CSC411 Project 1 - Face Recognition and Gender Classification with Regression

Che Liu, 1002246839

January 2018

A Note on Reproducing the Results

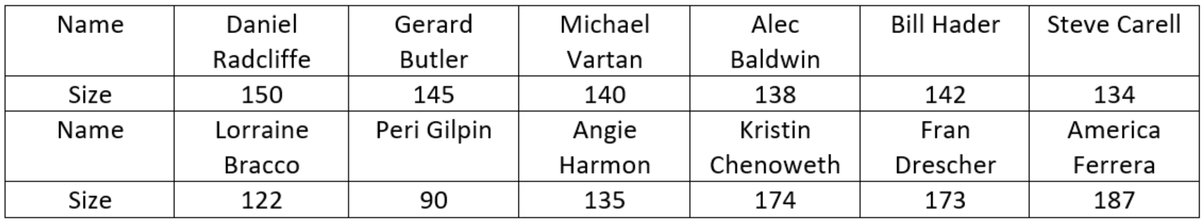
A zip file containing all the required documents and scripts was submitted for this project. The zip file consists of one pdf report named faces.pdf, one tex file named faces.tex, five py scripts, and two folders named final_male and final_female. The five .py files are get_data.py, part3.py, part5.py, part6.py, and part7.py.

get_data.py downloads the images necessary for this project and processes them so that they are usable for later parts. part3.py contains the code for Part 3 and Part 4 of this project. part5.py and part6.py contains the code for their respective parts. part7.py contains the code for Part 7 and Part 8 of this project.

Since all the paths are relative, all the codes should be able to run in any directory. If you wish to use the zip file containing all the processed images instead of downloading them, please move the two folders (final_male and final_female) to the same directory where you intend to run the Python scripts. All other scripts should be able to run properly after that.

Part 1

The images downloaded from the URLs provided in facescrub_actors.txt and facescrub_actresses.txt are of different sizes, angles, and color scales. Before cropping, converting to grayscale, and rescaling the images, the dataset of faces was composed of 1742 images of which 855 were of actors and 887 were of actresses. After manipulations mentioned above, the usable dataset shrink to 1688 images of which 834 were of actors and 854 were of actresses. The shrinkage was caused by factors such as invalid images. The exact sizes of the usable datasets for each actor and actress is shown in the table below:



Name	Daniel Radcliffe	Gerard Butler	Michael Vartan	Alec Baldwin	Bill Hader	Steve Carell
Size	150	145	140	138	142	134
Name	Lorraine Bracco	Peri Gilpin	Angie Harmon	Kristin Chenoweth	Fran Drescher	America Ferrera
Size	122	90	135	174	173	187

Figure 1. Sizes of usable images of actors and actresses.

Three examples of the images in the original dataset are as follows:

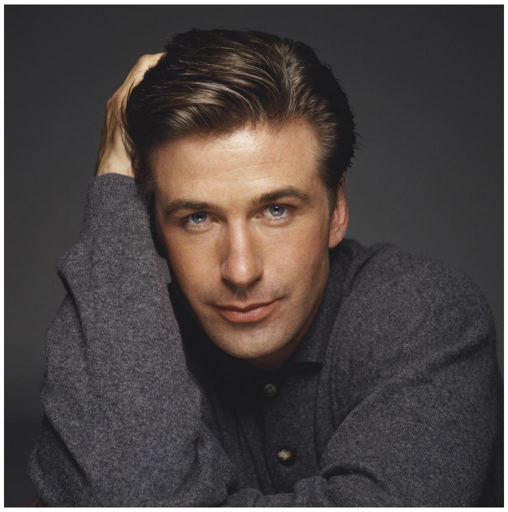


Figure 2. Original image of Alec Baldwin in the dataset.

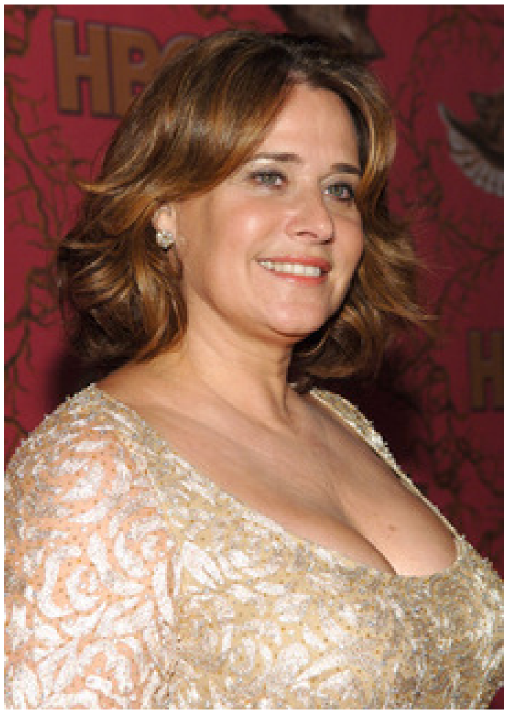


Figure 3. original image of Lorraine Bracco in the dataset.

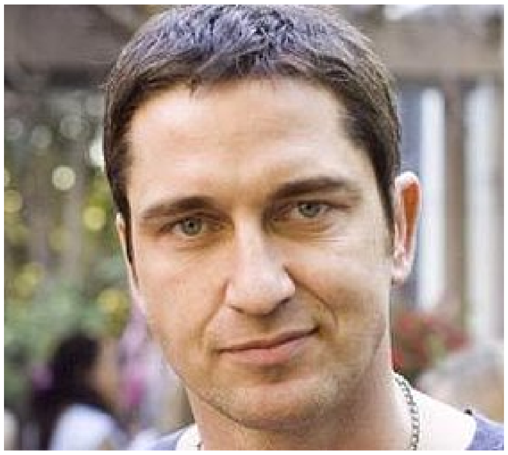


Figure 4. Original image of Gerard Butler in the dataset.

