CSC2515: Assignment 1

Due on Monday, January 29, 2018

Matthew Wong

January 29, 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 at the beginning of the file.

Additional Information: Due to time limitations, I was unable to put all of the cropped/uncropped folders into separate directories - instead, you will find these directories with a number appended (in most cases to highlight which question was attempted). For compilation purposes, the images that are saved as a result of code exeuction must be included in the same directory as the .tex file.

Problem 1

Dataset description

The uncropped dataset contains approximately 1500 images of 12 actors and actresses of varying shapes, sizes and colours. The various cropped dataset folders contain cropped 32×32, greyscale images that are relevant to that part outlined in the assignment specification. The cropped images were obtained by crawling through the FaceScrub dataset and applying the bounding boxes located those files. Overall, they seem to be accurate. There are numerous objectives in this assignment that pertain to this dataset - but in particular, the objective was the development of a linear regression classifier using Gradient Descent to distinguish between males and females as well as categorizing individual actors and actresses. The main data source to accomplish this task will be the greyscale pixel intensities of the cropped images. A few of the uncropped and cropped images of various actors will be shown below.

(a) Baldwin

(b) Drescher

(c) Hader

(d) Cropped Baldwin

(e) Cropped Drescher

(f) Cropped Hader

Figure 1: Cropped and Uncropped sample images

Based on additional images from the dataset, it can stated that some actors' and actresses' faces are sometimes not directly facing the camera - hence, a superposition of the images for each actor may not be representative of their face. Furthermore, a linear regression classifier may not be the best use case here - a logistic regression or the implementation of a convolution neural network would be more appropriate to handle rotated actors' and actresses' faces.

Problem 2

Splitting up the Data.

The algorithm that was used to shuffle the dataset originated from the *numpy.random* library. The *random.seed()* command was used and different seeds set in order to get the best results for training and validation in addition to allowing for reproducible results. A detailed explanation of this command can be located on the official Numpy documentation.

For parts 3, 4, and 5, the data was flattened into rows of 1024 pixels. The label (either Carell vs. Baldwin or Male vs. Female) was appended to the last column. The entire data matrix was shuffled using *numpy.random.shuffle* after using the *numpy.random.seed()* command. The last column from the resultant matrix was separated from the column. The new matrix, with the shuffled rows was then used for classification

Parts 7 and 8 used a different scheme and the matrix geometry taken from the assignment specification. The data matrix and data labels were obtained separately. The labels were then stacked below the data matrix to form a complete matrix. This matrix was also shuffled using *numpy.random.shuffle*, the last k rows (e.g. k=6) were extracted to form the labels and the number of columns was selected appropriately for the training and validation sets (e.g. 600 and 90 - approximately 100 and 15 for training and validation respectively.)

Post-Mortem Addendum/Future Changes: In retrospect, it would have been preferable to write a function to randomly select, based on actor name, directory, and number of files to generate the appropriate data matrix and label matrix. Unfortunately, due to time constraints, the *os.listdir*() command was used to traverse through a directory, and based on experimentation, it appears to do so alphabetically. The workflow would have been the following:

1. Specify number of training, validation, and test set sizes.
2. Specify directory containing the cropped images
3. Specify which actors to look up
4. Obtain data and label matrices for each actor individually
5. Concatenate the matrices vertically, then shuffle

By implementing the above method, a more accurate training, validation, and test set would have been acquired for each of the tasks, potentially resulting in results more aligned with the instructor/TA code.

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 output the results of the hypothesis from the binary classifier.

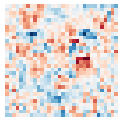
```
#Input - the training_hypothesis matrix, of shape m x 1, where m is the number of
    examples
   i = 0
   while i < training_hypothesis.shape[0]:
      if training_hypothesis[i] > 0.5:
5        training_hypothesis[i] = 1
      else:
         training_hypothesis[i] = 0
      i+=1
```

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). Furthermore, the above loop could further be optimized using a vectorization technique or by implementing one-hot initialization (e.g. as in the Tensorflow package).

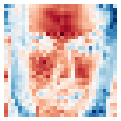
Problem 4

Visualizing Binary $\theta's$

(A) The two figures below show the visualized $\theta's$ for training on the complete set of Baldwin and Carell and of two images respectively.



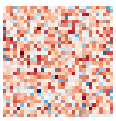
(a) $\theta's$ with full training set.



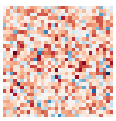
(b) $\theta's$ with training size 2.

Figure 2: Visualized $\theta's$

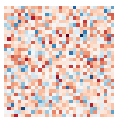
(B) Experiments were run by modifying both the initializations of θ and the number of iterations - however, it was noted that the experiments with an initialized θ matrix of 1's most closely resembled a face, indicating the the initialization of θ may be hindering the appearance of "face-like" θ values.



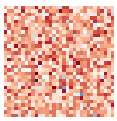
(a) 2500 iterations



(b) 5000 iterations

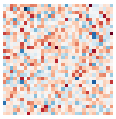


(c) 7500 iterations

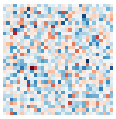


(d) 10000 iterations

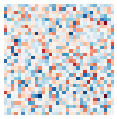
Figure 3: θ's after log-normal initialization of θ with $\mu = 0$ and $\sigma = 0.2$



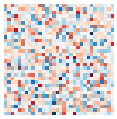
(a) 2500 iterations



(b) 5000 iterations

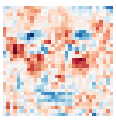


(c) 7500 iterations

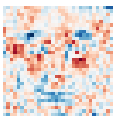


(d) 10000 iterations

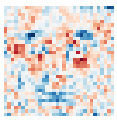
Figure 4: θ's after Normal initialization of θ with $\mu = 0$ and $\sigma = 0.2$



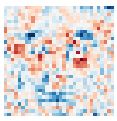
(a) 2500 iterations



(b) 5000 iterations



(c) 7500 iterations



(d) 10000 iterations

Figure 5: θ's 1 initialization

Problem 5

Investigating overfitting

The results from varying the training size on the training and validation accuracies are shown below.

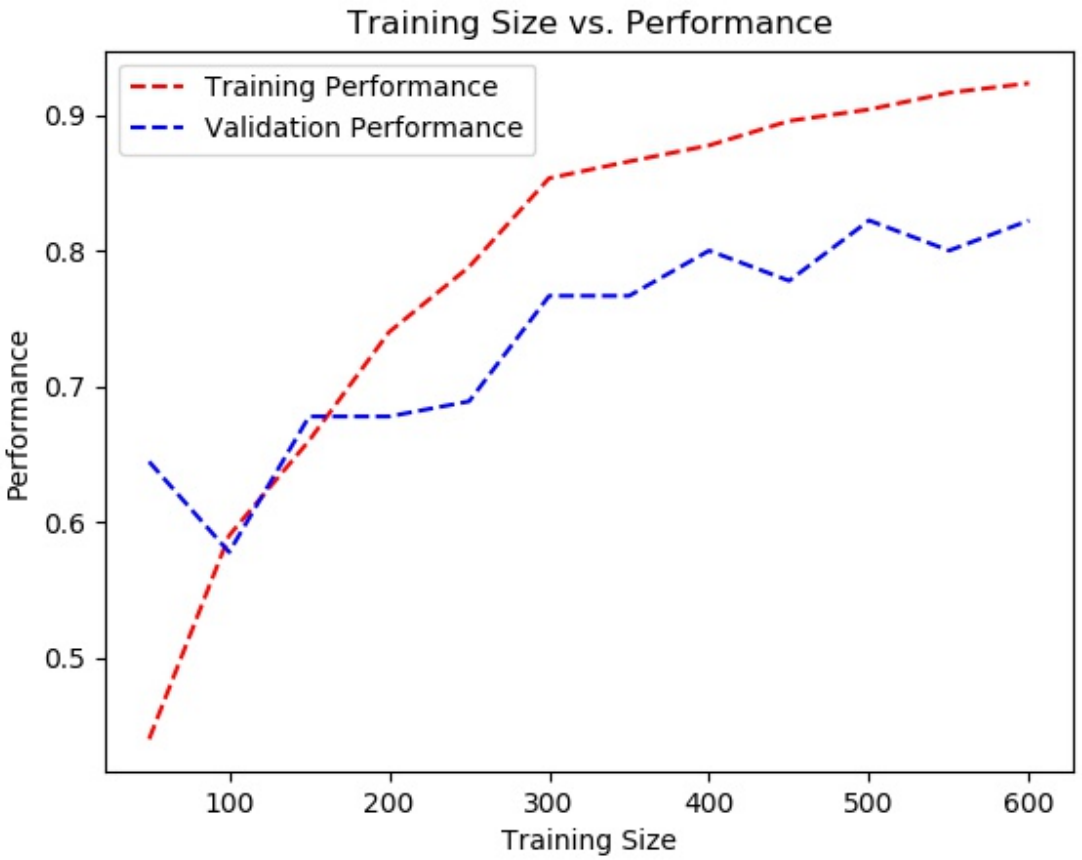


Figure 6: Graph comparing size of training set size vs. training accuracy and validation in increments of 50 images

A potential explanation for the results in the figure may be that the training and validation sets, when split amongst the individual actors, may be favouring some actors over others. There is an intersection at 100 training examples, and it appears the the accuracy between the two sets is almost identical. The fact that there is a large discrepancy between the accuracies between a training sample size of 180 to 300 (e.g. validation accuracy stays relatively constant) may reveal support for the above claim that actors are not being represented properly, presumably due to a lack of rigour in the splitting between data sets for the individual actors. Upon testing on the other actors and actresses not included in *act*, the validation accuracy obtained was 69%. As outlined in the **Post-Mortem Addendum** in Section 2, the selection of data from the images may also have played a role in this behaviour.

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:

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

(B) Based on the assignment specification, let X be a matrix of size $n \times m$, where n is the number of features and m is the number of training examples. Let θ be a matrix of size $k \times n$, where k is the number of features. Let Y be a $k \times m$ matrix. The provided expression taken from the assignment specification, with matrices substituted into their respective variables, gives the following expression:

$$2 \begin{bmatrix} x_{11} & \ldots & x_{1m} \\ \vdots & \ddots & \vdots \\ x_{n1} & \ldots & x_{nk} \end{bmatrix} \times \left[\begin{bmatrix} \theta_{11} & \ldots & \theta_{1n} \\ \vdots & \ddots & \vdots \\ \theta_{k1} & \ldots & \theta_{kn} \end{bmatrix}^T \times \begin{bmatrix} x_{11} & \ldots & x_{1m} \\ \vdots & \ddots & \vdots \\ x_{n1} & \ldots & x_{nm} \end{bmatrix} - \begin{bmatrix} y_{11} & \ldots & y_{1m} \\ \vdots & \ddots & \vdots \\ y_{k1} & \ldots & y_{km} \end{bmatrix} \right]^T$$

Simplification:

$$\begin{bmatrix} 2x_{11} & \ldots & 2x_{1m} \\ \vdots & \ddots & \vdots \\ 2x_{n1} & \ldots & 2x_{nk} \end{bmatrix} \times \begin{bmatrix} \theta_{11}x_{11} + \ldots + \theta_{k1}x_{n1} - y_{11} & \ldots \\ \vdots & \vdots \\ \ldots & \theta_{1n}x_{1m} + \ldots + \theta_{kn}x_{nm} - y_{km} \end{bmatrix}^T$$

If the above matrix computation is carried out, we can observe, from part A, that the pq term in the derivative of the cost function refers to an element within this gradient matrix. For example, the top left hand side derivative, e.g. $\frac{\partial J}{\partial \theta_{11}} = 2x_{11}(\theta_{11}x_{11} + \ldots + \theta_{k1}x_{n1} - y_{11}$. **Note:** The top left hand corner element of the gradient was selected for ease of computation, but any element within this gradient matrix would be valid.

```
#Cost Function
def f2(x,y,theta):
    ones = np.ones((1,x.shape[1]))
    x = np.vstack((ones,x))
    hypothesis = np.matmul(theta.T,x)
    loss = np.power((hypothesis-y),2)
    return np.sum(loss)
```

```
#Gradient Function
def df2(x,y,theta):
    ones = np.ones((1,x.shape[1]))
    x = vstack((ones,x))
```

```
    hypothesis = np.matmul(np.transpose(theta),x)-y
    hypothesis = np.transpose(hypothesis)
    gradient = 2*np.matmul(x,hypothesis)
    return gradient
```

