CSC411: Project #1

Due on Friday, February 3, 2017

JINGXI HAO

February 4, 2017

Part 1

Describe the dataset of faces. In particular, provide at least three examples of the images in the dataset, as well as at least three examples of cropped out faces. Comment on the quality of the annotation of the dataset: are the bounding boxes accurate? Can the cropped-out faces be aligned with each other?

Solution:

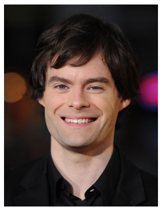
- First, provide three examples of the images in the dataset:

(a) chenoweth9.jpg

(b) chenoweth10.jpg

(c) chenoweth91.jpg

(d) hader33.jpg


(e) hader38.jpg

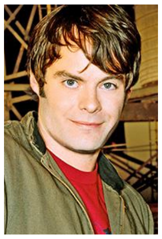
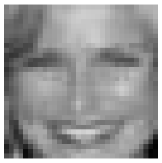
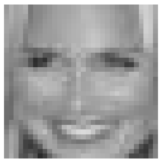

(f) hader114.jpg

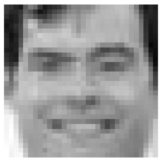
Figure 1: Examples of the Images in the Dataset

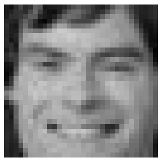
Then, we provide three examples of the cropped out faces:


(a) chenoweth9.jpg


(b) chenoweth10.jpg

(c) chenoweth91.jpg


(d) hader33.jpg


(e) hader38.jpg

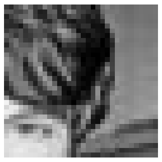

(f) hader114.jpg

Figure 2: Examples of the cropped out faces

- Based on the examples above, we can see that the bounding boxes are not very accurate since some of the faces of the images are not being cut fully (see $Figure 2(f)$) and some of the cropped-out images are not even and might not include the person's face (see $Figure2(c)$).

- According to the examples shown above, we can see that, for the actress Chenoweth, the cropped-out faces on $Figure2(a)$ and $Figure2(b)$ are aligned with each other and $Figure2(c)$ is not aligned with the others since there is no face being cut inside this image. Similarly, for the actor Hader, the cropped-out faces on $Figure2(c)$ and $Figure2(d)$ are aligned with each other and $Figure2(f)$ is not aligned with the others since the face cut in the image is not a full face. Thus, it is obvious that some of the cropped-out faces can be aligned with others for the same person and some of the cropped-out faces can not be aligned with others for the same person.

Part 2

Separate the dataset into three non-overlapping parts: the training set (100 face images per actor), the validation set (10 face images per actor), and the test set (10 face images per actor). For the report, describe how you did that. (Any method is fine). The training set will contain faces whose labels you assume you know. The test set and the validation set will contain faces whose labels you pretend to not know and will attempt to determine using the data in the training set.

Solution:

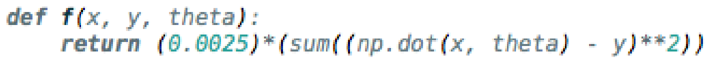
We use the function called *separate_dataset*() in the *faces.py* file to separate the dataset into three non-overlapping parts: the training set (100 face images per actor), the validation set (10 face images per actor), and the test set (10 face images per actor). In order to keep the randomness, for each actor or actress, we first randomly generate an array of 120 integers from the range 0 to 150. Also, for the reproducible purpose, we call *np.random.seed*(1) before every time we call *np.random.choice*. Each of these integers generated matches the number in the filename of each actor or actress (e.g. For actress Chenoweth, if the integer from the array is 25, then we put chenoweth20.jpg in the corresponding set). Then, based on this array of integers, we use first 100 integers to determine the images that we want to put into the training set, the next 10 integers to determine the images that we want to put into the validation set, and the last 10 images to determine the images that we want to put into the test set for each actor and actress. Then, we loop over this process for each actor and actress and store the corresponding images into the corresponding sets.

Part 3

Use Linear Regression in order to build a classifier to distinguish pictures of Bill Hader form pictures of Steve Carell. In your report, specify which cost function you minimized. Report the values of the cost function on the training and the validation sets. Report the performance of the classifier (i.e., the percentage of images that were correctly classified) on the training and the validation sets.
In your report, include the code of the function that you used to compute the output of the classifier (i.e., either Steve Carell or Bill Hader).
In your report, describe what you had to do in order to make the system to work. For example, the system would not work if the parameter α is too large. Describe what happens if is too large, and how you figure out what to set α too. Describe the other choices that you made in order to make the algorithm work.

Solution:

- First, we show the cost function that we minimized below.



```
def f(x, y, theta):
    return (0.0025)*(sum((np.dot(x, theta) - y)**2))
```

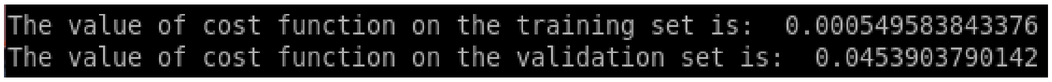
Figure 3: The Cost Function

Therefore, the cost function is

$$J(\theta) = \frac{1}{2m}\sum_{i}^{m}(x^{(i)}\theta - y^{(i)})^2 = \frac{1}{2*200}\sum_{i}^{200}(x^{(i)}\theta - y^{(i)})^2$$

.

