ECSE 6810 Final Project: Handwritten Digit Classification
Yogish Didgi 661538970

1. Introduction

The field of computer vision has been the beneficiary of the advances in machine learning research in the past decade. And one of the most important applications in computer vision is the optical character recognition (OCR) task. The OCR task involves the computer recognising the handwritten data by humans. The data may be alphabets, digits, symbols etc. They can be of any language, size, style etc. The motivation for the OCR task is the digitization of all the hand written scriptures, which may be used for further research. In this project, we will be trying to classify the handwritten digits, 0 to 9. We will be using the Bayesian network (BN) theory to solve this problem.

2. Theory

We will be following two approaches to solve the problem. In the first method, we will construct a simple BN with all the pixels of the image as a binary node and the label node being the parent of all the pixel nodes. In the second approach, there will be an addition of a layer with hidden nodes in between the layer of pixel nodes and the label node. We will discuss these methods in detail in the following sections.

2.1 Single layer Naive Bayes classifier

As the name suggests, this classifier has a single layer of pixel nodes all connected to the label node in the higher layer. The label node is the parent and all the pixel nodes being its children. Note that, there is no link between any of the pixels. Figure 1 depicts this classifier.

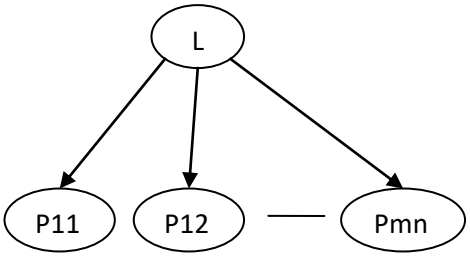

Figure 1: Single layer Naive Bayes classifier

In the figure, the pixel nodes are P11 to Pmn. P signifies that it is a pixel node. ‘m’ and ‘n’ indicates its position in the original image of size m x n. L indicates the label node. Since we are considering digits data, the pixel nodes are binary-valued, assuming only the values 0 or 1 with 0 indicating black and 1 indicating white on the digit image. There are 10 types of digits; 0 to 9. So, the label node is discrete with 10 possible values.

The data will be divided into training and testing data. During the training stage, both the pixel data and the label data is available. The task is to compute the maximum likelihood (ML) estimate of the parameters of each node. Since the pixel nodes are independent of each other given the label node, this computation becomes simpler. During the testing phase, we will compute the posterior of each label class given the pixel information.

The objective of ML estimation can be written as [1, 2],

$$\boldsymbol{\theta}^* = \arg\max_{\theta} LL(\boldsymbol{\theta}:\boldsymbol{D}) \tag{1}$$

where $LL(\boldsymbol{\theta}:\boldsymbol{D})$ represents the joint log-likelihood of $\boldsymbol{\theta}$ given pixel data $\boldsymbol{D}$. For a network with N nodes and M iid samples, we can write the log-likelihood as,

$$LL(\boldsymbol{\theta}:\boldsymbol{D}) = \log \prod_{m=1}^{M} p(X_1^m, X_2^m, \dots, X_N^m|\boldsymbol{\theta}) \tag{2}$$

By using the Markov Condition on the Bayesian network, we can write,

$$LL(\boldsymbol{\theta}:\boldsymbol{D}) = \log \prod_{n=1}^{N}\prod_{m=1}^{M} p(X_n^m|\pi(X_n^m),\theta_n) = \sum_{n=1}^{N}\sum_{m=1}^{M} \log p(X_n^m|\pi(X_n^m),\theta_n) = \sum_{n=1}^{N} LL(\theta_n:\boldsymbol{D}) \tag{3}$$

where $LL(\theta_n: \boldsymbol{D})$ is the marginal log-likelihood for node X_n. In equation (3), we have written the joint log-likelihood of all the parameters in the network as the sum of the marginal log-likelihood for each node. So, the parameters for each node can be learnt separately i.e.,

$$\theta_n^* = \arg\max_{\theta_n} LL(\theta_n: \boldsymbol{D}) \tag{4}$$

If the BN is discrete, θ_n can be further decomposed into $\theta_n = \{\theta_{nj}\}$ for $j = 1, 2, \dots, J$ for j th configuration of the parents for node X_n. Assuming the parameters for each parent configuration is independent of each other, we can write the marginal log-likelihood for node X_n as,

$$LL(\theta_n: \boldsymbol{D}) = \sum_{m=1}^{M} \sum_{j=1}^{J} \log p(X_n^m | \pi(X_n^m) = j, \theta_{nj}) \tag{5}$$

So, we can compute the log-likelihood for each parameter separately i.e.,

$$\theta_{nj}^* = \arg\max_{\theta_{nj}} LL(\theta_{nj}: \boldsymbol{D}) \tag{6}$$

Given each parent configuration, the counts of node X_n follow a multinomial distribution. So for a K-valued distribution, we have θ_{njk} as the probability that node X_n will have a value *k* with its parents in configuration *j*. We can write the log-likelihood as,

$$LL(\theta_{nj}: \boldsymbol{D}) = \sum_{m=1}^{M} \log p(X_n^m | \pi(X_n^m) = j, \theta_{nj}) = \log \frac{M!}{\prod_{k=1}^{K} M_{nj}} \prod_{k=1}^{K} \theta_{njk}^{M_{njk}} \tag{7}$$

where, M_{njk} is the number of samples with node X_n taking value *k* and its parents in configuration *j*. Also note that,

$$M_{nj} = \sum_{k=1}^{K} M_{njk} \tag{8}$$

$$\sum_{k=1}^{K} \theta_{njk} = 1 \tag{9}$$

Maximizing equation (7) with respect to the constraint (9) results in,

$$\theta_{njk} = \frac{M_{njk}}{M_{nj}} \tag{10}$$

Therefore, the parameter estimation is now reduced to a counting problem. Counting up all the samples where node X_n takes value *k* and its parents in configuration *j* and dividing by the total samples of X_n with its parents in configuration *j* gives the estimate of parameter θ_{njk}. Of course we need to have sufficient number of samples for each parameter to get a reliable estimate.

So, for every pixel location (i,j), we compute the parameter P(Pij|L) for each class label. And since the distribution parameters must follow the rules of probability, the other parameters will be computed so as to not violate equation (9).

During testing, we are given the pixel information and are interested in determining the corresponding label. We will compute the posterior as,

$$P(L = l \,|P_{11}, P_{12}, \dots, P_{mn}) \propto \prod_{i,j} P(P_{ij} | L = l) \tag{11}$$

The probabilities of each pixel are multiplied in the above equation because, each of the pixel is independent of the other given the parent i.e., label node. The posterior is computed for each class of each sample in the test dataset. Then, all the values are normalized by their sum to obtain actual probability values. The final classification is done by picking the class which has the highest posterior.

$$class = \arg\max_{l} P(L = l \,|P_{11}, P_{12}, \dots, P_{mn}) \tag{12}$$

2.2 Two-layer Naive Bayes classifier

In this network, the bottom layer has all the pixel nodes, similar to the previous classifier. The next higher layer is a feature layer and the top layer is the label node. Each node in the feature layer is connected to 4 nodes in the pixel node layer, with the feature node acting as the parent node and pixel nodes as children. All the feature nodes are connected to the label node, with label node as the parent. There are no interconnections among the feature nodes or the pixel nodes. Figure 2 depicts this classifier.

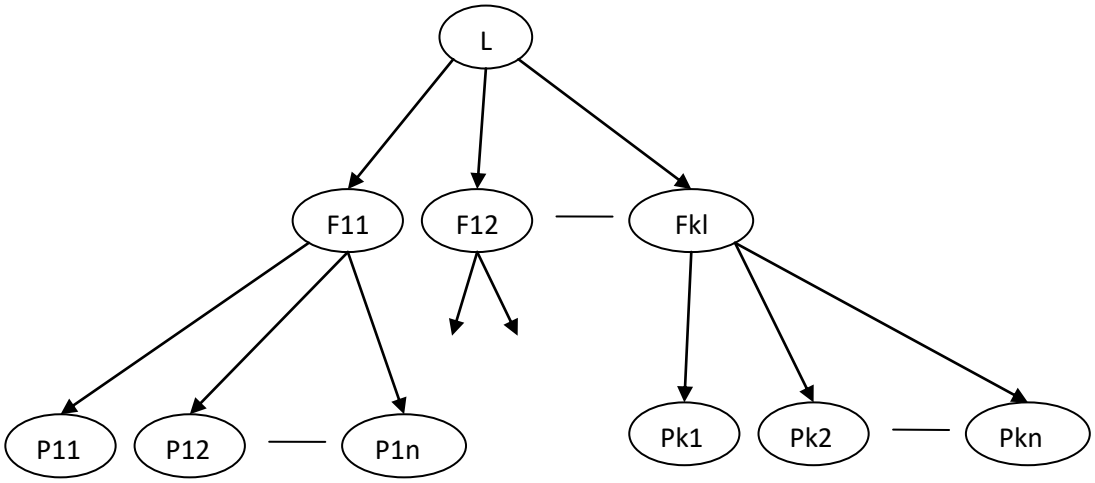

Figure 2: Two-layer Naive Bayes classifier

In the figure, Fij represents the feature and Pi1 to Pin its children. L is the label node. For the implementation of this classifier, the number of children for each feature node was chosen to be 4. So, each feature node is connected to 4 pixel nodes. The feature chosen was very simple. The feature node is binary with values 0 or 1. It will be set to 1 if atleast 2 of its children are set to 1. Otherwise, it will be set to 0. There are no probabilities involved when computing the feature nodes. The feature nodes layer along with the label node now forms a naive bayes classifier. Figure 3 indicates how features are connected to pixels from the digit image.

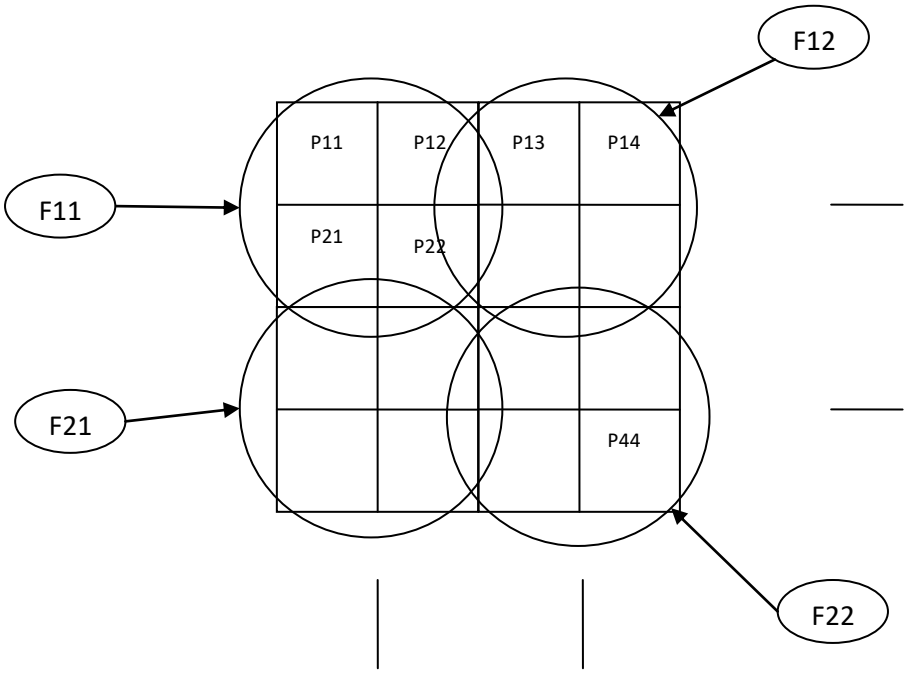

Figure 3: Forming features from pixels

During the training phase, we compute the feature values from the pixel values. Then, for every feature location (i,j), we compute the parameter P(Fij|L) for each given class label. Note that, L is a discrete node with 10 possible values and F is a binary valued node. The parameters are computed using the likelihood estimation procedure as described in the previous section.

During the testing phase, we again compute the feature values from the given pixel values. We will compute the posterior as,

$$P(L = l\,|F_{11}, F_{12}, \dots, F_{kl}) \propto \prod_{i,j} P(F_{ij}|L = l) \tag{13}$$

The probabilities of each feature are multiplied in the above equation because, each of the feature nodes is independent of the other given the parent i.e., label node. The posterior is computed for each class of each sample in the test dataset. Then, all the values are normalized by their sum to obtain actual probability values. The final classification is done by picking the class which has the highest posterior.

$$class = \arg\max_l P(L = l\,|F_{11}, F_{12}, \dots, F_{kl}) \tag{14}$$

2.3 Three-layer Naive Bayes classifier

The classifier described in this section is similar to the one in the previous section. The only difference is the addition of an extra feature layer. So, the first layer is the pixel nodes layer, second is the feature layer for pixels, third is the feature layer for the feature nodes. The label node is the last layer. It acts as the parent to all feature nodes in the layer below it. Each feature node is connected to 4 children in the layer below. The label node is discrete and can assume one of 10 values in the range 0 to 9. Feature nodes in both the layers are binary valued. Note that, there are no interconnections among the nodes in the same layer. For simplicity, the feature used in both the layers is the same, i.e., the feature node is set to 1 if atleast 2 of its children are set to 1; otherwise it is set to 0. Of course, there can be some modification to the feature used in the second feature layer. The feature was chosen to be the same for the ease of implementation. However, the ordering chosen was different. The children for each feature node in the second layer are picked from the feature nodes in the layer below in a serial format using row major order. Figure 4 depicts this classifier.