Three examples of the cropped out faces are as follows:

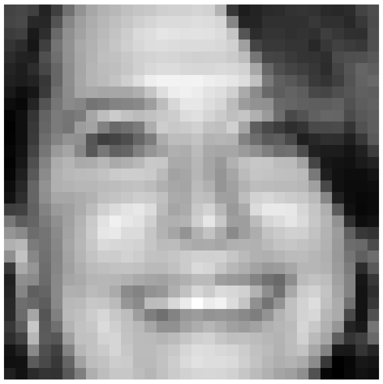


Figure 5. Cropped image of Lorraine Bracco in the dataset.

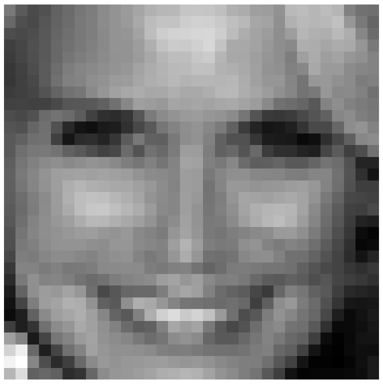


Figure 6. Cropped image of GKristin Chenoweth in the dataset.

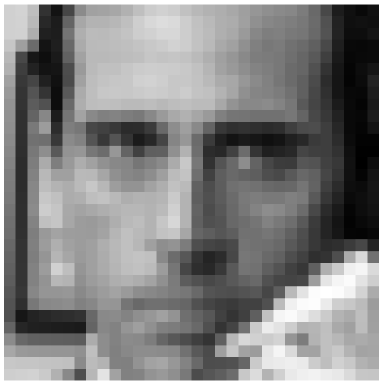


Figure 7. Cropped image of Steve Carell in the dataset.

The bounding boxes are accurate for the vast majority of images. Quantitatively, the faces in only 7 out of 1688 cropped images are incomplete or absent. Two of such examples are baldwin67.jpg and carell93.jpeg. The former is shown below:

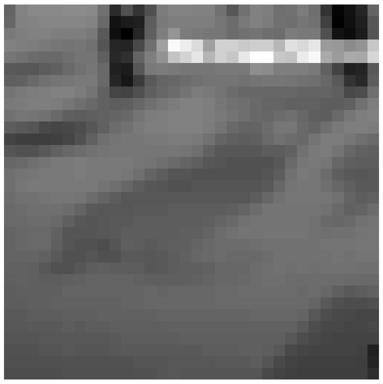


Figure 8. Baldwin67.jpg with incorrect bounding box.

The cropped out faces are mostly aligned with each other. In some images, however, the actor/actress is not facing the camera or is tilting his/her head. In some images, the actor/actress wears glasses, which may have a slight impact on the accuracy of regression. An example in which the actor is not facing forward is shown below:

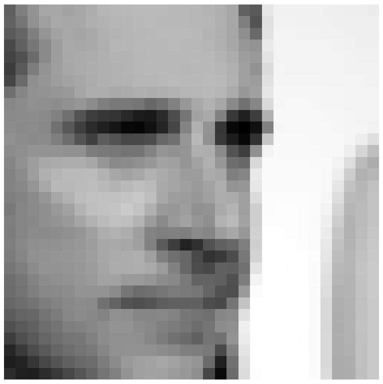


Figure 9. Carell89.jpg in which the actor's face does not align with others.

Part 2

90 images need to be selected for each actor/actress without overlapping. More specifically, 70 images for training, 10 for validation, and 10 for testing need to be chosen by means of randomness. The algorithm used is as follows:

- Initialize twelve empty lists corresponding to the twelve actors/actresses.
- Append the file names of all the cropped images to their corresponding lists.
- (Call np.random.seed (0) such that the result is reproducible.)
- To avoid potential bias, shuffle the lists using np.random.shuffle () for randomization.
- Initialize three more lists for each actor/actress, corresponding to the training, validation, and testing sets.
- For each list corresponding to the training set, pop the list containing all file names of images of that actor/actress 70 times and append to the training list.
- For each list corresponding to the validation set, pop the list containing all file names of images of that actor/actress 10 times and append to the validation list.
- For each list corresponding to the test set, pop the list containing all file names of images of that actor/actress 10 times and append to the test list.

Part 3

Linear regression is used to classify the images as of Alec Baldwin or Steve Carell, the following quadratic loss function is minimized:

$$J(\theta) = \sum_i (y^{(i)} - \theta^T x^{(i)})^2 \tag{1}$$

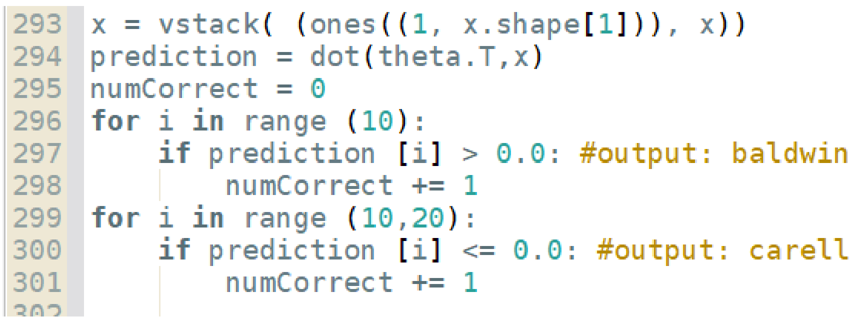
where $x^{(i)}$ corresponds to the pixel intensities of the i^{th} image divided by 255.0 of a particular image ($x_0 = 1$ for the offset θ_0), $y^{(i)}$ is the label of the image, $\theta^T = [\theta_0\ \theta_1\ ...\ \theta_{1024}]$ is the coefficients we need to run gradient descent on. $h_\theta = \theta^T x$ is the hypothesis.

The values of this cost function on training and validation sets are f(x) = 0.58813 and f(x) = 6.49813, respectively. This is consistent with our intuition since the tuned model fits the training set better than the validation set.

Performance-wise, the classifier can correctly classify 100% of images in the training set and 95% of images in the validation set.

Images of Baldwin and Carell are mapped to values of +1's and -1's, respectively. Thus, if the result of $\theta^T x^{(i)}$ is greater than 0, the algorithm believes that it is more probable for the image to be of Baldwin. And if the result is smaller than 0, the algorithm believes that it is more probable that the image is of Carell.

The code of the function used to compute the output of the classifier (on validation set) is as follows:



```
x = vstack( (ones((1, x.shape[1])), x))
prediction = dot(theta.T,x)
numCorrect = 0
for i in range (10):
    if prediction [i] > 0.0: #output: baldwin
        numCorrect += 1
for i in range (10,20):
    if prediction [i] <= 0.0: #output: carell
        numCorrect += 1
```

Figure 10. the function used to classify the output as Baldwin or Carell.

There are several factors that affect the performance of this classifier. In particular, a learning rate (α) that is too high can result in overflow which crashes the gradient descent algorithm, an indication that gradient descent does not converge. On the other hand, if the learning rate is set too low, gradient descent may converge too slowly or simply not be able to run many steps due to the EPS, although the cost function is convex and getting trapped in a local minimums is not a problem. What I did was starting from a rather large learning rate, and reduce the learning rate to half if the gradient descent algorithm crashes or the loss function ended up too big. I kept doing this until the loss cannot be reduced noticeably and the performance on both the training set and validation set are high. By then, the image generated by the θ's looks somewhat, but not overly resembles a face.

If I decrease EPS and increase the maximum number of iterations, gradient descent will produce θ's which models the training set more closely. However, this may result in a decrease in the performance on the validation set due to over-fitting. The approach I adopted was similar as before. I gradually increased the maximum number of interactions and decreased EPS such that the performance on the validation set did not decrease due to overfitting.

Furthermore, theta0 is drawn from a normal distribution centered at zero. Since the quadratic loss function is convex, its only minimum is its global minimum. The selection of theta0 need to minimize the initial cost of the cost function. By trying out normal distributions with different standard deviations centered at zero, it was found that the initial loss is generally minimized when the standard deviation is 0, or, in other words, when theta0 is all zeros.

Part 4

In this part, the coefficients θ_1 to θ_{1025} gotten from gradient descent are reshaped into a 32x32 matrix. This matrix is read as pixel intensities and displayed using the plt.imread () and plt.imshow () functions.

(a)

Using full training dataset (70 images per actor), the following image is obtained:

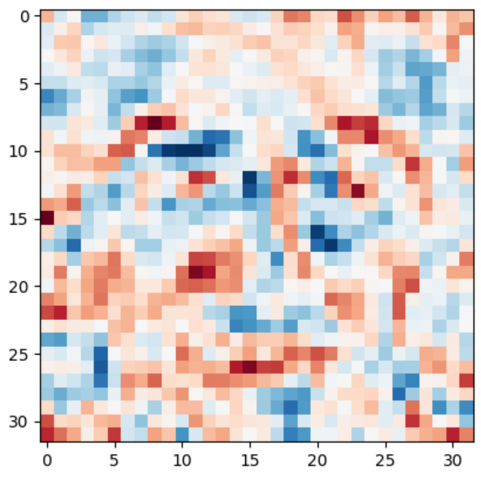


Figure 11. Visualization of the θ's obtained using full training set.

Using two images for each actor, the following image is obtained:

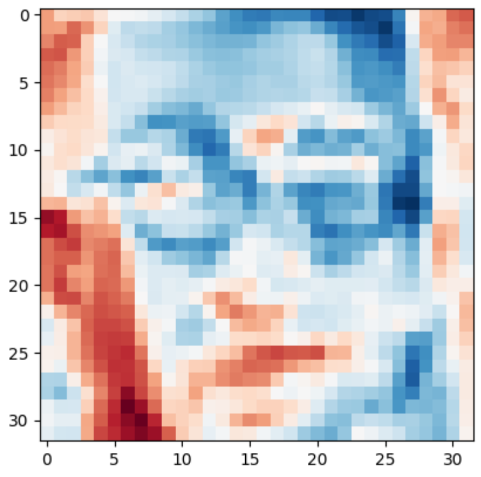


Figure 12. Visualization of the θ's obtained using two images of each actor.

(b)

In this part, we will obtain both a visualization that contains a face and a visualization that does not, using full training set (70 images for each actor). It was noticed that as stopping gradient descent earlier and earlier by setting the maximum number of iterations to a smaller and smaller value, the image resulting from the θ's looked increasingly like a face.

If I set the maximum number of iteration to 1000, the following visualization is obtained using the full training set:

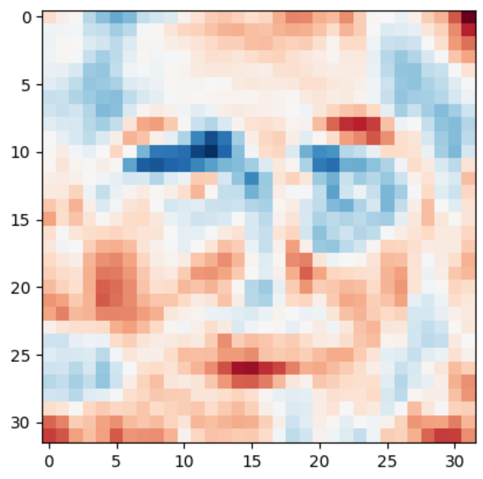


Figure 13. Visualization of the θ's obtained by ending gradient descent early.

On the other hand, if we end gradient descent later by setting the maximum number of iterations to a greater value and the EPS to a smaller value, visualization looked less like a face.

When I set the EPS to 1e-10 and the maximum number of iterations to 200000, the following visualization is obtained:

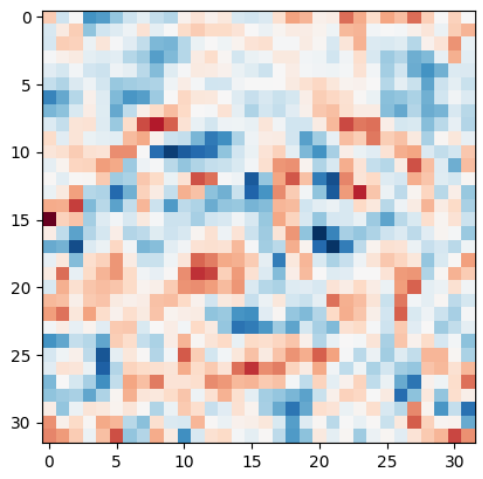


Figure 14. Visualization of the θ's obtained by ending gradient descent late.

Furthermore, theta0 also has an effect on whether the visualization appears to be a face. It seems that if we increase the standard deviation/variance of the normal distribution from which we draw theta0, the visualization looks more random. In particular, if we set the standard deviation to 0.1, while keeping the mean 0 (and restore all other settings back to those used in part 3), we get the following visualization:

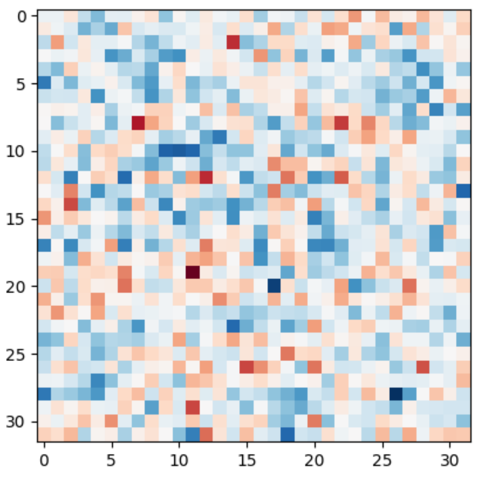


Figure 15. Visualization of the θ's obtained by increasing standard deviation.

In contrast, if we keep the distribution as N(0.,0.), the following visualization is obtained:

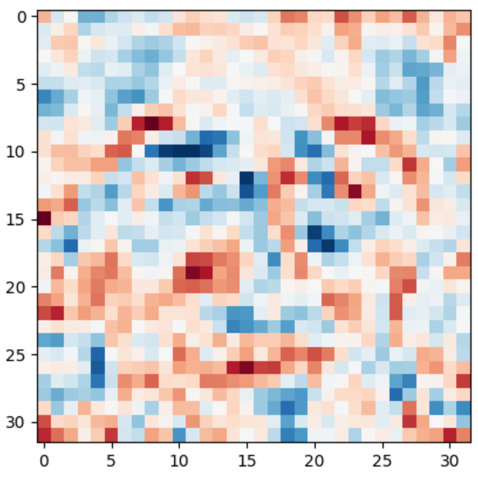


Figure 16. Visualization of the θ's obtained by having 0 standard deviation.

Part 5

Images are classified as of male or female in this part. To demonstrate overfitting, the percentages of images correctly classified on the training, validation, and unknown sets when the size of training set is between 40 and 200 (per gender) are plotted in increments of 10. The plot is shown below:

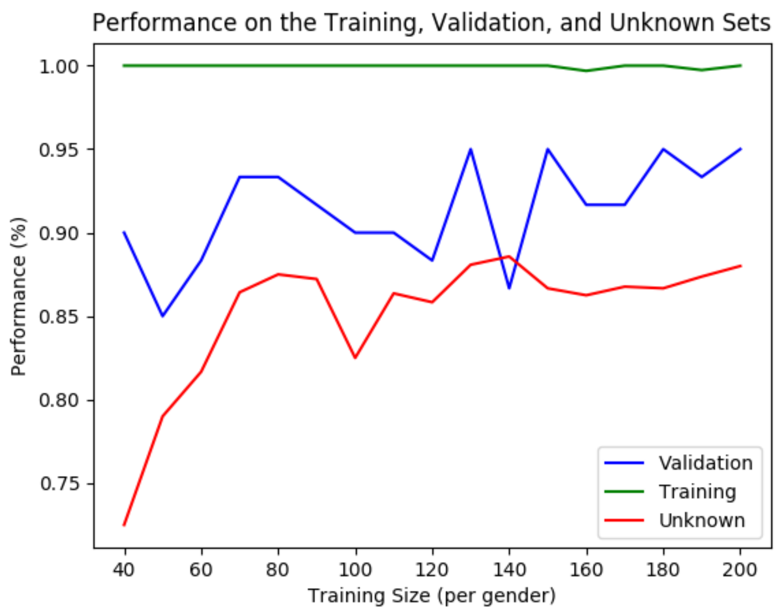


Figure 17. Performances of the gender classifier on the training, validation, and unknown set against size of the training set.

As can be clearly see, the percentage of images correctly classified on the validation and unknown sets steadily increases as the number of images we used to train the classifier increases. Specifically, the performances increased from around 85% and 70% for the validation and unknown sets to around 95% and 85%, respectively in this particular run. Nevertheless, the performance of the classifier on the training set drops slightly as we increase the size of training set.

The performance of the classifier on the validation and unknown sets are determined by the number of images correctly classified out of only sixty images hence explains the fluctuation. However, the raising trends on the validation and unknown sets are obvious to see.

Overfitting is an issue when the size of the training set is small, since the classifier models a small portion of the full training set more closely hence fails to model other images. The predictive power of the classifier is low when the size of the training set is small, because the performance on the training set ($\approx$ 100%) is significantly better than that on the validation/unknown sets ($\approx$ 80%). As the size of the training set increases, the performance also steadily increases because overfitting is less and less of a problem.

By the same reason, the performance of the classifier also increases on the actors not included in *act* when the size of the training set increases. In particular, when 200 images are used to train the classifier, around 95% of validation sets are correctly classified based on gender.

The maximum number of iterations is manually decreased to 10000 from 30000 to reduce the time it takes to run the program.

Part 6

a)

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

$$\frac{\partial J(\theta)}{\partial \theta_{pq}} = \frac{\partial}{\partial \theta_{pq}} \sum_i \left[(\theta^T x^{(i)} - y^{(i)})_1^2 + \dots + (\theta^T x^{(i)} - y^{(i)})_k^2 \right]$$

$$= \sum_i \frac{\partial}{\partial \theta_{pq}} \sum_j (\theta^T x^{(i)} - y^{(i)})_j^2$$

$$= \sum_i^m 2 \left(\sum_j^n \theta_{qj}^T x_j^{(i)} - y_q^{(i)} \right) \frac{\partial}{\partial \theta_{pq}} \left(\sum_j \theta_{qj}^T x_j^{(i)} - \cancel{y_q^{(i)}} \right)$$

$$= \sum_i^m 2 \sum_j^n \theta_{qj}^T x_j^{(i)} - y_q^{(i)}) \frac{\partial}{\partial \theta_{pq}} \left(\theta_{1q} x_1^{(i)} + \dots + \theta_{nq} x_n^{(i)} \right)$$

$$= 2 \sum_{i=1}^m \left(x_p^{(i)} \cdot \left(\sum_{j=1}^n \theta_{qj}^T x_j^{(i)} - y_q^{(i)} \right) \right).$$

As can be seen from the deviation above:

$$\frac{\partial J}{\partial \theta_{pq}} = 2 \sum_{i=1}^{m} (x_p^{(i)} \sum_{j=1}^{n} (\theta_{qj}^T x_j^{(i)} - y_q^{(i)})) \tag{2}$$

b)

$$\frac{\partial J}{\partial \theta_{pq}} = 2 \sum_{i=1}^m \left(x_p^{(i)} \cdot \sum_{j=1}^n (\theta_{qj}^T x_j^{(i)} - y_q^{(i)}) \right)$$

$$= 2 \sum_{i=1}^m \left(x_{pi} \sum_{j=1}^n (\theta_{qj}^T x_{ji} - y_{qi}) \right)$$

$$= 2 \sum_{i=1}^m \left(X_{pi} \sum_{j=1}^n (\theta_{qj}^T X_{ji} - Y_{qi}) \right).$$

$$= 2 \sum_{i=1}^m X_{pi} (\theta^T X - Y)_{qi}$$

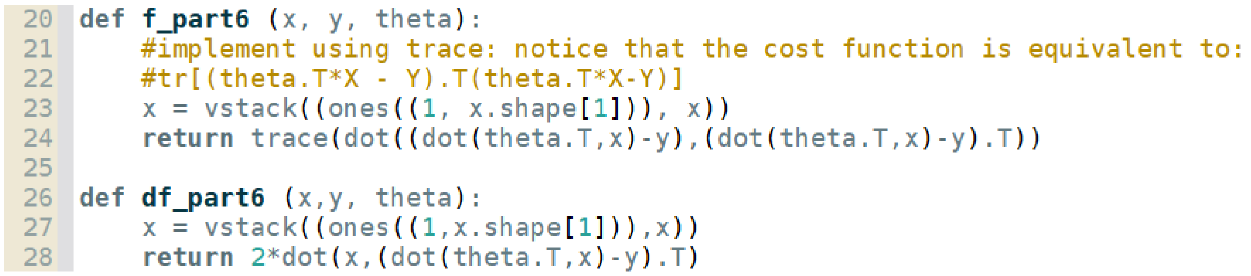
$$= 2 \sum_{i=1}^m X_{pi} (\theta^T X - Y)^T_{iq}$$

$$= \left(2X (\theta^T X - Y)^T \right)_{pq}.$$

Dimensions:
X: $n \times m$
Y: $k \times m$
θ: $n \times k$
where m is the number of training examples, n is the number of features (pixels) in each image (1025 including the bias term), k is the number of outputs.

c)

The implementation of the cost function from Part 6(a) and its vectorized gradient function in Python is shown below:



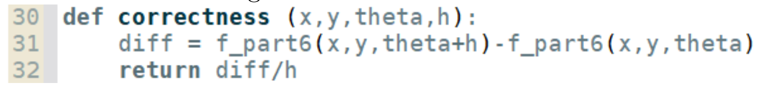
```python
def f_part6 (x, y, theta):
    #implement using trace: notice that the cost function is equivalent to:
    #tr[(theta.T*X - Y).T(theta.T*X-Y)]
    x = vstack((ones((1, x.shape[1])), x))
    return trace(dot((dot(theta.T,x)-y),(dot(theta.T,x)-y).T))

def df_part6 (x,y, theta):
    x = vstack((ones((1,x.shape[1])),x))
    return 2*dot(x,(dot(theta.T,x)-y).T)
```

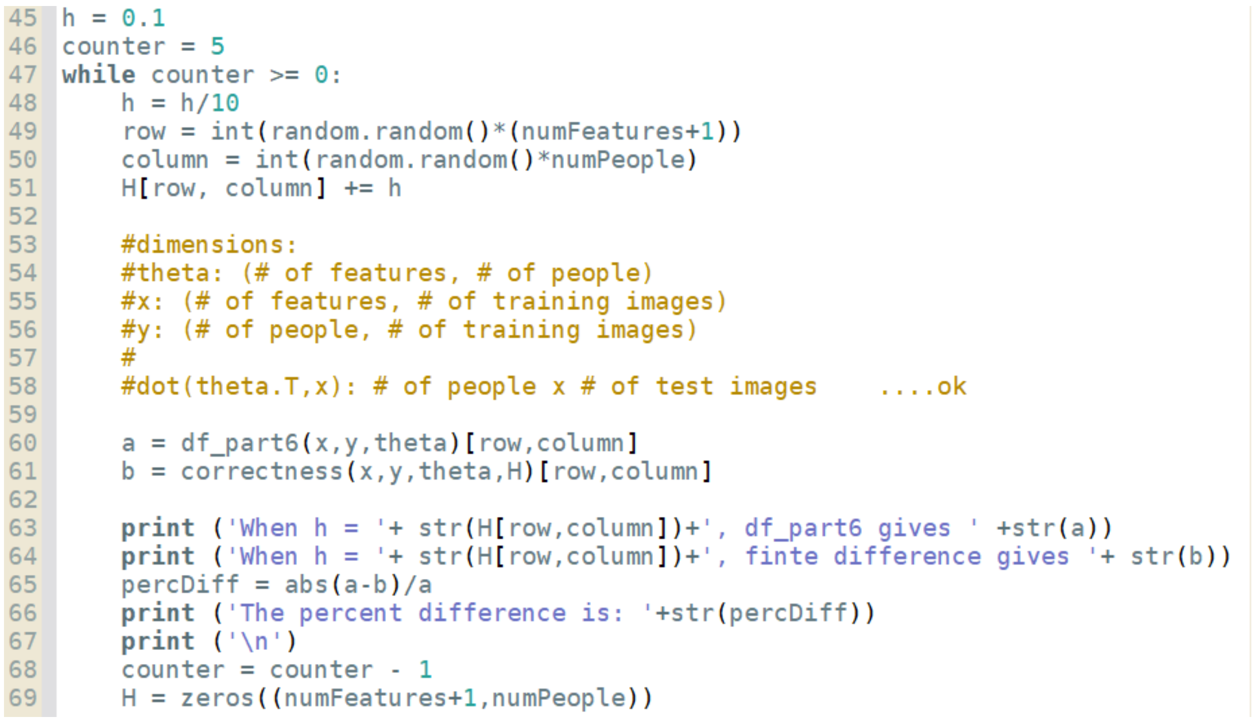
Figure 18. Python code for implementing the f and df functions.

(d)

Finite difference approximation is used to demonstrate that the implementation of the gradient function is correct. The following code is used to calculate the finite difference approximation:



```python
def correctness (x,y,theta,h):
    diff = f_part6(x,y,theta+h)-f_part6(x,y,theta)
    return diff/h
```



```python
h = 0.1
counter = 5
while counter >= 0:
    h = h/10
    row = int(random.random()*(numFeatures+1))
    column = int(random.random()*numPeople)
    H[row, column] += h

    #dimensions:
    #theta: (# of features, # of people)
    #x: (# of features, # of training images)
    #y: (# of people, # of training images)
    #
    #dot(theta.T,x): # of people x # of test images    ....ok

    a = df_part6(x,y,theta)[row,column]
    b = correctness(x,y,theta,H)[row,column]

    print ('When h = '+ str(H[row,column])+', df_part6 gives ' +str(a))
    print ('When h = '+ str(H[row,column])+', finte difference gives '+ str(b))
    percDiff = abs(a-b)/a
    print ('The percent difference is: '+str(percDiff))
    print ('\n')
    counter = counter - 1
    H = zeros((numFeatures+1,numPeople))
```

Figure 19. Python code for verifying the correctness of df.

In terms of selecting h, any h that is not too big and does not cause a numerical issue is acceptable.The output of this part is plotted for better visualization of the trend:

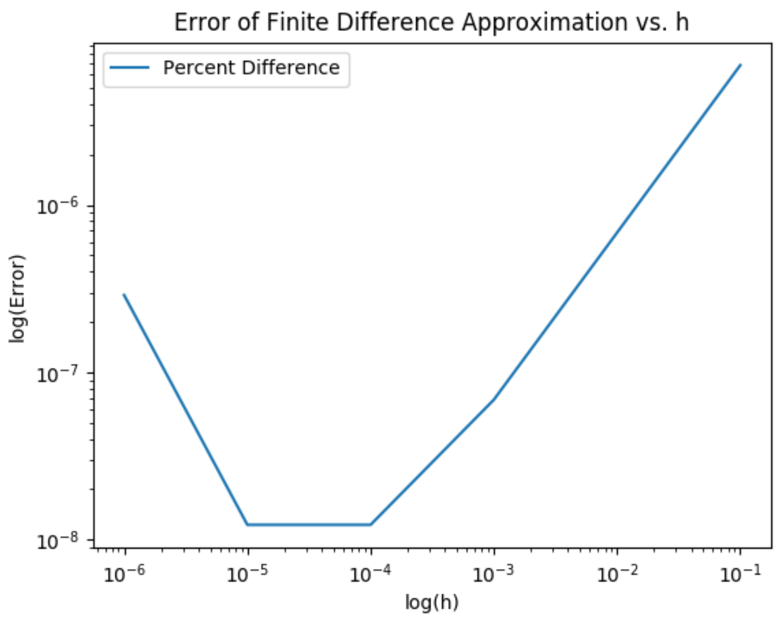


Figure 20. Graph of log (Error) and log (h) for visualizing the trend.

Part 7

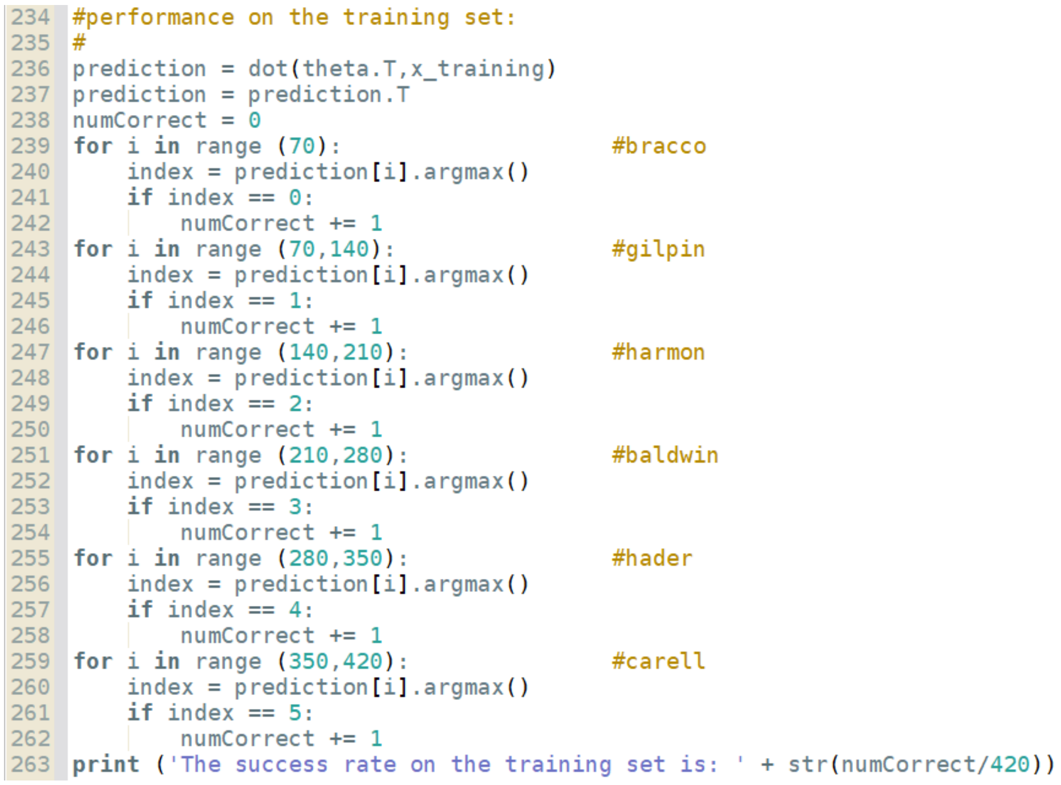
In this part, the classifier is trained to distinguish the six actors/actresses in the list act.

The performance that the classifier obtained on the training set is 97.38%

The performance that the classifier obtained on the validation set is 81.67%

A learning rate of 0.000001, EPS of 10e-5, max-iteration of 30000, and theta0 of all zeros are chosen using the same method as described in part 3. The value seems to make sense considering the performance on the validation set is excellent. If the algorithm simply guesses which actor/actress is in the picture, it would have a chance of merely 16.67% to get the correct answer. Given that the classifier gets 76.67% of the images correct in the validation set, clearly the algorithm is working hence gradient descent must be functioning as well.

Since one-hot coding is used to label the images when we generated y, to obtain the label in the classifier's prediction, we simply read $\theta^T x$ column by column, each of which corresponds to an image we want to find the predicted label of. The maximum value in each column indicates which actor/actress the classifier predicts. For instance, suppose a column in the output of the hypothesis function is [0.1, 0.5, 0.7, 0.4, 0.2, 0.9], it is an indication that the classifier classifies the image as one of Carell. This is because 0.9 is the highest value in the row, meaning the corresponding actor/actress is the most probable. The code used to achieve this functionality is given below: (I took the transpose of the hypothesis/prediction matrix so that it is easier to work with)



```python
#performance on the training set:
#
prediction = dot(theta.T,x_training)
prediction = prediction.T
numCorrect = 0
for i in range (70):                        #bracco
    index = prediction[i].argmax()
    if index == 0:
        numCorrect += 1
for i in range (70,140):                    #gilpin
    index = prediction[i].argmax()
    if index == 1:
        numCorrect += 1
for i in range (140,210):                   #harmon
    index = prediction[i].argmax()
    if index == 2:
        numCorrect += 1
for i in range (210,280):                   #baldwin
    index = prediction[i].argmax()
    if index == 3:
        numCorrect += 1
for i in range (280,350):                   #hader
    index = prediction[i].argmax()
    if index == 4:
        numCorrect += 1
for i in range (350,420):                   #carell
    index = prediction[i].argmax()
    if index == 5:
        numCorrect += 1
print ('The success rate on the training set is: ' + str(numCorrect/420))
```

Figure 21. Python code used for testing if the predictions are correct.

Part 8

We can get the θ visualization for each actor/actress by converting the corresponding column of θ to a 32x32 matrix and using the imread() function. The results are as shown in the images below:

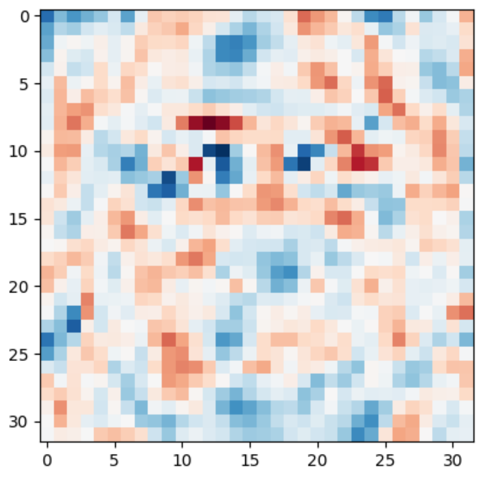


Figure 22. θ visualization for Bracco.

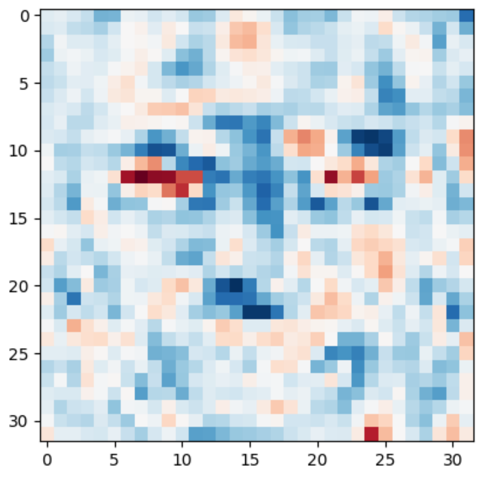


Figure 23. θ visualization for Gilpin..

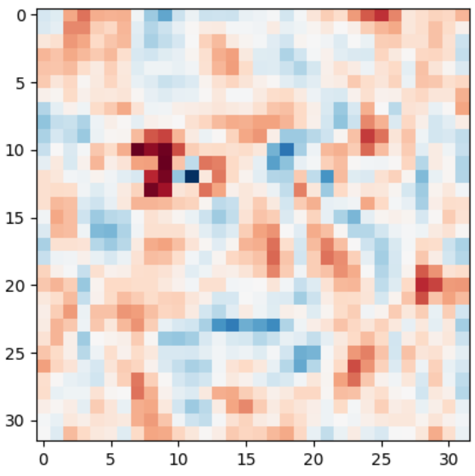


Figure 24. θ visualization for Harmon.

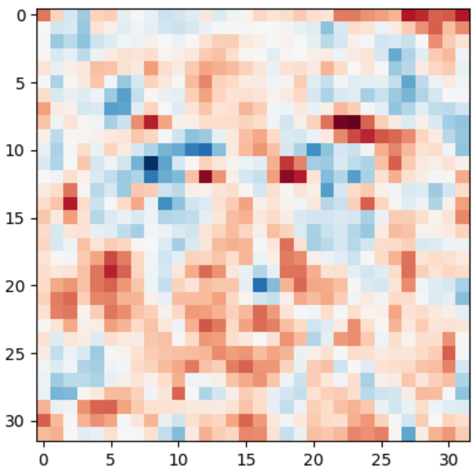


Figure 25. θ visualization for Baldwin.

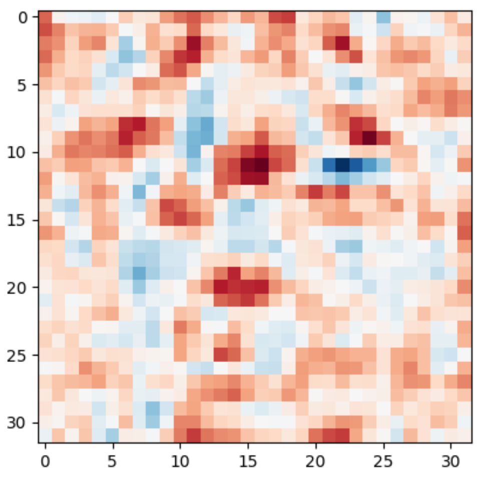


Figure 26. θ visualization for Hader.

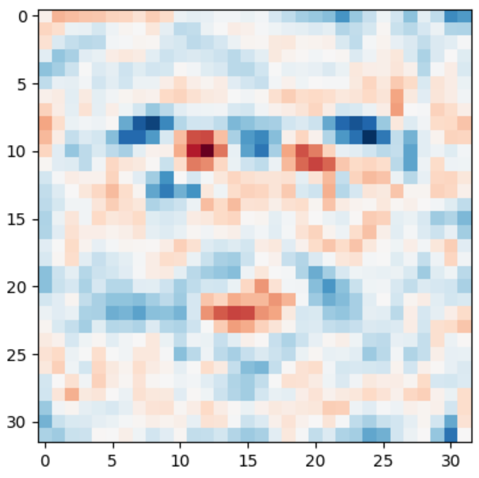


Figure 27. θ visualization for Carell.