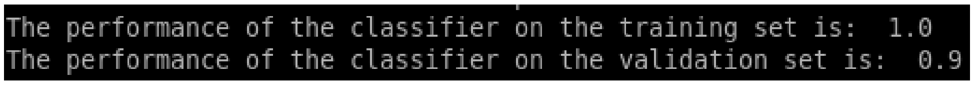
- Then, we report the values of the cost function on the training and the validation set.



```
The value of cost function on the training set is:  0.000549583843376
The value of cost function on the validation set is:  0.0453903790142
```

Figure 4: The Values of the Cost Function on the Training and the Validation Set

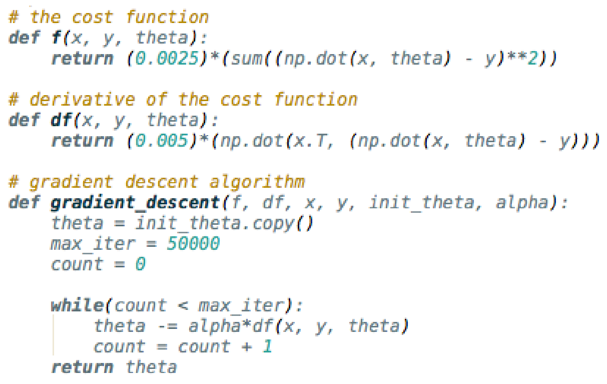
- Then, we report the performance of the classifier on the training set and the validation set.



```
The performance of the classifier on the training set is:  1.0
The performance of the classifier on the validation set is:  0.9
```

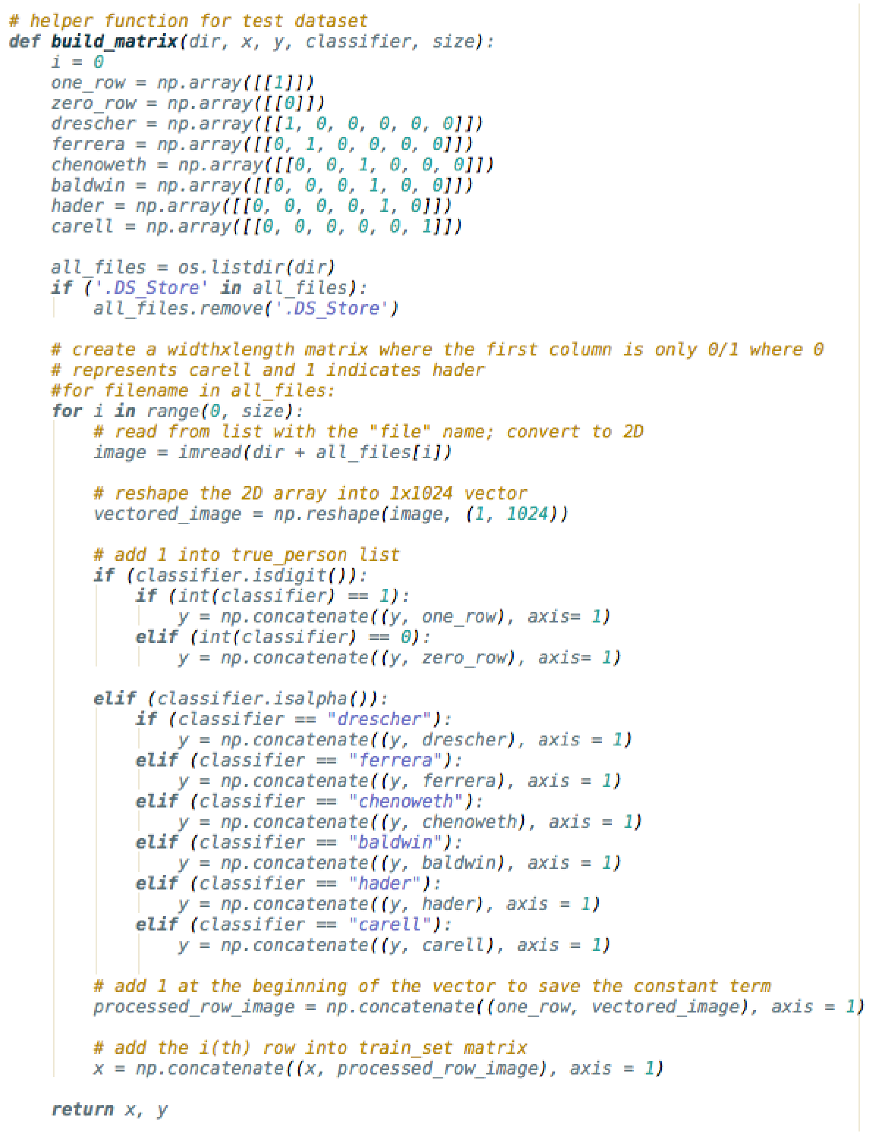
Figure 5: The performance of the Classifiers on the Training and the Validation Set

- Then, we report the code of the function that we used to compute the output of the classifier.