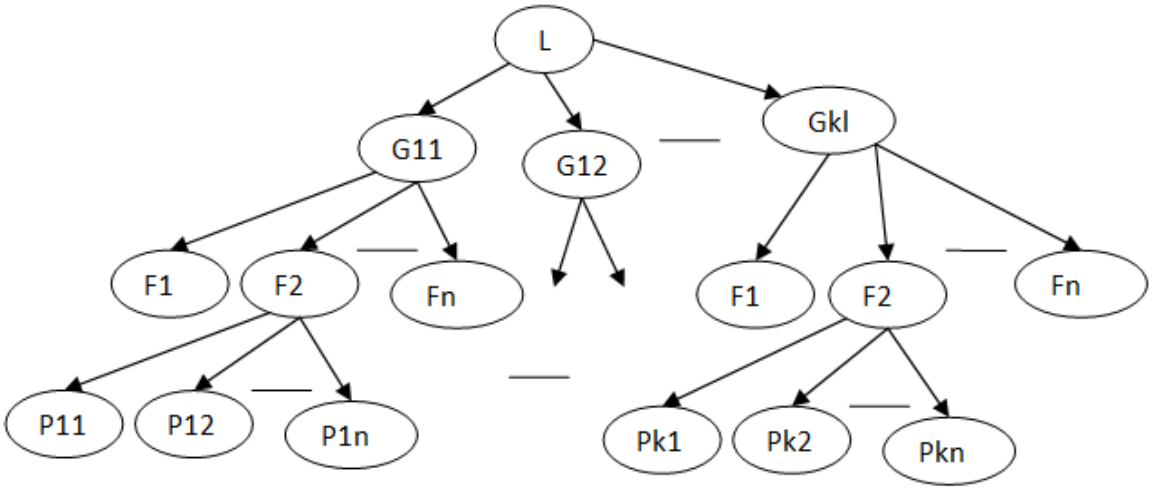

Figure 4: Three-layer Naive Bayes classifier

As in the two-layer naive bayes classifier, during both the training and testing phase, we first compute the feature nodes' values. We start off with the first layer nodes, F, since it is calculated solely based on the pixel values. Then, the feature nodes, G, at the second layer are computed based on values in the feature layer below. Then, for every feature location (i,j), we compute the parameter P(Gij|L) for each given class label. The parameters are computed using the likelihood estimation procedure as described in the previous sections.

During the testing phase, we will compute the posterior as,

$$P(L = l\,|G_{11}, G_{12}, \dots, G_{kl}) \propto \prod_{i,j} P(G_{ij}|L = l) \tag{15}$$

The probabilities of each feature are multiplied in the above equation because, each of the feature nodes is independent of the other given the parent i.e., label node. The posterior is computed for each class of each sample in the test dataset. Then, all the values are normalized by their sum to obtain actual probability values. The final classification is done by picking the class which has the highest posterior.

$$class = \arg\max_l P(L = l \,|G_{11}, G_{12}, \dots, G_{kl}) \tag{16}$$

2.4 Hidden-layer Bayes classifier

The hidden layer bayes classifier is similar to the two-layer Naive classifier in the sense that it has two layers in the BN. The image pixels form the nodes in the first layer as before. The second layer is made up of hidden nodes. However, for the second layer, the data is not directly computed as a feature. So, we have now an incomplete data set since some of the nodes values is missing. We follow the procedure of Expectation-Maximization (EM) to estimate the most likely value for all the hidden nodes such that it maximises the likelihood of the training data. The BN for this classifier is shown in Figure 5.

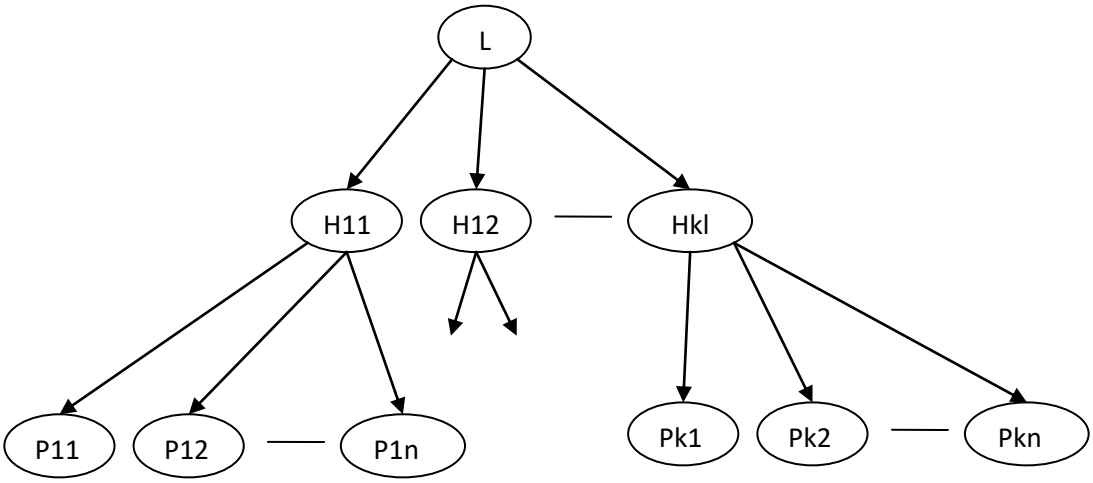

Figure 5: Hidden layer Bayes classifier

In the figure, Hij represents the hidden node and Pi1 to Pin its children. L is the label node. For the implementation of this classifier, the number of children for each hidden node was chosen to be 4. So, each hidden node is connected to only 4 pixel nodes. The hidden node is binary with values 0 or 1. Figure 3 indicates how hidden nodes are connected to pixels from the digit image. Note that, there are no interconnections between nodes in the same layer, i.e., pixel nodes are not connected to each other and hidden nodes are not connected to each other. So we have a tree structure for the network. We will now briefly outline the EM procedure [1, 2].

Given the incomplete data set, *D*, each sample can be represented as D_m, where *m* = 1, 2, ... , M. M is the total number of samples in the data set. We can decompose the training sample, $D_m = \boldsymbol{X}^m$, into two parts. One part has nodes' values which are completely observed, $\boldsymbol{Y}^m$, and another in which none of the nodes' have been observed, $\boldsymbol{Z}^m$, i.e., $\boldsymbol{X}^m = \{\boldsymbol{Y}^m, \boldsymbol{Z}^m\}$. Note that, $\boldsymbol{Z}^m$ can vary across samples i.e., the node observed in the current sample may be unobserved in the next sample and vice-versa. However, for our case, $\boldsymbol{Z}^m$ consists of nodes in the hidden layer only.

The EM method is the most widely used method for parameter estimation when the data is incomplete. The EM method maximizes the expected log-likelihood, which is the lower bound on the marginal log-likelihood. The marginal log-likelihood can be written as,

$$\log p(\boldsymbol{D}|\boldsymbol{\theta}) = \sum_{m=1}^{M} \log \sum_{\boldsymbol{Z}^m} p(\boldsymbol{Y}^m, \boldsymbol{Z}^m|\boldsymbol{\theta}) = \sum_{m=1}^{M} \log \sum_{\boldsymbol{Z}^m} q(\boldsymbol{Z}^m|\boldsymbol{Y}^m) \frac{p(\boldsymbol{Y}^m, \boldsymbol{Z}^m|\boldsymbol{\theta})}{q(\boldsymbol{Z}^m|\boldsymbol{Y}^m)} \tag{17}$$

where, $q(\boldsymbol{Z}^m|\boldsymbol{Y}^m)$ is an arbitrary density function chosen by the user. This is often chosen to be independent of the current estimate $\boldsymbol{\theta}$.

For a concave function, the Jensen's inequality states that, the function of the expectation is at least equal to the expectation of the function. Noting that in equation (17), the log function is the concave function, applying Jensen's inequality leads to,

$$\log \sum_{\boldsymbol{Z}^m} q(\boldsymbol{Z}^m|\boldsymbol{Y}^m) \frac{p(\boldsymbol{Y}^m,\boldsymbol{Z}^m|\boldsymbol{\theta})}{q(\boldsymbol{Z}^m|\boldsymbol{Y}^m)} \geq \sum_{\boldsymbol{Z}^m} q(\boldsymbol{Z}^m|\boldsymbol{Y}^m) \log \frac{p(\boldsymbol{Y}^m,\boldsymbol{Z}^m|\boldsymbol{\theta})}{q(\boldsymbol{Z}^m|\boldsymbol{Y}^m)} \tag{18}$$

So, instead of maximizing the log-likelihood given by equation (17), in EM method, we will maximise its lower bound given by equation (18). So, the parameter estimate is given by

$$\boldsymbol{\theta}^* = \arg\max_{\boldsymbol{\theta}} \sum_{m=1}^{M} \sum_{\boldsymbol{Z}^m} q(\boldsymbol{Z}^m|\boldsymbol{Y}^m) \log p(\boldsymbol{Y}^m, \boldsymbol{Z}^m|\boldsymbol{\theta}) \tag{19}$$

Note that $q(\boldsymbol{Z}^m|\boldsymbol{Y}^m) \log q(\boldsymbol{Z}^m|\boldsymbol{Y}^m)$ term is not included in equation (19) since it is independent of the current estimate. The choice for q is given by,

$$q(\boldsymbol{Z}^m|\boldsymbol{Y}^m) = p(\boldsymbol{Z}^m|\boldsymbol{Y}^m, \boldsymbol{\theta}^{t-1}) \tag{20}$$

Hence, q is dependent only on the estimate of the parameters from the previous step.

The maximization in equation (19) is done in two steps; E-step and M-step.

$$Q^t(\boldsymbol{\theta}^t|\boldsymbol{\theta}^{t-1}) = \sum_{m=1}^{M} \sum_{\boldsymbol{Z}^m} p(\boldsymbol{Z}^m|\boldsymbol{Y}^m, \boldsymbol{\theta}^{t-1}) \log p(\boldsymbol{Y}^m, \boldsymbol{Z}^m|\boldsymbol{\theta}^{t-1}) \tag{21}$$

$$\boldsymbol{\theta}^* = \arg\max_{\boldsymbol{\theta}} Q^t(\boldsymbol{\theta}^t|\boldsymbol{\theta}^{t-1}) \tag{22}$$

Equation (21) represents the E-step, where the lower bound (Q) is computed based on previous estimate ($\boldsymbol{\theta}^{t-1}$). Equation (22) represents the M-step, where the lower bound is maximised to obtain a new estimate ($\boldsymbol{\theta}^t$). Equation (22) is decomposable and hence each node's parameters can be estimated separately. Thus,

$$\theta_{njk}^t = \frac{M_{njk}}{M_{nj}} \tag{23}$$

where, M_{njk} is the number of samples with node X_n having value of k, with its parents being in jth configuration.

$$M_{njk} = \sum_{m=1}^{M} p(X_n^m = k|\pi(X_n^m) = j, \boldsymbol{Y}^m, \theta_n^{t-1}) \tag{24}$$

$$M_{nj} = \sum_{k=1}^{K} M_{njk} \tag{25}$$

The E-step and M-step in equations (21) and (22) are computed iteratively until convergence. This method is prone to converge to local maxima. Hence, parameter initializations for the first iterate becomes important, especially when the number of missing nodes' values is large, which is the case here. Random initialization was chosen in the implementation. The pseudo-code for this algorithm [1] is shown in Figure 6.

Algorithm 8 Pesudo-code for the EM algorithm

Given $X_1, X_2, ..., X_N$ nodes for a BN with each node assuming K values
$w_{m,j} = 1$, m=1,2, ..., M, j=1,2,..,K^N //initialize the weight for each sample
$\boldsymbol{\Theta}^0 = \boldsymbol{\Theta}_0$, //initialize the parameters for each node
while not converging **do**
  t=1
  E-step:
  for m=1 to M **do**
    if $\mathbf{X}^m$ contains variables $\mathbf{Z}^m$ with missing values **then**
      for j=1 to $K^{|\mathbf{Z}^m|}$ **do**
        $w_{m,j} = p(Z_j^m|\mathbf{Y}^m, \boldsymbol{\Theta}^{t-1})$ //Z_j^m is the jth configuration of $\mathbf{Z}^m$
      end for
    end if
  end for
  M-step:
  $\boldsymbol{\Theta}^t = \arg\max_{\boldsymbol{\Theta}^t} \sum_{m=1}^{M} \sum_{j=1}^{K^{|\mathbf{Z}^m|}} w_{m,j} \log p(Y^m, Z_j^m|\boldsymbol{\Theta}^t)$
  t=t+1
end while

Figure 6: Pseudo-code for EM algorithm [1]

Note that, the EM procedure is carried out separately for each hidden node. So, during the EM process, the hidden node, the label node and only its children are considered. This is valid because, the hidden nodes are independent of each other given the label nodes, since there are no interconnections. The EM task is now a little less computation-intensive. The EM procedure helps in determining the parameters of all the nodes in the network which best explains the training data. Now that we have the parameters, we now have to perform inference for the testing dataset given the parameters. This can be done using exact methods or approximate methods. They are described in the following sections.

2.5 Exact inference

The inference is computed as the posterior of the label given the pixel values. It is given by,

$$P(L = l\,|P_{11}, P_{12}, \dots, P_{kl}) \;\propto\; P(L = l, P_{11}, P_{12}, \dots, P_{kl}) \tag{26}$$

The proportionality in the above equation can be removed once all the values are computed using the right side of the equation and normalising by their sum. Since there is layer of hidden nodes, $\boldsymbol{H}$, between the pixel nodes and the label node, we can write equation (26) as,

$$\begin{aligned} P(L = l, P_{11}, P_{12}, \dots, P_{kl}) &= \textstyle\sum_{\boldsymbol{H}} P(L = l, P_{11}, P_{12}, \dots, P_{kl}, \boldsymbol{H}) \\ &= \textstyle\sum_{\boldsymbol{H}} P(L = l) \prod_{ij} P\big(H_{ij}|L = l\big) \prod_{kl} P(P_{kl}|\pi(P_{kl})) \end{aligned} \tag{27}$$

The probability of the label, $P(L = l)$, can be removed from the above equation and the equality replaced with proportionality, which of course can be removed at the end by normalising the computed posterior values for all labels. π in the above equation represents the parent of the node in the parentheses. The parents of the pixel nodes are only one of the hidden nodes which can be determined if the BN is known.

The above equation is computationally intensive. This is because the joint probability needs to be summed over all possible configuration values of the hidden nodes. The hidden nodes are binary. Suppose, the number of hidden nodes is N, then the number of additions in the above equation is of the order 2^N. This is exponential in nature. Although the value being computed is just a simple product, the number of additions makes it prohibitive to use this approach to compute the inference.

