SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

# 50.007 Machine Learning, Fall 2020
# Homework 1

Due 11 Oct 2020, 11:59 pm

This homework will be graded by TA Perry Lam

## 1. Linear classification [20 points]

Automatic handwritten digit recognition is an important machine learning task. The US Postal Service Zip Code Database (http://www.unitedstateszipcodes.org/zip-code-database/) provides $16 \times 16$ pixel images preprocessed from scanned handwritten zip codes (US zip codes are the analogues of Singapore postal codes). The task is to recognize the digit in each image. We shall consider the simpler goal of recognizing only two digits: 1 and 5. To simplify our task even further, let's consider only two features: intensity and symmetry. Digit 5 generally occupies more black pixels and thus have higher average pixel intensity than digit 1. Digit 1 is usually symmetric but digit 5 is not. By defining asymmetry as the average difference between an image and its flipped versions, and symmetry as the negation of asymmetry, we can get higher symmetry values for digit 1.

Write an implementation of the perceptron algorithm. Train it on the training set (`train_1_5.csv`), and evaluate its accuracy on the test set (`test_1_5.csv`). The training and test sets are posted on eDimension. csv stands for comma-separated values. In the files, each row is an example. The first value is the symmetry, the second is the average intensity, and the third is the label.

**Note: please do NOT shuffle the data. Visit the instances sequentially in the training set when running the perceptron algorithm.**

(a) [5 points] Run the perceptron algorithm with offset for 1 epoch (i.e., traversing the training set 1 time), report the $\theta$, offset and accuracy on the test set.

(b) [5 points] Run the perceptron algorithm with offset for 5 epochs, report the $\theta$, offset and accuracy on the test set.

(c) [10 points] Submit your code together with crystal clear instructions on how to run it. The TA will follow the instructions to run your code and grade accordingly.

## 2. Linear and polynomial regression [30 points]

For this exercise, you will experiment with linear and polynomial (in features) regression on a given data set. The inputs are in the file `hw1x.dat` and the desired outputs in `hw1y.dat`.

(a) [10 points] Load the data and add a column vector of 1s to the inputs, write a function implementing the closed form linear regression formula discussed in class to obtain the weight vector $\theta$ and report it. Plot both the linear regression line and the data on the same graph. Write a function that will evaluate the training error in terms of empirical risk of the resulting fit and report the error.

(b) [5 points] Write a function to calculate the weight vector $\theta$ using batch gradient descent algorithm. Consider learning rate as $\eta = 0.01$ and update it for $5$ times. Report $\theta$ and training error for the minimum empirical risk. Repeat the same with stochastic gradient descent algorithm for $5$ epochs (with replacement).

(c) [10 points] Write a function called PolyRegress(x,y,d) which adds the features $x^2, x^3, ...x^d$ to the inputs and performs polynomial regression using closed form solution. Use your function to get a quadratic fit of the data. Plot the data and the fit. Report the training error. Repeat the same for 3rd order fit to 15th order fit. After which order fit does the error get worse?

## 3. Ridge regression [20 points]

In this problem, we will explore the effects of ridge regression on generalization. We will use `hw1_ridge_x.dat` as the inputs and `hw1_ridge_y.dat` as the desired output. Please note that a column vector of 1s is already added to the inputs. Recall from Lecture Notes 4, the optimal weight for ridge regression is given by

$$\hat{\theta} = (n\lambda I + X^T X)^{-1} X^T Y \tag{1}$$

To find a suitable value for $\lambda$, we will set aside a small subset of the provided data set for estimating the test loss. This subset is called *validation set*, which we use to compute *validation loss*. The remainder of the data will be called the *training set*. Let the first 10 entries of the data set be the validation set, and the last 40 entries be the training set. Concatenate their features into matrices vX and tX, and their responses into vectors vY and tY.

(a) [10 points] Write a function *ridge_regression*$(tX, tY, l)$ that takes the training features, training responses and regularizing parameter $\lambda$, and outputs the exact solution $\theta$ for ridge regression. Report the resulting value of $\theta$ for $\lambda = 0.15$.

(b) [10 points] Use the following code to plot graphs of the validation loss and training loss as $\lambda$ varies on logarithmic scale from $\lambda = 10^{-5}$ to $\lambda = 10^{0}$. Write down the value of $\lambda$ that minimizes the validation loss.

```
import matplotlib.pyplot as plt

tn = tX.shape[0]
vn = vX.shape[0]
tloss = []
vloss = []
index = -np.arange(0,5,0.1)

for i in index:
    w = ridge_regression(tX,tY,10**i)
    tloss = tloss+[np.sum((np.dot(tX,w)-tY)**2)/tn/2]
    vloss = vloss+[np.sum((np.dot(vX,w)-vY)**2)/vn/2]
```

```
plt.plot(index,np.log(tloss),'r')
plt.plot(index,np.log(vloss),'b')
```

## 4. K-Means [30 points]

Consider the data in the file “kmeans-image.txt”. This file contains a large number (210,012) of length 3 vectors, each on one line. Each vector represents the red, green, and blue intensity values of one of the pixels in the image shown (Fig 1). The image has $516$ rows and $407$ columns. The pixels in the file are listed row by row from top to bottom, and within each row from left to right. For example, the first pixel in the file is the uppermost left pixel in the image. The second line of the file contains the pixel to the right of that one, and so on. In this question, we will explore clustering methods, applying them in particular to the problem of dividing the pixels of the image into a small number of similar clusters. Consider the K-means clustering algorithm, as described in class. In particular, consider a version in which the inputs to the algorithm are:

- The set of data to be clustered. (i.e., the vectors $x^{(1)}, x^{(2)}, x^{(3)}, ...$)
- The desired number of clusters, K.
- Initial centroids for the K clusters.

Then the algorithm proceeds by alternating: (1) assigning each instance to the class with the nearest centroid, and (2) recomputing the centroids of each class—until the assignments and centroids stop changing. Please use squared Euclidean distance (Lecture 5, Eq. 2) as the metric for clustering.

There are many implementations of K-means publicly available. However, please implement K-Means on your own. Then, use your implementation to cluster the data in the file mentioned above (“kmeans-image.txt”), using $K = 8$, and the initial centroids as given below in the table. Report the error and centroid IDs at each iteration of the algorithm. Lastly, display the image with each pixel intensity replaced by its assigned centroid.

| R | G | B |
|---|---|---|
| 255 | 255 | 255 |
| 255 | 0 | 0 |
| 128 | 0 | 0 |
| 0 | 255 | 0 |
| 0 | 128 | 0 |
| 0 | 0 | 255 |
| 0 | 0 | 128 |
| 0 | 0 | 0 |