98.46	98.29	98.58

Table 7: Top-1 performance (2nd order).

Font	Number of states in each model									
	4		5		6		7		8	
	VS	WVS	VS	WVS	VS	WVS	VS	WVS	VS	WVS
Chicago	99.71	99.71	99.71	99.71	99.71	99.71	99.71	99.71	99.99	99.99
Geneva	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
New York	99.87	99.91	99.95	99.97	99.96	99.96	99.98	99.99	99.92	99.93
Princeton	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
Toronto	99.99	100.00	100.00	99.97	100.00	99.97	100.00	100.00	100.00	100.00
Monaco	99.54	99.58	99.76	99.85	99.83	99.81	99.97	99.97	99.97	99.94
Athens	99.73	99.71	99.68	99.64	99.73	99.68	99.94	99.93	99.94	99.93
Courier	98.66	99.08	99.13	99.33	99.43	99.44	99.60	99.58	99.46	99.58
Helvetica	100.00	100.00	99.97	99.97	99.93	99.93	99.99	99.99	99.98	99.98
Times	99.89	99.93	99.75	99.97	99.99	99.99	99.92	99.98	99.99	99.98
Average	99.74	99.79	99.79	99.84	99.86	99.85	99.91	99.91	99.92	99.93

Table 8: Top-2 performance (2nd order).

Font	Number of states in each model									
	4		5		6		7		8	
	VS	WVS	VS	WVS	VS	WVS	VS	WVS	VS	WVS
Chicago	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
Geneva	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
New York	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
Princeton	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
Toronto	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
Monaco	99.83	99.86	99.89	99.92	99.98	100.00	100.00	100.00	100.00	100.00
Athens	99.98	99.95	100.00	99.97	99.98	99.98	100.00	100.00	100.00	100.00
Courier	99.56	99.68	99.78	99.93	99.96	99.89	99.97	99.93	99.97	99.96
Helvetica	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
Times	99.98	100.00	99.99	99.99	100.00	100.00	100.00	100.00	100.00	100.00
Average	99.93	99.95	99.97	99.98	99.99	99.99	100.00	99.99	100.00	100.00

Table 9: Top-3 performance (2nd order).