2.6 Approximate inference

There are several methods for computing the approximate inference. Likelihood weighted sampling approach was implemented in this project. It will be briefly explained now [1, 2].

One of the simplest techniques to obtain samples from a Bayesian network is to query all the nodes in the network in a pre-determined order. The predetermined order is that we start from the root node of the given network, work down to its children and its descendents until we reach the leaf nodes in the network. The state of all the nodes jointly defines one sample of the network. The inference task is simply to count the number of instances of the desired query variable with the desired state, and normalise it with all the obtained samples.

This method is known as Logic sampling. This method is proven to work correctly as it follows the Central Limit theorem, which states that, as we increase the number of samples of a random variable, the expectation is that it will settle to the mean of the random variable.

The problem with the simple procedure mentioned above arises when we have to incorporate the evidence observed in the network. One approach is to follow the above-mentioned approach, but to discard any sample which disagrees with the evidence. This seems very inefficient, because if the evidence observed has a very low probability of occurrence, according to the probability distribution, then we would be throwing away a lot of samples which don't agree with the evidence.

An improvement over the logic sampling is the likelihood weighting sampling technique. In this method, the sampling for the evidence node is not performed. Instead, the evidenced nodes values are taken directly into the sample of the network. But we also associate that sample with a weighting factor determined by the product of the probabilities of the evidence node taking that particular value given the current state of their parents in the network. Note that, all other nodes in the network, which are not in evidence, will be sampled as described before in the logic sampling. To perform inference, we sum the weights of all the samples in which our query node has the desired state. This is normalised by the total weight of all the samples. This gives the approximate inference. Again, as noted previously, the obtained probability value approaches the true estimate as we increase the number of samples. A pseudo code for likelihood weighted sampling technique is showed in Figure 7.

The advantage of likelihood weighted sampling technique is that it is simple. The inference is effectively a counting process. There are no disadvantages of this method. However, if the evidence has a low probability of occurrence, we have to obtain large number of samples, if we want the sample distribution to match the population distribution. However, this is true of any sampling method.

For the BN of hidden layer classifier, all the pixel nodes form the evidence and they are taken as is and their probability is accounted for in the weight calculation. The label node is assumed uniform random variable and is sampled as such to obtain values from 0 to 9. The hidden nodes are sampled from a distribution which is dependent on the label node value obtained in the previous step.

A source of frustration in approximate inference is the numerical precision. We are dealing with probability values, ranging from 0 to 1. If we have N hidden nodes and M pixel nodes, then the number of multiplications for each configuration of the hidden nodes is (N+M). The resulting product may be too small to be represented by the system and can be considered as an invalid entry or 0 depending on the implementation. This problem was taken care of by initializing the joint probability as the largest possible real number, and then multiplying each probability value to this value. As a result, the final probability value has a lower chance of losing out of the lowest possible precision of the machine. This is valid, since the values are all normalised at the end to obtain the correct probability values.

Algorithm 4 Weighted logic sampling

Order BN variables X_1,X_2,, X_N according to their topological order from the root nodes until leaf nodes
t: index to the number of samples with a total of T samples
i: index to the node number, with a total of N nodes
E: evidence nodes
Initialize weights w_1, w_2, ..., w_T to 1

```
for t=1 to T do
  for i to N do
    if X_i^t ∉ E then
      sample x_i^t from p(X_i^t|π(X_i^t))
    else
      x_i^t = e_i
      w_t = w_t * p(X_i^t = e_i|π(X_i^t))
    end if
  end for
  Form sample x^t = {x_1^t, x_2^t, ..., x_N^t} and compute its weight w_t
end for
```

Return $(\mathbf{x}^1, w_1), (\mathbf{x}^2, w_2), ..., (\mathbf{x}^T, w_T)$

Figure 7: Likelihood weighted sampling [1]

3. Data

The handwritten digit dataset was obtained from [3]. It provides 60000 training samples and 10000 samples in the test set. Due to significant amount of training time, we limit the number of samples used for training and testing to 10000 and 1000 respectively. The dataset consists of image file and label file. The image file contains an ordered set of integer values corresponding to each pixel in a 28x28 grid of the image for each sample. The values range from 0 to 255. To simplify the problem, the intensity values were thresholded to make each pixel a binary-valued random variable. The threshold chosen was 127. The label image consists of the ground truth labels i.e., the actual characters, the images represent. Some samples of the dataset are shown in Figure 8. The result of thresholding is also shown.

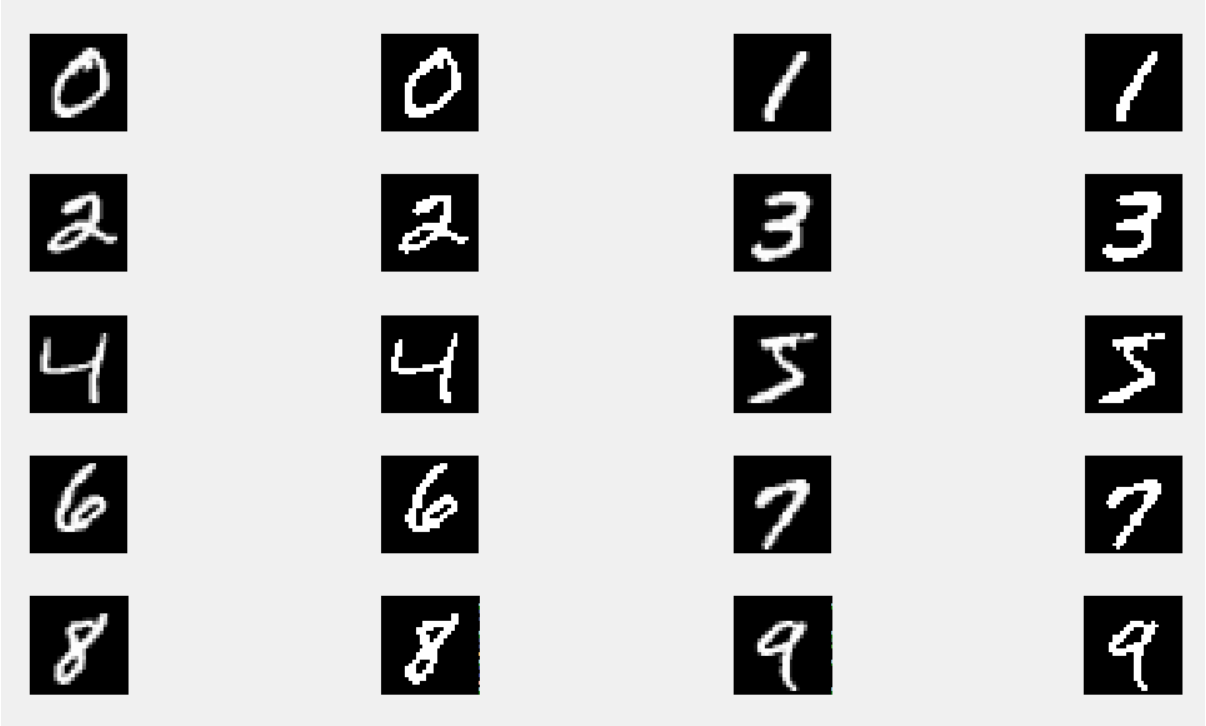

Figure 8: Digit images before and after thresholding

4. Results

Naive Bayes classifier was run with all the 28x28 pixels in the pixel node layer. The number of features in the 2-layer bayes classifier was 14x14 with each node having 4 pixel nodes as children. The number of features in the 3-layer bayes classifier was 14x14 in the first layer and 7x7 in the second layer. The image data was also resized to 16x16, 14x14, 8x8 and 7x7 and evaluated with single layer naive bayes classifier to make a suitable comparison across all the classifiers.

The classification accuracy for each digit for all the classifiers is shown in Table 1, and the same is plotted in Figure 9. The number of training samples was 10000, and the number of testing samples was 1000. The dataset was uniformly distributed across the digits, meaning there was same number of data samples for each digit during both training and testing. The best classifier for the particular digit is shown in bold in Table 1. Single layer bayes classifier outperformed all the others in most of the digits (six classes). Two layer naive bayes wasn't far behind with it classifying the best in 5 classes. It performed better than the 14x14 resized network. Similarly, the three layer bayes outperformed the one with 7x7 pixel nodes.

The running time of these classifiers was also very low, with the longest amount for both training and testing taking less than a few minutes. More time was taken for pixel data rearrangement as was shown in Figure 3.

Table 1: Accuracy of Naive Bayes classifier

Digit	Single layer Naive Bayes	Two layer Naive Bayes	Three layer Naive Bayes	Single layer Naive Bayes-Resize_16x16	Single layer Naive Bayes-Resize_14x14	Single layer Naive Bayes - Resize_8x8	Single layer Naive Bayes - Resize_7x7
0	0.86	0.86	0.75	**0.89**	0.85	0.72	0.54
1	0.95	**0.98**	0.97	0.95	0.94	0.85	0.93
2	**0.79**	**0.79**	0.56	0.75	0.74	0.56	0.45
3	0.83	**0.86**	0.56	0.83	0.8	0.58	0.48
4	**0.74**	**0.74**	0.69	0.72	0.69	0.56	0.6
5	**0.66**	0.59	0.33	0.62	0.61	0.57	0.46
6	0.82	0.81	0.55	**0.84**	**0.84**	0.66	0.53
7	**0.78**	0.77	0.66	0.77	0.77	0.57	0.55
8	**0.69**	**0.69**	0.37	**0.69**	**0.69**	0.51	0.38
9	**0.9**	0.86	0.82	0.85	0.86	0.53	0.48
Average Accuracy	**0.802**	0.795	0.626	0.791	0.779	0.611	0.54

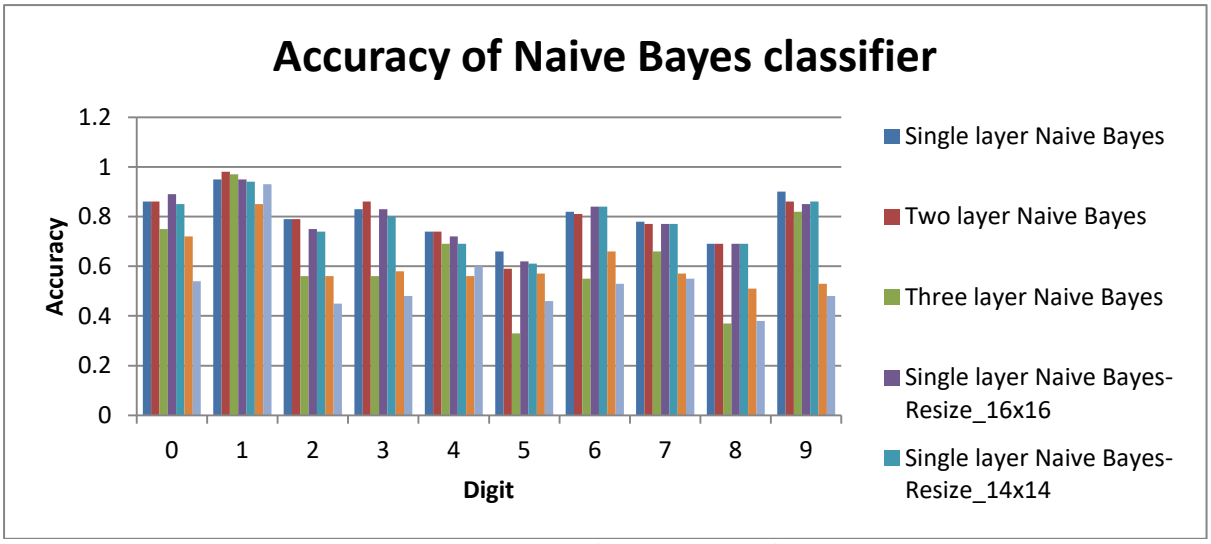

Figure 9: Accuracy of Naive Bayes classifier

For the hidden layer bayes classifier, only 10000 training samples and 1000 testing samples were considered. There was same number of data samples for each class during both training and testing. Three classifiers were built for this type. The first one had all the 28x28 pixel nodes in the bottom layer and 14x14 nodes in the hidden layer. For the second one, the digits pixel data was resized to 16x16 and used. The number of hidden nodes was 8x8 in that case. For the third one, 8x8 pixel nodes were used with 4x4 hidden nodes. As can be inferred, the number of children for each hidden node in all the classifiers is the same (equal to 4).

Due to significant running time for training the classifiers, the number of EM iterations was fixed to 10000. Only hidden layer-3 classifier was run for 20000 iterations. This is a little low and its effect is certainly seen in the accuracy results. The training time amount for these values is shown in Table 2. The number of iterations for likelihood weighted sampling was set to 10000. This seemed to be a reasonable number.

Approximate inference was chosen in all the cases. This is because the number of hidden nodes is very high in each of the 3 classifiers and the number of states to sum over was too high. For e.g., hidden layer-2 classifier has 64 hidden nodes. The number of possible states to iterate over is 2^64 which is of the order 10^19. Exact inference was computed only for hidden layer-3 classifier. It had 2^16 = 65536 states to iterate over for each

testing sample. Each sample took 12.5 sec to classify, leading to close to 3.5 hours for performing the classification for all the 1000 test samples.

Table 2: Details of hidden layer bayes classifier

Parameter	Hidden layer-1	Hidden layer-2	Hidden layer-3
Num Pixel nodes	28x28=784	16x16=256	8x8=64
Num Hidden nodes	14x14=196	8x8=64	4x4=16
Time for each EM iter(sec)	2.1	0.7	0.17
Number of EM iterations	10000	10000	20000
Total training time	5.83 hrs	1.94 hrs	57 min
Total classification time	370 sec	151 sec	51 sec (approximate) 3.47 hrs (exact)

Compared to the naive bayes classifier, the hidden layer bayes classifier performed a little worse. The accuracy results are shown in Table 3 and Figure 10. Results are shown for two trials for each of the classifier. The difference in the accuracy is due to the random component present in the code. All the nodes in the network are initialized to random values for their parameters for the first iteration before performing EM. Also, approximate inference involves sampling from the estimated parameters which is also a random event.