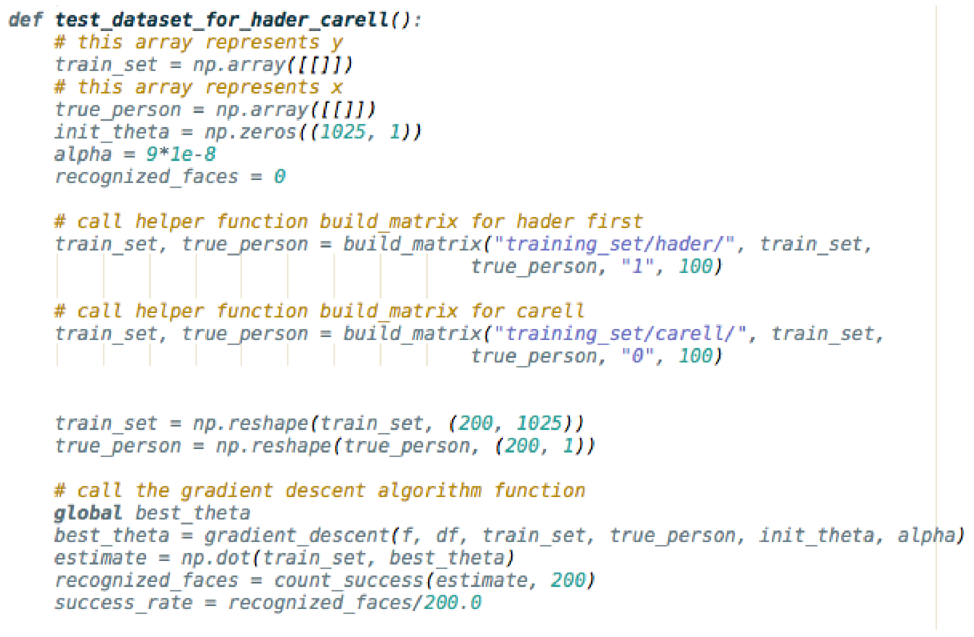
```python
# the cost function
def f(x, y, theta):
    return (0.0025)*(sum((np.dot(x, theta) - y)**2))

# derivative of the cost function
def df(x, y, theta):
    return (0.005)*(np.dot(x.T, (np.dot(x, theta) - y)))

# gradient descent algorithm
def gradient_descent(f, df, x, y, init_theta, alpha):
    theta = init_theta.copy()
    max_iter = 50000
    count = 0

    while(count < max_iter):
        theta -= alpha*df(x, y, theta)
        count = count + 1
    return theta
```

Figure 6: Linear Regression Function Definition and Gradient Descent Algorithm



```python
# helper function for test dataset
def build_matrix(dir, x, y, classifier, size):
    i = 0
    one_row = np.array([[1]])
    zero_row = np.array([[0]])
    drescher = np.array([[1, 0, 0, 0, 0, 0]])
    ferrera = np.array([[0, 1, 0, 0, 0, 0]])
    chenoweth = np.array([[0, 0, 1, 0, 0, 0]])
    baldwin = np.array([[0, 0, 0, 1, 0, 0]])
    hader = np.array([[0, 0, 0, 0, 1, 0]])
    carell = np.array([[0, 0, 0, 0, 0, 1]])

    all_files = os.listdir(dir)
    if ('.DS_Store' in all_files):
        all_files.remove('.DS_Store')

    # create a widthxlength matrix where the first column is only 0/1 where 0
    # represents carell and 1 indicates hader
    #for filename in all_files:
    for i in range(0, size):
        # read from list with the "file" name; convert to 2D
        image = imread(dir + all_files[i])

        # reshape the 2D array into 1x1024 vector
        vectored_image = np.reshape(image, (1, 1024))

        # add 1 into true_person list
        if (classifier.isdigit()):
            if (int(classifier) == 1):
                y = np.concatenate((y, one_row), axis= 1)
            elif (int(classifier) == 0):
                y = np.concatenate((y, zero_row), axis= 1)

        elif (classifier.isalpha()):
            if (classifier == "drescher"):
                y = np.concatenate((y, drescher), axis = 1)
            elif (classifier == "ferrera"):
                y = np.concatenate((y, ferrera), axis = 1)
            elif (classifier == "chenoweth"):
                y = np.concatenate((y, chenoweth), axis = 1)
            elif (classifier == "baldwin"):
                y = np.concatenate((y, baldwin), axis = 1)
            elif (classifier == "hader"):
                y = np.concatenate((y, hader), axis = 1)
            elif (classifier == "carell"):
                y = np.concatenate((y, carell), axis = 1)

        # add 1 at the beginning of the vector to save the constant term
        processed_row_image = np.concatenate((one_row, vectored_image), axis = 1)

        # add the i(th) row into train_set matrix
        x = np.concatenate((x, processed_row_image), axis = 1)

    return x, y
```

