CSC2515: Assignment 1

Due on Monday, January 29, 2018

Matthew Wong

January 25, 2018

Problem 0

Introductory information and readme instructions
Attached, you will find the required submissions *faces.py, faces.tex & faces.pdf* in addition to the various folders of uncropped and cropped images. Comments about the code are located in the *faces.py* file and the code is separated out by function and can be run as necessary from a command line argument. Outside of the function definitions for data retrieval, cleansing and storage, code for each section is labelled appropriately and can be run without internal dependencies. Code was written in Python 3.5 - the packages used are outlined in the file header.

Problem 1

Dataset description
The dataset consists of 2359 grayscale 25×25-pixel of images of the letter "a," rendered using various fonts. The letter appears in both uppercase and lowercase, and there is considerable variation in the appearance of the letters. However, examining the dataset, it appears that there are more print letters than letters in other fonts, and most of the letters are lowercase. A random sample of 25 "a"s is shown in Figure **??**.
Advice: when describing a dataset, give a general description, and mention the things that would be important for a person who is working on the same problem that you are working on.

Problem 2

Splitting up the Data.
The algorithm that was used to shuffle the dataset originated from the *numpy.random* library.

Problem 3

Binary Classification between Steve Carrel and Alec Baldwin

As outlined in the assignment documentation, linear regression was used to create a binary classifier. The cost function that was minimized was the quadratic cost function, also known as the least-squares cost function as outlined below.

$$J(\theta) = \sum_{i=1}^{n}(h_\theta(x^{(i)} - y^{(i)})^2$$

Using the Gradient-Descent algorithm, a training accuracy of 100% was obtained on the training data compared to a validation accuracy of 83%. The following code was used to implement the binary classifier.

```
//part3 function call within faces.py
 #Declare list of actors for processing
  act = ['Alec Baldwin', 'Steve Carell']
  getCroppedData(act,"facescrub_actors.txt",3)
  x = getDataMatrix(3)
  y = getDataLabels(act,3)

  #concatenate the data matrix and labels for processing

  complete = np.column_stack((x,y))
  np.random.seed(4)
  np.random.shuffle(complete)
  training = complete[0:200,:]
  validation = complete[200:230,:]
  test = complete[230:260,:]
  print("Number of Baldwin in training set: " +
      str((np.shape(np.where(training[:,-1]==0))[1])))
  print("Number of Baldwin in validation set: " +
      str((np.shape(np.where(validation[:,-1]==0))[1])))
  print("Number of Baldwin in test set: " +
      str((np.shape(np.where(test[:,-1]==0))[1])))
  print("Number of Carell in training set: " +
      str((np.shape(np.where(training[:,-1]==1))[1])))
  print("Number of Carell in validation set: " +
      str((np.shape(np.where(validation[:,-1]==1))[1])))
  print("Number of Carell in test set: " +
      str((np.shape(np.where(test[:,-1]==1))[1])))
  theta0 = np.ones((1025,))
  #theta0 = np.random.normal(0,0.2,1025)

  training_labels = training[:,-1]
  training = np.transpose(training[:,:-1])
  theta = grad_descent(f,df,training,training_labels,theta0,0.00001,10000)

  #Training hypothesis
  ones_t = np.ones((1,training.shape[1]))
  training_with_bias = vstack((ones_t,training))
  training_hypothesis = np.dot(theta.T,training_with_bias)
  i = 0
  while i < training_hypothesis.shape[0]:
     if training_hypothesis[i] > 0.5:
        training_hypothesis[i] = 1
```

```
        else:
            training_hypothesis[i] = 0
        i+=1
    print("Accuracy percentage in training set:" +
        str(np.sum(np.equal(training_hypothesis,training_labels))/200.0))

    #Validation hypothesis
    validation_labels = validation[:,-1]
    validation = np.transpose(validation[:,:-1])
    ones_v = np.ones((1,validation.shape[1]))
    validation_with_bias = vstack((ones_v,validation))
    validation_hypothesis = np.dot(theta.T,validation_with_bias)
    i=0
    while i < validation_hypothesis.shape[0]:
        if validation_hypothesis[i] > 0.5:
            validation_hypothesis[i] = 1
        else:
            validation_hypothesis[i] = 0
        i+=1

    print("Accuracy percentage in validation set:" +
        str(np.sum(np.equal(validation_hypothesis,validation_labels))/30.0))

    test_labels = test[:,-1]
    test = np.transpose(test[:,:-1])
    ones_test = np.ones((1,test.shape[1]))
    test_with_bias = vstack((ones_test,test))
    test_hypothesis = np.dot(theta.T,test_with_bias)
    i=0
    while i < test_hypothesis.shape[0]:
        if test_hypothesis[i] > 0.5:
            test_hypothesis[i] = 1
        else:
            test_hypothesis[i] = 0
        i+=1

    print("Accuracy percentage in test set:" +
        str(np.sum(np.equal(test_hypothesis,test_labels))/30.0))
    return theta
```

The Gradident Descent algorithm was modified for the purposes for this assignment was taken from the Galaxy code posted on the CSC2515 website. In order to get the system to work, a suitable number of maximum iterations was performed, in addition to modifying the learning rate, α. I did not formalize my selection of the two parameters, but tried experimenting with different values. I noticed that as α was increased by a factor of 10 (e.g. 10^{-4}), there were errors in the matrix operations being performed, resulting in numerical overflow issues. As α decreased, (e.g. 10^{-6}) it was observed that both the training and validation sets exhibited lower accuracy in correctly identifying the individuals - perhaps indicating a local minimum. However, this could certainly be investigated by modifying the number of maximum iterations (but in the cases mentioned, testing was completed on 10000 iterations).

Problem 4

Splitting up the Data.
The algorithm that was used to shuffle the dataset originated from the *numpy.random* library.

Problem 5

Splitting up the Data.
The algorithm that was used to shuffle the dataset originated from the *numpy.random* library.

Problem 6

Mathematical Derivations of Gradient Descent

(A) We can compute $\frac{\partial J}{\partial \theta_{pq}}$ by expanding the expression as outlined in the assignment documentation as follows:

$$J(\theta) = \sum_i \left((\theta^T x^{(i)} - y^{(i)})_1^2 + (\theta^T x^{(i)} - y^{(i)})_2^2 + \ldots + (\theta^T x^{(i)} - y^{(i)})_j^1 \right)$$
$$J(\theta) = (\theta^T x^{(1)} - y^{(1)})_1^2 + (\theta^T x^{(1)} - y^{(1)})_2^2 + \ldots (\theta^T x^{(1)} - y^{(1)})_j^1 + (\theta^T x^{(2)} - y^{2)})_1^2 + \ldots + (\theta^T x^{(i)} - y^{(i)})_j^2$$

Therefore, it appears we can model this double summation as a summation over the p^{th} column and q^{th} row. Therefore, the partial derivative of a specific element, say, element pq can be expressed as:

$$\boxed{\frac{\partial J}{\partial \theta_{pq}} = 2x_q^p(\theta^T x^p - y^p)_q^2}$$