The second implementation of the classifier with 64 hidden nodes performed the best achieving around 70% classification accuracy over all the digits. Still this was over 10% behind the best performing naive bayes classifier. We expect the accuracy to improve with changes introduced in the network architecture. For instance, we could introduce interconnections among the hidden nodes. Secondly, the number of EM iterations was restricted to 10000 to keep the training time reasonable. This may be increased further. Thirdly, it is possible that the EM procedure was getting stuck in local optima due to the random initialization of the nodes' parameters. For such a high dimensional space, this is quite possible.

The hidden layer-2 classifier gave the best performance for 6 of the classes compared to the other classifiers. It was surprising to see hidden layer-1 classifier performing worse than the hidden layer-2 classifier. The hidden layer-3 was significantly worse than the other two and could only achieve around 39% accuracy on average.

The exact inference was run for hidden layer-3 classifier. The resulting accuracy was around 44%. The approximate inference was underestimating the classifier by around 5 percentage points. This suggests that maximum likelihood weighted sampling with 10000 iterations can be used to get a reasonable inference. If we expect this trend to hold for the other classifiers as well, then the hidden layer-2 classifier almost matches the accuracy of the naive bayes classifiers.

Table 3: Accuracy of Hidden Layer Bayes classifier

Digit	hidden layer-1		hidden layer-2		hidden layer-3		hidden layer-3-Exact
	Run-1	Run-2	Run-1	Run-2	Run-1	Run-2	
0	**0.88**	**0.88**	0.85	**0.88**	0.56	0.57	0.55
1	0.95	**0.97**	0.91	0.96	0.27	0.27	0.61
2	0.49	0.55	0.6	**0.63**	0.5	0.52	0.51
3	0.76	0.73	**0.81**	0.8	0.58	0.56	0.58
4	0.47	0.5	**0.64**	0.55	0.34	0.33	0.48
5	0.26	0.27	0.32	**0.33**	0.27	0.25	0.27
6	0.65	0.64	**0.7**	0.68	0.52	0.5	0.5
7	0.58	0.58	0.71	**0.76**	0.24	0.24	0.3
8	0.55	**0.61**	0.59	**0.61**	0.32	0.32	0.31
9	0.65	0.65	**0.83**	**0.83**	0.35	0.32	0.33
Average Accuracy	0.624	0.638	0.696	**0.703**	0.395	0.388	0.444

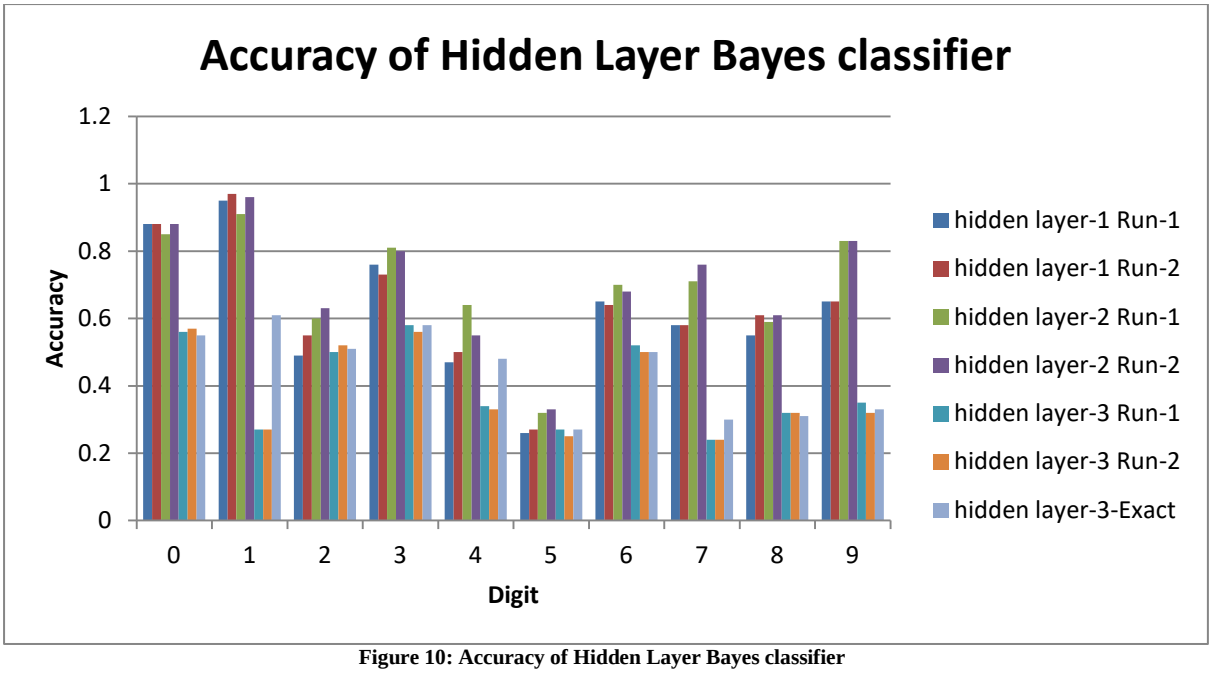

Figure 10: Accuracy of Hidden Layer Bayes classifier

The confusion matrix for the hidden layer-2 classifier for the second trial is shown in Table 4. This also shows some important information. The confusion matrix is constructed as follows. For each test sample, we get the label it was classified to and increment the corresponding column for the actual label row. As we expect from a good classifier, the diagonal should be filled with large numbers if the classifier is good. Indeed, this is what is seen in Table 4. The table also gives us information about why the classification accuracy of the digit 5 was only 33%. As can be seen in the table, the digit 5 was mostly mistaken for digit 3 in the testing dataset because 23 out of the 100 testing samples with label 5 was misclassified as 3. This is a very high number. This digit 5 was also mistaken for 8 and 9 for around 12 cases. Similarly, digit 4 was misclassified as 9 in large number of cases.

Table 4: Confusion matrix for hidden layer-2 classifier run-2

Digit	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	88	0	1	0	0	1	2	0	6	2
1	0	96	0	1	0	2	1	0	0	0
2	5	3	63	12	0	0	5	2	8	2
3	2	4	1	80	1	2	0	5	2	3
4	4	1	2	2	55	1	5	3	3	24
5	7	5	0	23	3	33	3	2	13	11
6	14	4	2	0	3	5	68	0	4	0
7	0	5	1	1	4	0	0	76	0	13
8	6	5	2	11	3	6	0	3	61	3
9	0	0	0	3	3	0	0	8	3	83

5. Conclusion

A handwritten digit classifier was implemented in this project. Two types of classifiers were considered; naive bayes and hidden layer bayes classifier. A MAP estimation procedure was used for inference in the naive bayes classifier. An EM procedure to estimate the values of the hidden nodes was used in the hidden layer classifiers and to subsequently estimate the distribution parameters. This was followed by likelihood weighted sampling to perform approximate inference. The exact inference was also performed for one of the implementations of the hidden layer bayes classifier. The single layer naive bayes classifier was highly accurate and achieved a classification accuracy of around 80%. The two layer bayes classifier also achieved similar accuracy. In comparison, the best hidden layer bayes classifier with 64 hidden nodes could achieve only 70% on average. The drop in accuracy can be attributed to the EM method's two problems; getting stuck in local optima based on the initialization and low number of EM iterations. These were inevitable due to the huge time complexity. We believe with enough trials and enough iterations, these problems could be solved. Another improvement could be adding more links in the network among the hidden nodes to make it more complex.

References

[1] Prof. Qiang Ji, ECSE 6810 Lecture notes
[2] Yogish Didgi, ECSE 6810 Reports
[3] Y. LeCun, C. Cortes, C.J.C. Burges, “The MNIST database of handwritten digits”, http://yann.lecun.com/exdb/mnist/

Appendix

A.1 Naive Bayes classifier - 1 layer, 28x28 pixel nodes, 1 label node

```
function naiveBayes
clear all;

load 'labelDataSubset_Train.mat';%Goes from 0 to 9
load 'pixelDataSubset_Train.mat';
load 'labelDataSubset_Test.mat';
load 'pixelDataSubset_Test.mat';

%Threshold the pixel values
pixelDataSubset_Train = (pixelDataSubset_Train > 127).*1;%Scale to 255 for display
pixelDataSubset_Test = (pixelDataSubset_Test > 127).*1;
% pixelDataSubset_Train = resizem(pixelDataSubset_Train,[size(pixelDataSubset_Train,1) 16]);
% pixelDataSubset_Test = resizem(pixelDataSubset_Test,[size(pixelDataSubset_Test,1) 16]);

numChars = 10;
numTrainingSamples = size(pixelDataSubset_Train,1);
numTestingSamples = size(pixelDataSubset_Test,1);
numNodes = size(pixelDataSubset_Train,2);%Number of pixel nodes
numSamplesLimitTrain = 1000;%Per digit
numSamplesLimitTest = 100;

CPD_Count = zeros(numChars, numNodes);
CPD = zeros(numChars, numNodes, 2);

fprintf('Starting training ...\r');
%ML estimate
for i = 1:numTrainingSamples
    CPD_Count(labelDataSubset_Train(i)+1,:) = CPD_Count(labelDataSubset_Train(i)+1,:) +
pixelDataSubset_Train(i,:);
end
%Those whose values were 0, assign them to be a very low non-zero value;
%uniform is giving low accuracy
CPD_Count(find(CPD_Count == 0)) = 10;%numSamplesLimitTrain/2;
CPD(:,:,2) = CPD_Count/numSamplesLimitTrain;
CPD(:,:,1) = 1 - CPD(:,:,2);

fprintf('Starting testing ...\r');
AccuracyCount = zeros(numChars,2);
ConfusionMatrix = zeros(numChars,numChars);
%Classification
for i = 1:numTestingSamples
    for class = 1:numChars
        pbty(class,:) = pixelDataSubset_Test(i,:).*CPD(class,:,2) + (1 -
pixelDataSubset_Test(i,:)).*CPD(class,:,1);
    end
    %pbtyLog = log(pbty);
    %jointPbty = sum(pbtyLog,2);
    jointPbty = prod(pbty,2);
    [value, classID] = max(jointPbty);
    if((classID-1) == labelDataSubset_Test(i))
        AccuracyCount(labelDataSubset_Test(i)+1,1) =
AccuracyCount(labelDataSubset_Test(i)+1,1) + 1;
    else
        AccuracyCount(labelDataSubset_Test(i)+1,2) =
AccuracyCount(labelDataSubset_Test(i)+1,2) + 1;
    end
    ConfusionMatrix(labelDataSubset_Test(i)+1,classID) =
ConfusionMatrix(labelDataSubset_Test(i)+1,classID) + 1;
end
Accuracy = AccuracyCount/numSamplesLimitTest;
fprintf('Accuracy ...\r');
for class = 1:numChars
    fprintf('Char: %d\tCorrect: %d\tIncorrect: %d\tPercent: %g\r',class-
1,AccuracyCount(class,1),AccuracyCount(class,2),Accuracy(class));
end

%Confusion matrix
for i = 1:numChars
    fprintf('%d:\r\t',i-1);
    for j = 1:numChars
        fprintf('%d:%d; ',j-1,ConfusionMatrix(i,j));
    end
    fprintf('\r');
```

```
end
keyboard;
end
```

