CSC411: Assignment #1

Due on Monday, January 29, 2018

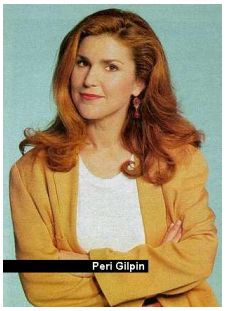
Yifei Dong

Part 1

Dataset description
The original images are from a subset of FaceScrub dataset. The images are cropped by the provided dimensions and converted into grayscale. The final dataset consists of 5400 grayscale 32×32-pixel of images of 6 actor and actresses, each of them with 90 images. The uncropped images are saved in a folder called uncropped and the cropped images are saved in a folder called cropped. Samples of uncropped and cropped images are shown in Figure 1.

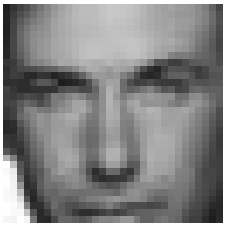
(a) Uncropped Baldwin1

(b) Uncropped Baldwin2



(c) Uncropped Gilpin

(d) Cropped Baldwin1



(e) Cropped Baldwin2

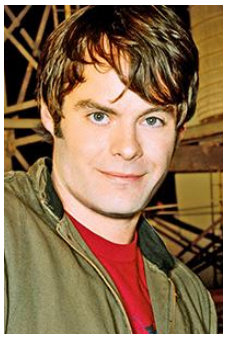
(f) Cropped Gilpin

Figure 1

From the samples above, we could see that the faces are cropped correctly. From the two images of Baldwin, the similarity of the two iamges can be seen by comparing the eyes, nose and mouth. And by comparing the cropped image of Baldwin to the cropped image of Gilpin we can see that both of them have only the face cropped. So the images can be compared between actor and actresses as well.

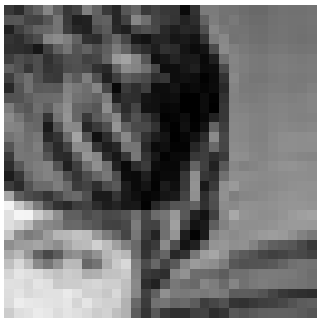
However, there are some issues of cropped images. Some examples are shown below in Figure 2. The first issue I faced is that some images are not correct images of the person, so the cropped image is something random. Another issue is that although the face is in the cropped image but the face is not shown properly. A third issue is that some original images are not downloaded properly so the images could not be read and thus cannot be cropped.

(a) Uncropped Baldwin



(b) Uncropped Harmon

(c) Cropped Baldwin



(d) Cropped Harmon

Figure 2

Part 2

Separate the dataset into three non-overlapping parts
The cropped grayscale faces are separated into three non-overlapping parts, which are training set, validation set and test set. I put all the cropped images into one folder called cropped but build three separate dictionaries for each set. The dictionaries are called tra, val and tes. I used the name of the images as the keys of dictionaries. For each key, there are two parameters. The first is the name of the actor or actress, the second is the gender. If it's an actor, the second parameter will be "M", otherwise it will be "F".

For example, a dictionary containing two images would look like this:

```
dict = {'bracco72.jpg': ['bracco', 'F'], 'baldwin79.jpg': ['baldwin', 'M']}
```

The dictionaries are built while saving the images. When a cropped image is successfully saved, it will be added to the corresponding dictionary.

I first built three empty dictionaries call tra, val and tes. Images are read and saved in the order of the act list, which contains the names of the actor and actresses we want. The outer for loop read and saved images of a single person each iteration. Therefore, in the inner for loop, I recorded the number of saved cropped images. When the number is smaller than 70, I save the images to tra dictionary. When the number is between 70 and 80, I save the images to val dictionary. When the number is between 80 and 90, I save the iamges to the tes dictionary. Thus, the three non-overlapping parts are created.

Part 3

Use Linear Regression in order to build a classifier to distinguish pictures of Alec Baldwin form pictures of Steve Carell.
The cost function I minimized is:

$$J(\theta_0, \theta_1, ..., \theta_{1024}) = \sum_{i=1}^{140} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2$$

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

The theta is of size $(1025, 1)$ since there are $32 \times 32 = 1024$ pixels per image and θ_0 is added as the intercept of Linear regression.$(h_\theta(x^{(i)}) - y^{(i)})^2$ is added 140 times because there are 140 images.

The value of the cost function on training set is $f_{min}(x) = 0.69$. The value of the cost function on validation set id $f_{min}(x) = 0.04$.

The accuracy of the Part 3 classifier on training set is 100%. The accuracy of the Part 3 classifier on validation set is 90%.

