Feature evaluation for handwritten character recognition with regressive and generative Hidden Markov Models

Kalyan Ram Ayyalasomayajula, Carl Nettelblad, Anders Brun

Center for Image Analysis, Division of Scientific Computing
Uppsala University
Sweden

Abstract. Hidden Markov Models constitute an established approach often employed for offline handwritten character recognition in digitized documents. The current work aims at evaluating a number of procedures frequently used to define features in the character recognition literature, within a common Hidden Markov Model framework. By separating model and feature structure, this should give a more clear indication of the relative advantage of different families of visual features used for character classification. The effects of model topologies and data normalization are also studied over two different handwritten data-sets. The Hidden Markov Model framework is then used to generate images of handwritten characters, to give an accessible visual illustration of the power of different features[1].

1 Introduction

Transcription of digitized handwritten documents is an important task gaining lot of attention from the image processing and pattern recognition research community. Degradation of the documents over time, variation in writing styles, and inconsistent document background based on the medium, are but a few of the challenges encountered when working in this field in particular historical handwriting. One of the methods that has been successful in this area are the hidden Markov models (HMMs) [1]. Within natural-language processing, these models were used for speech recognition early on, and are good at handling discrete and continuous sequences of data. Since text is often written along one direction, handling text as a sequence of characters that can be explained with a language model fits well within the HMM paradigm.

The process of training an HMM with a finite, discrete state space, depends on computing a matrix of *state transition probabilities* denoted by $\mathbf{A}$, a vector of *start probabilities* denoted by π, a matrix of state-specific *emission probabilities* distributions denoted by $\mathbf{B}$ which are computed by training relevant examples. In the current context, we consider the Baum-Welch and Bakis algorithms. The

[1] The research is funded by ”Vetenskapsrådet grant 2012-5743” and the ”q2b – From Quill 2 Bytes initiative at Uppsala University”.

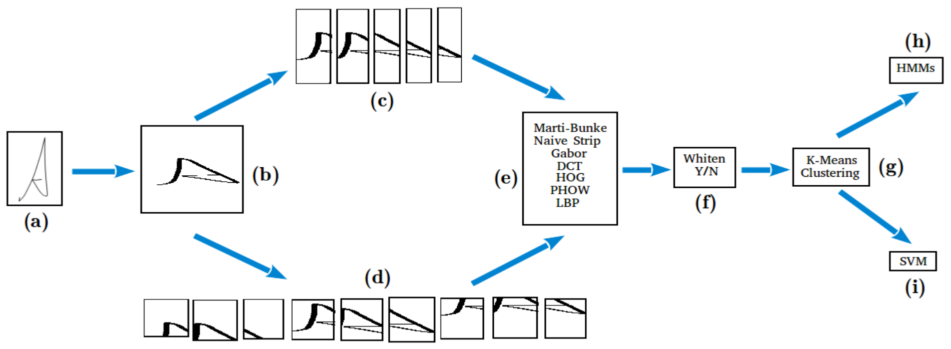

Fig. 1. An overview of the experimental setup with individual process blocks

Baum-Welch algorithm is an Expectation-Maximization procedure that tries to answer question of determining the $(\pi, \mathbf{A}, \mathbf{B})$ parameters which, help maximizing the HMM performance. However, there is an inherent directionality in Latin based text which is exploited in the *left-right* topology using Bakis algorithm. Here the states move left to right but not the other way around. Further details on HMM implementation can be found in [2].

As HMMs are efficient at handling sequential information, images are often processed using a sliding window. The sliding window approach can be used at word or character level to identify words or characters respectively [1]. In the current approach the content within each window is used to extract features, which are then labeled. By concatenating labels of successive sliding windows, a string is formed corresponding to the original image. A HMM-bank, containing multiple disjoint HMMs is trained on the labeled training sequences. A query image is then classified based on a scoring mechanism over all the HMMs in the bank. The HMM in the bank that is trained on instances of a character giving the most favorable likelihood for the whole sequence of labels is selected as the classifier output. This setup provides an ideal testing environment to understand the strengths of different image features when used along with a sliding-window HMM. A Support Vector Machine (SVM) classifier is also trained on the features under study, in order to provide a benchmark for comparison (Fig. 1(i)).