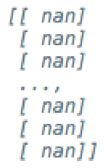
Figure 7: Helper function that Helps to Build X and Y



```
def test_dataset_for_hader_carell():
    # this array represents y
    train_set = np.array([[]])
    # this array represents x
    true_person = np.array([[]])
    init_theta = np.zeros((1025, 1))
    alpha = 9*1e-8
    recognized_faces = 0

    # call helper function build_matrix for hader first
    train_set, true_person = build_matrix("training_set/hader/", train_set,
                                          true_person, "1", 100)

    # call helper function build_matrix for carell
    train_set, true_person = build_matrix("training_set/carell/", train_set,
                                          true_person, "0", 100)


    train_set = np.reshape(train_set, (200, 1025))
    true_person = np.reshape(true_person, (200, 1))

    # call the gradient descent algorithm function
    global best_theta
    best_theta = gradient_descent(f, df, train_set, true_person, init_theta, alpha)
    estimate = np.dot(train_set, best_theta)
    recognized_faces = count_success(estimate, 200)
    success_rate = recognized_faces/200.0
```

Figure 8: Function that Computes Output of the Classifier

- Lastly, we describe what we had to do in order to make the system to work. First, we describe how we choose α. The system is not able to work with the large α. When α is large, the system produces the poor performance for both sets and the value of θ can not be computed.



```
[[ nan]
 [ nan]
 [ nan]
 ...,
 [ nan]
 [ nan]
 [ nan]]
```

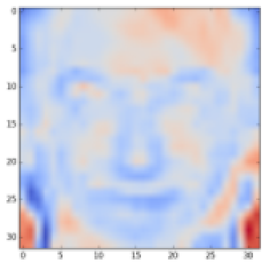
Figure 9: Theta obtained by Using Large Alpha

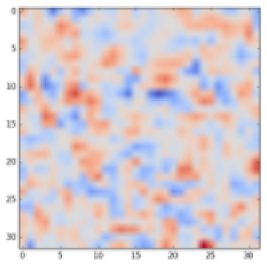
Also, too small α offers the poor accuracy for both sets as well. Then, we figure out the value of α by trying distinct values of α and choose the value that produces the higher performance. In our case, we pick α equals to $9 * 1e - 8$. In addition, we need to choose the value of iterations. By trying different values for iteration, we are able to conclude that more iterations higher accuracy the system produces. But, too large iterations may produce overfitting which reduces the performance for both sets as well. Thus, we choose the value of iteration by running the system with different values and set the value of iteration to the one that produces the higher performance for both sets. In our case, we choose the value of iteration equals to 50000. Moreover, we also have to pick the value for the initial θ which is the weight. This time we also pick the value that produces the higher performance after trying different values of initial θ and then we let each component of initial θ be zero.

Part 4

In Part 3, you used the hypothesis function $h_\theta(x) = \theta_0 + \theta_1 x_1 + ... + \theta_n x_n h_\theta(x) = \theta_0 + \theta_1 x_1 + ... + \theta_n x_n$. If $(x_1, ..., x_n)(x_1, ..., x_n)$ represents a flattened image, then $(\theta_1, ..., \theta_n)(\theta_1, ..., \theta_n)$ can also be viewed as an image. Display the $\theta's$ that you obtain by training using the full training dataset, and by training using a training set that contains only two images of each actor.

The images could look as follows.





Solution:

First, we display the $\theta's$ obtained by training using the full training dataset.

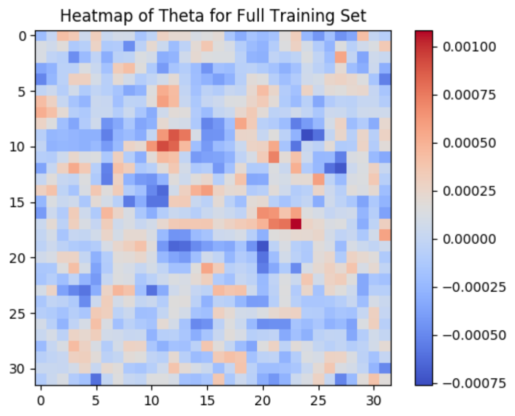


Figure 10: Theta obtained by Using Full Training Dataset

Then, we display the $\theta's$ obtained by training using a training set that contains only two images of each actor.

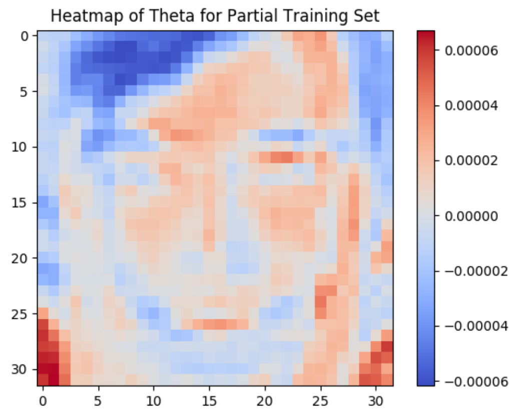


Figure 11: Theta obtained by Using Two Images of Each Actor

As we can see that, these two heatmaps have big difference. The image of the heatmap generated by using the full dataset is noisy since due to the diverse quality of the cropped-out images of the faces for the actor, the more data used the more diverse the characteristics are chosen and determined. This causes each feature to not be able to conform, hence makes it hard to display all features together. Thus, the heatmap of the θ produced by using the full training dataset is noisy. Also, if we use a smaller size of the dataset, the less features of the actor is produced and easy to conform. Therefore, the image of the heatmap of the θ obtained by using two images of each actor looks more like a face.

