Machine Learning (CS771A)
Homework 2 (Due date: Oct 5, 2016, 11:59pm)

Instructions

- Each late submission will receive a 10% penalty per day for up to 5 days. No submissions will be accepted after the 5th late day.
- We will only accept electronic submissions and the main writeup must be as a PDF file. If you are handwriting your solutions, please scan the hard-copy and convert it into PDF. Your name and roll number should be clearly written at the top. In case you are submitting multiple files, all files must to be zipped and **submitted as a single file** (named: your-roll-number.zip). Please do not email us your submissions. Your submissions have to be uploaded here: `http://tinyurl.com/gqhar47`.
- Note: There are a total of 4 problems. Problems 1 and 2 involve programming/experimentation. Some skeleton code and a lot of helper code is provided, so the actual coding effort shouldn't be that much (however, please carefully go through the problem descriptions to understand what is being asked).

Problem 1 (30 marks)

(Multiclass Perceptron with Ruppert-Polyak Averaging) For this part, we will implement a multiclass version of the Perceptron algorithm. Assuming $C > 2$ classes (the provided MNIST digits data in `mnist_big.mat` file has $C = 10$ classes), in multiclass Perceptron, we will have a weight matrix $\mathbf{W} = [\boldsymbol{w}_1\ \boldsymbol{w}_2\ \ldots\ \boldsymbol{w}_C]$ of size $D \times C$ and a bias vector $\boldsymbol{b} = [b_1\ b_2\ \ldots\ b_C]$ of size $1 \times C$. Note that now we have a separate weight vector and bias for each class. The prediction is based on which class has the highest score: $y_n = \arg\max_c(\boldsymbol{w}_c^\top \boldsymbol{x}_n + b_c)$

The mistake-driven update rule for multiclass Perceptron is as follows: If, for the current training example $(\boldsymbol{x}_n, y_n)$, the current model predicts $\hat{y}_n \in \{1, 2, \ldots, C\}$ and if $\hat{y}_n \neq y_n$ then we update $\mathbf{W}$ and $\boldsymbol{b}$ as follows

$$\begin{aligned} \boldsymbol{w}_{y_n} &= \boldsymbol{w}_{y_n} + \boldsymbol{x}_n \\ b_{y_n} &= b_{y_n} + 1 \\ \boldsymbol{w}_{\hat{y}_n} &= \boldsymbol{w}_{\hat{y}_n} - \boldsymbol{x}_n \\ b_{\hat{y}_n} &= b_{\hat{y}_n} - 1 \end{aligned}$$

These updates basically modify two of the columns of $\mathbf{W}$ (and the corresponding entries of $\boldsymbol{b}$) such that $\mathbf{W}, \boldsymbol{b}$ move *away* from the wrongly predicted class and *closer* to the true class (similar to binary Perceptron).

Your job is to implement this algorithm and test it on the data set I have provided (it's again the MNIST digits data). I have also provided some skeleton code in `perceptron_multiclass.m`. The folder **PP1** contains the skeleton code and the data for this part. Note that you just have to complete the TODO parts.

In addition, within the same code, you also have to implement the **Ruppert-Polyak averaging** for $\mathbf{W}$ and $\boldsymbol{b}$. The idea of doing Ruppert-Polyak averaging is to make Percepton's "stochastic" updates robust (please see lecture-7, slide-3 which shows how Ruppert-Polyak averaging can be done efficiently).

The code computes the classification accuracies of test data after each update, for both simple and averaged versions, and plots the accuracy curves for both version. *What trends to you see in both the curves. Do the two curves look different? If so, how?* Submit your code and plot and answer the questions in your writeup.

Problem 2 (40 marks)

Playing with K-means and PCA: In this part, we will experiment with clustering and linear dimensionality reduction. We will see how they work on their own, as well as how they work when used as a pre-processing step for a supervised learning algorithm (we will use a simple 1-nearest neighbor classifier). There are two "programming" aspects to this part: (1) Clustering with K-means, (2) Dimensionality reduction with PCA.

To save your time and help you focus on the "core" implementation components, you are provided a lot of pre-written MATLAB code (which will generate some plots and figures). You can find it all (along with the data set to be used for this part) within the folder **PP2**. **The only parts you have to code by yourself are in the skeleon codes `kmeans.m` and `PCA.m` to implement K-means and PCA, respectively**. Those places in the code are marked "TODO". You only have to complete those parts in these two files.

K-means Clustering: In this task, you will implement basic K-means clustering. The skeleton code is in `kmeans.m`. There are essentially two things you need to implement within `kmeans.m`. First is the actual K-means algorithm. Second is the initialization based on the furthest point heuristic. You should follow the comments in `kmeans.m` for hints on how to do the implementation.

There is another script, `test_kmeans.m` for (not surprisingly!) testing your kmeans function. This function reads in some simple two dimensional data and then tries to cluster it. The figures it produces are:

- A plot of the data, unclustered
- The data clustered with $K = 2$ and random initialization
- The data clustered with $K = 3$ and random initialization (this is run 16 times... you should see a small amount of variability in the outputs). The plots also include scores.
- The data clustered with $K = 3$ and "furthest" initialization (this is run 16 times... you should see a small amount of variability in the outputs). The plots also include scores.
- A plot of $K \in \{2, 3, 4, 5, 6, 8, 10, 15, 20\}$ versus score.
- The data clustered for $K \in \{2, 3, 4, 5, 6\}$ with together with scores.

