# ON-LINE UNCONSTRAINED HANDWRITING RECOGNITION BASED ON PROBABILISTIC TECHNIQUES

**Homayoon S.M. Beigi**, **Krishna Nathan**,
and **Jayashree Subrahmonia**

IBM T.J. Watson Research Center
P.O. Box 704
Yorktown Heights, New York 10598 U.S.A.
Phone: +1 (914) 784-6626 Fax: +1 (914) 784-7007
EMail: beigi@watson.ibm.com

## Abstract

This paper discusses a probabilistic on-line handwriting recognition scheme, based on Hidden Markov Models (HMM's), and its implementation for recognizing handwritten words captured from a tablet. Statistical methods, such as HMM's have been used successfully for speech recognition. These methods have recently been applied to the problem of handwriting recognition as well. This paper, discusses a general recognition system for large vocabulary, writer-independent, unconstrained handwritten text. A key characteristic of the recognition system described here is its *real-time* performance on Intel 80486 class PC platforms without the large memory requirements of traditional HMM-based systems. A word-error rate of 18.9% is achieved for a writer-independent 21,000 word vocabulary task in the absence of any language model. This recognizer is modified to handle digits written in Persian or Arabic, producing error rates less than 7%. The grounds for developing a full-scale Persian recognition system are also set.

## Keywords

Cursive, Unconstrained, Handwriting, Recognition, Hidden Markov Models, Digit, Persian, Arabic

# 1 Introduction

Recently, a lot of attention has been given to producing pen-computers which rely on the pen as their primary human interface. A few of these machines have recently been introduced into the market-place by major companies. Among these computers some are intended to be more of Personal Digital Assistants (PDA's) while others are actual note-books computers which have an additional pen-interface. Apple Newton, Canon AINote, Epson Cyber Media, and Sony Palmtop are examples of PDA's. Full-functioned pen-computers such as IBM ThinkPad 750P and 360P, and GRID are also available.†

A probabilistic on-line unconstrained handwriting recognition scheme, based on Hidden Markov Models (HMM's), and its implementation are presented. Words may be written naturally and without constraints on a digitizer tablet. The motion of the tip of the stylus (pen) is sampled at equal time intervals using a digitizer tablet and the sampled points are passed to a computer which performs the handwriting recognition. The data signal undergoes some filtration, it is normalized to a standard size, and its slant and slope are corrected. After normalization, the writing is segmented into basic units and each segment is classified and labeled. Using a search algorithm in the context of a word list, the most likely path is returned to the user as the intended string hypothesis (see Figure 1).