2 Architecture HMM Classifier

In order to evaluate various features we have used identical pre-processing and classification procedures for all feature sets tested. As shown in Fig. 1 each image is converted to a gray-scale image rescaled to 100×100 pixels (Fig. 1(a),(b)). Binarization is carried out if it is necessary to to compute specific feature such as background to foreground transitions. The image is then divided into overlapping patches allowing for about 50% overlap over adjacent windows (can be stripes favored by Marti-Bunke type features or patch based Fig. 1(c),(d)). Features are then extracted on each patch independently. The features thus accumulated over the training data are then clustered using k-Means and a label is assigned to each patch as per the cluster to which it belongs. The experiments are repeated with and without whitening the feature vector to understand its behavior.

In the current context HMMs are primarily used as classifiers of labeled sequences. In transcription of documents in Latin based script one is likely to encounter alphabetic letters in both upper and lower case, so an individual class for each character in each case is defined. This architecture leads to an HMM-bank with 52 individual HMMs, one for each letter in upper and lower case. Virtual beginning and end states are introduced, denoted by *start* and *end*, respectively. In case the feature extraction step produces n segments there will be $n + 2$ states in the HMMs. These special beginning and end states are included to facilitate concatenation multiple training observation sequences to form one observation sequence on which individual HMM is trained. The classification of a query image is done by converting it to a query string using the labels obtained from the k-Means clustering. The query string is then passed to each character HMM in the bank and likelihood score is returned. A decision based on the maximum score returned from HMM-bank results in the classification of the query image to a letter.

3 Feature Extraction

In the following section we provide a brief description of seven different feature extraction methods. They have all found previous use in character recognition. Some of these are unique to character recognition using HMMs, while others are widely known as generic feature extraction methods in image analysis and computer vision.

3.1 Naive Strip features

Each image is binarized and then divided into vertical segments. Within each segment connected components are identified and three of the maximal components are picked based on the component length, C_l. These components are then identified as long, $\mathcal{L}$ if $C_l \geq n \cdot w_d$, short, $\mathcal{S}$ if $w_d \leq C_l < n \cdot w_d$, or none, $\mathcal{N}$ if $C_l < w_d$, where w_d is the width of the segment and n is a scale-factor. Each segment can thus be identified with a triplet formed by $\mathcal{L}, \mathcal{S}, \mathcal{N}$. There can be 10 combinations of these triplets and these form the class labels for this feature.

3.2 Marti-Bunke features

This is a nine-dimensional feature that is obtained for each vertical segment of the image. This feature captures rough shape, texture and span information of a character by computing some statistics and estimates over each segment.

The shape information is based on computing the upper and lower contour position of the character in each segment, and the gradient of the upper and lower contours of the segment. The texture information is retained in the number of background-foreground transitions and by computing the number of foreground pixels between the upper and lower contour divided by the height of the contour. The character span is estimated by computing mean, the center of gravity and the second order moment of the segment [3]. For efficient computation of these features binarized as well as gray-scale images are required. This feature is a compact representation of the shape information, but lacks a scale estimate that is required in differentiating the upper and lower case alphabet for instance in the case of *x, o, c* etc.

3.3 Gabor features

The feature is built by dividing the gray-scale image into a grid. A complex Gaussian kernel is built with varying sigma at different angles between the real and imaginary part of the kernel (the argument of a complex number) [4]. The mean of the absolute value of the convolution output is set as the threshold and the count of instances that have exceeded this threshold in each grid at each scale and orientation is cascaded to form the overall feature vector [5]. In the current framework it is forty-dimensional (5 scales $\times$ 8 orientations) feature per patch are the default parameter settings.

