CSC 411: Introduction to Machine Learning
ASSIGNMENT # 2

Out: March 5, 2016

Due: March 18 (Friday), 2016 [6pm Toronto time]

Submission

Hand in answers to all the questions in the parts below. The goal of your write-up is to document the experiments you have done and your main findings. Be sure to explain the results. The answers to your questions should be in pdf form and handed in along with your code (a directory with code is preferable). Only include functions and scripts that you modified.

Submit your assignment on MarkUs. Do not hand in a hard copy of the write-up (if you wrote part of your solution on paper, please scan it and submit it electronically).

Overview

In this assignment, you will experiment with a neural network and the mixture of Gaussians model. Some code that implements a neural network with one hidden layer, and the mixture of Gaussians model will be provided for you (both MATLAB and Python).

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 Backpropagation for Convnets (15 pts)

You are training a Convolutional Neural Network (CNN) by minimizing the cross-entropy loss. The CNN has the following architecture:

- The input is a 32×32 image (grayscale, not RGB)
- The first (and only) hidden layer is convolutional. There are F number of filters with size $w \times h$. The activation function is a ReLU.
- The output layer is fully-connected and has 3 units. It has the soft-max activation function.

How many weights are there in the model? Explain how backpropagation works, and derive equations for the updates for each weight in the model. How many operations does the forward pass require?

2 Neural Networks (40 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.

2.1 Basic generalization [8 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? Show a plot of error curves (training and validation) to support your argument.

2.2 Classification error [8 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, for both training and validation.

2.3 Learning rate [8 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?

2.4 Number of hidden units [8 points]

Set the learning rate ϵ to .02, momentum to 0.5 and try different numbers of hidden units on this problem (you might also need to adjust `num_epochs` accordingly in `init_nn.m`). 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.

2.5 Compare k-NN and Neural Networks (8 points)

Try k-NN on this digit classification task using the code you developed in the first assignment, and compare the results with those you got using neural networks. Briefly comment on the differences between these classifiers.

3 Mixtures of Gaussians (45 points)

3.1 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 `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.

As always, read and understand code before using it.

3.2 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 also need to use `colormap(gray)` to obtain grayscale image. 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.

3.3 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.

3.4 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?

3.5 Bonus Question: 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 3.4 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 3.2 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.