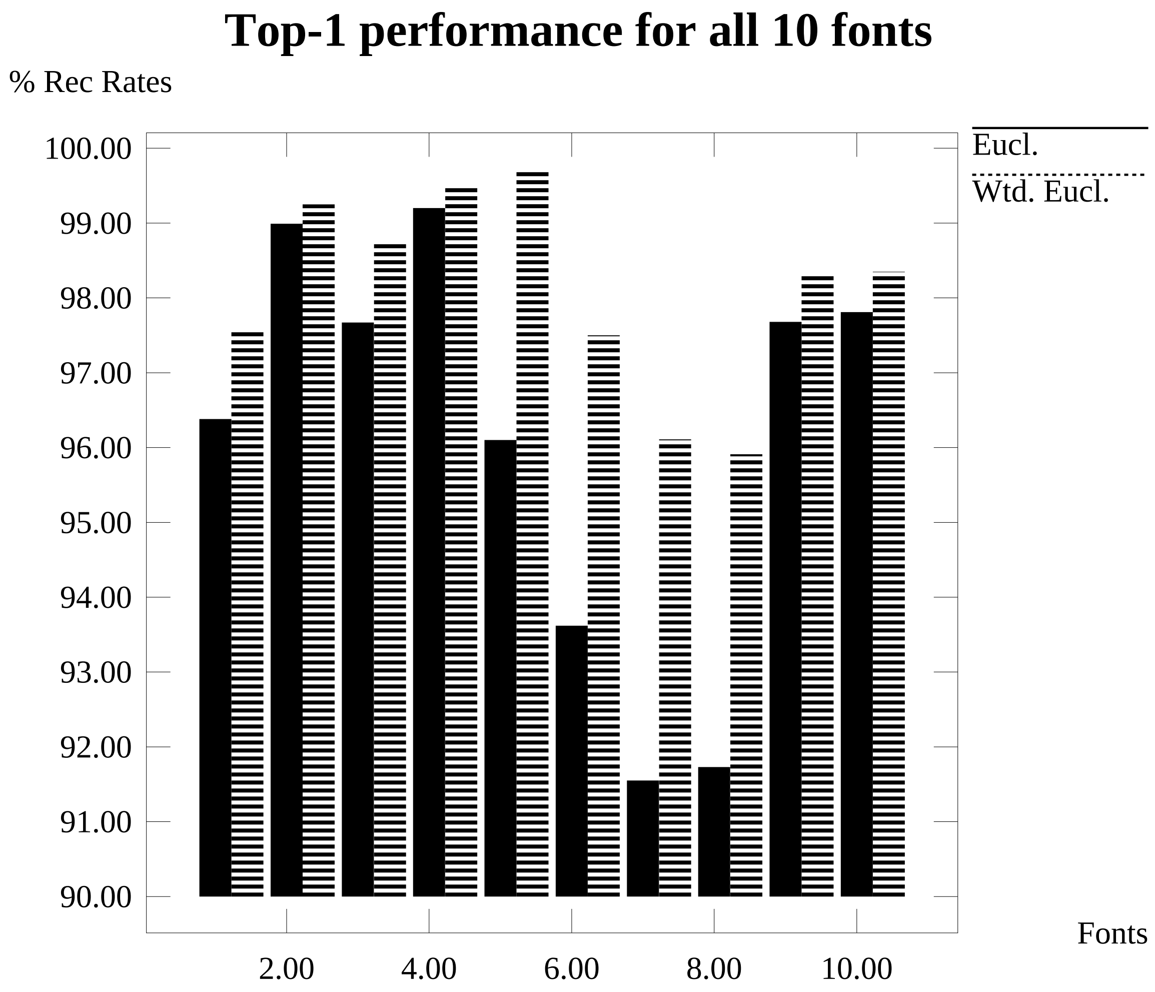


Figure 13: Performance of distance measures.

Font	Performance of the different methods						
	HMM1		HMM2		ED		MAJ
	VS	WVS	VS	WVS	ED	WED	MAJ
Chicago	97.77	97.77	98.17	98.17	96.38	97.54	98.18
Geneva	99.83	99.96	99.83	99.95	98.99	99.25	99.83
New York	98.77	99.25	98.64	99.13	97.67	98.72	98.62
Princeton	99.62	99.65	99.62	99.65	99.20	99.52	99.62
Toronto	99.54	99.54	99.51	99.51	96.10	99.72	99.52
Monaco	97.61	98.00	97.64	97.89	93.62	97.50	97.76
Athens	96.85	97.02	96.39	96.78	91.55	96.11	96.67
Courier	96.80	97.13	96.44	96.99	91.73	95.91	96.71
Helvetica	99.50	99.57	99.47	99.58	97.68	98.31	99.50
Times	98.28	98.75	97.94	98.87	97.81	98.35	98.14
Average	98.46	98.66	98.37	98.65	96.07	98.09	98.45

Table 10: Top-1 performance after 14 iterations.

States	Characters/sec		
	HMM1	HMM2	MAJ
4	610	147	63
5	445	83	37
6	336	50	23
7	263	33	15
8	212	23	11

Table 11: Speed versus number of states.

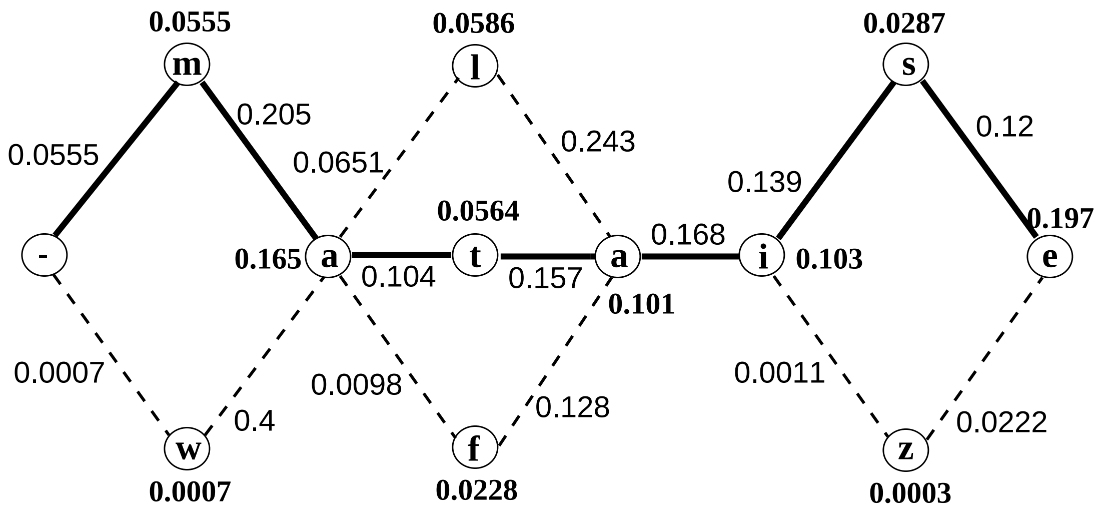


Figure 14: Viterbi Net for a 7-letter output sequence.