Part 5

In this part, you will demonstrate overfitting. Build classifiers that classify the actors as male or female using the training set with the actors from

```
act =['Fran Drescher', 'America Ferrera', 'Kristin Chenoweth', 'Alec Baldwin', 'Bill Hader', 'Steve Carell']
```

and using training sets of various sizes. Plot the performance of the classifiers on the training and validation sets vs the size of the training set.

Report the performance of the classifier on actors who are not included in *act* :

```
act_test = ['Gerard Butler', 'Daniel Radcliffe', 'Michael Vartan', 'Lorraine Bracco', 'Peri Gilpin', 'Angie Harmon']
```

Solution:

- First, we show that plot graph of the performance of the classifiers on the training set vs the size of the training set.

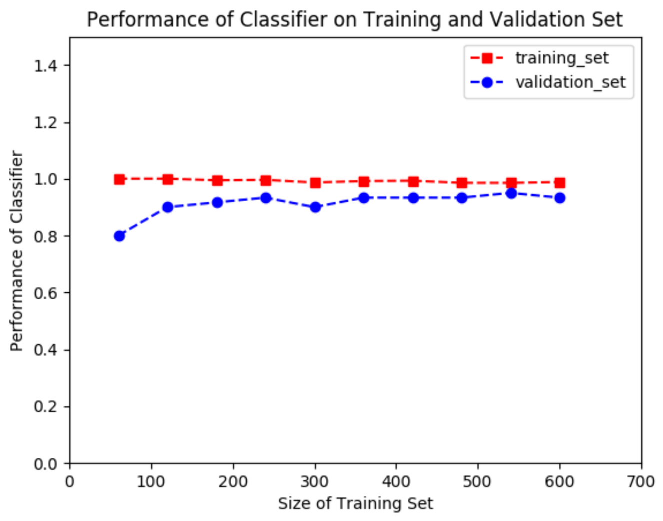
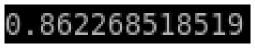


Figure 12: The Performance of the Classifiers on the Training and Validation Set vs the Size of the Training Set

We try 10 different sizes of the training set and generated 10 different values of θ. Also, we use each different θ to compute the corresponding performance of the classifiers and then plot them. Based on the graph shown above, we can see that the performance of the classifier on the validation set gradually increases at first and then gradually decreases after the size of the training set equals to 540 and the performance of the classifier on the training set is always very close to 1. Therefore, this demonstrates the overfitting.

- Then, we report the performance of the classifier on actors who are not included in *act*. Then, the performance is

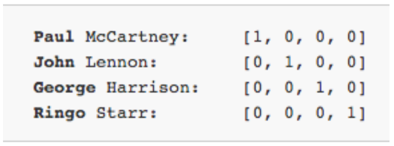


```
0.862268518519
```

Figure 13: The Performance of the Classifiers on Actors who are not Included in *act*

Part 6

Now, consider a different way of classifying inputs. Instead of assigning the output value $y = 1$ to images of Paul McCartney and the output value $y = 1$ to images of John Lennon, which would not generalize to more than 2 labels, we could assign output values as follows:



Paul McCartney:	[1, 0, 0, 0]
John Lennon:	[0, 1, 0, 0]
George Harrison:	[0, 0, 1, 0]
Ringo Starr:	[0, 0, 0, 1]

The output could still be computed using $\theta^T x$, but θ would now have to be a $n \times k$ matrix, where k is the number of possible labels, with x being a $n \times 1$ vector.

The cost function would still be the sum of squared differences between the expected outputs and the actual outputs:

$$J(\theta) = \sum_i \left(\sum_j (\theta^T x^{(i)} - y^{(i)})_j^2 \right).$$

(a) Compute $\partial J / \partial \theta_{pq}$. Show your work. Images of neatly hand-written derivations are acceptable, though you are encouraged to use LaTeX.

Solution:

We show the detailed work for computing $\partial J / \partial \theta_{pq}$.

$$\begin{aligned}
\frac{\partial J(\theta)}{\partial \theta_{pq}} &= \frac{\partial \sum_i (\sum_j (\theta^T x^{(i)} - y^{(i)})_j)^2}{\partial \theta_{pq}} \\
&= \frac{\partial \sum_i ((\theta^T x^{(i)} - y^{(i)})_0 + (\theta^T x^{(i)} - y^{(i)})_1 + ... + \theta^T x^{(i)} - y^{(i)})_n)^2}{\partial \theta_{pq}} \quad \text{\# where } j \text{ is in range from 0 to } n \\
&= \frac{\partial \sum_i (\theta^T x^{(i)} - y^{(i)})_q{}^2}{\partial \theta_{pq}} \\
&= \sum_i \frac{\partial (\theta^T x^{(i)} - y^{(i)})_q{}^2}{\partial \theta_{pq}} \\
&= 2 \sum_i \frac{\partial (\theta^T x^{(i)} - y^{(i)})_q}{\partial \theta_{pq}} (\theta^T x^{(i)} - y^{(i)})_q \\
&= 2 \sum_i x_p{}^{(i)} (\theta^T x^{(i)} - y^{(i)})_q
\end{aligned}$$

(b) Show, by referring to *Part* 6(a), that the derivative of $J(\theta)$ with respect to all the components of θ can be written in matrix form as

$$2\mathbf{X}(\theta^T \mathbf{X} - \mathbf{Y})^T$$

.

