CSC 411
Machine Learning and Data Mining
Assignment 2
Out: Oct. 11
Due: Nov. 1

Overview

In this assignment, you will experiment with a neural network and clustering methods, and derive a simple EM algorithm. Some code that implements a neural network with one hidden layer, and the EM algorithm for a mixture of Gaussians (MoG) model will be provided for you (both MATLAB and Python). You will use multiple MoGs to model the data distribution and then combine them to make a classifier, and compare this to the neural networks.

You will be working with the following dataset:

Digits: The file `digits.mat` contains 6 sets of 16×16 greyscale images in vector format (the pixel intensities are between 0 and 1 and were read into the vectors in a raster-scan manner). The images contain centered, handwritten 2's and 3's, scanned from postal envelopes. `train2` and `train3` contain examples of 2's and 3's respectively to be used for training. There are 300 examples of each digit, stored as 256×300 matrices. Note that each data vector is a column of the matrix. `valid2` and `valid3` contain data to be used for validation (100 examples of each digit) and `test2` and `test3` contain test data to be used for final evaluation **only** (200 examples of each digit).

1 Neural Networks (15 points)

Code for training a neural network with one hidden layer of logistic units, logistic output units and a cross entropy error function is included. The main components are:

MATLAB

- `init_nn.m`: initializes the weights and loads the training, validation and test data.
- `train_nn.m`: runs `num_epochs` of backprop learning.
- `test_nn.m`: Evaluates the network on the test set.

Python

- `nn.py` : Methods to perform initialization, backprop learning and testing.

a) Basic generalization [6 points]

Train a neural network with 10 hidden units. You should first use `init_nn` to initialize the net, and then execute `train_nn` repeatedly (more than 5 times). Note that `train_nn` runs 100 epochs each time and will output the statistics and plot the error curves. Alternatively, if you wish to use Python, set the appropriate number of epochs in `nn.py` and run it. Examine the statistics and plots of training error and validation error (generalization). How does the network's performance differ on the training set versus the validation set during learning ?

b) Classification error [3 points]

You should implement an alternative performance measure to the cross entropy, the mean classification error. You can consider the output correct if the correct label is given a higher probability than the incorrect label. You should then count up the total number of examples that are classified incorrectly according to this criterion for training and validation respectively, and maintain this statistic at the end of each epoch. Plot the classification error vs. number of epochs.

c) Learning rate [3 points]

Try different values of the learning rate ϵ ("eps") defined in `init_nn.m` (and in `nn.py`). You should reduce it to .01, and increase it to 0.2 and 0.5. What happens to the convergence properties of the algorithm (looking at both cross entropy and %Correct)? Try momentum of $\{0.0, 0.5, 0.9\}$. How does momentum affect convergence rate ? How would you choose the best value of these parameters?

d) Number of hidden units [3 points]

Set the learning rate ϵ to .002 and try different numbers of hidden units on this problem. You should use two values $\{2, 5\}$, which are smaller than the original and two others $\{30, 100\}$, which are larger. Describe the effect of this modification on the convergence properties, and the generalization of the network.

2 Mixtures of Bernoulli distributions (30 points)

In this section, you will derive the Expectation-Maximization (EM) algorithm to perform inference in a *mixture of Bernoulli distributions model*.

Consider a set of D binary, Bernoulli distributed variables x_i with parameter μ_i, where

$i = 1, \ldots, D$, such that

$$p(\mathbf{x}|\boldsymbol{\mu}) = \prod_{i=1}^{D} \mu_i^{x_i}(1-\mu_i)^{(1-x_i)},$$

where $\mathbf{x} = \{x_1, \ldots, x_D\}^T$ and $\boldsymbol{\mu} = \{\mu_1 \ldots, \mu_D\}^T$. We see that the variables x_i are independent given μ and that the mean and covariance are given by

$$\begin{aligned} \mathbb{E}[\mathbf{x}] &= \boldsymbol{\mu} \\ \text{cov} &= \text{diag}\{\mu_i(1-\mu_i)\}. \end{aligned}$$