A.2 Naive Bayes classifier - 2 layer, 28x28 pixel nodes, 14x14 accumulation nodes, 1 label node

```
function naiveBayes_2Layer
clear all;

load 'labelDataSubset_Train.mat';%Goes from 0 to 9
load 'pixelDataSubset_Train.mat';
load 'labelDataSubset_Test.mat';
load 'pixelDataSubset_Test.mat';

%Threshold the pixel values
pixelDataSubset_Train = (pixelDataSubset_Train > 127).*1;%Scale to 255 for display
pixelDataSubset_Test = (pixelDataSubset_Test > 127).*1;
% pixelDataSubset_Train = resizem(pixelDataSubset_Train,[size(pixelDataSubset_Train,1) 16]);
% pixelDataSubset_Test = resizem(pixelDataSubset_Test,[size(pixelDataSubset_Test,1) 16]);

numChars = 10;
numTrainingSamples = size(pixelDataSubset_Train,1);
numTestingSamples = size(pixelDataSubset_Test,1);
numNodes_L1 = size(pixelDataSubset_Train,2)/4;%Number of feature nodes
numNodes_L2 = size(pixelDataSubset_Train,2);%Number of pixel nodes
numSamplesLimitTrain = 1000;%Per digit
numSamplesLimitTest = 100;
charWidth = sqrt(numNodes_L2); charHeight = sqrt(numNodes_L2);

CPD_Count_L1 = zeros(numChars, numNodes_L1);
CPD_L1 = zeros(numChars, numNodes_L1, 2);
CPD_Count_L2 = zeros(2, numNodes_L2);
CPD_L2 = zeros(2, numNodes_L2, 2);
numChildren = numNodes_L2/numNodes_L1;

fprintf('Starting training ...\r');
if(0)
    %Rearrange pixel data for easy access
    tic;
    pixelDataSubset_TrainNew = pixelDataSubset_Train;
    for i = 1:numTrainingSamples
        pixelDataSubset_TrainNew(i,:) = rearrangePixelData(pixelDataSubset_Train(i,:),
numNodes_L1,charWidth,charHeight);
    end
    toc;
    tic;
    pixelDataSubset_TestNew = pixelDataSubset_Test;
    for i = 1:numTestingSamples
        pixelDataSubset_TestNew(i,:) = rearrangePixelData(pixelDataSubset_Test(i,:),
numNodes_L1,charWidth,charHeight);
    end
    toc;
    save('pixelDataSubset_TrainRearrange.mat','pixelDataSubset_TrainNew');
    save('pixelDataSubset_TestRearrange.mat','pixelDataSubset_TestNew');
else
    load 'pixelDataSubset_TrainRearrange.mat';
    load 'pixelDataSubset_TestRearrange.mat';
end

pixelDataTrain_L1 = zeros(numTrainingSamples,numNodes_L1);%Hidden/Average of L2
pixelDataTrain_L2 = pixelDataSubset_TrainNew;%pixelDataSubset_Train;

%ML estimate
for i = 1:numTrainingSamples
    %Determine L1 nodes' values
    for j = 1:numNodes_L1
        pixelDataTrain_L1(i,j) = ceil(sum(pixelDataTrain_L2(i,(j-1)*numChildren + 1:(j-
1)*numChildren + numChildren))/numChildren);
    end
    CPD_Count_L1(labelDataSubset_Train(i)+1,:) = CPD_Count_L1(labelDataSubset_Train(i)+1,:) +
pixelDataTrain_L1(i,:);
    for j = 1:numNodes_L2
        parentID = floor((j-1)/numChildren)+1;
        CPD_Count_L2(pixelDataTrain_L2(i,parentID)+1,j) =
CPD_Count_L2(pixelDataTrain_L2(i,parentID)+1,j) + pixelDataTrain_L2(i,j);
    end
end
```

```
%Those whose values were 0, assign them to be a very low non-zero value;
%uniform is giving low accuracy
CPD_Count_L1(find(CPD_Count_L1 == 0)) = 10;%numSamplesLimitTrain/2;
CPD_Count_L2(find(CPD_Count_L2 == 0)) = 10;%numSamplesLimitTrain/2;

CPD_L1(:,:,2) = CPD_Count_L1/numSamplesLimitTrain;
CPD_L1(:,:,1) = 1 - CPD_L1(:,:,2);
CPD_L2(:,:,2) = CPD_Count_L2/numSamplesLimitTrain;
CPD_L2(:,:,1) = 1 - CPD_L2(:,:,2);

fprintf('Starting testing ...\r');
AccuracyCount = zeros(numChars,2);
ConfusionMatrix = zeros(numChars,numChars);
%Classification
pixelDataTest_L1 = zeros(numTestingSamples,numNodes_L1);%Hidden/Average of L2
pixelDataTest_L2 = pixelDataSubset_TestNew;%pixelDataSubset_Test

for i = numTestingSamples:-1:1
    %Determine L1 nodes' values
    for j = 1:numNodes_L1
        pixelDataTest_L1(i,j) = ceil(sum(pixelDataTest_L2(i,(j-1)*numChildren + 1:(j-
1)*numChildren + numChildren))/numChildren);
    end
    for class = 1:numChars
        pbty(class,:) = pixelDataTest_L1(i,:).*CPD_L1(class,:,2) + (1 -
pixelDataTest_L1(i,:)).*CPD_L1(class,:,1);
    end
    %pbtyLog = log(pbty);
    %jointPbty = sum(pbtyLog,2);
    jointPbty = prod(pbty,2);
    [value, classID] = max(jointPbty);
    if((classID-1) == labelDataSubset_Test(i))
        AccuracyCount(labelDataSubset_Test(i)+1,1) =
AccuracyCount(labelDataSubset_Test(i)+1,1) + 1;
    else
        AccuracyCount(labelDataSubset_Test(i)+1,2) =
AccuracyCount(labelDataSubset_Test(i)+1,2) + 1;
    end
    ConfusionMatrix(labelDataSubset_Test(i)+1,classID) =
ConfusionMatrix(labelDataSubset_Test(i)+1,classID) + 1;
end
Accuracy = AccuracyCount/numSamplesLimitTest;
fprintf('Accuracy ...\r');
for class = 1:numChars
    fprintf('Char: %d\tCorrect: %d\tIncorrect: %d\tPercent: %g\r',class-
1,AccuracyCount(class,1),AccuracyCount(class,2),Accuracy(class));
end

%Confusion matrix
for i = 1:numChars
    fprintf('%d:\r\t',i-1);
    for j = 1:numChars
        fprintf('%d:%d; ',j-1,ConfusionMatrix(i,j));
    end
    fprintf('\r');
end
keyboard;
end

%Rearrange pixel data for easy access so that the children nodes are
%accesible easily (next to each other in the new array)
function pixelDataNew = rearrangePixelData(pixelData, numHiddenNodes,charWidth,charHeight)
pixelDataNew = pixelData';
numPixelNodes = size(pixelDataNew,1);
numChildren = numPixelNodes/numHiddenNodes;
widthPixel = sqrt(numPixelNodes);
heightPixel = sqrt(numPixelNodes);

pixelDataMat = reshape(pixelData,charWidth,charHeight)';

widthHidden = sqrt(numHiddenNodes);
lengthChild = sqrt(numChildren);
index = 1;
for i = 1:numHiddenNodes
    quotient = floor((i-1)/widthHidden);
    reminder = mod(i-1,widthHidden);
    [X,Y] = ndgrid(lengthChild*quotient+1:lengthChild*quotient+lengthChild, ...
```

```
                    lengthChild*reminder+1:lengthChild*reminder+lengthChild);
    index = sub2ind([widthPixel,heightPixel],X,Y);
    pixelDataNew((i-1)*numChildren+1:(i-1)*numChildren+numChildren) = pixelData(index);
    index = index + 1;
end
end
```

A.3 Naive Bayes classifier - 3 layer, 28x28 pixel nodes, 14x14, 7x7 accumulation nodes, 1 label node

```
function naiveBayes_3Layer
clear all;

load 'labelDataSubset_Train.mat';%Goes from 0 to 9
load 'pixelDataSubset_Train.mat';
load 'labelDataSubset_Test.mat';
load 'pixelDataSubset_Test.mat';

%Threshold the pixel values
pixelDataSubset_Train = (pixelDataSubset_Train > 127).*1;%Scale to 255 for display
pixelDataSubset_Test = (pixelDataSubset_Test > 127).*1;
% pixelDataSubset_Train = resizem(pixelDataSubset_Train,[size(pixelDataSubset_Train,1) 16]);
% pixelDataSubset_Test = resizem(pixelDataSubset_Test,[size(pixelDataSubset_Test,1) 16]);

numChars = 10;
numTrainingSamples = size(pixelDataSubset_Train,1);
numTestingSamples = size(pixelDataSubset_Test,1);
numNodes_L1 = size(pixelDataSubset_Train,2);%Number of pixel nodes
numNodes_L2 = size(pixelDataSubset_Train,2)/4;%Number of feature nodes in L2
numNodes_L3 = size(pixelDataSubset_Train,2)/16;%Number of feature nodes in L3
numSamplesLimitTrain = 1000;%Per digit
numSamplesLimitTest = 100;
charWidth = sqrt(numNodes_L1); charHeight = sqrt(numNodes_L1);

CPD_Count_L1 = zeros(2, numNodes_L1);
CPD_L1 = zeros(2, numNodes_L1, 2);
CPD_Count_L2 = zeros(2, numNodes_L2);
CPD_L2 = zeros(numChars, numNodes_L2, 2);
CPD_Count_L3 = zeros(numChars, numNodes_L3);
CPD_L3 = zeros(numChars, numNodes_L3, 2);
numChildren_L1_L2 = numNodes_L1/numNodes_L2;
numChildren_L2_L3 = numNodes_L2/numNodes_L3;

fprintf('Starting training ...\r');
if(0)
    %Rearrange pixel data for easy access
    tic;
    pixelDataSubset_TrainNew = pixelDataSubset_Train;
    for i = 1:numTrainingSamples
        pixelDataSubset_TrainNew(i,:) = rearrangePixelData(pixelDataSubset_Train(i,:),
numNodes_L1,charWidth,charHeight);
    end
    toc;
    tic;
    pixelDataSubset_TestNew = pixelDataSubset_Test;
    for i = 1:numTestingSamples
        pixelDataSubset_TestNew(i,:) = rearrangePixelData(pixelDataSubset_Test(i,:),
numNodes_L1,charWidth,charHeight);
    end
    toc;
    save('pixelDataSubset_TrainRearrange.mat','pixelDataSubset_TrainNew');
    save('pixelDataSubset_TestRearrange.mat','pixelDataSubset_TestNew');
else
    load 'pixelDataSubset_TrainRearrange.mat';
    load 'pixelDataSubset_TestRearrange.mat';
end

pixelDataTrain_L1 = pixelDataSubset_TrainNew;%pixelDataSubset_Train;
pixelDataTrain_L2 = zeros(numTrainingSamples,numNodes_L2);%Hidden/Average of L2
pixelDataTrain_L3 = zeros(numTrainingSamples,numNodes_L3);%Hidden/Average of L3

%ML estimate
for i = 1:numTrainingSamples
    %Determine L2 nodes' values
    for j = 1:numNodes_L2
        pixelDataTrain_L2(i,j) = ceil(sum(pixelDataTrain_L1(i,(j-1)*numChildren_L1_L2 + 1:(j-
1)*numChildren_L1_L2 + numChildren_L1_L2))/numChildren_L1_L2);
    end
```

```
    CPD_Count_L2(labelDataSubset_Train(i)+1,:) = CPD_Count_L2(labelDataSubset_Train(i)+1,:) +
pixelDataTrain_L2(i,:);
    %Determine L3 nodes' values
    for j = 1:numNodes_L3
        pixelDataTrain_L3(i,j) = ceil(sum(pixelDataTrain_L2(i,(j-1)*numChildren_L2_L3 + 1:(j-
1)*numChildren_L2_L3 + numChildren_L2_L3))/numChildren_L2_L3);
    end
    CPD_Count_L3(labelDataSubset_Train(i)+1,:) = CPD_Count_L3(labelDataSubset_Train(i)+1,:) +
pixelDataTrain_L3(i,:);
    for j = 1:numNodes_L1
        parentID = floor((j-1)/numChildren_L1_L2)+1;
        CPD_Count_L1(pixelDataTrain_L1(i,parentID)+1,j) =
CPD_Count_L1(pixelDataTrain_L1(i,parentID)+1,j) + pixelDataTrain_L1(i,j);
    end
end
%Those whose values were 0, assign them to be a very low non-zero value;
%uniform is giving low accuracy
CPD_Count_L1(find(CPD_Count_L1 == 0)) = 10;%numSamplesLimitTrain/2;
CPD_Count_L2(find(CPD_Count_L2 == 0)) = 10;%numSamplesLimitTrain/2;
CPD_Count_L3(find(CPD_Count_L3 == 0)) = 10;%numSamplesLimitTrain/2;

CPD_L1(:,:,2) = CPD_Count_L1/numSamplesLimitTrain;
CPD_L1(:,:,1) = 1 - CPD_L1(:,:,2);
CPD_L2(:,:,2) = CPD_Count_L2/numSamplesLimitTrain;
CPD_L2(:,:,1) = 1 - CPD_L2(:,:,2);
CPD_L3(:,:,2) = CPD_Count_L3/numSamplesLimitTrain;
CPD_L3(:,:,1) = 1 - CPD_L3(:,:,2);

fprintf('Starting testing ...\r');
AccuracyCount = zeros(numChars,2);
ConfusionMatrix = zeros(numChars,numChars);
%Classification
pixelDataTest_L1 = pixelDataSubset_TestNew;%pixelDataSubset_Test
pixelDataTest_L2 = zeros(numTestingSamples,numNodes_L2);%Hidden/Average of L2
pixelDataTest_L3 = zeros(numTestingSamples,numNodes_L3);%Hidden/Average of L3

for i = numTestingSamples:-1:1
    %Determine L2 nodes' values
    for j = 1:numNodes_L2
        pixelDataTest_L2(i,j) = ceil(sum(pixelDataTest_L1(i,(j-1)*numChildren_L1_L2 + 1:(j-
1)*numChildren_L1_L2 + numChildren_L1_L2))/numChildren_L1_L2);
    end
    %Determine L3 nodes' values
    for j = 1:numNodes_L3
        pixelDataTest_L3(i,j) = ceil(sum(pixelDataTest_L2(i,(j-1)*numChildren_L2_L3 + 1:(j-
1)*numChildren_L2_L3 + numChildren_L2_L3))/numChildren_L2_L3);
    end
    for class = 1:numChars
        pbty(class,:) = pixelDataTest_L3(i,:).*CPD_L3(class,:,2) + (1 -
pixelDataTest_L3(i,:)).*CPD_L3(class,:,1);
    end
    %pbtyLog = log(pbty);
    %jointPbty = sum(pbtyLog,2);
    jointPbty = prod(pbty,2);
    [value, classID] = max(jointPbty);
    if((classID-1) == labelDataSubset_Test(i))
        AccuracyCount(labelDataSubset_Test(i)+1,1) =
AccuracyCount(labelDataSubset_Test(i)+1,1) + 1;
    else
        AccuracyCount(labelDataSubset_Test(i)+1,2) =
AccuracyCount(labelDataSubset_Test(i)+1,2) + 1;
    end
    ConfusionMatrix(labelDataSubset_Test(i)+1,classID) =
ConfusionMatrix(labelDataSubset_Test(i)+1,classID) + 1;
end
Accuracy = AccuracyCount/numSamplesLimitTest;
fprintf('Accuracy ...\r');
for class = 1:numChars
    fprintf('Char: %d\tCorrect: %d\tIncorrect: %d\tPercent: %g\r',class-
1,AccuracyCount(class,1),AccuracyCount(class,2),Accuracy(class));
end

%Confusion matrix
for i = 1:numChars
    fprintf('%d:\r\t',i-1);
    for j = 1:numChars
        fprintf('%d:%d; ',j-1,ConfusionMatrix(i,j));
```

```
    end
    fprintf('\r');
end
keyboard;
end

%Rearrange pixel data for easy access so that the children nodes are
%accesible easily (next to each other in the new array)
function pixelDataNew = rearrangePixelData(pixelData, numHiddenNodes,charWidth,charHeight)
pixelDataNew = pixelData';
numPixelNodes = size(pixelDataNew,1);
numChildren = numPixelNodes/numHiddenNodes;
widthPixel = sqrt(numPixelNodes);
heightPixel = sqrt(numPixelNodes);

pixelDataMat = reshape(pixelData,charWidth,charHeight)';

widthHidden = sqrt(numHiddenNodes);
lengthChild = sqrt(numChildren);
index = 1;
for i = 1:numHiddenNodes
    quotient = floor((i-1)/widthHidden);
    reminder = mod(i-1,widthHidden);
    [X,Y] = ndgrid(lengthChild*quotient+1:lengthChild*quotient+lengthChild, ...
                   lengthChild*reminder+1:lengthChild*reminder+lengthChild);
    index = sub2ind([widthPixel,heightPixel],X,Y);
    pixelDataNew((i-1)*numChildren+1:(i-1)*numChildren+numChildren) = pixelData(index);
    index = index + 1;
end
end
```