3.4 Discrete Cosine Transform features

Each patch is then subjected to a discrete cosine transform (DCT) [6]. As most of the energy content of the image is contained in the low frequency, the coefficients are reordered by zig-zag scan in each patch. The most significant coefficients per patch can be picked up and cascaded to form the feature vector. In the default settings ten most significant coefficients are picked for each patch.

3.5 Histogram of Oriented Gradients features

For each of the patch gradients are computed along various orientations [7],[8]. A histogram over the computed gradients in the given patch is cascade into a feature vector. In the current setup this is a 31-dimensional vector per patch.

3.6 Pyramid Histogram Of visual Words features

This feature is a bag of dense Scale Invariant Feature Transform (SIFT) features at various scales [9]. This is a 512 dimensional feature vector due to the accumulation of 128 dimensional SIFT features at 4 scales.

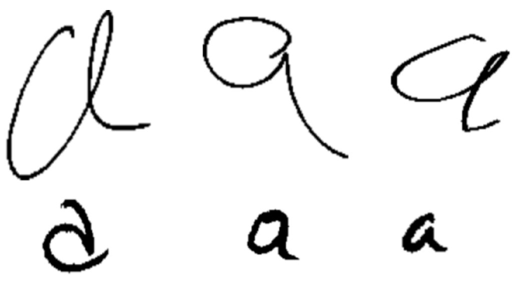

Fig. 2. Instances of 'a' in UniPenn(first row), NIST (second row)

3.7 Local Binary Patterns features

Within each selected path the center pixel is compared with every other pixel in a 3 × 3 neighborhood. Comparison between the pixels in encoded into a 8-dimensional vector of 0s and 1s depending on the gray scale value of the central pixel being higher or lower than the neighbor respectively. The vector can thus be a 128 dimensional but is quantized into 58 possible patterns thus resulting in 58 dimensional vector averaging over the patch. [10].

4 Experiments

All the experiments were conducted using two data sets as described below.

4.1 Data-sets

UJIPenchar2 The UJIPenchar2 dataset of online handwritten characters [11] captures the stylus co-ordinates of 60 writers when writing two instances of each character in upper and lower case. This data-set has recorded information of 120 instances of each character as an (x,y)-coordinate trace of the pen. This on-line pen information is converted to offline images of characters using spline interpolation tracing the pen trajectory from captured coordinates. These images are subjected to morphological erosion with a 3 × 3 cross structuring element to create characters of varying stroke width. Then a series of affine transformations are applied to the resulting images such as clockwise and counter-clockwise rotation about the vertical axis by 10 degrees, skewing the image in horizontal and vertical direction and adding noise along the edges of the character thus creating 3600 instances of each letter as shown in Fig.2.

NIST-19 The NIST[2] dataset consists of handwritten forms from 2100 different users provided with a form based handprint recognition system. About 1472

[2] The authors would like to thank Alicia Fornes and Computer Vision Center at Universitat Autonoma de Barcelona for their help in extracting the character images from the NIST data-set.

instances per lowercase character as shown in Fig.2, were extracted from these forms using the underlying recognition system [12].

4.2 Evaluation

On each of the datasets a random sample of 1012 images are picked of which 512 are used for training and 500 images are used for testing. The results reported here are percentage of classification accuracy $\%accuracy = \frac{T_c}{N}$, where T_c are images correctly labeled in the test set and N total number of test images averaged over all the letters.

The experiments can be broadly classified into two classes. First, a regressive evaluation of classification accuracy of the HMM classifier trained on each of the features described previously are benchmarked against SVM using a polynomial kernel of degree three. The focus has been on not only understanding the classification capability of the features, but also capture the parameter settings that are well suited for each method such as effect of whitening and k-Means clusters which directly influence the number of labels at each state in the HMMs. These parameter settings are then used in the experiments where the HMMs are used as a generative models for characters in order to further understand the feature in capturing the variance in the handwriting over characters. The results from the generative model are useful in visually interpreting the performance of features and also the role of better resolution on performance of HMMs.