Now consider a mixture of K such distributions such that

$$p(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\pi}) = \sum_{k=1}^{K} \pi_k p(\mathbf{x}|\boldsymbol{\mu}_k)$$

where $\boldsymbol{\mu} = \{\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K\}$, $\boldsymbol{\pi} = \{\pi_1, \ldots, \pi_K\}$, and

$$p(\mathbf{x}|\boldsymbol{\mu}_k) = \prod_{i=1}^{D} \mu_{ki}^{x_i}(1-\mu_{ki})^{(1-x_i)}.$$

The expectation and covariance of the mixture are given by

$$\begin{aligned} \mathbb{E}[\mathbf{x}] &= \sum_{k=1}^{K} \pi_k \boldsymbol{\mu}_k \\ \text{cov}[\mathbf{x}] &= \sum_{k=1}^{K} \pi_k \boldsymbol{\mu}_k \boldsymbol{\mu}_k^T - \mathbb{E}[\mathbf{x}]\,\mathbb{E}[\mathbf{x}]^T \end{aligned}$$

Note that the covariance matrix is no longer diagonal, so the mixture model introduced dependence between x_i's. The log likelihood function of this mixture distribution is

$$\log p(\boldsymbol{X}|\boldsymbol{\mu}, \boldsymbol{\pi}) = \sum_{n=1}^{N} \log \left\{ \sum_{k=1}^{K} \pi_k p(\mathbf{x}_n|\boldsymbol{\pi}_k) \right\}.$$

As for mixtures of Gaussians, we employ the trick of associating with each instance of $\mathbf{x}$ a latent variable $\boldsymbol{z} = (z_1, \ldots, z_K)$, where $z_k = 1$ if $\mathbf{x}$ belongs to the k-th cluster and all other components are equal to 0. Using this, we can write the conditional distribution of $\mathbf{x}$ as

$$p(\mathbf{x}|\mathbf{z}, \boldsymbol{\mu}) = \prod_{k=1}^{K} p(\mathbf{x}|\boldsymbol{\mu}_k)^{z_k}.$$

The prior distribution for the latent variables is given by the mixing proportions π_k:

$$p(\mathbf{z}|\boldsymbol{\pi}) = \prod_{k=1}^{K} \pi_k^{z_k}.$$

Now we can write down the complete-data log-likelihood function

$$\log p(\boldsymbol{X}, \boldsymbol{Z}|\boldsymbol{\mu}, \boldsymbol{\pi}) = \sum_{n=1}^{N} \sum_{k=1}^{K} z_{nk} \left\{ \log \pi_k + \sum_{i=1}^{D} [x_{ni} \log \mu_{ki} + (1 - x_{ni}) \log(1 - \mu_{ki})] \right\}.$$

If we take the expectation with respect to the posterior distribution of the latent variables, we get

$$\mathbb{E}_{\boldsymbol{Z}}[\log p(\boldsymbol{X}, \boldsymbol{Z}|\boldsymbol{\mu}, \boldsymbol{\pi})] = \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} \left\{ \log \pi_k + \sum_{i=1}^{D} [x_{ni} \log \mu_{ki} + (1 - x_{ni}) \log(1 - \mu_{ki})] \right\},$$

where $r_{nk} = \mathbb{E}[z_{nk}]$ is the *responsibility*, or posterior probability, of component k given data point $\mathbf{x}_n$.

1. *Derive the E-step of the EM algorithm: Use Bayes' theorem to show that the responsibilities take the form*

$$r_{nk} = \frac{\pi_k p(\mathbf{x}_n|\boldsymbol{\mu}_k)}{\sum_{j=1}^{K} \pi_j p(\mathbf{x}_n|\boldsymbol{\mu}_j)}.$$

   Now define

$$N_k = \sum_{n=1}^{N} r_{nk}$$

$$\bar{\mathbf{x}}_k = \frac{1}{N_k} \sum_{n=1}^{N} r_{nk} \mathbf{x}_n$$

   The next two problems concern the M-step.