A.4 Naive Bayes classifier - 1 layer, resize factor enabled

```
function naiveBayesWithResize
clear all;

load 'labelDataSubset_Train.mat';%Goes from 0 to 9
load 'pixelDataSubset_Train.mat';
load 'labelDataSubset_Test.mat';
load 'pixelDataSubset_Test.mat';

%Resize factor
resizeValue = 16;
resizeWidth = resizeValue;
resizeHeight = resizeValue;

numChars = 10;
numTrainingSamples = size(pixelDataSubset_Train,1);
numTestingSamples = size(pixelDataSubset_Test,1);
numNodes_Orig = size(pixelDataSubset_Train,2);
numSamplesLimitTrain = 1000;%Per digit
numSamplesLimitTest = 100;
charWidth = sqrt(numNodes_Orig); charHeight = sqrt(numNodes_Orig);
numNodes = resizeWidth*resizeHeight;
CPD_Count = zeros(numChars, numNodes);%Number of pixel nodes
CPD = zeros(numChars, numNodes, 2);

%Rearrange pixel data for easy access
for i = 1:numTrainingSamples
    trainSample = reshape(imresize(reshape(pixelDataSubset_Train(i,:),[charWidth
charHeight])',[resizeWidth resizeHeight])',[1 numNodes]);
    pixelDataSubset_TrainNew(i,:) = rearrangePixelData(trainSample,
numNodes,resizeWidth,resizeHeight);
end

for i = 1:numTestingSamples
    testSample = reshape(imresize(reshape(pixelDataSubset_Test(i,:),[charWidth
charHeight])',[resizeWidth resizeHeight])',[1 numNodes]);
    pixelDataSubset_TestNew(i,:) = rearrangePixelData(testSample,
numNodes,resizeWidth,resizeHeight);
end

%Threshold the pixel values
pixelDataSubset_TrainNew = (pixelDataSubset_TrainNew > 127).*1;%Scale to 255 for display
pixelDataSubset_TestNew = (pixelDataSubset_TestNew > 127).*1;

fprintf('Starting training ...\r');
```

```
%ML estimate
for i = 1:numTrainingSamples
    CPD_Count(labelDataSubset_Train(i)+1,:) = CPD_Count(labelDataSubset_Train(i)+1,:) +
pixelDataSubset_TrainNew(i,:);
end
%Those whose values were 0, assign them to be a very low non-zero value;
%uniform is giving low accuracy
CPD_Count(find(CPD_Count == 0)) = 10;%numSamplesLimitTrain/2;
CPD(:,:,2) = CPD_Count/numSamplesLimitTrain;
CPD(:,:,1) = 1 - CPD(:,:,2);

fprintf('Starting testing ...\r');
AccuracyCount = zeros(numChars,2);
ConfusionMatrix = zeros(numChars,numChars);
%Classification
for i = 1:numTestingSamples
    for class = 1:numChars
        pbty(class,:) = pixelDataSubset_TestNew(i,:).*CPD(class,:,2) + (1 -
pixelDataSubset_TestNew(i,:)).*CPD(class,:,1);
    end
    %pbtyLog = log(pbty);
    %jointPbty = sum(pbtyLog,2);
    jointPbty = prod(pbty,2);
    [value, classID] = max(jointPbty);
    if((classID-1) == labelDataSubset_Test(i))
        AccuracyCount(labelDataSubset_Test(i)+1,1) =
AccuracyCount(labelDataSubset_Test(i)+1,1) + 1;
    else
        AccuracyCount(labelDataSubset_Test(i)+1,2) =
AccuracyCount(labelDataSubset_Test(i)+1,2) + 1;
    end
    ConfusionMatrix(labelDataSubset_Test(i)+1,classID) =
ConfusionMatrix(labelDataSubset_Test(i)+1,classID) + 1;
end
Accuracy = AccuracyCount/numSamplesLimitTest;
fprintf('Accuracy ...\r');
for class = 1:numChars
    fprintf('Char: %d\tCorrect: %d\tIncorrect: %d\tPercent: %g\r',class-
1,AccuracyCount(class,1),AccuracyCount(class,2),Accuracy(class));
end

%Confusion matrix
for i = 1:numChars
    fprintf('%d:\r\t',i-1);
    for j = 1:numChars
        fprintf('%d:%d; ',j-1,ConfusionMatrix(i,j));
    end
    fprintf('\r');
end
keyboard;
end

%Rearrange pixel data for easy access so that the children nodes are
%accesible easily (next to each other in the new array)
function pixelDataNew = rearrangePixelData(pixelData, numHiddenNodes,charWidth,charHeight)
pixelDataNew = pixelData';
numPixelNodes = size(pixelDataNew,1);
numChildren = numPixelNodes/numHiddenNodes;
widthPixel = sqrt(numPixelNodes);
heightPixel = sqrt(numPixelNodes);

pixelDataMat = reshape(pixelData,charWidth,charHeight)';

widthHidden = sqrt(numHiddenNodes);
lengthChild = sqrt(numChildren);
index = 1;
for i = 1:numHiddenNodes
    quotient = floor((i-1)/widthHidden);
    reminder = mod(i-1,widthHidden);
    [X,Y] = ndgrid(lengthChild*quotient+1:lengthChild*quotient+lengthChild, ...
                   lengthChild*reminder+1:lengthChild*reminder+lengthChild);
    index = sub2ind([widthPixel,heightPixel],X,Y);
    pixelDataNew((i-1)*numChildren+1:(i-1)*numChildren+numChildren) = pixelData(index);
    index = index + 1;
end
end
```

A.5 Bayesian classifier with hidden nodes - 2 layer, 28x28 pixel nodes, 14x14 hidden nodes, 1 label node

```
function hiddenLayerBayes_28x28_EMChanges
clear all;

load 'labelDataSubset1_Train.mat';%Goes from 0 to 9
load 'pixelDataSubset1_Train.mat';
load 'labelDataSubset1_Test.mat';
load 'pixelDataSubset1_Test.mat';
load 'pixelDataSubset1_TrainRearrange_28x28.mat';
load 'pixelDataSubset1_TestRearrange_28x28.mat';

numChars = 10;
charWidth = 28; charHeight = 28;
numHiddenNodes = charWidth*charHeight/4;
numPixelNodes = charWidth*charHeight;
numChildren = numPixelNodes/numHiddenNodes;
numTrainingSamples = size(pixelDataSubset_Train,1);
numTestingSamples = size(pixelDataSubset_Test,1);

CPD_Hidden = zeros(numChars, numHiddenNodes, 2);%binary-valued hidden state
CPD_Hidden_Count = zeros(numChars, numHiddenNodes, 2);
CPD_Pixel = zeros(2, numPixelNodes, 2);
CPD_Pixel_Count = zeros(2, numPixelNodes, 2);%num of parent states=2
weightVct = zeros(numHiddenNodes,2*numTrainingSamples);
numEMIters = 10000;%2000;
numInferenceIters = 10000;

% keyboard;
%Init-random
CPD_Hidden(:,:,1) = rand(numChars, numHiddenNodes);
CPD_Hidden(:,:,2) = 1 - CPD_Hidden(:,:,1);
CPD_Pixel(:,:,1) = rand(2, numPixelNodes);
CPD_Pixel(:,:,2) = 1 - CPD_Pixel(:,:,1);

fprintf('Starting training ...\r');
for iter = 1:numEMIters
    fprintf('Iter: %d out of %d ... ', iter, numEMIters);
    %{
    NOTE: !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!! IMPORTANT
    1. Initialize counts everytime?
    2. Previous iteration weight is not used in the E-code
    %}
    CPD_Hidden_Count = zeros(numChars, numHiddenNodes, 2);
    CPD_Pixel_Count = zeros(2, numPixelNodes, 2);%num of parent states=2
    tic;
    weightVct =
computeExpectation(pixelDataSubset_TrainNew,labelDataSubset_Train,CPD_Pixel,CPD_Hidden,weightV
ct);
    [CPD_Pixel_Count,CPD_Pixel,CPD_Hidden_Count,CPD_Hidden] =
computeMaximization(pixelDataSubset_TrainNew,labelDataSubset_Train,...

weightVct,CPD_Pixel_Count,CPD_Pixel,CPD_Hidden_Count,CPD_Hidden);
    t1 = toc;
    fprintf('%g sec\r',t1);
%     reminder = mod(iter,1000);
%     if(reminder == 0)
%         CPD_string = sprintf('CPD_Pixel_EM_28x28_new_%d.mat',iter);
%         CPD_Pixel_EM = CPD_Pixel;
%         save(CPD_string,'CPD_Pixel_EM');
%         CPD_string = sprintf('CPD_Hidden_EM_28x28_new_%d.mat',iter);
%         CPD_Hidden_EM = CPD_Hidden;
%         save(CPD_string,'CPD_Hidden_EM');
%     end
end

fprintf('Starting testing %d iters (approx.)... ',numInferenceIters);
%keyboard;
% ConfusionMatrix = zeros(numChars,numChars);
tic;
ConfusionMatrix1 = approxInference(pixelDataSubset_TestNew, labelDataSubset_Test, CPD_Pixel,
CPD_Hidden, numInferenceIters);
t1 = toc;
fprintf('%g sec\r\n',t1);
for i = 1:numChars
    fprintf('%d: Accuracy: %g\r\t',i-1,ConfusionMatrix1(i,i)/sum(ConfusionMatrix1(i,:)));
```

```
    for j = 1:numChars
        fprintf('%d:%d; ',j-1,ConfusionMatrix1(i,j));
    end
    fprintf('\r');
end
keyboard;
end

%Function to perform expectation of the hidden nodes, return the weight
%vector
function weightVct =
computeExpectation(pixelDataSubset_TrainNew,labelDataSubset_Train,CPD_Pixel,CPD_Hidden,weightV
ct)
numTrainingSamples = size(pixelDataSubset_TrainNew,1);
numPixelNodes = size(CPD_Pixel,2);
numHiddenNodes = size(CPD_Hidden,2);
numChildren = numPixelNodes/numHiddenNodes;

for sampleID = 1:numTrainingSamples
    for hiddenNodeID = 1:numHiddenNodes
        %Compute the weight for two possible states of the hidden node
        pbty_0 = CPD_Hidden(labelDataSubset_Train(sampleID)+1,hiddenNodeID,1);
        pbty_1 = CPD_Hidden(labelDataSubset_Train(sampleID)+1,hiddenNodeID,2);
        for i = 1:numChildren
            pbty_0 = pbty_0*CPD_Pixel(1,(hiddenNodeID-
1)*numChildren+i,pixelDataSubset_TrainNew(sampleID,(hiddenNodeID-1)*numChildren+i)+1);
            pbty_1 = pbty_1*CPD_Pixel(2,(hiddenNodeID-
1)*numChildren+i,pixelDataSubset_TrainNew(sampleID,(hiddenNodeID-1)*numChildren+i)+1);
        end
        pbty_sum = pbty_0 + pbty_1;
        weightVct(hiddenNodeID,sampleID) = pbty_0/pbty_sum;%+
weightVct(hiddenNodeID,sampleID);
        weightVct(hiddenNodeID,sampleID+numTrainingSamples) = pbty_1/pbty_sum;%+
weightVct(hiddenNodeID,sampleID+numTrainingSamples);
    end
end
end

%Perform maximization; Compute the distribution parameters as a counting
%problem from the weight vector
function [CPD_Pixel_Count,CPD_Pixel,CPD_Hidden_Count,CPD_Hidden] =
computeMaximization(pixelDataSubset_TrainNew,labelDataSubset_Train,...

weightVct,CPD_Pixel_Count,CPD_Pixel,CPD_Hidden_Count,CPD_Hidden)
numTrainingSamples = size(pixelDataSubset_TrainNew,1);
numPixelNodes = size(CPD_Pixel,2);
numHiddenNodes = size(CPD_Hidden,2);
numChildren = numPixelNodes/numHiddenNodes;

for sampleID = 1:numTrainingSamples
    for hiddenNodeID = 1:numHiddenNodes
        CPD_Hidden_Count(labelDataSubset_Train(sampleID)+1,hiddenNodeID,1) =
CPD_Hidden_Count(labelDataSubset_Train(sampleID)+1,hiddenNodeID,1) ...
                                                                    +
weightVct(hiddenNodeID,sampleID);
        CPD_Hidden_Count(labelDataSubset_Train(sampleID)+1,hiddenNodeID,2) =
CPD_Hidden_Count(labelDataSubset_Train(sampleID)+1,hiddenNodeID,2) ...
                                                                    +
weightVct(hiddenNodeID,sampleID+numTrainingSamples);
        for i = 1:numChildren
            CPD_Pixel_Count(1,(hiddenNodeID-
1)*numChildren+i,pixelDataSubset_TrainNew(sampleID,(hiddenNodeID-1)*numChildren+i)+1) = ...
                CPD_Pixel_Count(1,(hiddenNodeID-
1)*numChildren+i,pixelDataSubset_TrainNew(sampleID,(hiddenNodeID-1)*numChildren+i)+1) ...
                + weightVct(hiddenNodeID,sampleID);
            CPD_Pixel_Count(2,(hiddenNodeID-
1)*numChildren+i,pixelDataSubset_TrainNew(sampleID,(hiddenNodeID-1)*numChildren+i)+1) = ...
                CPD_Pixel_Count(2,(hiddenNodeID-
1)*numChildren+i,pixelDataSubset_TrainNew(sampleID,(hiddenNodeID-1)*numChildren+i)+1) ...
                + weightVct(hiddenNodeID,sampleID+numTrainingSamples);
        end
    end
end
for hiddenNodeID = 1:numHiddenNodes
    normalizationFactor = CPD_Hidden_Count(:,hiddenNodeID,1) +
CPD_Hidden_Count(:,hiddenNodeID,2);
    CPD_Hidden(:,hiddenNodeID,1) = CPD_Hidden_Count(:,hiddenNodeID,1)./normalizationFactor;
```

```
    CPD_Hidden(:,hiddenNodeID,2) = CPD_Hidden_Count(:,hiddenNodeID,2)./normalizationFactor;
    for i = 1:numChildren
        normalizationFactor = CPD_Pixel_Count(:,(hiddenNodeID-1)*numChildren+i,1) +
CPD_Pixel_Count(:,(hiddenNodeID-1)*numChildren+i,2);
        CPD_Pixel(:,(hiddenNodeID-1)*numChildren+i,1) = CPD_Pixel_Count(:,(hiddenNodeID-
1)*numChildren+i,1)./normalizationFactor;
        CPD_Pixel(:,(hiddenNodeID-1)*numChildren+i,2) = CPD_Pixel_Count(:,(hiddenNodeID-
1)*numChildren+i,2)./normalizationFactor;
    end
end
end
```