4.3 Regression Tests

Number of kMeans clusters: In state of the art HMMs for handwriting transcription labeling the feature vector is done through training Gaussian Mixture models. The features are used to train a mixture model with 4 distributions. This model is in turn used to initialize and train a mixture with 8 distributions and this procedure is repeated successively to obtain a model with either 16 or 32 distributions [13]. In a similar spirit we have clustered the features using kMeans with k=5,10,...40 and found that going beyond k=20 does not yield any significant improvement in classification accuracy but makes the HMM training slower due to more labels. Effect of moving from k=10 to k=20 is shown in Fig. 3. For all the experiments that follow k=20 during the k-Means clustering step.

Topology and whitening of data: Two topologies tested in these experiments are the Ergodic and the Bakis topologies. The classification accuracy for these topologies are mostly similar. The results with and without of data whitening on the two HMMS topologies are available in Table1. The results from HMM classifier are compared with SVM with a polynomial kernel of degree three which is one of the top performing classifier on NIST digits dataset [14] with the entire image is used as input. This result is reported in Table 1. However, to make the comparison more fair, we also feed the SVM with feature data, in Table 2.

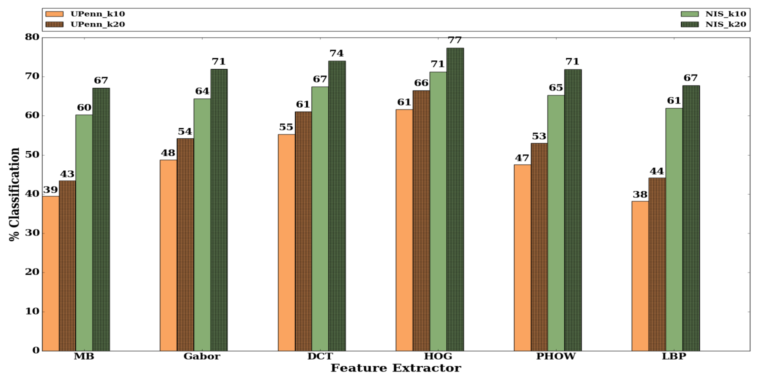

Fig. 3. Comparison of classification accuracy percentage of HMM classifier wiht k-Means Clustering with k=10 and k=20

Table 1. Classification accuracy percentages for HMM and SVM Classifiers

Feature	Ergodic		Bakis		Whitened
	UniPenn	NIST	UniPenn	NIST	
Naive-Strip	12.9	26.13	12.92	25.49	Yes
Marti-Bunke	43.36	66.68	43.46	67.09	
Gabor	52.24	70.48	52.24	70.5	
DCT	59.37	72.82	60.89	73.76	
HOG	63.36	77.68	64.58	76.82	
PHOW	52.24	70.48	48.38	68.68	
LBP	42.37	65.82	41.56	65.76	
Naive-Strip	12.9	26.13	12.92	25.49	No
Marti-Bunke	37.68	65.47	38.2	66.0	
Gabor	54.24	71.99	54.23	71.93	
DCT	61.43	74.18	61.09	74.0	
HOG	65.16	79.66	66.48	77.32	
PHOW	55.94	72.38	53.03	71.86	
LBP	45.30	69.22	44.16	67.76	
SVM poly. deg. 3	75.75	85.63	-	-	-

Sliding windows: It is a common practice in word and character HMMs to apply the sliding window from left to right and feed the HMM with features from thin overlapping image stripes. In this paper we extend this paradigm, also sweeping top-down and testing square shaped patches, and investigate whether the performance of the HMMs is improved. The size and step length of the sliding window in these cases are calculated such that length of the label sequence generated per image in both the methods are almost same, thus no bias in introduced in the HMM training. Table2 shows the comparison between Marti-Bunke and DCT. As the dimensionality of these features are nine and ten respectively, they are comparable.

Table 2. Comparison of features for patch vs strip based sliding window approach

Feature	UniPenn	NIST
HMM + MB + Strip	43.46	67.09
HMM + MB + Patch	45.90	75.83
HMM + DCT + Strip	44.77	67.85
HMM + DCT + Patch	59.79	76.09
SVM + MB + Strip	59.56	82.23
SVM + MB + Patch	65.79	85.67
SVM + DCT + Strip	58.54	84.34
SVM + DCT + Patch	60.32	85.0

4.4 Generative Tests

In the final experiments, we use the trained HMMs as generative models instead of classifiers. The results are synthesized instances of characters, which gives a glimpse of what a given HMM model is able to capture. It also enable us to make a qualitative comparison between features through the analysis of the emission matrix (**B**). The most probable state transitions are generated from the transition matrix (**A**) and the label at each state is generated from the emission matrix (**B**). The image corresponding to the label are generated from the k-Means cluster center of that label. This approach has two benefits. Firstly, it provides a qualitative way to visualize the results by showing the extent of variation in the writing of each character as captured by the features. Fig. 4 helps in comparing the letters generated with strip and patch mode, by comparing top and middle rows, and also features from Marti-Bunke and DCT features by comparing the middle and bottom rows.

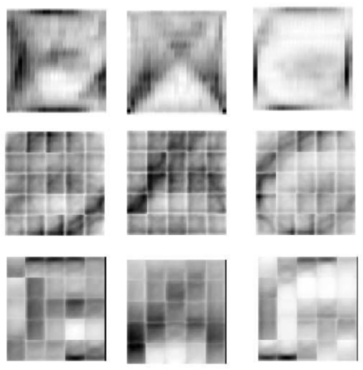

Fig. 4. Character instances of B,A,G from HMMs. top row: DCT+strip, middle row: DCT+patch, bottom row: MB+patch

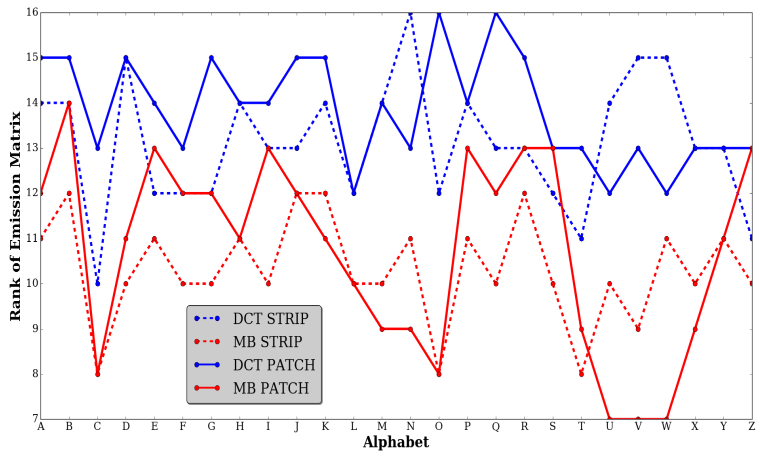

Fig. 5. Reduced rank of emission matrices over various character HMMs

Secondly, a the generative experiments helps to analyze the learning transfered to HMMs through the features. By construction, for an HMM with with N_s states its transition matrix ($\mathbf{A}$) is square and diagonal dominant as we are forcing a inherent directionality through the way the image of the character is swept. The variation in the various handwritten characters are captured in the emission matrix ($\mathbf{B}$), which is a rectangular matrix of size $N_s \times N_l$, where N_l are number of emission labels at each state. In order to capture this variation efficiently, matrix $\mathbf{B}$ has to be of full rank. By performing a low rank approximation of $\mathbf{B}$, we can determine which feature is efficient. In the current experiment $N_s = 25$, $N_l = 20$. For an efficient feature, the rank for the low rank approximation of $\mathbf{B}$ needs to be as high as possible, i.e as close to 20 as possible. Fig. 5 shows these results in parallel coordinates [15] representation of the rank for Marti-Bunke and DCT features respectively to limit the width of the graph only the result for HMMs trained for upper case letters is provided.