2. *Find the derivative of* $\mathbb{E}_{\boldsymbol{Z}}[\log p(\boldsymbol{X}, \boldsymbol{Z}|\boldsymbol{\mu}, \boldsymbol{\pi})]$ *w.r.t* $\boldsymbol{\mu}$*, set it to zero and solve to show that the estimates for the Bernoulli parameters are*

$$\boldsymbol{\mu}_k = \bar{\mathbf{x}}_k.$$

3. *Lastly, maximize* $\mathbb{E}_{\boldsymbol{Z}}[\log p(\boldsymbol{X}, \boldsymbol{Z}|\boldsymbol{\mu}, \boldsymbol{\pi})]$ *with respect to the mixing weights* π_k*. You will need to use a Lagrange multiplier to enforce the constraint that* $\sum_k \pi_k = 1$*. Show that*

$$\pi_k = \frac{N_k}{N}.$$

Mixtures of Gaussians

Code

The Matlab file `mogEM.m` implements the EM algorithm for the MoG model.
The file `mogLogProb.m` computes the log-probability of data under a MoG model.
The file `pcaimg.m` implements the PCA algorithm.
The file `kmeans.m` contains the k-means algorithm.
The file `distmat.m` contains a function that efficiently computes pairwise distances between sets of vectors. It is used in the implementation of k-means.

Similarly, `mogEM.py` implements methods related to training MoG models.
The file `kmeans.py` implements k-means and `pcaimg.py` implements PCA.

As always, read and understand code before using it.

3 Training (15 points)

The Matlab variables `train2` and `train3` each contain 300 training examples of handwritten 2's and 3's, respectively. Take a look at some of them to make sure you have transferred the data properly. In Matlab, plot the digits as images using
`imagesc(reshape(vector,16,16))`, which converts a 256-vector to an 16x16 image. You may need to also say `colormap(gray)`. Look at `kmeans.py` to see an example of how to do this in Python.

For each training set separately, train a mixture-of-Gaussians using the code in `mogEM.m`. Let the number of clusters in the Gaussian mixture be 2, and the minimum variance be 0.01. You will also need to experiment with the parameter settings, e.g. `randConst`, in that program to get sensible clustering results. And you'll need to execute `mogEM` a few times for each digit, and see the local optima the EM algorithm finds. Choose a good model for each digit from your results.

For each model, show both the mean vector(s) and variance vector(s) as images, and show the mixing proportions for the clusters within each model. Finally, provide $\log P(TrainingData)$ for each model.

4 Initializing a mixture of Gaussians with k-means (10 points)

Training a MoG model with many components tends to be slow. People have found that initializing the means of the mixture components by running a few iterations of k-means tends to speed up convergence. You will experiment with this method of initialization. You should do the following.

- Read and understand `kmeans.m` and `distmat.m` (Alternatively, `kmeans.py`).
- Change the initialization of the means in `mogEM.m` (or `mogEm.py`) to use the k-means algorithm. As a result of the change the model should run k-means on the training data and use the returned means as the starting values for `mu`. Use 5 iterations of k-means.
- Train a MoG model with 20 components on all 600 training vectors (both 2's and 3's) using both the original initialization and the one based on k-means. Comment on the speed of convergence as well as the final log-prob resulting from the two initialization methods.

5 Classification using MoGs (20 points)

Now we will investigate using the trained mixture models for classification. The goal is to decide which digit class d a new input image $\mathbf{x}$ belongs to. We'll assign $d = 1$ to the 2's and $d = 2$ to the 3's.

For each mixture model, after training, the likelihoods $P(\mathbf{x}|d)$ for each class can be computed for an image $\mathbf{x}$ by consulting the model trained on examples from that class; probabilistic inference can be used to compute $P(d|\mathbf{x})$, and the most probable digit class can be chosen to classify the image.