A.6 Bayesian classifier with hidden nodes - 2 layer, 16x16 pixel nodes, 8x8 hidden nodes, 1 label node

```
function hiddenLayerBayes_16x16_EMChanges
clear all;

load 'labelDataSubset1_Train.mat';%Goes from 0 to 9
load 'pixelDataSubset1_Train.mat';
load 'labelDataSubset1_Test.mat';
load 'pixelDataSubset1_Test.mat';
load 'pixelDataSubset1_TrainRearrange_16x16.mat';
load 'pixelDataSubset1_TestRearrange_16x16.mat';

numChars = 10;
charWidth = 16; charHeight = 16;
numHiddenNodes = charWidth*charHeight/4;
numPixelNodes = charWidth*charHeight;
numChildren = numPixelNodes/numHiddenNodes;
numTrainingSamples = size(pixelDataSubset_Train,1);
numTestingSamples = size(pixelDataSubset_Test,1);

CPD_Hidden = zeros(numChars, numHiddenNodes, 2);%binary-valued hidden state
CPD_Hidden_Count = zeros(numChars, numHiddenNodes, 2);
CPD_Pixel = zeros(2, numPixelNodes, 2);
CPD_Pixel_Count = zeros(2, numPixelNodes, 2);%num of parent states=2
weightVct = zeros(numHiddenNodes,2*numTrainingSamples);
numEMIters = 10000;%2000;
numInferenceIters = 10000;

% keyboard;
%Init-random
CPD_Hidden(:,:,1) = rand(numChars, numHiddenNodes);
CPD_Hidden(:,:,2) = 1 - CPD_Hidden(:,:,1);
CPD_Pixel(:,:,1) = rand(2, numPixelNodes);
CPD_Pixel(:,:,2) = 1 - CPD_Pixel(:,:,1);

fprintf('Starting training ...\r');
for iter = 1:numEMIters
    fprintf('Iter: %d out of %d ... ', iter, numEMIters);
    %{
    NOTE: !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!! IMPORTANT
    1. Initialize counts everytime?
    2. Previous iteration weight is not used in the E-code
    %}
    CPD_Hidden_Count = zeros(numChars, numHiddenNodes, 2);
    CPD_Pixel_Count = zeros(2, numPixelNodes, 2);%num of parent states=2
    tic;
    weightVct =
computeExpectation(pixelDataSubset_TrainNew,labelDataSubset_Train,CPD_Pixel,CPD_Hidden,weightV
ct);
    [CPD_Pixel_Count,CPD_Pixel,CPD_Hidden_Count,CPD_Hidden] =
computeMaximization(pixelDataSubset_TrainNew,labelDataSubset_Train,...

weightVct,CPD_Pixel_Count,CPD_Pixel,CPD_Hidden_Count,CPD_Hidden);
    t1 = toc;
    fprintf('%g sec\r',t1);
%     reminder = mod(iter,1000);
%     if(reminder == 0)
%         CPD_string = sprintf('CPD_Pixel_EM_16x16_new_%d.mat',iter);
%         CPD_Pixel_EM = CPD_Pixel;
%         save(CPD_string,'CPD_Pixel_EM');
%         CPD_string = sprintf('CPD_Hidden_EM_16x16_new_%d.mat',iter);
%         CPD_Hidden_EM = CPD_Hidden;
%         save(CPD_string,'CPD_Hidden_EM');
%     end
end
```

```
fprintf('Starting testing %d iters (approx.)... ',numInferenceIters);
%keyboard;
% ConfusionMatrix = zeros(numChars,numChars);
tic;
ConfusionMatrix1 = approxInference(pixelDataSubset_TestNew, labelDataSubset_Test, CPD_Pixel,
CPD_Hidden, numInferenceIters);
t1 = toc;
fprintf('%g sec\r\n',t1);
for i = 1:numChars
    fprintf('%d: Accuracy: %g\r\t',i-1,ConfusionMatrix1(i,i)/sum(ConfusionMatrix1(i,:)));
    for j = 1:numChars
        fprintf('%d:%d; ',j-1,ConfusionMatrix1(i,j));
    end
    fprintf('\r');
end
keyboard;
end

%Function to perform expectation of the hidden nodes, return the weight
%vector
function weightVct =
computeExpectation(pixelDataSubset_TrainNew,labelDataSubset_Train,CPD_Pixel,CPD_Hidden,weightV
ct)
numTrainingSamples = size(pixelDataSubset_TrainNew,1);
numPixelNodes = size(CPD_Pixel,2);
numHiddenNodes = size(CPD_Hidden,2);
numChildren = numPixelNodes/numHiddenNodes;

for sampleID = 1:numTrainingSamples
    for hiddenNodeID = 1:numHiddenNodes
        %Compute the weight for two possible states of the hidden node
        pbty_0 = CPD_Hidden(labelDataSubset_Train(sampleID)+1,hiddenNodeID,1);
        pbty_1 = CPD_Hidden(labelDataSubset_Train(sampleID)+1,hiddenNodeID,2);
        for i = 1:numChildren
            pbty_0 = pbty_0*CPD_Pixel(1,(hiddenNodeID-
1)*numChildren+i,pixelDataSubset_TrainNew(sampleID,(hiddenNodeID-1)*numChildren+i)+1);
            pbty_1 = pbty_1*CPD_Pixel(2,(hiddenNodeID-
1)*numChildren+i,pixelDataSubset_TrainNew(sampleID,(hiddenNodeID-1)*numChildren+i)+1);
        end
        pbty_sum = pbty_0 + pbty_1;
        weightVct(hiddenNodeID,sampleID) = pbty_0/pbty_sum;%+
weightVct(hiddenNodeID,sampleID);
        weightVct(hiddenNodeID,sampleID+numTrainingSamples) = pbty_1/pbty_sum;%+
weightVct(hiddenNodeID,sampleID+numTrainingSamples);
    end
end
end

%Perform maximization; Compute the distribution parameters as a counting
%problem from the weight vector
function [CPD_Pixel_Count,CPD_Pixel,CPD_Hidden_Count,CPD_Hidden] =
computeMaximization(pixelDataSubset_TrainNew,labelDataSubset_Train,...

weightVct,CPD_Pixel_Count,CPD_Pixel,CPD_Hidden_Count,CPD_Hidden)
numTrainingSamples = size(pixelDataSubset_TrainNew,1);
numPixelNodes = size(CPD_Pixel,2);
numHiddenNodes = size(CPD_Hidden,2);
numChildren = numPixelNodes/numHiddenNodes;

for sampleID = 1:numTrainingSamples
    for hiddenNodeID = 1:numHiddenNodes
        CPD_Hidden_Count(labelDataSubset_Train(sampleID)+1,hiddenNodeID,1) =
CPD_Hidden_Count(labelDataSubset_Train(sampleID)+1,hiddenNodeID,1) ...
                                                                    +
weightVct(hiddenNodeID,sampleID);
        CPD_Hidden_Count(labelDataSubset_Train(sampleID)+1,hiddenNodeID,2) =
CPD_Hidden_Count(labelDataSubset_Train(sampleID)+1,hiddenNodeID,2) ...
                                                                    +
weightVct(hiddenNodeID,sampleID+numTrainingSamples);
        for i = 1:numChildren
            CPD_Pixel_Count(1,(hiddenNodeID-
1)*numChildren+i,pixelDataSubset_TrainNew(sampleID,(hiddenNodeID-1)*numChildren+i)+1) = ...
                CPD_Pixel_Count(1,(hiddenNodeID-
1)*numChildren+i,pixelDataSubset_TrainNew(sampleID,(hiddenNodeID-1)*numChildren+i)+1) ...
                + weightVct(hiddenNodeID,sampleID);
```

```
            CPD_Pixel_Count(2,(hiddenNodeID-
1)*numChildren+i,pixelDataSubset_TrainNew(sampleID,(hiddenNodeID-1)*numChildren+i)+1) = ...
                CPD_Pixel_Count(2,(hiddenNodeID-
1)*numChildren+i,pixelDataSubset_TrainNew(sampleID,(hiddenNodeID-1)*numChildren+i)+1) ...
                + weightVct(hiddenNodeID,sampleID+numTrainingSamples);
        end
    end
end
for hiddenNodeID = 1:numHiddenNodes
    normalizationFactor = CPD_Hidden_Count(:,hiddenNodeID,1) +
CPD_Hidden_Count(:,hiddenNodeID,2);
    CPD_Hidden(:,hiddenNodeID,1) = CPD_Hidden_Count(:,hiddenNodeID,1)./normalizationFactor;
    CPD_Hidden(:,hiddenNodeID,2) = CPD_Hidden_Count(:,hiddenNodeID,2)./normalizationFactor;
    for i = 1:numChildren
        normalizationFactor = CPD_Pixel_Count(:,(hiddenNodeID-1)*numChildren+i,1) +
CPD_Pixel_Count(:,(hiddenNodeID-1)*numChildren+i,2);
        CPD_Pixel(:,(hiddenNodeID-1)*numChildren+i,1) = CPD_Pixel_Count(:,(hiddenNodeID-
1)*numChildren+i,1)./normalizationFactor;
        CPD_Pixel(:,(hiddenNodeID-1)*numChildren+i,2) = CPD_Pixel_Count(:,(hiddenNodeID-
1)*numChildren+i,2)./normalizationFactor;
    end
end
end
```

A.7 Bayesian classifier with hidden nodes - 2 layer, 8x8 pixel nodes, 4x4 hidden nodes, 1 label node

```
function hiddenLayerBayes_8x8_EMChanges
clear all;

load 'labelDataSubset1_Train.mat';%Goes from 0 to 9
load 'pixelDataSubset1_Train.mat';
load 'labelDataSubset1_Test.mat';
load 'pixelDataSubset1_Test.mat';
load 'pixelDataSubset1_TrainRearrange_8x8.mat';
load 'pixelDataSubset1_TestRearrange_8x8.mat';

numChars = 10;
charWidth = 8; charHeight = 8;
numHiddenNodes = charWidth*charHeight/4;
numPixelNodes = charWidth*charHeight;
numChildren = numPixelNodes/numHiddenNodes;
numTrainingSamples = size(pixelDataSubset_Train,1);
numTestingSamples = size(pixelDataSubset_Test,1);
CPD_Hidden = zeros(numChars, numHiddenNodes, 2);%binary-valued hidden state
CPD_Hidden_Count = zeros(numChars, numHiddenNodes, 2);
CPD_Pixel = zeros(2, numPixelNodes, 2);
CPD_Pixel_Count = zeros(2, numPixelNodes, 2);%num of parent states=2
weightVct = zeros(numHiddenNodes,2*numTrainingSamples);
numEMIters = 20000;%2000;
numInferenceIters = 10000;

%Init-random
CPD_Hidden(:,:,1) = rand(numChars, numHiddenNodes);
CPD_Hidden(:,:,2) = 1 - CPD_Hidden(:,:,1);
CPD_Pixel(:,:,1) = rand(2, numPixelNodes);
CPD_Pixel(:,:,2) = 1 - CPD_Pixel(:,:,1);

fprintf('Starting training ...\r');
for iter = 1:numEMIters
    fprintf('Iter: %d out of %d ... ', iter, numEMIters);
    %{
    NOTE: !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!! IMPORTANT
    1. Initialize counts everytime?
    2. Previous iteration weight is not used in the E-code
    %}
    CPD_Hidden_Count = zeros(numChars, numHiddenNodes, 2);
    CPD_Pixel_Count = zeros(2, numPixelNodes, 2);%num of parent states=2
    tic;
    weightVct =
computeExpectation(pixelDataSubset_TrainNew,labelDataSubset_Train,CPD_Pixel,CPD_Hidden,weightV
ct);
    [CPD_Pixel_Count,CPD_Pixel,CPD_Hidden_Count,CPD_Hidden] =
computeMaximization(pixelDataSubset_TrainNew,labelDataSubset_Train,...

weightVct,CPD_Pixel_Count,CPD_Pixel,CPD_Hidden_Count,CPD_Hidden);
    t1 = toc;
    fprintf('%g sec\r',t1);
```

```
%     reminder = mod(iter,1000);
%     if(reminder == 0)
%         CPD_string = sprintf('CPD_Pixel_EM_8x8_new_%d.mat',iter);
%         CPD_Pixel_EM = CPD_Pixel;
%         save(CPD_string,'CPD_Pixel_EM');
%         CPD_string = sprintf('CPD_Hidden_EM_8x8_new_%d.mat',iter);
%         CPD_Hidden_EM = CPD_Hidden;
%         save(CPD_string,'CPD_Hidden_EM');
%     end
end

fprintf('Starting testing %d iters (approx.)... ',numInferenceIters);
tic;
ConfusionMatrix1 = approxInference(pixelDataSubset_TestNew, labelDataSubset_Test, CPD_Pixel,
CPD_Hidden, numInferenceIters);
t1 = toc;
fprintf('%g sec\r\n',t1);
for i = 1:numChars
    fprintf('%d: Accuracy: %g\r\t',i-1,ConfusionMatrix1(i,i)/sum(ConfusionMatrix1(i,:)));
    for j = 1:numChars
        fprintf('%d:%d; ',j-1,ConfusionMatrix1(i,j));
    end
    fprintf('\r');
end
keyboard;
end

%Function to perform expectation of the hidden nodes, return the weight
%vector
function weightVct =
computeExpectation(pixelDataSubset_TrainNew,labelDataSubset_Train,CPD_Pixel,CPD_Hidden,weightV
ct)
numTrainingSamples = size(pixelDataSubset_TrainNew,1);
numPixelNodes = size(CPD_Pixel,2);
numHiddenNodes = size(CPD_Hidden,2);
numChildren = numPixelNodes/numHiddenNodes;

for sampleID = 1:numTrainingSamples
    for hiddenNodeID = 1:numHiddenNodes
        %Compute the weight for two possible states of the hidden node
        pbty_0 = CPD_Hidden(labelDataSubset_Train(sampleID)+1,hiddenNodeID,1);
        pbty_1 = CPD_Hidden(labelDataSubset_Train(sampleID)+1,hiddenNodeID,2);
        for i = 1:numChildren
            pbty_0 = pbty_0*CPD_Pixel(1,(hiddenNodeID-
1)*numChildren+i,pixelDataSubset_TrainNew(sampleID,(hiddenNodeID-1)*numChildren+i)+1);
            pbty_1 = pbty_1*CPD_Pixel(2,(hiddenNodeID-
1)*numChildren+i,pixelDataSubset_TrainNew(sampleID,(hiddenNodeID-1)*numChildren+i)+1);
        end
        pbty_sum = pbty_0 + pbty_1;
        weightVct(hiddenNodeID,sampleID) = pbty_0/pbty_sum;%+
weightVct(hiddenNodeID,sampleID);
        weightVct(hiddenNodeID,sampleID+numTrainingSamples) = pbty_1/pbty_sum;%+
weightVct(hiddenNodeID,sampleID+numTrainingSamples);
    end
end
end

%Perform maximization; Compute the distribution parameters as a counting
%problem from the weight vector
function [CPD_Pixel_Count,CPD_Pixel,CPD_Hidden_Count,CPD_Hidden] =
computeMaximization(pixelDataSubset_TrainNew,labelDataSubset_Train,...

weightVct,CPD_Pixel_Count,CPD_Pixel,CPD_Hidden_Count,CPD_Hidden)
numTrainingSamples = size(pixelDataSubset_TrainNew,1);
numPixelNodes = size(CPD_Pixel,2);
numHiddenNodes = size(CPD_Hidden,2);
numChildren = numPixelNodes/numHiddenNodes;

for sampleID = 1:numTrainingSamples
    for hiddenNodeID = 1:numHiddenNodes
        CPD_Hidden_Count(labelDataSubset_Train(sampleID)+1,hiddenNodeID,1) =
CPD_Hidden_Count(labelDataSubset_Train(sampleID)+1,hiddenNodeID,1) ...
                                                                    +
weightVct(hiddenNodeID,sampleID);
        CPD_Hidden_Count(labelDataSubset_Train(sampleID)+1,hiddenNodeID,2) =
CPD_Hidden_Count(labelDataSubset_Train(sampleID)+1,hiddenNodeID,2) ...
```

```
                                                                    +
weightVct(hiddenNodeID,sampleID+numTrainingSamples);
        for i = 1:numChildren
            CPD_Pixel_Count(1,(hiddenNodeID-
1)*numChildren+i,pixelDataSubset_TrainNew(sampleID,(hiddenNodeID-1)*numChildren+i)+1) = ...
                CPD_Pixel_Count(1,(hiddenNodeID-
1)*numChildren+i,pixelDataSubset_TrainNew(sampleID,(hiddenNodeID-1)*numChildren+i)+1) ...
                + weightVct(hiddenNodeID,sampleID);
            CPD_Pixel_Count(2,(hiddenNodeID-
1)*numChildren+i,pixelDataSubset_TrainNew(sampleID,(hiddenNodeID-1)*numChildren+i)+1) = ...
                CPD_Pixel_Count(2,(hiddenNodeID-
1)*numChildren+i,pixelDataSubset_TrainNew(sampleID,(hiddenNodeID-1)*numChildren+i)+1) ...
                + weightVct(hiddenNodeID,sampleID+numTrainingSamples);
        end
    end
end
for hiddenNodeID = 1:numHiddenNodes
    normalizationFactor = CPD_Hidden_Count(:,hiddenNodeID,1) +
CPD_Hidden_Count(:,hiddenNodeID,2);
    CPD_Hidden(:,hiddenNodeID,1) = CPD_Hidden_Count(:,hiddenNodeID,1)./normalizationFactor;
    CPD_Hidden(:,hiddenNodeID,2) = CPD_Hidden_Count(:,hiddenNodeID,2)./normalizationFactor;
    for i = 1:numChildren
        normalizationFactor = CPD_Pixel_Count(:,(hiddenNodeID-1)*numChildren+i,1) +
CPD_Pixel_Count(:,(hiddenNodeID-1)*numChildren+i,2);
        CPD_Pixel(:,(hiddenNodeID-1)*numChildren+i,1) = CPD_Pixel_Count(:,(hiddenNodeID-
1)*numChildren+i,1)./normalizationFactor;
        CPD_Pixel(:,(hiddenNodeID-1)*numChildren+i,2) = CPD_Pixel_Count(:,(hiddenNodeID-
1)*numChildren+i,2)./normalizationFactor;
    end
end
end
```

A.8 Approximate inference using likelihood weighted sampling

```
%Perform inference using likelihood weighted sampling
%Return the confusion matrix
function confusionMat = approxInference(pixelDataSubset_Test, labelDataSubset_Test, CPD_Pixel,
CPD_Hidden, numIters)
numPixelNodes = size(CPD_Pixel,2);
numHiddenNodes = size(CPD_Hidden,2);
numChildren = numPixelNodes/numHiddenNodes;
numChars = 10;
numSamples = size(pixelDataSubset_Test,1);
confusionMat = zeros(numChars,numChars);
for sampleID = 1:numSamples
    numerator = zeros(numChars,1);
    denominator = 0;
%     for label = 1:numChars
        for iter = 1:numIters
            weight = realmax;%Not one bcoz the value was losing precision because of large num
of nodes
            hiddenNode = zeros(numHiddenNodes,1);
            %Sample label node
            %Sample hidden nodes one by one
            %Take pixel nodes as evidence
            label = randi(numChars);
            for hiddenNodeID = 1:numHiddenNodes
                hiddenNode(hiddenNodeID) = rand > CPD_Hidden(label,hiddenNodeID, 1);
            end
            for i = 1:numPixelNodes
                parentHiddenNode = floor((i-1)/numChildren)+1;
                weight =
weight*CPD_Pixel(hiddenNode(parentHiddenNode)+1,i,pixelDataSubset_Test(sampleID,i)+1);
            end
            denominator = denominator + weight;
            for i = 1:numChars
                numerator(i) = numerator(i) + weight*(label ==
i);%(labelDataSubset_Test(sampleID)+1));
            end
        end
        ratio = numerator/denominator;
%     end
    [val,ID] = max(ratio);
    confusionMat(labelDataSubset_Test(sampleID)+1,ID) =
confusionMat(labelDataSubset_Test(sampleID)+1,ID) + 1;
end
end
```

A.9 Exact inference

```
%Perform exact inference and return the confusion matrix
%Sum over all possible values of hidden nodes for exact inference
function confusionMat = exactInference(pixelDataSubset_Test, labelDataSubset_Test, CPD_Pixel,
CPD_Hidden)
numPixelNodes = size(CPD_Pixel,2);
numHiddenNodes = size(CPD_Hidden,2);
numChildren = numPixelNodes/numHiddenNodes;
numChars = 10;
numSamples = size(pixelDataSubset_Test,1);
confusionMat = zeros(numChars,numChars);
parentHiddenNode = zeros(numPixelNodes);
for i = 1:numPixelNodes
    parentHiddenNode(i) = floor((i-1)/numChildren) + 1;
end
numStates = 2^numHiddenNodes;%Number of possible hidden nodes' states
for sampleID = 1:numSamples
    jointPbty = zeros(numChars,1);
    stateHidden = zeros(numHiddenNodes,1);
    for stateID = 0:numStates-1%Compute joint pbty for each state
        tempPbty = ones(numChars,1);
        state = dec2bin(stateID, numHiddenNodes);%Get the state in binary for all nodes
        for i = 1:numHiddenNodes
            stateHidden(i) = str2double(state(i));%State in numeric type instead of ascii
format
            for char = 1:numChars
                tempPbty(char) = tempPbty(char)*CPD_Hidden(char,i,stateHidden(i)+1);
            end
        end
        for i = 1:numPixelNodes
            tempPbty =
tempPbty*CPD_Pixel(stateHidden(parentHiddenNode(i))+1,i,pixelDataSubset_Test(sampleID,i)+1);
        end
        jointPbty = jointPbty + tempPbty;%Accumulate pbty
    end
    jointPbty = jointPbty/sum(jointPbty);
    [val,ID] = max(jointPbty);
    confusionMat(labelDataSubset_Test(sampleID)+1,ID) =
confusionMat(labelDataSubset_Test(sampleID)+1,ID) + 1;
end
end
```

A.10 Prepare dataset

```
%Read data from MNIST dataset file and store in .mat format
function getData
clear all;
%Train data
%Label
fileID = fopen('..\data\train-labels.idx1-ubyte');
headerData = fread(fileID,2,'uint32','b');
fprintf('%d\t%d\r',headerData(1),headerData(2));
labelData_Train = fread(fileID,headerData(2),'uint8','b');
save('labelData_Train.mat','labelData_Train');
fclose(fileID);
%Pixel
fileID = fopen('..\data\train-images.idx3-ubyte');
headerData = fread(fileID,4,'uint32','b');
fprintf('%d\t%d\r',headerData(1),headerData(2));
pixelData_Train = uint8(zeros(headerData(2),headerData(3)*headerData(4)));
for i = 1:headerData(2)
    pixelData_Train(i,:) = fread(fileID,headerData(3)*headerData(4),'uint8','b');
end
save('pixelData_Train.mat','pixelData_Train');
fclose(fileID);

%Test data
%Label
fileID = fopen('..\data\t10k-labels.idx1-ubyte');
headerData = fread(fileID,2,'uint32','b');
fprintf('%d\t%d\r',headerData(1),headerData(2));
labelData_Test = fread(fileID,headerData(2),'uint8','b');
save('labelData_Test.mat','labelData_Test');
fclose(fileID);
%Pixel
```

```
fileID = fopen('..\data\t10k-images.idx3-ubyte');
headerData = fread(fileID,4,'uint32','b');
fprintf('%d\t%d\r',headerData(1),headerData(2));
pixelData_Test = uint8(zeros(headerData(2),headerData(3)*headerData(4)));
for i = 1:headerData(2)
    pixelData_Test(i,:) = fread(fileID,headerData(3)*headerData(4),'uint8','b');
end
save('pixelData_Test.mat','pixelData_Test');
fclose(fileID);
keyboard;
end

%Divide full dataset into manageable smaller dataset
function getSubsetData
clear all;
load 'labelData_Train.mat';
load 'pixelData_Train.mat';
load 'labelData_Test.mat';
load 'pixelData_Test.mat';

numSamplesLimitTrain = 10;
numSamplesLimitTest = 1;

for i = 1:10
    %Train subset
    %Label
    index = find(labelData_Train == (i-1), numSamplesLimitTrain);
    labelDataSubset_Train(((i-1)*numSamplesLimitTrain + 1):(i*numSamplesLimitTrain),:) =
labelData_Train(index);
    %Pixel
    pixelDataSubset_Train(((i-1)*numSamplesLimitTrain + 1):(i*numSamplesLimitTrain),:) =
pixelData_Train(index,:);

    %Test subset
    %Label
    index = find(labelData_Test == (i-1), numSamplesLimitTest);
    labelDataSubset_Test(((i-1)*numSamplesLimitTest + 1):(i*numSamplesLimitTest),:) =
labelData_Test(index);
    %Pixel
    pixelDataSubset_Test(((i-1)*numSamplesLimitTest + 1):(i*numSamplesLimitTest),:) =
pixelData_Test(index,:);
end
save('labelDataSubset3_Train.mat','labelDataSubset_Train');
save('pixelDataSubset3_Train.mat','pixelDataSubset_Train');
save('labelDataSubset3_Test.mat','labelDataSubset_Test');
save('pixelDataSubset3_Test.mat','pixelDataSubset_Test');
keyboard;
end

%Rearrange pixel data for easy access so that the children nodes are
%accesible easily (next to each other in the new array)
function pixelDataNew = rearrangePixelData(pixelData, numHiddenNodes,charWidth,charHeight)
pixelDataNew = pixelData';
numPixelNodes = size(pixelDataNew,1);
numChildren = numPixelNodes/numHiddenNodes;
widthPixel = sqrt(numPixelNodes);
heightPixel = sqrt(numPixelNodes);

pixelDataMat = reshape(pixelData,charWidth,charHeight)';

widthHidden = sqrt(numHiddenNodes);
lengthChild = sqrt(numChildren);
index = 1;
for i = 1:numHiddenNodes
    quotient = floor((i-1)/widthHidden);
    reminder = mod(i-1,widthHidden);
    [X,Y] = ndgrid(lengthChild*quotient+1:lengthChild*quotient+lengthChild, ...
                   lengthChild*reminder+1:lengthChild*reminder+lengthChild);
    index = sub2ind([widthPixel,heightPixel],X,Y);
    pixelDataNew((i-1)*numChildren+1:(i-1)*numChildren+numChildren) = pixelData(index);
    index = index + 1;
end
end
```