CSC411: Assignment 1

Due on monday, January 29, 2018

Roustem Khamitov

January 30, 2018

Problem 1

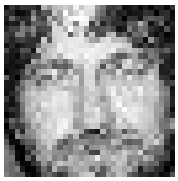
System configuration
All the code submitted for this assignment and referenced in this report was developed using python 2.7.13 on macOS Sierra Version 10.12.6.
Dataset description
The FaceScrub dataset contains images of 265 male celebrities and 265 female celebrities for a total of 106,863 images. The two .txt files provided information on a subset of this data that contained the name, URL for download, shape and the coordinates of the bounding box around their face for 6 male and 6 female actors. The file getdata.py is a short program that when run with the paths to the .txt files from the current directory as arguments will download the images and create a raw uncropped folder containing the entire data set, and then create a cropped folder that separates the pictures based on gender, crops the images, resizes them to 32 by 32 and converts them to grayscale.The first 3 images below are some examples of the raw unedited photos downloaded from the dataset, and the next 3 are examples that have been cropped, resized and gray scaled.

(a) A random raw image from the data set. It is higher quality than most at 1.9mb

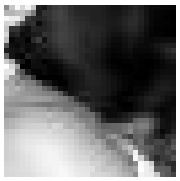
(b) Another random raw image of a lower quality image that is 8kb:

(c) An 320kb raw image

(d) A cropped, resized and grayscaled image



(e) A cropped, resized and grayscaled image



(f) A cropped, resized and grayscaled image

The raw images vary in size from around 3mb to under 10kb, and so do not constitute a dataset of uniform quality. Additionally, the bounding boxes aren't always accurate; the image in figure 5 has a significant portion of the jawline removed, a facial feature that is often very different among men and women. Even more problematic is the fact that some of the bounding boxes refer to portions of an image that do not contain a face, such as the image in figure 6. This, in combination with the fact in many of the photos the actors are facing different directions, makes it impossible to accurately align the faces in the dataset.

Problem 2

Separating the photos into sets
The algorithm that sorts the photos is very basic: I simply traverse the directories containing the cropped out images and retrieve the first 70 images of each actor for the training set, the next 10 images of each actor for the validation set and then another 10 for the test set. No randomization occurs here as this is implemented in the classifier code. The program to generate the 3 directories containing the sets is called getsets.py and it is run without any arguments. Run it after running the getdata.py mentioned in part 1. Otherwise, Sets.zip is a file that contains the training, validation and test sets that were obtained by running

getsets.py. Ensure that these 3 files are placed in the same repository as faces.py to run the code referenced in this report.

Problem 3

Hypothesis function
We are attempting to use linear regression to make a binary classifier that separates photos of Steve Carell from photos of Alec Baldwin. After we resize and flatten our images, each picture that we input is a flattened image vector that looks as follows:

$$X^{(i)} = [x_1, x_2, ..., x_{1024}] \tag{1}$$

where each x_i represents a pixel. In our equations, we make each $X^{(i)}$ a column vector and thus transpose our flattened vector. From now on in the report, whenever we refer to $X^{(i)}$ we are referring to a column vector that looks as follows:

$$X^{(i)} = \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ \vdots \\ x_{1024} \end{bmatrix} \tag{2}$$

We pad the first entry with a 1 to allow us to represent our hypothesis function as a multiplication of two vectors. Our hypothesis function that we will use to classify each input will be:

$$h_\theta(X_{(i)}) = \theta_0 + \theta_1 x_1 + ... + \theta_{1024} x_{1024} = \theta^T X^i \tag{3}$$

where :

$$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_{1024} \end{bmatrix} \tag{4}$$

Cost function:
Given m inputs, our cost function for any choice of θ is

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(X^{(i)}) - Y^i)^2 \tag{5}$$

where:

$$Y^{(i)} = \begin{cases} -1 & \text{if } X^{(i)} \text{ is Steve Carell} \\ 1 & \text{if } X^{(i)} \text{ is Alec Baldwin} \end{cases}$$

Then the gradient of our cost function:

$$\nabla J(\theta) = \frac{1}{m} \sum_{i=1}^{m} (h_\theta(X^{(i)}) - Y^i) X^{(i)} \tag{6}$$

We then use this gradient function when we run gradient descent to compute our θ.
Computing the output of the classifier:
After we compute our values of θ, we compute the predictions for a set of inputs X with the following function:

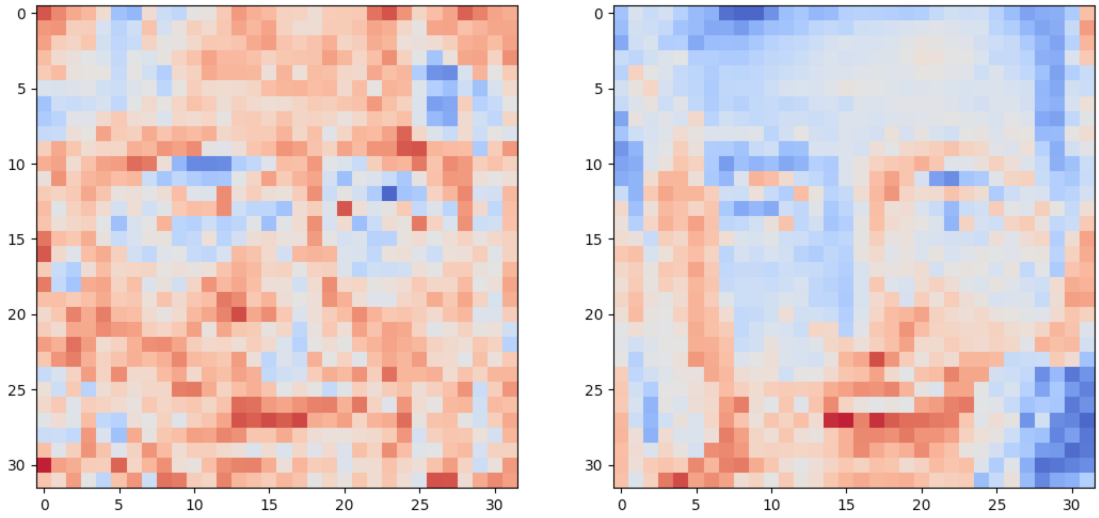
```
def get_prediction_binary(theta, x):
    x = vstack( (ones((1, x.shape[1])), x))
    h = dot(theta.T,x)
    predictions = []
    for i in range(len(h[0])):
        if h[0][i] > 0:
            predictions.append(1)
        if h[0][i] < 0:
            predictions.append(-1)

    return predictions
```

Perfomance of our classifer:
We run gradient descent on an initial θ that is a 0 vector with the following parameters:
max interations $= 30,000$
$\alpha = 1 * 10^{-3}$
After training on the full training set, the performance of the classifier is as follows:
The percentage of correct predictions on the training set is: 100.0
The percentage of correct predictions on the validation set is: 90.0
To reproduce the above result, enter the following in your terminal:
python faces.py part3
Value of the cost function:
The value of the cost function on the training set is: 0.00506538998835783
The value of the cost function on the validation set is: 0.16937626854552446
Decisions to make the classifier work:
Initially, we had to divide every value in our image vector by 255 so that every value was between 0 and 1. The initial value of α that we tried was too large and resulted in values that grew to quickly and resulted in a stack overflow. A value of $\alpha = 1 * 10^{-3}$ produced optimal results. Additionally, the initial values of θ did not affect the performance of the classifier as long as they were small enough. They were initialized to be zero.