Write a program that computes $P(d = 1|\mathbf{x})$ and $P(d = 2|\mathbf{x})$ based on the outputs of the two trained models. You can use `mogLogProb.m` (or the method `mogLogProb` in `mogEm.py`) to compute the log probability of examples under any model.

You will compare models trained with the same number of mixture components. You have trained 2's and 3's models with 2 components. Also train models with more components: 5, 15 and 25. For each number, use your program to classify the validation and test examples.

For each of the validation and test examples, compute $P(d|\mathbf{x})$ and classify the example. Plot the results. The plot should have 3 curves of classification error rates versus number of mixture components (averages are taken over the two classes):

- The average classification error rate, on the training set

- The average classification error rate, on the validation set
- The average classification error rate, on the test set

Provide answers to these questions:

1. You should find that the error rates on the training sets generally decrease as the number of clusters increases. Explain why.
2. Examine the error rate curve for the test set and discuss its properties. Explain the trends that you observe.
3. If you wanted to choose a particular model from your experiments as the best, how would you choose it? If your aim is to achieve the lowest error rate possible on the new images your system will receive, which model (number of clusters) would you select? Why?

6 Mixture of Gaussians vs Neural Network (10 points)

Choose the best mixture of Gaussian classifier you have got so far according to your answer to question 3 in the section above. Compare this mixture of Gaussian classifier with the neural network. For this comparison, set the number of hidden units equal to the number of mixture components in the mixture model (digit 2 and 3 combined). Visualize the input to hidden weights as images to see what your network has learned. You can use the same trick as mentioned in section 3 to visualize these vectors as images.

You can visualize how the input to hidden weights change during training. Discuss the classification performance of the two models and compare the hidden unit weights in the neural network with the mixture components in the mixture model.

Bonus Question: Classification using PCA (10 points)

Apply the PCA algorithm to the digit images (computing all 256 of the eigenvalues and eigenvectors). Then you should plot the (sorted) eigenvalues as a descending curve. Then show the first 3 eigen-images (reshape each of the first 3 eigenvectors and use imagesc to see these as images). And do the same for the mean of each digit. This part is for you to gain some intuition about how PCA works. *You do not need to write this part up!*

For each image, you can project it into the low-dimensional space spanned by the first m principal components. After projection, we can use a 1-NN classifier to classify the digit in the low-dimensional space. You need to implement the classifier based on the code you have.

You will do the classification under different m values to see the effect of m. Here, let $m = 2, 5, 10, 20$, and under each m, classify the validation digits. Plot the results, where the plot should show the curve of average validation set classification error rates versus number

of eigenvectors you keep, i.e., m (averages are taken over the two classes). If you wanted to choose a particular model from your experiments as the best, how would you choose it? If your aim is to achieve the lowest error rate possible on the new images your system will receive, which model (number of eigenvectors) would you select? Why? Compare your results with the mixture models.

Write up

Hand in answers to all the questions in the parts above. The goal of your write-up is to document the experiments you've done and your main findings. Do not require us to read your code. We will look at the code only if it is not clear from the report that the implementation has been done correctly.

Submit your code and write-up in a zip file by email to `csc411ta@cs.toronto.edu`, and turn in a hard copy of the write-up before class on November 1 as well.

A checklist of things to submit -

- Question 1
  - (a) one plot + one paragraph
  - (b) one plot + one code modification
  - (c) one paragraph (3-4 lines)
  - (d) one paragraph (3-4 lines)
- Question 2 - Derivations (1-2 pages)
- Question 3
  - Four mean and variance images (2 clusters in each of th 2 models)
  - Mixing proportions (4 numbers)
  - 2 plots showing log P(training data), one plot for each digit
- Question 4 - Paragraph (5-6 lines)
- Question 5
  - Code
  - Plot with 3 curves
  - Answers to the three questions (3-4 lines each)
- Question 6 - Paragraph (5-6 lines)
- Bonus Question
  - Code
  - Plot of classification error vs number of eigen values
  - Paragraph (5-6 lines)