CSC411 Project 1

Zili Xie

Jan 29th 2018

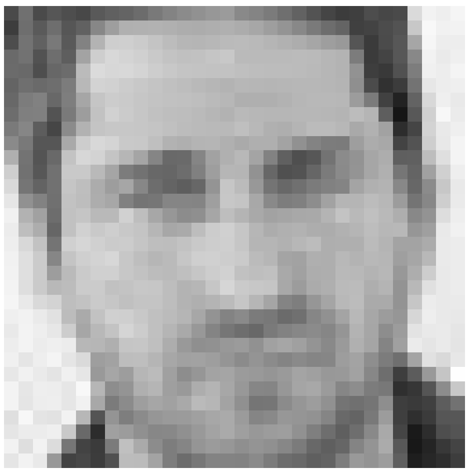
Part 1

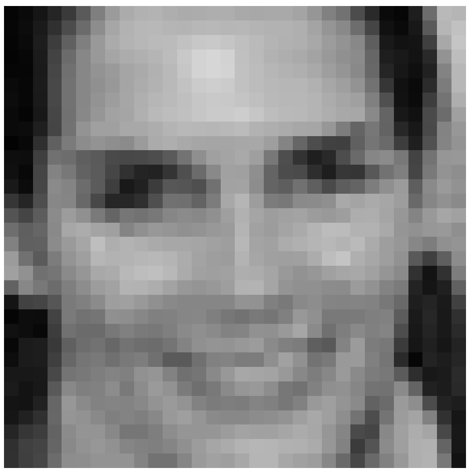
The data set we used for this assignment is a collection of images of 6 actors and 6 actresses. The file provided to us contains 3105 links to images of actors. Each link is along with a name of actor or actress and coordinates that crop the their face out. Most of the link in the given data file is accessible, while there still exists some broken links and therefore some images can not be downloaded. The quality of images are varies significantly, because some images are photo shoot in award ceremony but some images are just screenshots from videos or movies, so this kind of images might have a lower qualities. The qualities of images will affect cropped out resolutions of face image, and product noises in regression analysis.
Three example images from the data set are listed below, along with their corresponding cropped out face. Apparently, the faces that were cropped from images are generally correct.

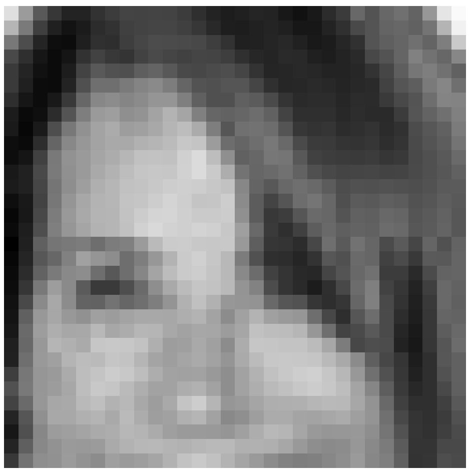
butler0

harmon112

bracco63


butler0


harmon112


bracco63

However, there are some cases, where the four coordinates of bounding boxes is not accurate. For example, bracco's cropped face image from the above listing lost her chin which means that the bounding box should actually move down a little bit.

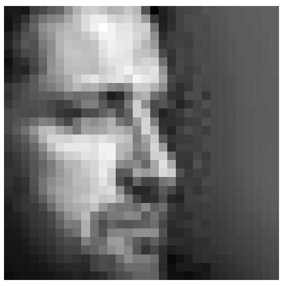
The face matrix below has 25 cropped out faces from 5 actors randomly choose from data set, 5 face images per actor. Again we see that not all the bounding boxes crop the faces accurately, there are some boxes only bound part of the actor's face. Moreover, due to the camera angle of some image, we can only see the side of face in some images. Some actors were not facing towards the camera when they are being photographed. Some of them accidentally put something over their face such as hands, hairs. All this can be factors that affecting the correctness of detecting face. The faces are not completely aligned but in most cases they can be aligned with each other and can be used as data source for linear regression.

butler48

butler49

butler109

butler114


butler129

harmon43

harmon57

harmon59

harmon66

harmon127

bracco33

bracco58

bracco87

bracco105

bracco108

radcliffe2

radcliffe9

radcliffe11

radcliffe12

radcliffe27

drescher0

drescher49

drescher63

drescher71

drescher109

Part 2

Our objective in this part is to separate all the actors' cropped faces into 3 non-overlaping sets: training set, validation set, test set, which respectively contains 100 images, 10 images and 10 images. I used the get_data script provided on course website to obtain images i needed for this project(note: i got fewer than 100 images for Gilpin so i manually downloaded some images that were missed by the download script) To separate the images, I first group cropped images actor by actor, faces from different person will be separate into different folders. Then my program loops through the actor list. In each iteration, read all the file name under a specific actor's folder into a list and shuffle the order. I call this list act_images. My script also keeps 3 dictionary data structure: train_set, valid_set, test_set. Each of these 3 sets uses actor name as key, a list of images in the set as value corresponding to the key. For each actor, assign file names in his/her act_images list to 3 different dictionaries. 10 image names go to test set, another 10 go to valid set. and select 100 image file names from the rest of the list, assign them to the train set. At last, base on what we have in these 3 dictionaries, respectively copy all the files to 3 separate folders: train, valid, test. Easy to use next time.

Part 3

In this part, linear regression is used to build a classifier to distinguish Alec Baldwin and Steve Carell. First we need to read each image from training set into a 32×32 grayscale matrix using imread() function. Then we need to reshape the resolution of image from 32 by 32 to a 1×1024 vector. For this part, I use the reshape() function imported from numpy library. The hypothesis function we use is:

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + ... + \theta_n x_n$$

There is a constant term θ_0 at the beginning of right hand side equation. we need to concatenate a 1 at the very first of each vector. After that, we stack all the vectors together. Since we have 200 images for these 2 actor in training set, a 200×1025 matrix is created whose first row to 100 row are images of Steve Carell, 101 to 200 row hold images of Alec Baldwin. I call this matrix x. Different actor has different label. Steve Carell's images are label as one and images of Alec Baldwin are all labelled as zero, so we can create a vector y which contains all the expected labels. y is a 200×1 matrix having 100 consecutive ones and 100 consecutive zeros. having x and y, we can rewrite the hypothesis function in vector form:

$$h_\theta(x) = \theta x$$

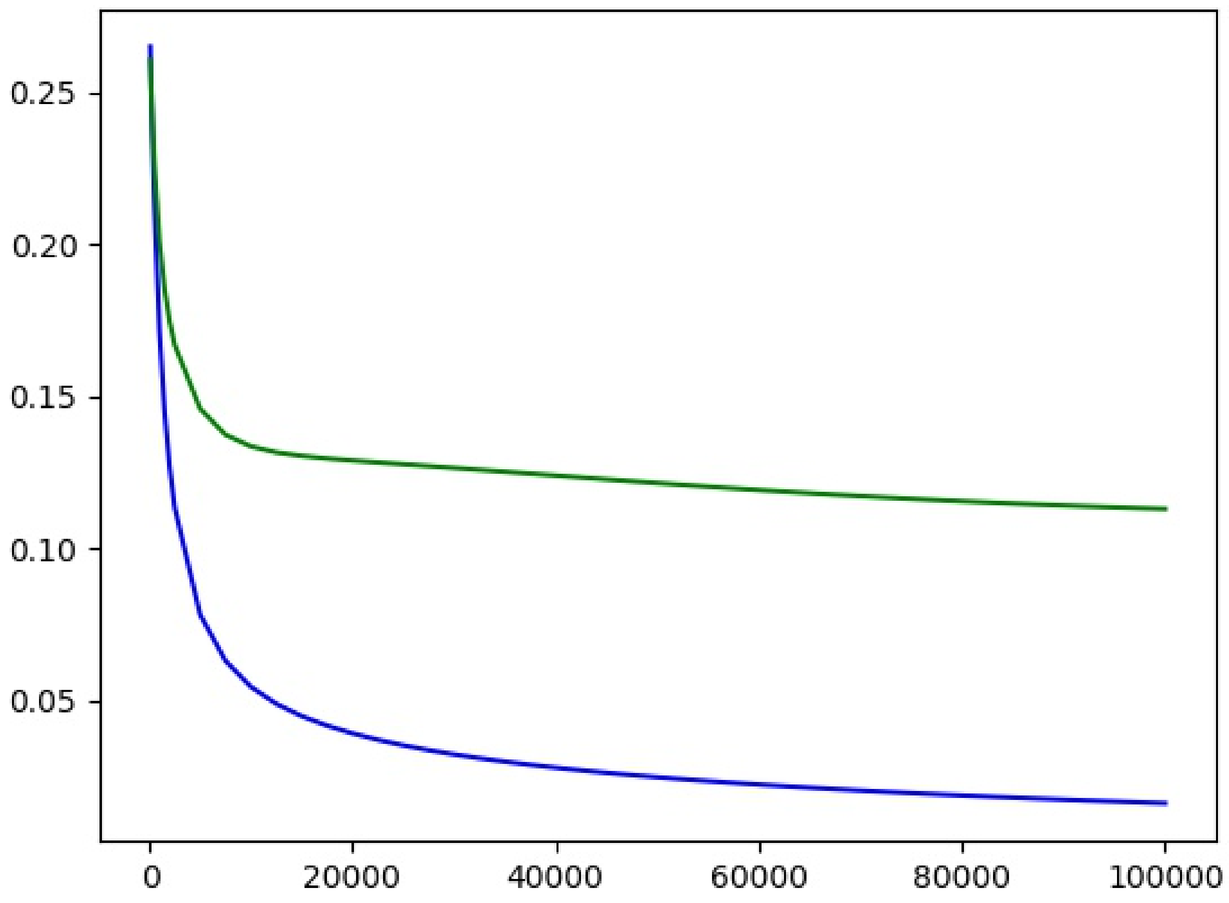
and the cost function that we want to minimize can be defined as follow:

$$f(x, y, \theta) = \frac{1}{2m} \sum_{n=1}^{m} (h_\theta(x) - y)^2$$

We can use gradient descent to minimize the cost. Below are some codes that defines the cost function and gradient descent. The alpha chosen for gradient really matter because if the alpha is too small, the step we move in each iteration would be small as well. More iterations are needed for reaching the minimum point. On the other hand, if the alpha is too large, which means in each iteration, the algorithm will subtract theta by a larger number, Then the good thing is that gradient descent algorithm can approach the minimum in a faster speed. However, there might be a case when gradient descent jump across the global minimum and goes to the other side of cost function, causing a higher cost. This would lead to a divergence of global minimum.

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

def df(x, y, theta):
    return -2 * (dot(x.T , (y - dot(x, theta))))

def grad_descent(x, y, init_t , df , iter_time , a):
    alpha = a
    t = init_t.copy()
    max_iteration = iter_time
    iter  = 0
    while iter < max_iteration:
        t -= alpha*df(x, y, t)
        iter += 1
    return t
```

In my gradient descent, α is set to be 9e-6. The initial theta is set equals to a 1025×1 vector of all zeros, which will be another input for gradient descent. After 100000 iterations time, the value returned by grad_descent() will be the θ we want. The cost function value on two sets are computed as training set: 0.0162878, validate set: 0.1131115



Green: Valid Set, Blue: Train Set
X: iteration times, Y: cost

Shown in picture part3 plot above, as the iterations increase, both of the costs for training set and validate set go down to a level and keep stable. The cost associated with the training set is minimized more than validation set, which indicates that gradient descent on training set couldn't minimize the cost of validation set as much as possible.

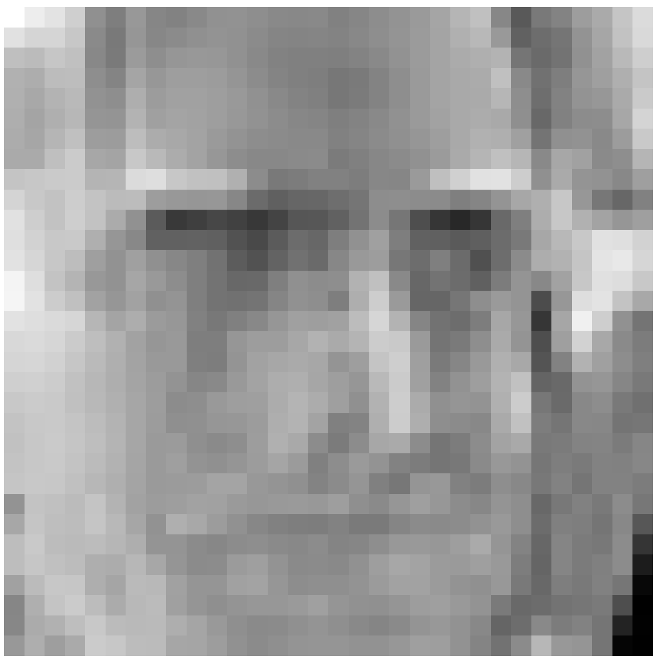
This is a limitation of linear regression when using this cost function.
As for the performance of classifier on each set, similar method was used previously in the set up phase to obtain x_valid and x_test. They are both 20×1025 matrices. And since size of validation and test sets are both 20, first I compute the dot product between validate x matrix (also test x matrix) and theta to compute a vector representing the result. Then for each cell in results if the cell has value greater than 0.5, then treat it as one. Other cases, treat it as zero, and after that, compare it will the same cell in y. Count the occurrence of same 1s and 0s.
Performance on each sets:
training set accuracy: 0.99, validate set accuracy: 0.95, test set accuracy: 0.90
And below are some segment of code I use to generate classifer:

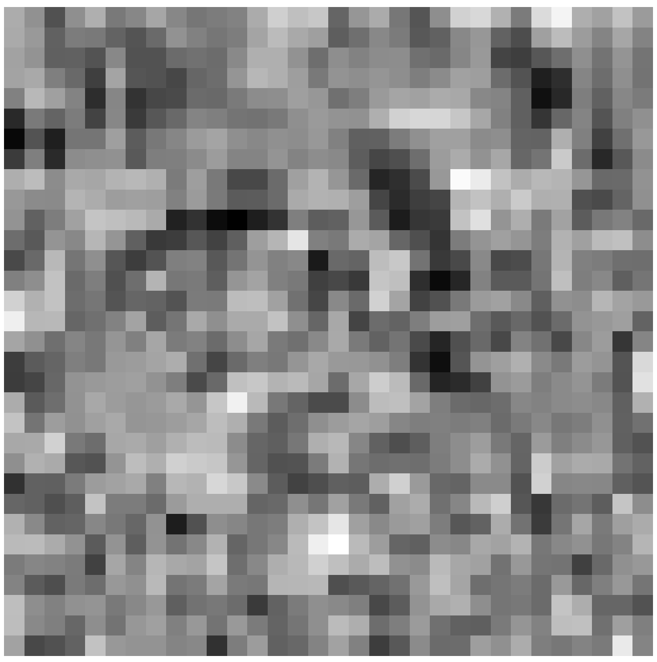
```
def Part3_1():
    x = np.array([[]])
    y = np.array([[]])
    for i in range(0, 200):
        if i < 100:
            img = imread("train/"+train_set['carell'][i]
            flat_img = np.reshape(img,(1, 1024))
            flat_img = np.concatenate((one, flat_img/255.0),1)
            x = np.concatenate((x, flat_img), 1)
        else:
            img = imread("train/"+train_set['baldwin'][i-100])
            flat_img = np.reshape(img,(1, 1024))
            flat_img = np.concatenate((one, flat_img/255.0),1)
            x = np.concatenate((x, flat_img), 1)
    for i in range(0, 200):
        if i < 100:
            y = np.concatenate((y, one), 1)
        else:
            y = np.concatenate((y, zero), 1)
    x = np.reshape(x, (200, 1025))
    y = np.reshape(y, (200, 1))
    return x, y

x, x_valid, x_test, y, y_valid = Part3_1()

def Part3_2(x, x_valid, x_test, y):
    init_t = np.zeros((1025,1))
    theta = grad_descent(x, y, init_t, df, 100000, 9e-6)
    square_theta = np.reshape(theta[1:], (32,32))
    imsave("100img_theta.jpg", square_theta)
    count = get_accuracy(x, theta, 100, "train␣set")
    count = get_accuracy(x_valid, theta, 10, "validate␣set")
    count = get_accuracy(x_test, theta, 10, "test␣set")
Part3_2(x, x_valid, x_test, y)
```

Part 4

(a)

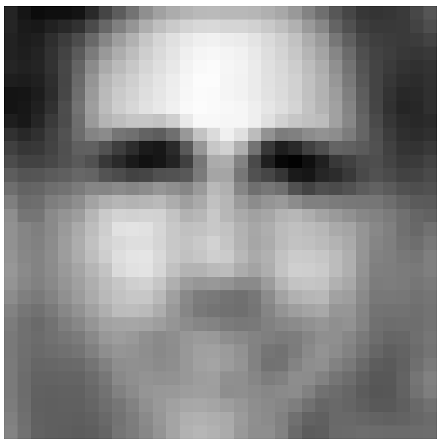
Using the hypothesis function in part 3, the theta images are represented below. The image generated by gradient descent using the complete training set looks more disorderly and unsystematic. while the theta image for training set with only 2 images per actor is more like a human face.


Theta from Training set of size 2


Theta from Training set of size 100

(b)

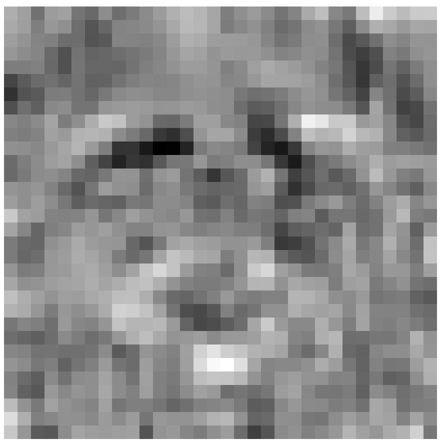
Use the training set as input, stop the gradient descent in early iterations. In my program, I stopped the process in the first, 10th, 100th, 1000th, 10000th, 100000th and 1000000th iterations and generate the corresponding theta image of that stage.

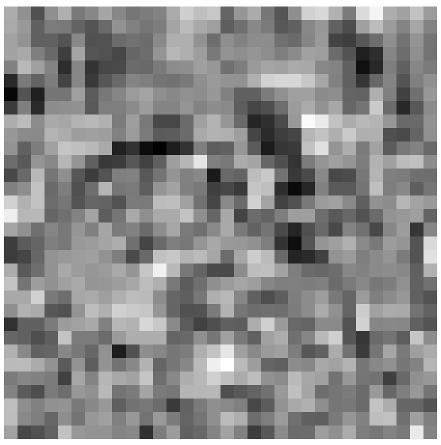

1st iteration

10th iteration

100th iteration

10000th iteration

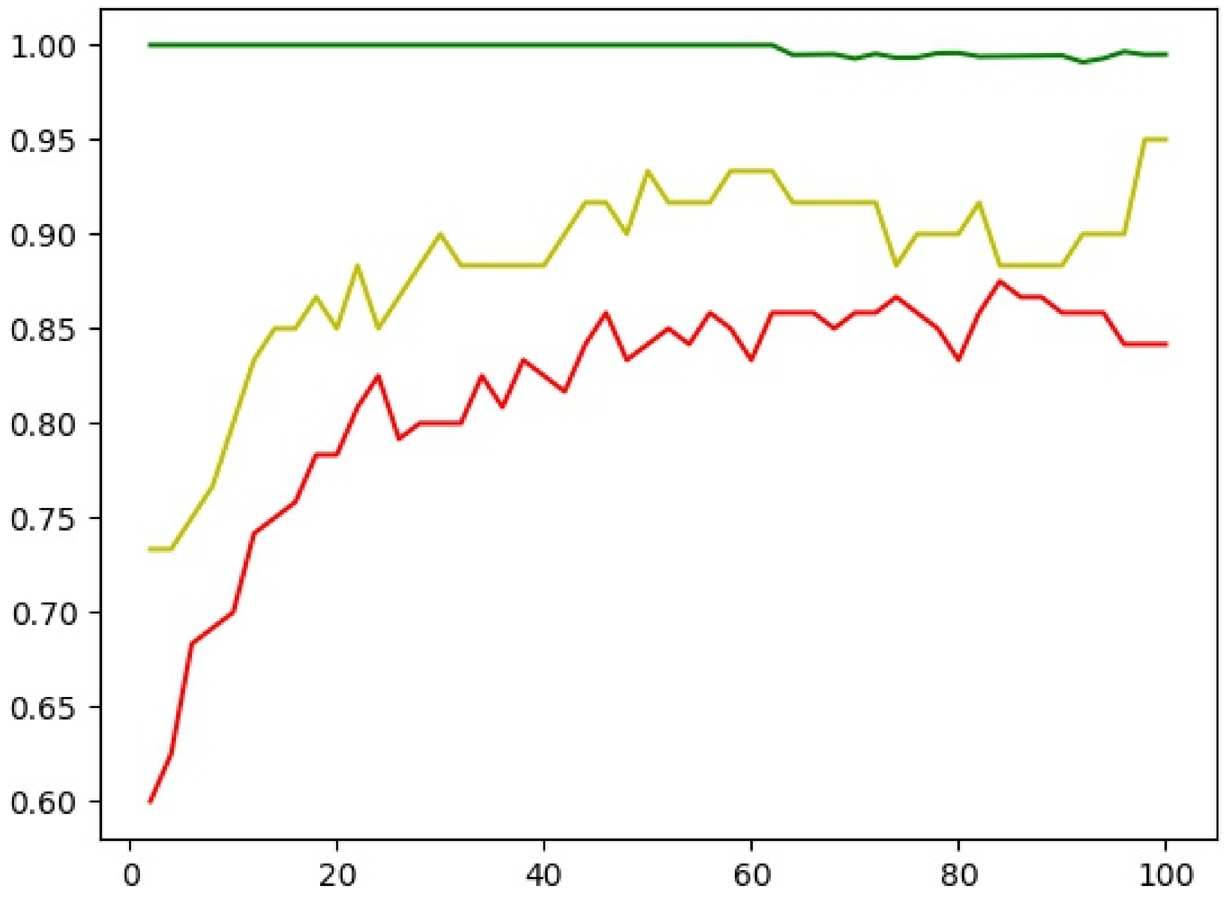

100000th iteration


1000000th iteration

The image for the 1st iteration obviously looks more like a human face, while the theta after all iterations is unsystematic just like the previous part.

Part 5

Below the image demonstrate the gender classification performance on training set, validate set, test set, against the number of images used in training set to generate theta. For this part, I used the same way to handle the input images data by reshaping them into flat images and concatenate them as a matrix. Meanwhile, I labelled actors as one and actresses as zero and put all the expected results in a vector. Finally, the test set was acquired by randomly selecting 20 images from the actors who are not included training set. The plot shows that training set always has better performance than validate set and test set, because the classifier is generated by minimizing the cost on training set, so it make sense that train set has the best performance. We can also see a growth trend for both validate set and test set. The accuracy on validate set and test set grown and reach a plateau at respectively 0.95 and 0.85 (so the performance on actors not in act is 0.85)



Green: Train Set, Yellow: Valid Set, Red: Test Set
X: Train Image Size per Actor, Y: cost

In the plot above, we can see some fluctuations on yellow line and red line, but in overall, they have upward trends and they are both in a relatively low level at the beginning stage when the training size is small. The low performance is actually a result of overfitting, when there are just few images in training set, the theta we obtain by linear regression perfectly fit the images in train set, but can not successfully predict the gender of actors/actresses from other data.

Part 6

(a)

The derivative of cost function can be represented as:

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

The θ x, y can be represented as:

$$x^{(i)} = \begin{bmatrix} x_1^{(i)} & x_2^{(i)} & x_3^{(i)} & \dots & x_{1025}^{(i)} \end{bmatrix}; y^{(i)} = \begin{bmatrix} y_1 & y_2 & y_3 & \dots & y_k \end{bmatrix}$$

$$\theta = \begin{bmatrix} \theta_{1,1} & \theta_{1,2} & \theta_{1,3} & \dots & \theta_{1,k} \\ \theta_{2,1} & \theta_{2,2} & \theta_{2,3} & \dots & \theta_{2,k} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \theta_{1025,1} & \theta_{1025,2} & \theta_{1025,3} & \dots & \theta_{1025,k} \end{bmatrix}$$

so we have,

$$\theta^T(x)^{(i)} - y^{(i)} = \begin{bmatrix} e_1 & e_2 & e_3 & \dots & e_k \end{bmatrix}; e_k = \sum_{n=1}(\theta_{pk}x_p^{(i)}) - y_k$$

thus,

$$\sum_j(\theta^T(x)^i - y^i)_j^2 = (\theta^T(x)^i - y^i)^T(\theta^T(x)^i - y^i)$$

and

$$\frac{dJ(\theta)}{d\theta_{pq}} = 2\sum_{n=1}(\theta^T(x)^i - y^i)_q \cdot \frac{d(\theta^T(x)^i - y^i)_q}{d\theta_{pq}}$$

$$\frac{dJ(\theta)}{d\theta_{pq}} = 2\sum_{n=1}(x_p)^i(\theta^T(x)^i - y^i)_q$$

(b)

First we can define $X = a \times b$, $\theta = a \times k$, $\theta^T = k \times a$. Thus, $M = a \times k$ Then we can let

$$M = 2X(\theta^T X - Y)^T$$

$$M_{pq} = 2\sum_b(X_{pb})((\theta^T X - Y)^T)_{aq}$$

$$N = (\theta^T X - Y)^T$$

and then we have

$$M_{pq} = 2\sum_b(X_{pb})(N)_{aq}$$

$O = \theta^T X - Y$ is a k by b matrix,

$$(\theta^T X - Y)_{qb} = ((\theta^T X - Y)^T)_{bq}$$

So $O_{qb} = (N^T)_{bq}$ and we also have $B_{qb} = (\theta^T x^b - y^b)^T)_q$

$$M_{pq} = 2\sum_b x_p^b \cdot (\theta^T x^b - y_b)_q$$

Therefore,

$$M_{pq} = 2\sum_i x_p^i \cdot (\theta^T x^i - y_i)_q$$

$$M_{pq} = \frac{dJ(\theta)}{d\theta_{pq}}$$

(c)

The cost function and gradient are implemented as follows. They are similar to the cost function and gradient that we had in the previous part.

```
def f_p6(x, y, theta):
    return sum((y - dot(theta.T, x))**2)

def df_p6(x, y, theta):
    return 2*np.dot(x, (dot(theta.T, x) - y).T)
```

(d)

The approximation formula uses the definition of derivative. To compute the gradient of one specific point, get the value of functions of another point close to that point. And then calculate the average difference between them.

```
def part6d(n,m):
    y = np.ones([6, 600])
    x = np.ones([1025,600])
    theta = np.ones([1025, 6])
    theta_h = np.ones([1025,6])
    theta_h[n,m] = theta_h[n,m] + 1e-5

    print (f_p6(x, y, theta_h) - f_p6(x, y, theta))/ 1e-5
    print df_p6(x, y, theta)[n,m]
part6d(1,2)
```

the result of (1,2) shows:
finite-difference approximation: 1228799.963
computed from gradient: 1228800.0
so the difference is 0.0370

Part 7

In this part, we will need to create a classifier that classify different actors and report the performance on test set. The full training set with the size of each actor 100 and the test set with the size of 10 are used for this part. I use the matrix format similar to part 6. x is also a combination of flat images from 6 different actors, and y is a 600×6 matrix that looks like the one below. Each image is labelled by a 6 dimensions binary vector.

$$Baldwin \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ \dots & \dots & 100 & rows & \dots & \dots \\ 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$Hader \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ \dots & \dots & 100 & rows & \dots & \dots \\ 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

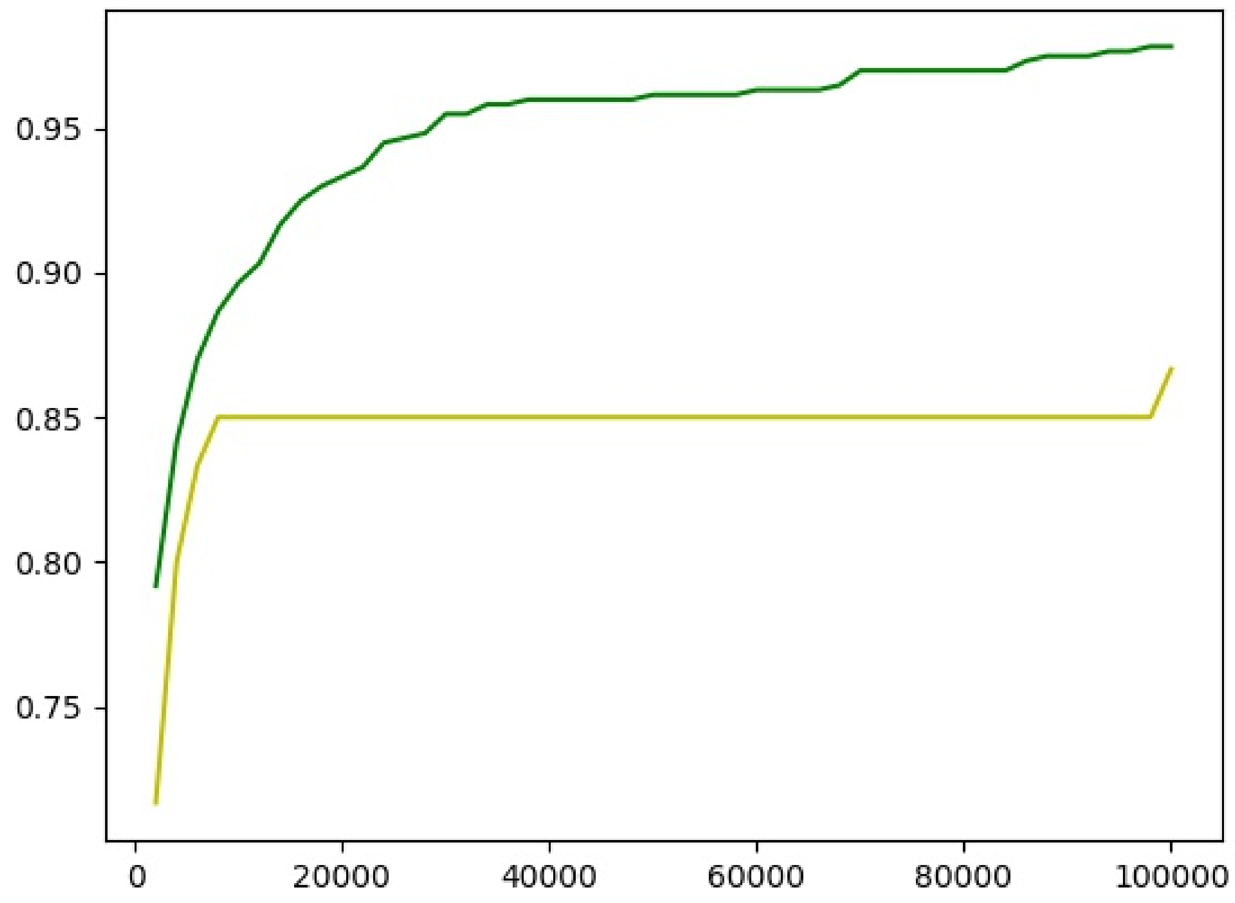
$$Carell \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ \dots & \dots & 100 & rows & \dots & \dots \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

$$Bracco \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ \dots & \dots & 100 & rows & \dots & \dots \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

$$Harmon \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ \dots & \dots & 100 & rows & \dots & \dots \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

$$Gilpin \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ \dots & \dots & 100 & rows & \dots & \dots \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

As the result shows, the accuracy of training set reaches 0.98 and the validation set accuracy stands at 0.86. Below is a plot showing how the performance changes as the iteration times increase.

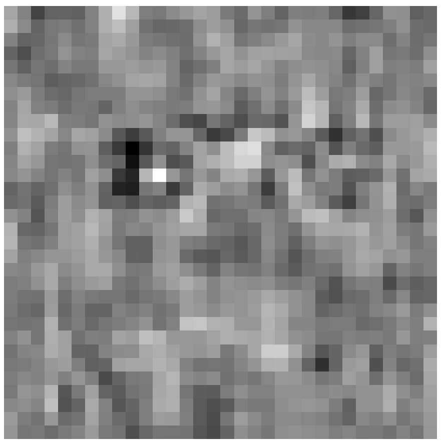


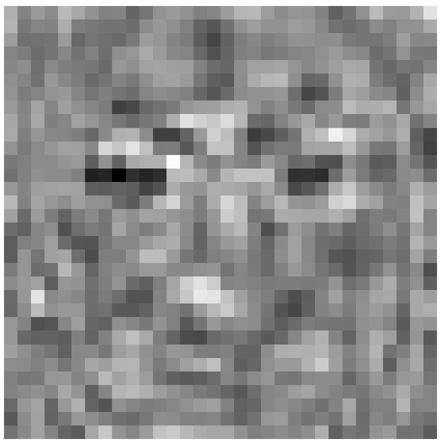
Green: Train Set, Yellow: Valid Set
X: Iteration Time, Y: Accuracy

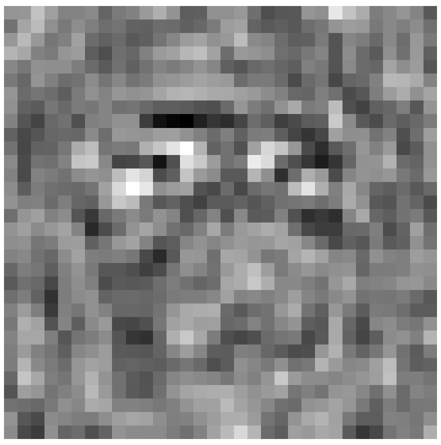
Similar to the procedure producing x in part5, in this part x has fixed size of 100, so x is a 600 by 1025 matrix and the initial theta is a 1025 by 6 matrix whose cells are all set to be zero. I set alpha to be 9e-7, same as the value used in part3 to generate iteration vs cost plot. This alpha value makes sense because the gradient value in the last iteration is very close to zero. Other parameters are good because the dimension of matrices align well. The result we obtained from computing the dot product of theta and x is a 600 by 6 matrix. To count the accuracy, for each 6 dimension vector in the matrix, find the index of maximum value in a row by using the argmax() function. After that, check the same row of y, see whether the same index in this vector from y is one. If it is one, then we can increment the counter by one, if not, then it means that our prediction for this image fail. At the end of all loops, divide the counter by size of test set and get accuracy.

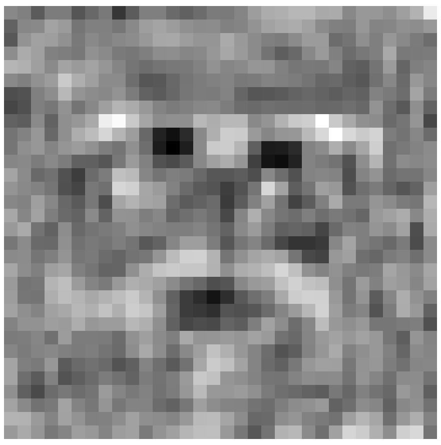
Part 8

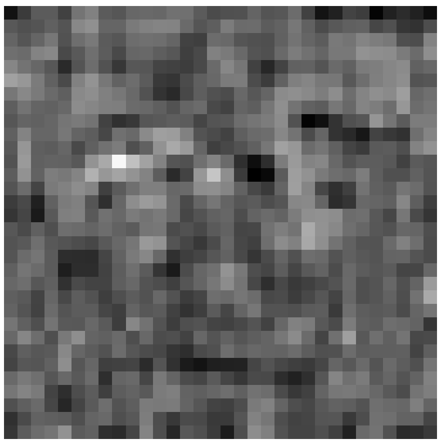
Below are the 6 theta images obtain from 6 actor in part 7.

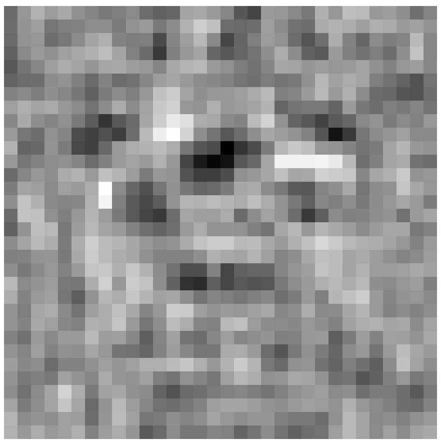

theta of harmon


theta of gilpin


theta of bracco


theta of carell


theta of baldwin


theta of hader