Problem 4

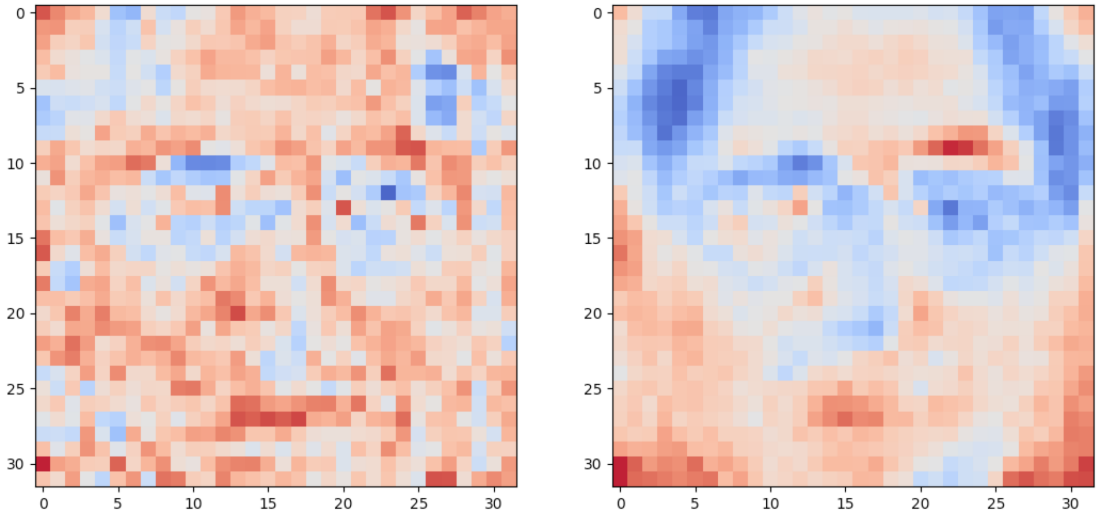
Part (a)
The following visualizations of θ was produced by running gradient descent with the same parameters as described in part 3 on a full training set of 70 pictures of each actors, and then on 2 images of each actor. The visualization on the left is from the full training set, and the visualization on the right is from trainining on a set of only 2 pictures per actor.



To reproduce the following visualizations, run faces.py with the argument part4a as follows:
python faces.py part4a

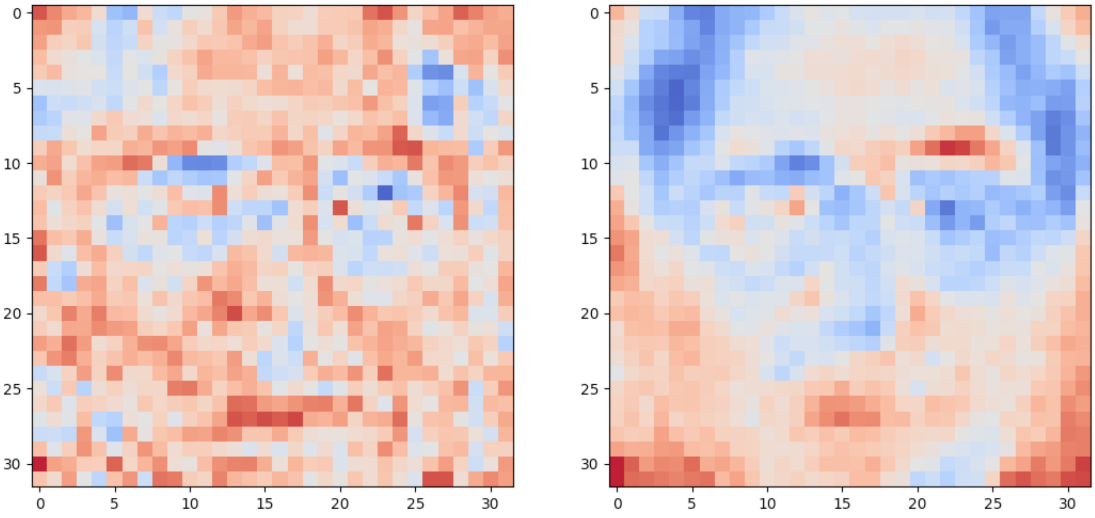
Part (b)

To obtain a visualization that contains a face and one that doesn't, we can change the max number of iterations that the gradient descent algorithm runs. The visualization on the left is obtained from running gradient descent on the full training set for 30000 iterations, and the one on the right is obtained by running it on 10 iterations only. That is the only difference between the two visualizations:



To reproduce the above visualization, run faces.py with the argument part4b as follows:
python faces.py part4b

Additionally, reducing the value of α from $1 * 10^{-3}$ to $1 * 10^{-6}$ while keeping the number of iterations the same produces a similar result as above:

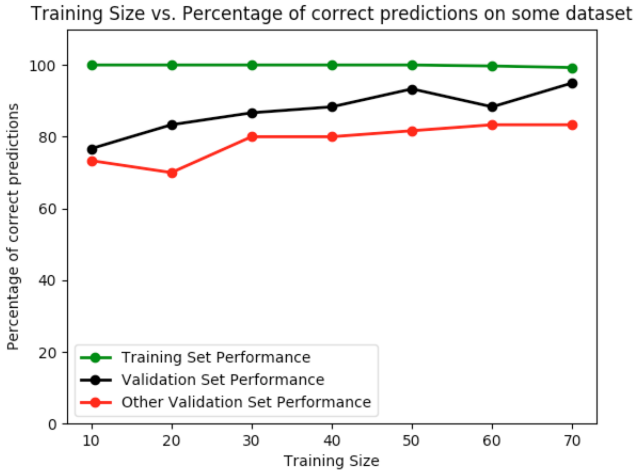


Problem 5

For this problem, we trained our classifier on training sets of size 10, 20, 30, 40, 50, 60 and 70 images for each actor in the set. At each iteration, we ran our classifier:

1. first on the same set we trained on(if we trained on a set of size 10, then we ran our classifier on that same set)
2. then on the validation set(always of size 10) for the same 6 actors we trained on
3. then on the validation set(always of size 10) of the 6 actors we haven't trained on

We plotted the percentage of correct predictions for 1, 2 and 3 against the size of the set we trained on:



Additionally, the program prints the result of 3 after each iteration to the terminal. This looks as follows:

The percentage of correct predictions on the validation set of actors not trained on after training on a set of size 10 is: 73.3333333333

The percentage of correct predictions on the validation set of actors not trained on after training on a set of size 20 is: 70.0

The percentage of correct predictions on the validation set of actors not trained on after training on a set of size 30 is: 80.0

The percentage of correct predictions on the validation set of actors not trained on after training on a set of size 40 is: 80.0

The percentage of correct predictions on the validation set of actors not trained on after training on a set of size 50 is: 81.6666666667
The percentage of correct predictions on the validation set of actors not trained on after training on a set of size 60 is: 83.3333333333
The percentage of correct predictions on the validation set of actors not trained on after training on a set of size 70 is: 83.3333333333
To reproduce the above graph, as well as the result of running the classifier on the validation set of 6 actors who we didn't train on, run the following:
python faces.py part5

Problem 6

Part (a)
Now, θ is a 1025 by k matrix where k is the number of labels(or categories), each $X^{(i)}$ is a 1025 by 1 column vector defined as before(the first entry is a 1) and each $Y^{(i)}$ is a k by 1 vector. Then:

$$\theta^T X^i - Y^i = \begin{bmatrix} [\sum_{z=1}^{1025} \theta_{z1} x_z^i] - y_1^i \\ \vdots \\ [\sum_{z=1}^{1025} \theta_{zk} x_z^i] - y_k^i \end{bmatrix} \tag{7}$$

So

$$J(\theta) = \sum_{i=1}^{m} [([\sum_{z=1}^{1025} \theta_{z1} x_z^i] - y_1^i)^2 + ... + ([\sum_{z=1}^{1025} \theta_{zk} x_z^i] - y_k^i)^2] \tag{8}$$

Now, when calculating the partial derivative for this function with respect to any entry θ_{pq}, every squared term that doesn't contain the θ_{pq} is treated as a constant and thus disappears. Hence the partial derivative with respect to θ_{pq} is :

$$\frac{\partial J}{\partial \theta_{pq}} = 2 \sum_{i=1}^{m} [([\sum_{z=1}^{1025} \theta_{zp} x_z^i] - y_p^i) x_p^{(i)}] \tag{9}$$

Part (b) Recall that θ is a 1025 by 6 matrix, X is a 1025 by m matrix and Y is a 6 by m matrix, where m is the number of images.
Then:

$$\theta^T X - Y = [\theta^T X^{(1)} - Y^{(1)} \quad \theta^T X^{(2)} - Y^1 \quad .... \quad \theta^T X^{(m)} - Y^{(m)}] \tag{10}$$

Hence:

$$2X(\theta^T X - Y)^T = 2X^{(1)}(\theta^T X^{(1)} - Y^{(1)}) + 2X^{(2)}(\theta^T X^{(2)} - Y^{(2)} + ... + 2X^{(m)}(\theta^T X^{(m)} - Y^{(m)} \tag{11}$$

Which is a matrix whose entries are equivalent to the partial derivative found earlier evaluated at the entries of θ
Therefore:

$$DJ(\theta) = 2X(\theta^T X - Y)^T \tag{12}$$

Part (c)
Below is the above cost function and its derivative:

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


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

Part (d) We compute the difference between the gradient and the approximation at 5 different coordinates using the following code:

```
 def get_finite_diff(x, y, theta, h):
    G = df_m(x, y, theta)
    row = [i for i in range(1025)]
    col = [i for i in range(6)]
    random.shuffle(row)
    random.shuffle(col)
    for i in range(5):
        H = np.zeros([1025, 6])
        H[row[i], col[i]] = h
        finite_diff = G[row[i], col[i]] - (f_m(x, y, theta + H) - f_m(x, y, theta))/h
        print("The difference between the gradient and approximation at" + "(" + str(row[i]
```

This code was run with $h = 1 * 10^{-6}$ to obtain the following result:

```
The difference between the gradient and approximation at(770, 4) is: -9.7934094022811e-05
The difference between the gradient and approximation at(893, 5) is: -9.0701904053247e-05
The difference between the gradient and approximation at(636, 2) is: -0.000147700377123527
The difference between the gradient and approximation at(55, 1) is: -0.000151650920130030
The difference between the gradient and approximation at(281, 3) is: -0.000157224510213715
```

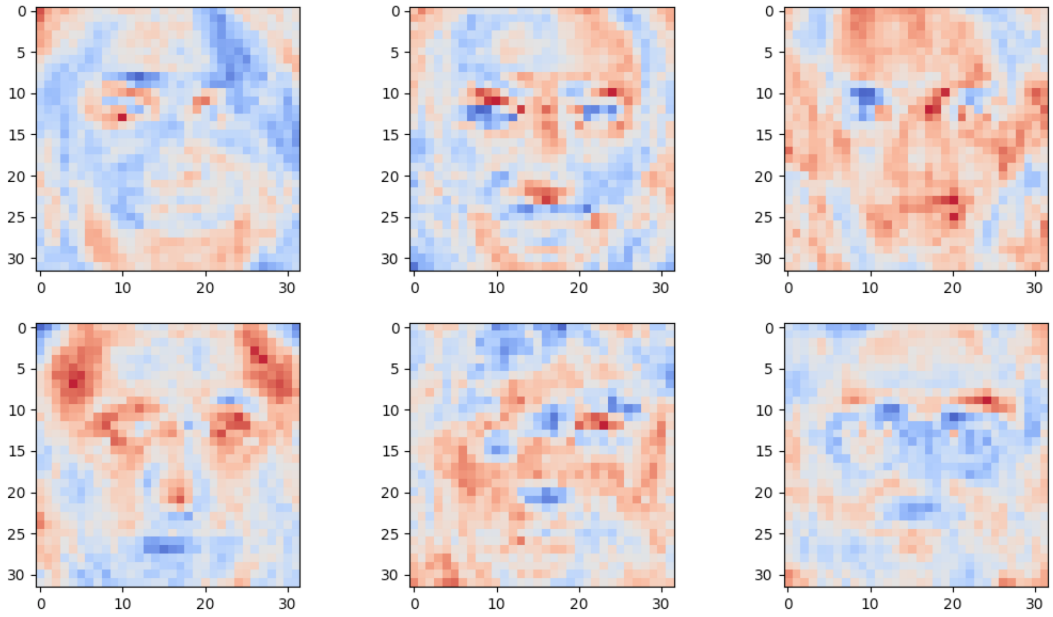
To reproduce the above result, run the following:
python faces.py part6

Problem 7

Gradient descent is run on a max of 30000 as in the previous parts, but the value of α had to be decreased since in the multi-class cost function, we do not divide by the size of the data as we did in the previous parts.
The performance of the classifier on the training and validation sets is:
The percentage of correct predictions on the training set is: 99.0476190476
The percentage of correct predictions on the validation set is: 81.6666666667
The above results can be reproduced by running the folllowing code:
python faces.py part7
To obtain the predicted label from the output of the model, I found the index in each output that corresponded to the highest value which I then took to be the predicted position of the 1. Below is the code for this:

Problem 8

Below are the visualizations of the θ obtained from by running the code using the same parameters as in part 7 but on 3000 iterations, not 30000, as 30000 did not produce recognizable faces.



Starting from the top left and going right, the visualizations represent Lorraine Bracco, Peri Gilpin, Angie Harmon, Alec Baldwin, Bill Hader, and Steve Carell.
To reproduce the above visualizations, run:
python faces.py part8
Below are the visualizations for the same θ obtained in part 7: