CSC411: Assignment 1

Due on Monday, January 29, 2018

Calvin Sanghera

January 31, 2018

Project Setup and Data Retrieval

The project was coded using the Python 2.7 interpreter and compiled on a Macbook with OS 10.13.2
.
The images used in this assignment were obtained from the FaceScrub dataset using the modified *get_data.py* script. I modified the script such that it cropped, resized and converted the image to gray scale (if necessary) as the images were downloading and being saved to the cropped and uncropped folders. In some cases, for Radcliffe and Gilpin, less than 90 images were obtained. Consequently, other parts of the assignment were affected.

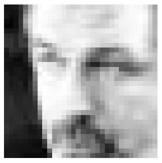
I have also submitted a folder of the zipped cropped images I used in this assignment.

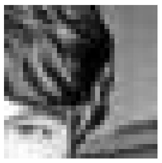
Problem 1

(a) The actor is not the sole member of picture.

(b) The actresses' face is slightly obscured.

(c) The actor's face is clearly in focus in the picture.


(d) After cropping we can see bounding box was not very accurate.


(e) Cropping eliminated a majority of the face of the actor.

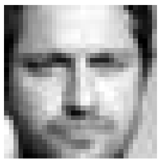

(f) Cropped headshot shows the bounding box was not accurate.

Figure 1

The uncropped dataset of images consists of a mixture of headshot and half body shots for the most part. Some instances of the images were full body shots. There are some pictures that have the actor/actress of focus and other subjects. For example in Figure 1a) we can see its a shot of Alec Baldwin in one half of the picture and the other half has a female. Also in Figure 1b) we can see that a dog is licking Bracco's face. A majority of the uncropped photos clearly show the face of the actor or actress like the one in Figure 1c). In the cropped data set, it is evident that the bounding boxes are not all that accurate in some cases. In figure 1d) a large majority of the face is cut off and in 1e) three quarters of the face is cut off. Based on these observations, it is highly unlikely that we can easily align the cropped faces with each other. Moreover, the bounding boxes cut off the jaw/chin area like in figure f. This may affect how effective our algorithm can differentiate between men and woman because the jaw is typically a distinct difference between men and women.

Problem 2

The algorithm I used (can be found in sorter.py) to separate the images into training set, test set and validation set was taking the first 70 images for each actor or actress if possible and saved them to the training set folder. Then I took the next 10 images for each actor or actress and saved them to the test set folder. I then took the next 10 images for each actor and actress and saved it to the validation set folder. In the case of Gilpin and Radcliffe, due to the fact that there were less than 90 images retrieved from FaceScrub, less than 70 images were added to the test set for them. Then using a function faces.py these images were retrieved from their respective folders and put into sets. The test set and validation set were not affected. I chose not to randomize this process because I knew nothing about why the image urls were ordered the way they were in the actor and actresses text file. Therefore, I did not think my current implementation of the algorithm would negatively impact the results in the assignment.

Note: This algorithm was used in the makeset functions in faces.py. Also the scrip in sorter.py assumes the training, test, and validation folders have already been made.

Problem 3

Cost function = $J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})$ where m is the number of images, x^i represents a specific flattened image and y^i is the expected output for that image. In our case, it is 1 for Baldwin and -1 for Carell.

Gradient Descent Implementation: Our training set had 140 images: 70 for Baldwin and 70 for Carell. As a result, our matrix x was initially a 140 x 1024 matrix, where each row was a flattened image $x^i = \begin{bmatrix} p_1 & p_2 .... & p_{1024} \end{bmatrix}$. Each p is a pixel in the image. Our initial θ for gradient descent was a 1025 x 1 matrix of 0s. In order to get the dot product between x and θ^T to be compatible, the transpose of X that was padded with a row of 1s to account for bias was inputted into the gradient descent function.

After initially running the gradient descent algorithm, I was getting a NaN error. I realized that this was a result from me not dividing the initial input images by 255 and a large α, so with each iteration, the numbers were getting extremely large. Too find the ideal α I kept on changing the power of the exponent until I got ideal results. When the α was too small, it was evident that gradient descent was getting stuck in a local minimum as the results of the validation were poorer.

The cost on the training set was 0.004704194478195664.

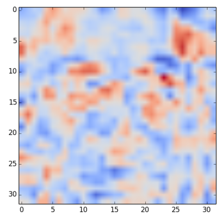
The cost on the validation set was 0.11458766716449731.

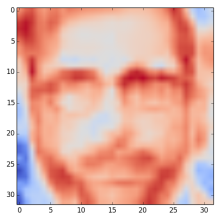
The performance for the classifier was perfect for both the training set and validation set.
Below is the used to compute the output of the classifier:

```python
def validate_results(theta, x, correct_y):
    correct_y = correct_y.tolist()
    result = dot(theta.T, x)
    predicted = []

    for i in range(len(correct_y)):
        if result[:,i] < 0:
            predicted.append(-1)
        else:
            predicted.append(1)
    correct = 0
    for j in range(len(correct_y)):
        if correct_y[j] == predicted[j]:
            correct += 1
    return correct
```

Note: The rest of the functions and code used for this section can be found in the faces.py in the part34() function.

Problem 4

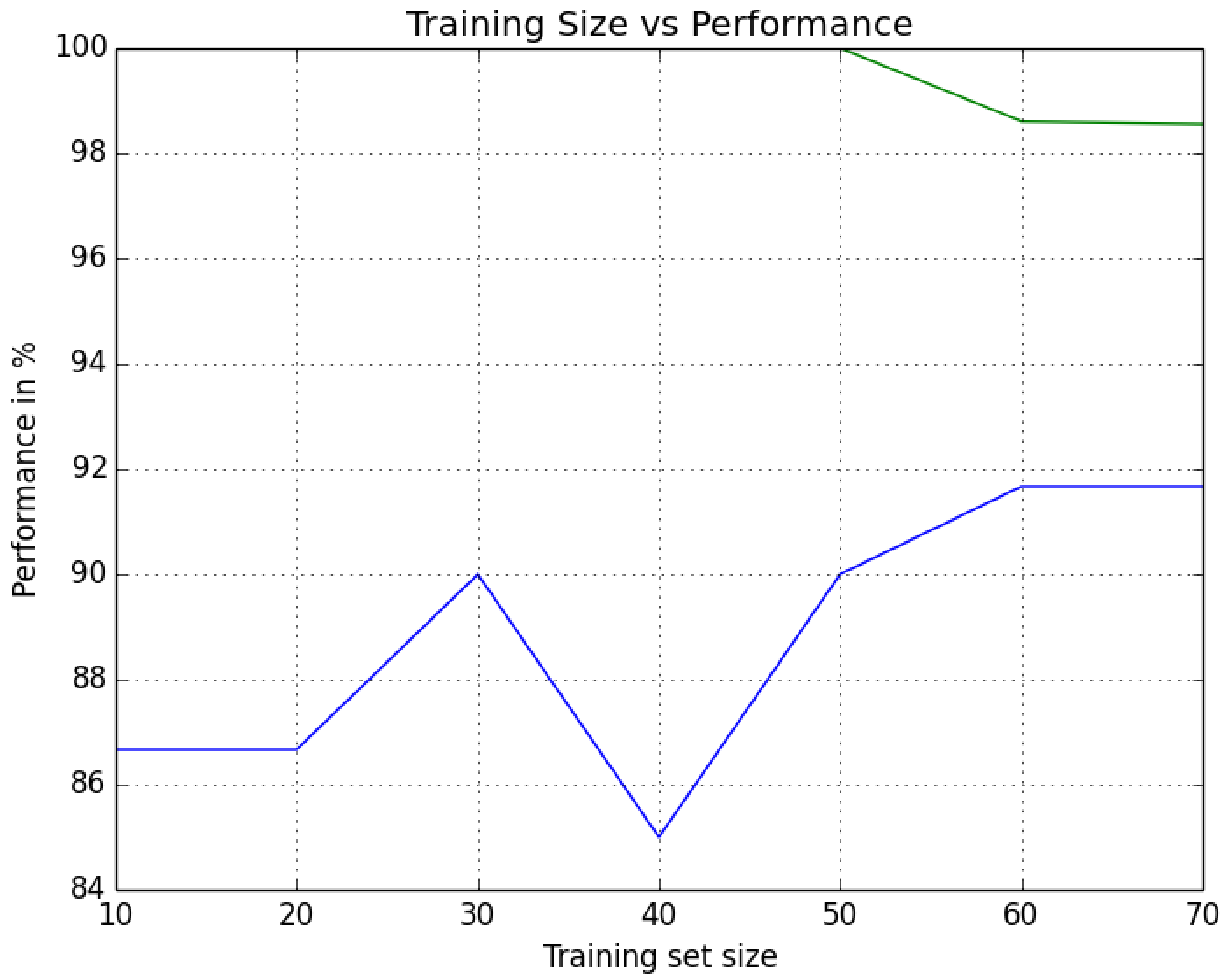

(a) Image from full training set


(b) Image from two image per actor training set

Under normal settings, on a full training set, it is expected to get an image generated like the one from figure a. In order to get that result I used an α of 0.0010 and a maximum of 300,000 iterations. We can see that figure a is a lot more pixilated and the face is not as well defined. In order to get an image that looks like figure b on a training set, decreasing the α and decreasing the number of iterations. Doing these two things, helped produce a clearer image because of over fitting. To get over fitting on a full training set I used an α of 0.00000010 and a maximum of 300 iterations.

Note: The code used to generate this data can be found in the problem34() function in faces.py and generated by running the main method.

Problem 5



In the above graph, the green line represents the result of the classifier on the training set and the blue line represent the result of the classifier on the validation set. It is evident that the classifier from the largest training set performed best on the validation set, but there is not strong enough evidence to suggest a correlation with the size of the training set and performance.

The validation result on other set of actors/actresses is $\frac{54}{60}$ of the images were correctly categorized as male or female. The classifier used was the θ obtained from the largest training set.

Note: The graph and other data from this problem is under the function5 in faces.py and can be obtained by running the main method.

Problem 6

a. $J(\theta) = \sum_i(\sum_j(\theta^T x^{(i)} - y^{(i)})_j^2)$
$\frac{\partial J}{\partial \theta_{pq}} = \sum_i(\sum_j \frac{\partial J}{\partial \theta_{pq}}(\theta^T x^{(i)} - y^{(i)})_j^2$
$\frac{\partial J}{\partial \theta_{pq}} = \sum_i(\sum_j(2(\theta^T x^{(i)} - y^{(i)})(\frac{\partial J}{\partial \theta_{pq}}\theta^T x^{(i)} - \frac{\partial J}{\partial \theta_{pq}}y^{(i)}))_j)$
$\frac{\partial J}{\partial \theta_{pq}} = \sum_i((2(\theta_p^T q x_q^{(i)} - y_q^{(i)})(x_p^{(i)})))$ is the partial derivative.

b. Lets define m as the number of training examples, and our θ will be a 1025 x 6 matrix and our X will be a 1025 x m matrix. This is because we have 1024 pixes in addition to the row of additional 1s to account for bias. In addition to this Y will be a m x 6 matrix.

So $2X(\theta^T - Y)^T$ can be expanded to :
$2X^1(\theta^T X^1 - Y^1) + 2X^2(\theta^T X^2 - Y^2) + ... + 2X^m(\theta^T X^m - Y^m)$:
Which is equivalent to $2\sum_i x^i(\theta^T x^i - y^i)$ and this is similar to the partial derivative we found in part a.

c. Here are the cost and vector functions:

```
def multi_f(x, y, theta):
    x = vstack( (ones((1, x.shape[1])), x))
    return sum(np.square((y - dot(theta.T,x).T)))

def multi_df(x, y, theta):
    x = vstack( (ones((1, x.shape[1])), x))
    return 2*dot(x, np.transpose(dot(theta.T,x)-y))
```

Note: The functions can be found in faces.py

Problem 7

After running gradient descent on the six actors, the success of actors and actresses identified correctly from the training set was 397 out of 408. The success rate for the validation set was 47 out of 60.

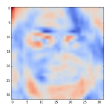
The parameters for gradient descent were both equations that were implemented from 6c. Otherwise, it took in the same parameters as the regular gradient descent function that was used earlier on in the assignment.

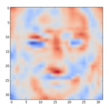
So the output of the model was a m x 6 matrix, where m was the number of images inputted. For each row in the matrix, I found the index with the larges value and in a new matrix, I set that index in the corresponding row to 1 and the rest of the elements in the row to 0. I continued to build a matrix by doing this and the result was what the expected output should look like. I then took this and compared it to what the actual output was and determined the correctness of the θ obtained. This can be seen in the code shown below in the validatemult function.

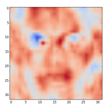
```

def validate_mult(theta, x, y):
    predicted = np.empty([0,6])
    x = vstack( (ones((1, x.T.shape[1])), x.T))
    b = dot(theta.T,x)
    for i in b.T:
         max = -10000000
         max_i = 0
         for j in range(len(i)):
              if i[j] > max:
                 max = i[j]
                 max_i = j
         a = [0 ] * 6
         a[max_i] = 1
         np.array(a)
         predicted = vstack((predicted, a))
    correct = 0
    for i in range(len(predicted)):
        if np.array_equal(predicted[i], y[i]):
            correct +=1
    return correct
```

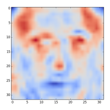
Note: All related code to this section can be found in the a1.py file in the part7 function and can be run by executing the main method.

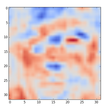
Problem 8


(a) Bracco


(b) Gilpin


(c) Harmin


(d) Baldwin


(e) Hader

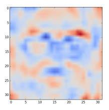

(f) Carell

Figure 3

From figure 3 it can be seen that the 6 θs used to generate these images do resemble faces. However the images generated were not clear enough to identify the actor or actress that they were meant to represent. Possible ways to better this would be over fitting by decreasing the number of iterations or making the α even smaller.

Note: Running the main method in faces.py will generate the images from problem 8.