Specify the dimensions of each matrix that you are using, and define each variable (e.g., we defined m as the number of training examples.) **X** is a matrix that contains all the input training data (and additional 1s), of the appropriate dimensions.

Solution:

First, we specify the dimensions of each matrix that we are using, and define each variable. Then, **X** is an input matrix and its dimensions are $n \times m$ and **Y** is an actual output and its dimensions are $k \times m$. Also, θ is a weight matrix and its dimensions are $n \times k$, therefore, the dimensions of θ^T are $k \times n$.

Then, we show that the derivative of $J(\theta)$ with respect to all the components of θ can be written in matrix form as $2\mathbf{X}(\theta^T\mathbf{X} - \mathbf{Y})^T$. In order to prove this, we need to show that the each value in the matrix $\partial J(\theta)/\partial\theta_{pq}$ agrees with the value in the matrix $2\mathbf{X}(\theta^T\mathbf{X}-\mathbf{Y})^T$ at the same position. Therefore, we have that,

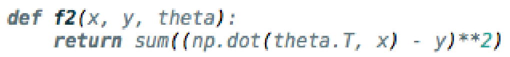
$$\begin{aligned}(2\mathbf{X}(\theta^T\mathbf{X} - \mathbf{Y})^T)_{pq} &= 2\textstyle\sum_i x_{pi}(\theta^T x - y)_{iq}{}^T \qquad \text{\# where } i \text{ is in range from 0 to } m\\ &= 2\textstyle\sum_i x_p{}^{(i)}(\theta^T x - y)_{iq}{}^T\\ &= 2\textstyle\sum_i x_p{}^{(i)}(\theta^T x - y)_{qi}\\ &= 2\textstyle\sum_i x_p{}^{(i)}(\theta^T x^{(i)} - y^{(i)})_q\\ &= \partial J(\theta)/\partial\theta_{pq} \qquad \text{\# By } \textit{Part } 6\ (a)\end{aligned}$$

Thus, we have proven that the derivative of $J(\theta)$ with respect to all the components of θ can be written in matrix form as $2\mathbf{X}(\theta^T\mathbf{X} - \mathbf{Y})^T$.

(c) Implement the cost function from Part 6 and the vectorized gradient function in Python. Include the code in your report.

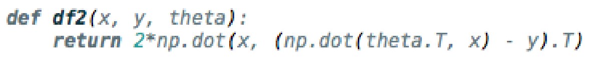
Solution:

We show the code that implements the cost function and the vectorized gradient.



```
def f2(x, y, theta):
    return sum((np.dot(theta.T, x) - y)**2)
```

Figure 14: The cost function



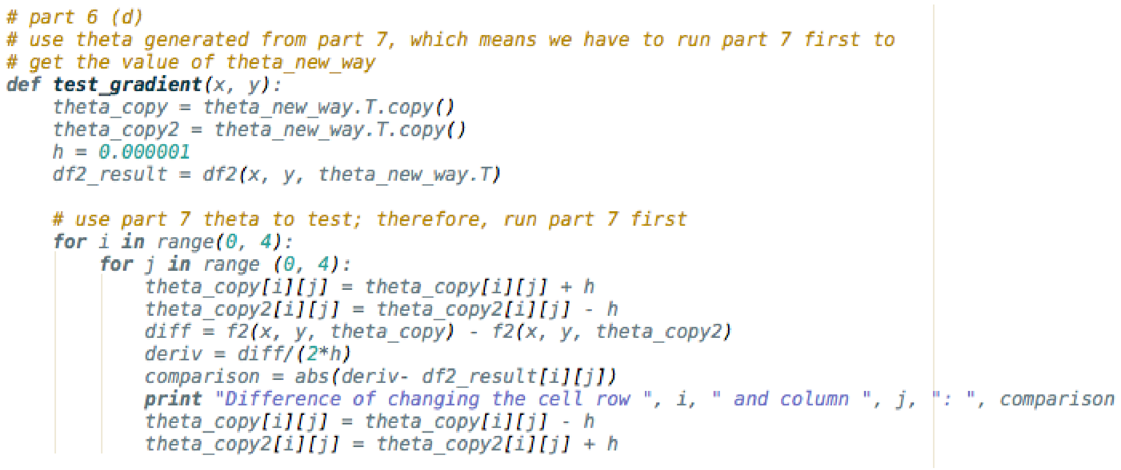
```
def df2(x, y, theta):
    return 2*np.dot(x, (np.dot(theta.T, x) - y).T)
```

Figure 15: The Vectorized Gradient

(d) Demonstrate that the vectorized gradient function works by computing several components of the gradient using finite differences. In your report, include the code that you used to compute the gradient components using finite differences, and to compare them to the gradient that you computed using your function.

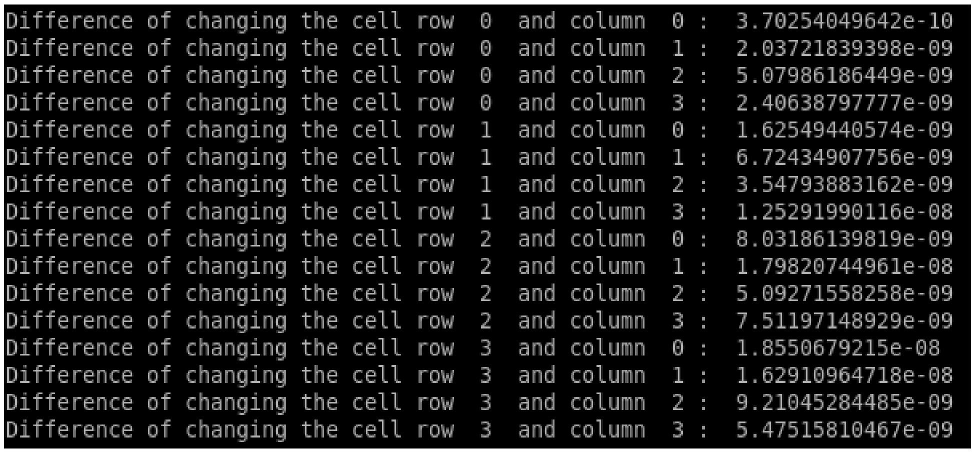
Solution:

First, we show the code that we used to compute the gradient components using finite differences and to compare them to the gradient that we computed using *test_dataset_new_way*() function in *Part* 7.



```
# part 6 (d)
# use theta generated from part 7, which means we have to run part 7 first to
# get the value of theta_new_way
def test_gradient(x, y):
    theta_copy = theta_new_way.T.copy()
    theta_copy2 = theta_new_way.T.copy()
    h = 0.000001
    df2_result = df2(x, y, theta_new_way.T)

    # use part 7 theta to test; therefore, run part 7 first
    for i in range(0, 4):
        for j in range (0, 4):
            theta_copy[i][j] = theta_copy[i][j] + h
            theta_copy2[i][j] = theta_copy2[i][j] - h
            diff = f2(x, y, theta_copy) - f2(x, y, theta_copy2)
            deriv = diff/(2*h)
            comparison = abs(deriv- df2_result[i][j])
            print "Difference of changing the cell row ", i, " and column ", j, ": ", comparison
            theta_copy[i][j] = theta_copy[i][j] - h
            theta_copy2[i][j] = theta_copy2[i][j] + h
```

Figure 16: The Code to Compute and Compare



```
Difference of changing the cell row  0  and column  0 :  3.70254049642e-10
Difference of changing the cell row  0  and column  1 :  2.03721839398e-09
Difference of changing the cell row  0  and column  2 :  5.07986186449e-09
Difference of changing the cell row  0  and column  3 :  2.40638797777e-09
Difference of changing the cell row  1  and column  0 :  1.62549440574e-09
Difference of changing the cell row  1  and column  1 :  6.72434907756e-09
Difference of changing the cell row  1  and column  2 :  3.54793883162e-09
Difference of changing the cell row  1  and column  3 :  1.25291990116e-08
Difference of changing the cell row  2  and column  0 :  8.03186139819e-09
Difference of changing the cell row  2  and column  1 :  1.79820744961e-08
Difference of changing the cell row  2  and column  2 :  5.09271558258e-09
Difference of changing the cell row  2  and column  3 :  7.51197148929e-09
Difference of changing the cell row  3  and column  0 :  1.8550679215e-08
Difference of changing the cell row  3  and column  1 :  1.62910964718e-08
Difference of changing the cell row  3  and column  2 :  9.21045284485e-09
Difference of changing the cell row  3  and column  3 :  5.47515810467e-09
```

Figure 17: The Results From the Code Above

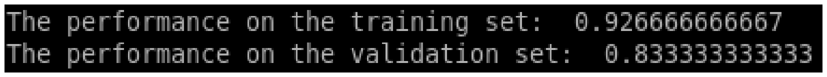
We compute 16 components of the gradient using finite differences and compare them to the gradient that we computed from *Part* 7 by calculating the absolute value of the difference between the corresponding components from each matrix. Based on the results shown above, we can see that the absolute value of difference is really small. Thus, the vectorized gradient function works by computing several components of the gradient using finite differences.

Part 7

Run gradient descent on the set of six actors *act* in order to perform face recognition. Report the performance you obtained on the training and validation sets. Indicate what parameters you chose for gradient descent and why they seem to make sense.

Solution:

- First, we report the performance obtained on the training and the validation sets.



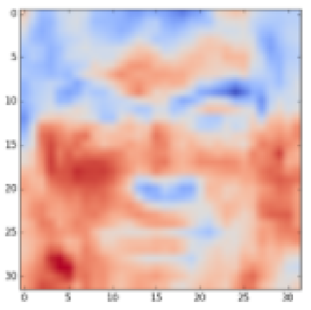
```
The performance on the training set:  0.926666666667
The performance on the validation set:  0.833333333333
```

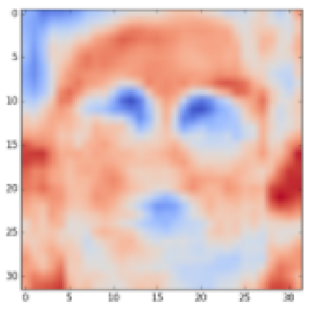
Figure 18: The Performance on the Training and Validation Sets

- Then, we indicate what parameters chosen for gradient descent and explain why they seen to make sense. In this case, we let α be $4 * 1e - 11$ and iterations be 4000. Also, assign zero to every component in the initial θ which is the weight. We choose these values by the same picking approach, which is to try out different values and pick the one offers the relatively higher performances and produces the heatmap (generated in *Part* 8) that resembles a face.