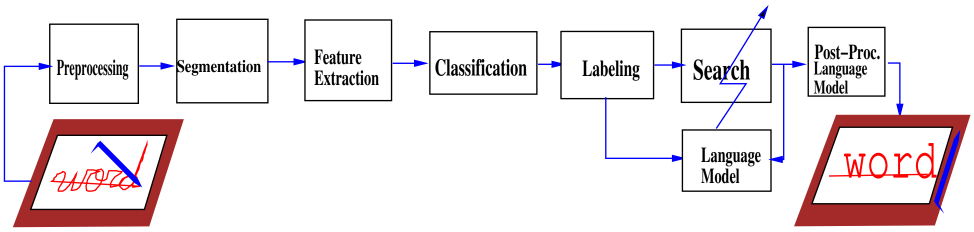


Figure 1: Generic On-Line Handwriting Recognition Process

Earlier techniques of on-line handwriting recognition could not handle naturally written handwriting, since they assumed a pen-lift between letters. [1, 2, 3] In this restriction, input units (strokes) were considered to be sub-units of letters. Thus, a collection of letters could not be represented by a single stroke. In the literature, this is referred to as **discrete handwriting recognition**. The recognition method discussed here handles connected writing as well as discrete handwriting, hence called **unconstrained handwriting recognition** (see Figure 2). In this case the only imposed restriction is the need for a sufficient gap between adjacent words on a common base-line to aid the recognition engine in conducting word segmentation. Results from this type of recognition are more readily extended to the recognition of Persian (Farsi) handwriting which is unconstrained by nature (a mixture of discrete and cursive). There are several ways of solving the unconstrained recognition problem. One approach is to create a template (model) for each word. In this strategy, the input corresponding to a word is compared to individual word models. This is a *simple* but *impractical* strategy which does not require any character or sub-character alignment or segmentation. However, it requires developing several models for each word, covering many possible writing styles. This approach could be very expensive for a large-vocabulary implementation and therefore it is not practically acceptable. In contrast to the above method, the objective is to be able to handle large vocabulary recognition and to have fewer, more general models. After undergoing some pre-processing which will be discussed in the next section, our basic prototypes are constructed from sub-character units. These prototypes are shared by different character segments. Therefore, a sub-character segmentation is performed which is discussed in a later section. Certain features are then extracted from each segment. From these features, a probability is computed for membership of that particular segment to different character classes (lexemes). A search algorithm is then used to come up with the most likely hypotheses given these sub-character membership probabilities (referred to as fast-match probabilities). These word hypotheses are then once more refined, using more detailed,

†These are trademarks of their corresponding corporations

multi-state, Hidden Markov Models. The hypotheses are then sorted such that the word hypothesis with the highest detailed-match probability would be placed on top of the list. Each of theses steps is briefly described in the following sections.


Figure 2: Samples of Different Types of Handwriting

Later, an experiment is presented in extending this recognition system to handle Persian handwriting. A Persian digit recognizer is implemented and tested, based on the fast-match techniques presented in this paper. (N.B., In this paper, by Persian, we mean to include similarly written styles such as Arabic, Ordu, Pashtu, Maghrebi, Uigur, etc.) The grounds for the development of a full-scale Persian handwriting recognizer are also set.

# 2 Pre-Processing

Most experiments reveal that sampling frequencies of 100Hz are well sufficient for obtaining good recognition results. For a good survey of digitizer technology please see [1]. All the data discussed in this paper was, therefore, collected as a stream of $(x, y)$ points indexed in time, sampled at rates between 70Hz and 100Hz. This section describes the pre-processing that the sampled points undergo in the first few stages of the recognition process. This includes smoothing and filtering, size normalization, and slope correction.


Figure 3: Principal lines of a word

## 2.1 Smoothing and Filtering

Most digitizer tablets have built-in low-pass filters in hardware form which take away the jagged nature of the handwriting signal. These filters in some cases generate more problem than they solve. For example, smoothing of the data at the hardware level could sometimes smooth away natural cusps and corners which are very important in recognizing certain characters such as $v$'s and $s$'s from their counterparts $u$'s and $r$'s. It is sometimes more desirable to do minimal filtering at the hardware level and to leave most of the filtering at the discretion of the recognition algorithm designer.

## 2.2 Size Normalization

Please refer to Figure 3. For size normalization, the base-line (line 1) and the mid-line (line 2) need to be estimated. The area surrounded by the base-line and the mid-line is the only non-empty part of any word. This is the most reliable portion of the data for usage in size normalization. Once accurate estimates of the base-line and the mid-line are obtained, a magnification factor may be computed from the ratio of the desired mid-portion size and that estimated from the input. The entire input data may then be scaled using this magnification factor. Furthermore, it is important to use the information provided by lines 0 and 3 of Figure 3. These lines are specially useful in cases where the whole word is either totally made up of upper-case letters or lower-case letters with no ascenders or descenders – such as $\{a, c, e, o, \cdots\}$. The relative positions of line 0 versus line 1 and line 2 versus line 3 inform us of the presence of descenders or ascenders respectively. Additional treatment should be given to special cases to achieve a high accuracy.

### Principal Line Estimation

Figure 3 shows the words "principal lines" written in English and Persian (Farsi). Both these scripts and those of similarly written languages (German, French, Spanish, Italian,

..., and Arabic, Ordu, ...) possess common features related to the method of size normalization which is used here. These techniques are, therefore, readily applicable to many other languages. Japanese, Chinese, Korean, and Hindi are among languages for which a different algorithm should be employed to determine corresponding principal lines (see Figure 4). For these languages, the entire height of the writing is basically fixed and the mean total height of words may be used to determine the normalization factor (i.e., only lines 0 and 3 are meaningful). Due to the nature of techniques used here, the basic normalization scheme is also applicable to OCR (Optical Character Recognition) without any changes. Please refer to [4] for details of the normalization algorithm.

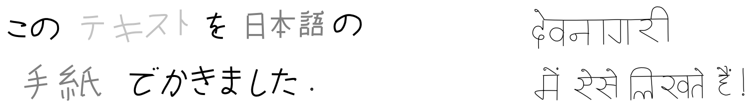


Figure 4: Samples of Japanese [left] and Hindi (Devanagari) [right] Writing

### 2.3 Slope Correction

The slope correction algorithm used here is based on evaluation of the mean velocities in the $x$ and $y$ directions. The angle between these velocity vectors is used to estimate the angle by which the words should be rotated. This rotation on most occasions re-orients the writing into a horizontal left to right flow. For special cases such as in words with all block capital letters, special provisions should be taken in estimating the rotation angle. [4]

## 3 Segmentation and Feature Extraction

Once the writing is normalized, the points are interpolated and then decimated such that the new stroke will have equidistant points in the Euclidean space. This removes inconsistencies due to velocity variations inherent to the different writer-independent samples. Geared toward the recognition of unconstrained handwriting, in this recognizer, segments are defined to be portions of each character and they are generated based on points of extreme velocities (minimum $x$ and $y$ velocities). These segments are then sent to the core of the recognition engine for feature extraction and labeling or hypothesis generation. Centers of these segments are given by the above segmentation process and a window of equal number of equidistant points is associated with each segment. This produces a number of overlapping windows each having the same number of equidistant points.

The actual features are the differences between adjacent coordinates ($\Delta x$ and $\Delta y$), the sine and cosine of the angle that the tangent line makes with the stroke path at each point, and the absolute $y$-coordinate of each point offseted by the computed baseline value from figure 3. Thus, there is a five dimensional feature vector which is associated with each of the filtered points. Since the segments are picked to have the same number of points, an augmentation of the local feature vectors will produce same-length feature vectors associated with each window. Therefore, each of the overlapping windows has a long vector (of dimension 5x# *of points in window*) associated with it.

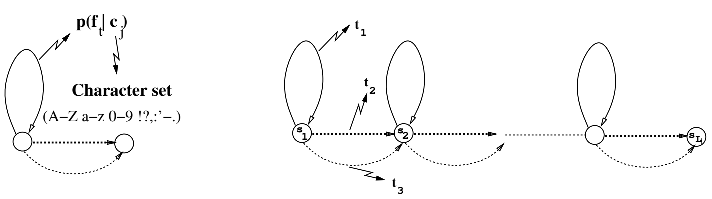


Figure 5: Single state model for a character [left] Multiple-State Model [right]

These long vectors are later projected onto a lower dimensional space given by the principal components of all the data gathered in the training process. These lower ($n$) dimensional vectors are the feature vectors used in the recognition process (frames). For the training data, the character label associated with each window is known. These labeled $n$-dimensional feature vectors are clustered in the $n$-dimensional space and each cluster is modeled by an $n$-dimensional Gaussian distribution. The means and covariances of these Gaussians are stored in the training process to be used later as prototypes for decoding.

# 4 Modeling

## 4.1 Fast-Match Model

In order to reduce computation, a degenerate single-state model is used to generate a short list of candidate hypotheses. Figure 5 shows the topology for the single-state model. The feature extraction yields a sequence of frames, $f_1, f_2, \ldots$ corresponding to the handwritten text. A Gaussian mixture model determines the probability that a particular character $c_j$ gives rise to a frame $f_t$ viz. $p(f_t|c_j) = \sum_i p(f_t|g_i)p(g_i|c_j)$. $g_i$ is a Gaussian with a diagonal covariance matrix. The distributions $g_i$ are obtained during the training phase by un-supervised clustering of all the frames. Also during training, the mixture coefficients $p(g_i|c_j)$ are estimated via the EM algorithm. During decoding, each frame is assumed to be independent of all the others and the probability that a particular character gives rise to a set of frames is merely the product of the individual $p(f_t|c_j)$ that compose that block of frames. Note that this simplified model does not provide duration modeling. In addition, there is no distinction between the beginning of a character and its end. A single output distribution characterizes the entire character. The notion of sub-models for states or sub-divisions of a character is introduced in the detailed-match, discussed next. Note that there is no notion of character segmentation prior to recognition. The segmentation arises as a natural consequence of recognition using a search algorithm.

## 4.2 Detailed-Match Model

The fast-match model, described above, can be shown to be equivalent to a single state HMM; there is no notion of relative position of frames within a character. In the detailed-match model this state degeneracy is removed and each character is modeled by a series of states, each of which has associated with it an output distribution corresponding to the portion of the character that it models. Figure 5 shows the HMM for any character $i$. The HMM has $L_i$ states labeled $s_1, s_2, \ldots, s_{L_i}$, where $L_i$ is the average number of frames for character $i$. Associated with each of the $L_i$ states, $s_1, s_2, \ldots, s_{L_i}$, is a set of three transitions labeled $t_1, t_2,$ *and* $t_3$. Transitions $t_1$ and $t_2$ result in the emission of an observation feature vector. The number of states $L_i$, the state transition probabilities, $p(s_i, t_j)$ and the output probability distributions $p(f_t \mid s_i, t_j)$ completely specify the model. The transitions $t_1$ and $t_2$ for a given state are tied, i.e., $p(f_t \mid s_i, t_1) = p(f_t \mid s_i, t_2)$. Hence the output distribution is associated with the state alone and can be written as $p(f_t \mid s_i)$. The output probabilities are determined from a mixture of tied Gaussian distributions, and can be written as, $p(f_t \mid s_i) = \sum_k p(f_t \mid g_k)p(g_k \mid s_i)$, where the summation is over the entire pool of distributions. Hence, the HMM is completely specified by the state-transition probabilities, $p(s_i, t_j)$, and the mixture coefficients, $p(g_k \mid s_i)$.

The HMM's are initialized from the fast-match models by replicating the single-state model $L_i$ times. The HMM's are trained using isolated characters and words written in an unconstrained manner using either Viterbi or forward-backward training. In our system, there is no significant difference in accuracy between the two training schemes. In decoding, the probability of a set of frames given a character $i$ is given by the probability of the most probable state sequence that could generate those frames. The optimal character sequence that make up the word is determined by a time synchronous beam search.

# 5 Beam Search

The 20,000+ word lexicon is stored in a structure that merges common prefixes and suffixes of the words. Since the search is lexicon driven, only those paths corresponding to valid words in the vocabulary are expanded. Associated with each frame is a stack containing all possible partial paths ending at that frame. The stacks are sorted by probability.

Naturally, thresholds are used to prune out low probability elements. The top element of the final stack corresponds to the recognized string. The search space is made more tractable by the introduction of constraints that limit the number of expanded nodes. One such constraint is a character-length distribution. These length histograms specify the range of the number of frames that a given character may span and are generated during training. These distributions are used in the fast-match stage since the single-state model has no way of modeling duration or the number of frames. In the context of the multi-state model, the external length distribution is not used since the length and state transition probabilities of individual models define duration. Delayed strokes pose a problem when using left to right HMM's to model characters. Examples of delayed strokes are the dots in the characters "i" and "j", and crosses in the characters "x" and "t". These are strokes that are often temporally separated from the body of the character. In most cases, these are added after the whole word is written. The data points for these characters are not necessarily contiguous in time, thus posing a problem when using left to right models for these characters. Therefore, we train the HMM on only the non-delayed strokes for these characters. Delayed strokes are stripped off before training the HMM. In decoding, the search mechanism first expands the non-delayed strokes based on the frame probabilities and the character models. The delayed strokes are then incorporated based on their position relative to the non-delayed strokes and their fast-match probabilities.

# 6 Experiments and Discussion

## 6.1 Data Sets and Training

Since we were primarily interested in the writer-independent performance of the recognizer, our first task was to collect data from a sufficiently large pool of writers. Approximately 100,000 characters of data were collected from a pool of 100 writers. The training set consisted of words chosen from a 20,000+ word lexicon and discrete characters written in isolation. The subjects were asked to write in their natural style and were encouraged to write on a horizontal line. *No* other instructions or directions pertaining to writing style were given. The test set was composed of data from a separate set of 25 writers collected in the same manner. This data words chosen at *random* from the same lexicon. Both native and non-native writers were included in both test and training sets. The data was collected on convertible IBM pen notebook computers. As expected, the data fell into three broad categories: purely discrete, mixed discrete and cursive, and purely cursive. Some examples are shown in Figure 6. As apparent, there is a wide range in the 'quality' or human readability of the data. The alphabet consisted of upper and lower case characters, numbers and a few punctuation symbols and special characters.

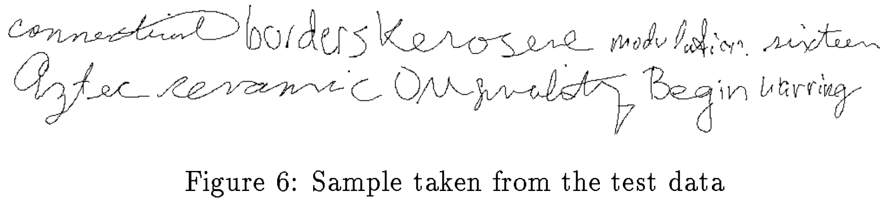

Figure 6: Sample taken from the test data

In order to capture the writing styles across a broad range of writers we chose to build models not for each character, but for each significant variation of each character. For example, a character may differ in the number of pen strokes, direction of pen movement or in actual shape itself. An automatic un-supervised procedure is used to identify these variations which we call lexemes. For this set, approximately 150 lexemes were generated. Individual baseforms, single and multiple state, are trained for each lexeme. Since on average each HMM consists of 6 states, this would result in 900 distinct states. In order to reduce the system parameters, individual states are shared across and within baseforms.

## 6.2 Pre-Processing

Significant word-error rate reductions of 54.3% and 35.8% were obtained for writer-dependent and writer-independent samples through using the normalization scheme of [4].

Desloping plays a big role in the general case as a pre-processor to size normalization. Even if the size is optimal, slope correction by itself is an important factor.

### 6.3 Writer-Independent Tests

Casual users of handwriting recognition are unwilling to invest the time and effort required to train user-specific systems. It is also easy to envision scenarios where training is not an option (e.g. public kiosks, points of sale etc.). This was our motivation for concentrating on the writer-independent task. Table 1 summarizes results for various vocabulary sizes. The detailed-match results were obtained by taking the top few hypotheses from the fast-match and presenting them to the multiple-state model. The fast-match effectively acts as a pruner. Since the test words were randomly selected we do not make use of a grammar to improve recognition performance by reducing the perplexity. Hence, the perplexity of the large vocabulary task reported is over 21,000. For the small vocabulary task, we obtain a writer-independent word-error rate of under 9%. As expected, this increases with the size of the lexicon (or perplexity). The error rate for the large vocabulary task is about 19%. We expect this number to decrease significantly if the recognizer were used in conjunction with statistical language models suited for the task. The recognition times per word range from 0.4 sec. for the small vocabulary task to 0.48 sec. for the large vocabulary task on an IBM RS6000 workstation. On standard 486 based PC's, we observe recognition times of 4 to 5 times these figures (still sufficient for real-time recognition).

| | **Small Vocab. (3K)** | **Medium Vocab. (12K)** | **Large Vocab. (21K)** |
|---|---|---|---|
| **Fast-Match** | 14.9% | 25.1% | 27.9% |
| **Detailed-Match** | 9.0% | 14.9% | 18.9% |

Table 1 : Writer-Independent word error rates (No grammar)

### 6.4 Persian Digits

The fast-match model of the system was trained on 600 samples of each digit written by native Iranian and Arab writers. The training data was generated by 20 different writers on a ThinkPad 750P$^{TM}$ running the OS2 with Pen Extension$^{TM}$. The system was tested by 14 writers who each wrote every digit five times. This produced an average accuracy of 93.1% for these writers (see Figure 8). All writers were able to get 100% accuracy once they got used to the system and observed some of the errors it was making.