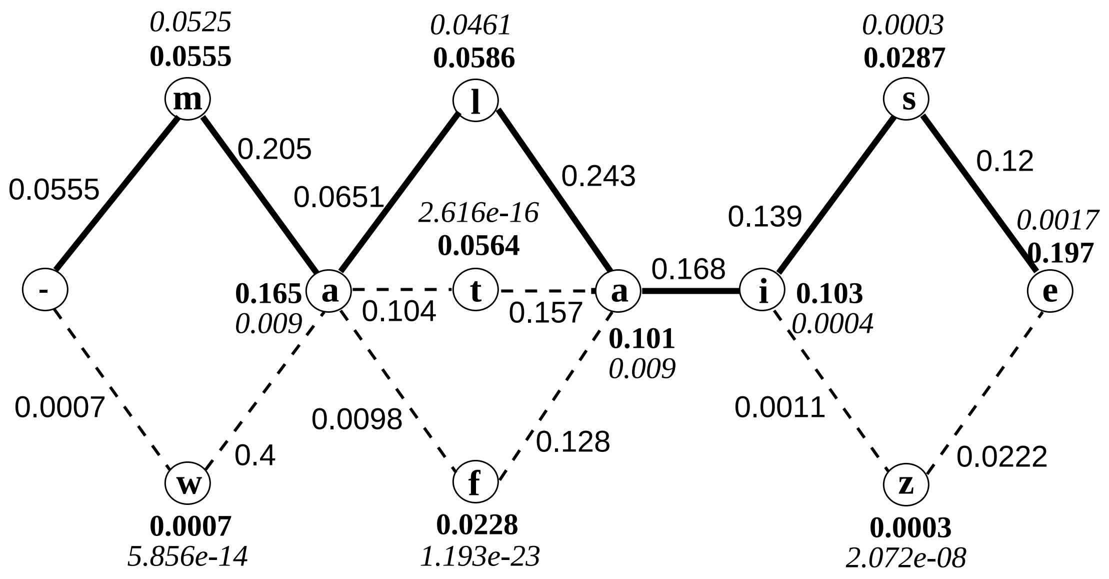


Figure 15: Weighted Viterbi Net for a 7-letter output sequence.

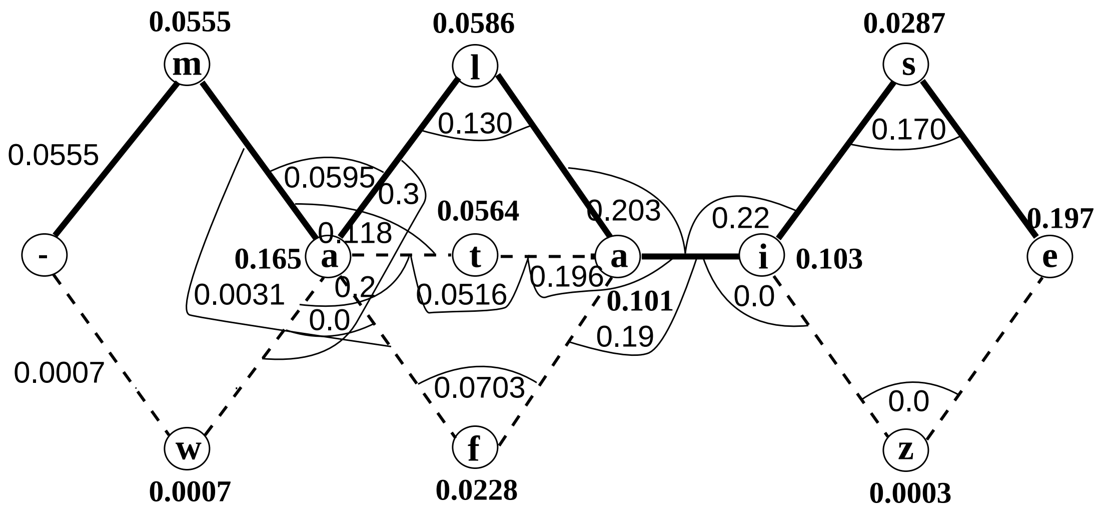


Figure 16: Viterbi Net based on trigrams.

Word	Performance of three word-correction methods					
	First Order HMM		Weighted First Order HMM		Second Order HMM	
	Probability	Choice	Probability	Choice	Probability	Choice
This	1.393e-11	fhis	9.648e-25	This	6.555e-12	This
paper	5.628e-11	paper	5.113e-23	paper	1.607e-11	paper
exposes	4.607e-17	exposes	1.054e-40	exposes	6.252e-16	exposes
stochastic	2.48e-24	stochastis	3.072e-52	stochastic	2.826e-24	stochastic
modeling	9.005e-18	modeling	6.465e-33	modeling	2.5e-16	modeling
recognition	1.255e-25	recognition	3.385e-46	recognition	2.096e-21	recognition
hidden	5.36e-15	hidden	1.376e-26	hidden	2.438e-14	hidden
script	2.813e-16	scripl	1.138e-35	script	7.551e-16	script
numerous	4.07e-21	numerous	2.099e-43	numerous	4.425e-19	numerous
applications	2.59e-22	apptications	1.225e-50	applications	8.898e-20	applizations
such	2.058e-11	suck	2.733e-24	such	4.487e-11	suck
character	2.04e-19	character	4.317e-44	character	9.141e-20	character
based	2.603e-11	bazed	6.449e-26	based	1.191e-11	based
the	2.635e-8	the	2.029e-18	the	8.343e-09	the

Table 12: Word recognition results.