Part 8

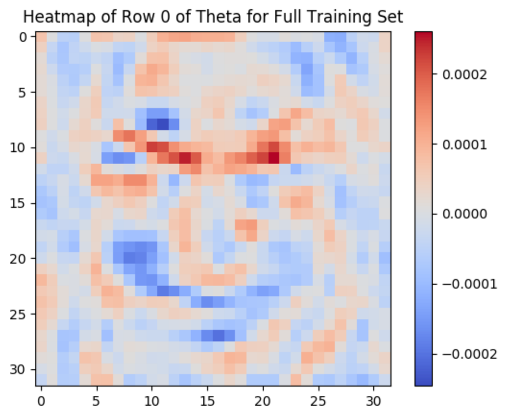
Visualize the θs that you obtained. Note that if θ is a $k \times n$ matrix, where k is the number of possible labels and $n-1$ is the number of pixels in each image, the rows of could be visualized as images. Your outputs could look something like the ones below. Label the images with the appropriate actor names.

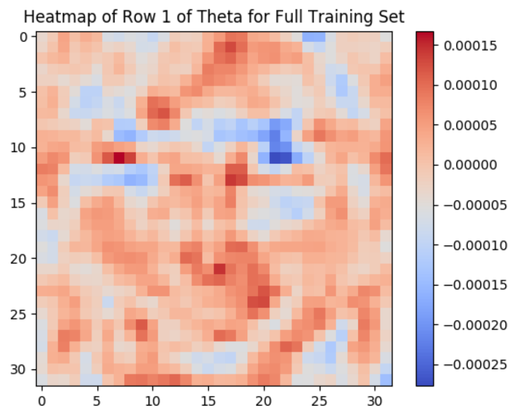


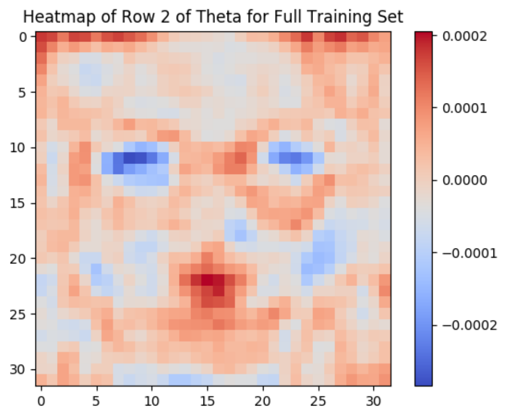


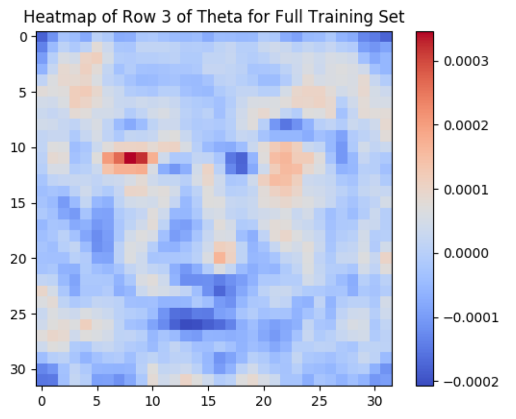
Solution:

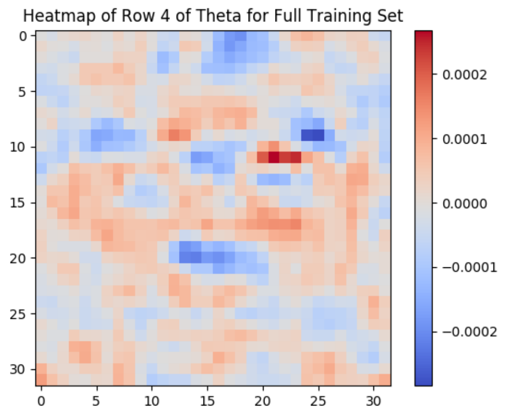
We visualize the θ obtained.

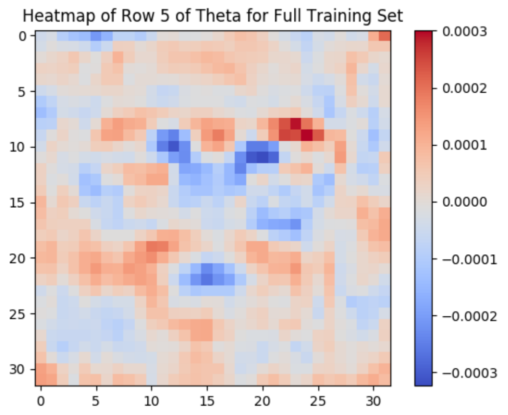









Heatmap of Row 4 of Theta for Full Training Set
0.0002
0.0001
0.0000
−0.0001
−0.0002


Heatmap of Row 5 of Theta for Full Training Set
0.0003
0.0002
0.0001
0.0000
−0.0001
−0.0002
−0.0003