## 7 A Full-Scale Persian Recognizer

Unfortunately, there has not been much work done on on-line Persian or Ordu recognition. Some minor work has been done on on-line Arabic recognition. [5] To build a recognizer for these languages, lots of effort is needed. Persian and other similarly written languages are unconstrained by nature (Figure 3). This means that they are the most difficult styles to recognize. A feature of these languages which makes them even harder than English and other Latin style writings is the fact that dots carry a lot of information and based on the number of dots put under or above a basic structure, it could turn into different characters with totally different sounds (Figure 7). Another basic problem is that vowels are not written in these languages. This reduces the average length of words and in turn creates more confusion and less context to help alleviate these confusions. A sentence-level language model is, therefore, essential for recognizing these forms of writing.

## 8 Conclusion

We have developed an HMM-based system for writer-independent handwriting recognition. The writing may be unconstrained (any mixture of print or cursive styles). The system achieves a word-error rate of 19% on a 21,000 word vocabulary task with no grammar (perplexity of 21,000). In conjunction with statistical language models that reduce the perplexity of the task, the error rate is expected to decrease significantly. Equally importantly, the recognition is performed in real-time on standard PC platforms.

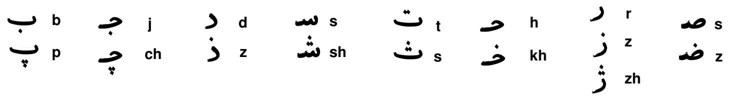


Figure 7: Some Confusing Characters in Persian and Their Sounds

Handwriting recognition allows more efficient drafting and document generation as well as applications such as form filling and keyboardless interface with a computer. In desktop systems, the pen could be a very important complement to the keyboard for editing, marking, drawing, etc. As the handwriting recognition technology becomes more mature, applications such as longhand note-taking in the class-room are going to be closer to reality. Professionals could write their documents and have them converted to text instantly without having to go through a few iterations of having their secretary type the document for them. This is specially useful to countries such as Iran where almost all office transactions are done through handwriting and very little training would therefore be necessary for using pen-computers in those environments.

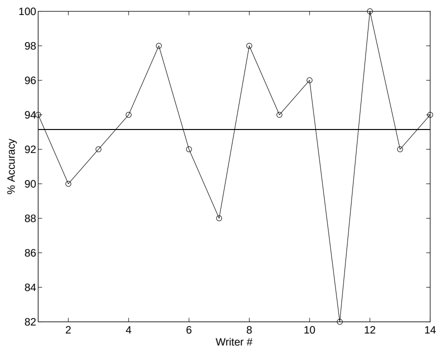


Figure 8: Accuracy of the Digit Recognition System

A very important problem in developing handwriting recognition systems is usually the insufficiency of data. In the process of creating a recognizer, different types of data are required such as, a text corpus for language model generation, training data, test data, etc. This is specially hard for languages for which not much on-line data is available, such as in Iran, Arab countries, Pakistan, Afghanistan, etc. For these languages, it is very time consuming to generate language models or to put together a training data set.

## References

[1] Charles C. Tappert, Ching Y. Suen, and Toru Wakahara, “The State of the Art in On-Line Handwriting Recognition,” IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 12, No. 8, Aug. 1990, pp. 787-808.

[2] T. Fujisaki, H.S.M. Beigi, C.C. Tappert, M. Ukelson, and C.G. Wolf, “On-line Recognition of Unconstrained Handprinting: a stroke-based system and its evaluation,” From Pixels to Features III: Frontiers in Handwriting Recognition, S. Impedovo and J. C. Simon (eds.), Elsevier Publishers, New York, 1992, pp. 297-312.

[3] Fathallah Nouboud and Réjean Plamondon, “On-line Recognition of Handprinted Characters: Survey and Beta Tests,” Pattern Recognition, Vol. 23, No. 9, 1990, pp. 1031-1044.

[4] Homayoon S.M. Beigi, Krishna Nathan, Gregory J. Clary, and Jayashree Subrahmonia, “Size Normalization in On-Line Unconstrained Handwriting Recognition,” International Conference on Image Processing, Austin, Texas, Nov. 1994, pp. 169-173.

[5] Samir Al-Emami and Mike Urber, “On-line Recognition of Handwritten Arabic Characters,” IEEE Transaction on Pattern Analysis and Machine Intelligence, Vol. 12, No. 7, July 1990, pp. 704-710.