(C) The code for testing the implementation of the cost and gradient is shown below.

```
#Testing the cost and gradient functionality
def part6():
    #Testing the loss function
    x = np.random.normal(0,0.6,(20,15))
    y = np.random.normal(0.2,0.4,(5,15))
    theta = np.random.normal(-0.1,0.3,(21,5))
    h = 0.00001
    #Cost of individual component
    testarr1 = np.zeros(theta.shape)
    testarr1[3,4] = h
    print("Cost is:")
    print((f2(x,y,theta+testarr1)-f2(x,y,theta-testarr1))/(2*h))
    print("Gradient is:")
    print(df2(x,y,theta)[3,4])
    testarr2 = np.zeros(theta.shape)
    testarr2[1,4] = h
    print("Cost is:")
    print((f2(x,y,theta+testarr2)-f2(x,y,theta-testarr2))/(2*h))
    print("Gradient is:")
    print(df2(x,y,theta)[1,4])
    testarr3 = np.zeros(theta.shape)
    testarr3[2,3] = h
    print("Cost is:")
    print((f2(x,y,theta+testarr3)-f2(x,y,theta-testarr3))/(2*h))
    print("Gradient is:")
    print(df2(x,y,theta)[2,3])
    testarr4 = np.zeros(theta.shape)
    testarr4[8,4] = h
    print("Cost is:")
    print((f2(x,y,theta+testarr4)-f2(x,y,theta-testarr4))/(2*h))
    print("Gradient is:")
    print(df2(x,y,theta)[8,4])
    testarr5 = np.zeros(theta.shape)
    testarr5[10,1] = h
    print("Cost is:")
    print((f2(x,y,theta+testarr5)-f2(x,y,theta-testarr5))/(2*h))
    print("Gradient is:")
    print(df2(x,y,theta)[10,1])
```

The output that results from execution of this code is below:

```
Cost is:
-8.507892229658864
Gradient is:
-8.5078922306346
Cost is:
```

```
-5.070817460506305
Gradient is:
-5.070817460432679
Cost is:
-11.655092452400593
Gradient is:
-11.655092451975934
Cost is:
-3.5024079892309596
Gradient is:
-3.5024079896724483
Cost is:
-0.24353752436923057
Gradient is:
-0.2435375237734189
```

The h was selected so as to try to compute a small difference, similar to the definition of the derivative - as h further decreases, you would observe that the gradient and incremental difference in cost appear to converge.

Problem 7

Linear Regression with Multiple Actors

Running the linear regression classifier results in a training accuracy of 89% and a validation accuracy of 56%. Both the α and number of iterations were modified, to varying degrees of success - however, it is worth noting that the validation accuracy is significantly lower than the training accuracy which may be a symptom of overfitting.

Two loops were written in order to obtain the training hypothesis and validation hypothesis labels. The steps are written below.

1. A matrix of 0's was initialized to the same shape as that of the label matrix
2. Each column in the label matrix was scanned for it's maximum value and the index of that value was stored in a new variable, *max*
3. Starting at the 0^{th} column, a 1 was stored into the 0's matrix for each index in *max*, and iterated through until the last column was reached.
4. The resultant matrix now had only one value of 1 in each column and was compared to the label matrix.

An example of this for the training data is shown below.

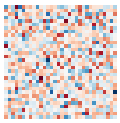
```
#Training hypothesis
ones_t = np.ones((1,training.shape[1]))
training_with_bias = vstack((ones_t,training))
training_hypothesis = np.matmul(theta.T,training_with_bias)
max = training_hypothesis.argmax(axis=0)
max = np.array(max)
training_hypothesis_labels = np.zeros((training_hypothesis.shape))
print(training_hypothesis_labels.shape)
i = 0
while i < training_hypothesis_labels.shape[1]:
    index = max[i]
    training_hypothesis_labels[index,i]=1
    i+=1

correct = 0
i = 0
while i < training_hypothesis_labels.shape[1]:
    if np.array_equal(training_hypothesis_labels[:,i],training_labels[:,i]) is True:
        correct += 1
    i+=1
print("Training accuracy is: " + str(correct/600.0))
```

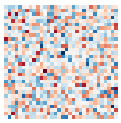
Problem 8

Visualizing $\theta's$ with multiple label matrix

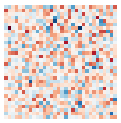
The 6 figures below show the visualized $\theta's$ for training on the full set of actors in *act* with the label matrix, as outlined in the assignment specification.



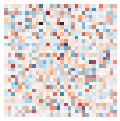
(a) Baldwin $\theta's$



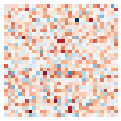
(b) Bracco $\theta's$



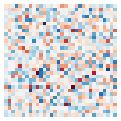
(c) Carell $\theta's$



(d) Gilpin $\theta's$



(e) Hader $\theta's$



(f) Harmon $\theta's$

Figure 7: θ values for each individual actor

A potential explanation for the lack of "face-like" appearance in $\theta's$ obtained after training may be that the size of the training set is too large for each actor. As observed in part 4, with only two actors from the Baldwin and Carell sets, the $\theta's$ visualized more resembled faces.