To verify that things seem to be going okay, aside from just checking to see if your clusters look reasonable in the plots, the scores that I get in Figure 6 are: 120, 58, 47, 38, 32. There will be a small amount of variation due to the random initialization, but they should be reasonably close.

Dimensionality Reduction with PCA: In the second task, you will implement PCA; the skeleton code is in `PCA.m`. In order to do this (easily), you should use the built-in `eigs` function in MATLAB (which gives you eigenvectors of a covariance matrix).

There is a script `test_pca.m` for testing your PCA function. This again reads in two dimensional data and runs PCA on it. It generates three plots:

- A plot of the data
- A plot of the data and the projection onto the first eigenvector
- A plot of the data and the projection onto the second eigenvector

The second figure, if PCA is implemented correctly, should project down onto a line that runs from the bottom-left to the top-right. The lines show the points that are projected. The third figure should project onto a line that runs upper-left to lower-right, with large projection distances.

Putting it together: There is a `run.m` script that calls your code `kmeans.m` and `PCA.m`. This will generate a large number of plots. All of this is based on a smaller version of the MNIST digits data from HW1 (this smaller version is in `mnist.mat` file and is provided to you). **For this part, you do not have to write any new code** and only have to answer the questions associated with the figures generated by the call to `run.m`.

In the first set of figures, we are considering clustering of digits:

Fig 1 displays the *score* (distortion; lower is better) as we change the number of clusters. It also displays that regularized score with AIC and BIC. (All of the scores are normalized so that they take a maximum value of 1 so they fit on the same plot.) *What are the trends of the plots as K increases? Based on each of score, BIC and AIC, what value of K would you choose to use?*

Fig 2 displays four notions of *clustering accuracy* as a function of K. The "gold truth" clusters are the *digit classes*. In all cases, higher is better. The *rand index* is computed by looking a the binary decision "same cluster?" for every pair of points and evaluating accuracies. The *F-score* is also computed on pairs of points, but looks at a balance between precision and recall of the "same cluster?" decision. The *edit score* reflects the number of edit operations required to get from the hypothesized clustering to the true clustering. The *normalized information* is a measure of the information in the hypothesized cluster about the true cluster. *Which of these seem most reasonable? Which value of K would you select based on these results?*

Fig 3-5 display the means of each cluster for K = 5, K = 10 and K = 15, respectively. *For each of the figures, how many real digits can you recognize? Which ones are missing? Which ones seem to have been merged?*

In the next set of figures, we are considering dimensionality reduction:

Fig 6 displays the scale of the top 50 eigenvalues associated with PCA. *At what point does the "bang for the buck" start to significantly decrease?*

Fig 7 diplays the results of projecting down onto the first three eigenvectors. The left figure is onto dimen- sions 1 and 2; the middle is onto 3 and 2; the right is onto 1 and 3. We only plot five classes (digits 1-5). *Which digits, in this representation, are the most confusable? Does this make sense?*

Fig 8-12 display a subset of the data, first projected down, and then restored. The number of dimensions used are 3, 6, 12, 25 and 50, respectively. *At what point do the images actually start looking like the right digits? How does this related to Figure 6? What sort of artifacts remain at 50 dimensions?*

Finally, we use both k-means and PCA as input to a 1-nearest-neighbor classifier. A baseline of training on all of the data should yield an accuracy of about 85%.

Fig 13 displays accuracies (top) and test time (bottom) when k-means is used as a preprocessing step to 1NN. In particular, we take each class in the training data, and cluster the points. We use the means of these clusters as meta training points, and run 1NN against those. K = 1 is equivalent to just taking the mean image for each class as the representative. The baseline is equivalent to take K = some-big-number. *How is accuracy and test time affected by the number of clusters? How many clusters per class do you need in order to achieve roughly the same accuracy as the baseline, and how much faster does this make it? Would you use this in practice?*

Fig 14 displays accuracies (top) and test time (bottom) when PCA is used as a preprocessing step to 1NN. *How do these results compare? Would you use this in practice?*

Problem 3 (15 marks)

For the multiclass logistic (or "softmax") regression model (which we briefly talked about in lecture-6, slide 15), derive the MLE solution for the weight matrix $\mathbf{W}$. In particular, you need to write down the log likelihood, take its derivative w.r.t. each column $\boldsymbol{w}_k$ of $\mathbf{W}$ to compute the gradient, and give the gradient based update rule for each $\boldsymbol{w}_k$. Give an intuitive meaning of the form of the final gradient-based update rule.

Problem 4 (15 marks)

Given a set of data points $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$, we define the convex hull to be the set of all points $\boldsymbol{x}$ given by $\boldsymbol{x} = \sum_n \alpha_n \boldsymbol{x}_n$ where $\alpha_n \geq 0$ and $\sum_n \alpha_n = 1$ (Intuitively, the convex hull of a set of points in the solid region that they enclose.) Consider a second set of points $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_M$ together with their corresponding convex hull. Show that the set of $\boldsymbol{x}$s and the set of $\boldsymbol{y}$s are linearly separable *if and only if* the convex hulls do not intersect.