5 Conclusions

The HMMs performance of the ergodic and left-right topologies are almost identical over all the features. The Marti-Bunke features performs well when increasing the number of segments that divides the image. However, Marti-Bunke features suffers from performance issues when the sequence length get longer. Features that are able to encode the scaling information, such as HoG, Gabor and DCT, are able to outperform the other features such as Marti-Bunke and local binary patterns. This is particularly due to effectively handling the scaling that occurs over upper and lower case characters such as c,k,o,p,x. The best performing feature extration appears to be HoG. DCT has only slightly worse performance than HoG, but is more compact (1/3 of the dimensionality). The results indicate that performance of character recognition HMMs could be improved, by moving from a strip based sliding window approach to a patch based. Finally, The SVM classifier, which processes all feature values in parallel rather than as a sequence, consistently beats HMM. This indicates that HMMs may suffer from their inherently one-dimensional data processing, in this particular application.

References

1. Plotz, T., Fink, G.A.: Markov models for offline handwriting recognition: A survey. Int. J. Docu. Analy. Reco. **12** (2009) 269–298
2. Rabiner, L.R.: A tutorial on hidden markov models and selected applications in speech recognition. Proc. Of the IEEE **70** (1989) 257–286
3. Marti, U., Bunke, H.: Using a statistical language model to improve the performance of an hmm-based cursive handwriting recognition systems. in hidden markov models: applications in computer vision. World Scientic Publishing Co., Inc. – (2002) 65–90
4. Jin Chen, Huaigu Cao, R.P.A.B., Natarajan, P.: Gabor features for offline arabic handwriting recognition. In: Proc. Int. Work. Doc. Analy. Sys. Volume 9th., IAPR (2010) 53–58
5. Cheng-Lin Liu, M.K., Fujisawa., H.: Gabor feature extraction for character recognition: Comparison with gradient feature. In: Proc. Int. Conf. Doc. Analy. Reco. Volume 8th., IEEE (2005) 121–125
6. N. Ahmed, T.N., Rao., K.: Discrete cosine transfom. In: Trans. on Computers. Volume 23th., IEEE (1974) 90–93
7. Dalal, N., Triggs, B.: Histograms of oriented gradients for human detection. Proc. CVPR (2005)
8. P. F. Felzenszwalb, R. B. Grishick, D.M., Ramanan, D.: Object detection with discriminatively trained part based models. In: PAMI. Volume 32nd., IEEE (2009) 1627 – 1645
9. Bosch, A., Zisserman, A., Munoz, X.: Image classification using random forests and ferns. In: IEEE International Conference on Computer Vision. (2007)
10. T. Ojala, M.P., Harwood, D.: Performance evaluation of texture measures with classification based on kullback discrimination of distributions. Proc. ICPR (1994)
11. D. Llorens, e.a.: The ujipenchars database: A pen-based database of isolated handwritten characters. In: Proc. Int. Lang. Res. Eval. Volume 6th., LREC (2008) 2647–2651
12. Grother, P.J.: Nist special database 19 handprinted forms and characters database, National Institute of Standards and Technology (1995)
13. S. Young, J. Odell, D.O.V.V., Woodland, P.: The HTK Book: Hidden Markov Models Toolkit V2.1. Wiley-Interscience. Wiley (1997)
14. Y. LeCun, L. Bottou, Y.B., Haffner, P.: Gradient-based learning applied to document recognition. In: Proceedings of the IEEE. Volume 86th., IEEE (1998) 2278–2324
15. Moustafa, R.: Parallel coordinate and parallel coordinate density plots. In: Interdisciplinary Reviews: Computational Statistics. Volume Vol 3(2)., Wiley (2011) 134–148