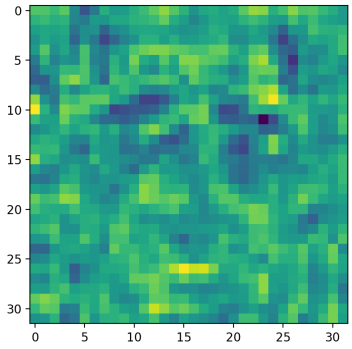
```
def part3(image):
    """
    Compute the output of the classifier. Either Steve Carell or Alec Baldwin is returned.
    """
    result = ""
    theta = classifierBC(tra, 70)
    im = imread(image)
    data = (np.reshape(im, (1024, 1)))/255.0
    data = np.vstack( (ones((1, data.shape[1])), data))
    y = np.dot(theta.T, data)
    if y <= 0.5:
        result = "Alec Baldwin"
    elif y > 0.5:
        result = "Steve Carell"
    return result
```

The above is the code I used to compute the output of the classifier. Theta was computed by the classifier called classifierBC. And the result y is compute by the formula $y = \theta^T x$, where x is matrix of flattened image.

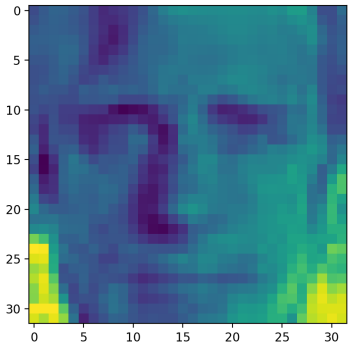
There are a few things I did in order to get the system to work.
1. The threshold ϵ in Gradient Descent function could change the behaviour of the system, as well as the maximum number of iterations. When the threshold is too high and the max_iter is too low, the gradient descent function would stop before the minimum value of cost function is found.
2. When α was set too large or too small. The minimum is passed and the gradient descent function diverged. I first set $\alpha = 10^{-8}$ and the gradient descent function stopped almost immediately. I finally set α to 0.00001 after a few times of testing.
3. I set the initial theta as an array of size $(1025, 1)$ with the same value. When I set the initial theta value as an array of 0s. The accuracy of training set is 100% as well as the test set. But the accuracy for validation set is only 85%. This might be overfitting. I tried with a few theta values. When the initial theta is set to 0.3, the accuracy is 100% for training set, 90% for validation set, and 95% for test set.

Part 4

a)



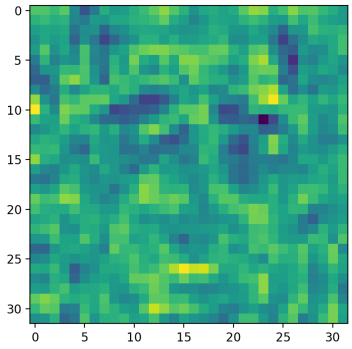
(a) θ got from full training set



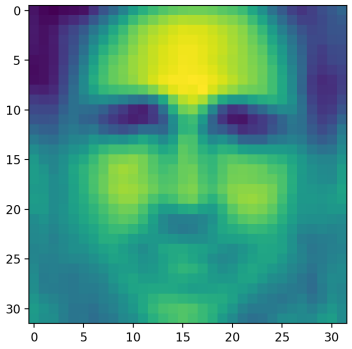
(b) θ got from a training set with two images

Figure 3

b)



(a) θ got from full training set with larger α



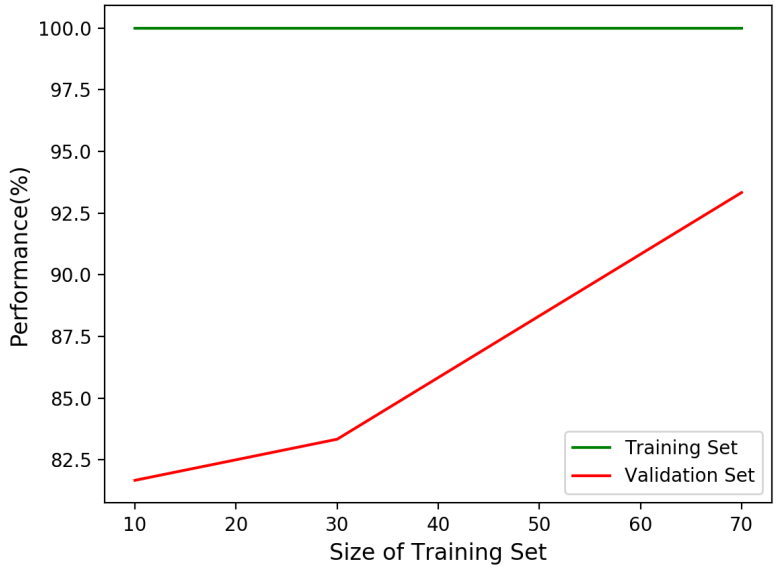
(b) θ got from a full training set with smaller α

Figure 4

Figure 4(b) looks similar to Figure 3(b). The faces are both obvious in them compare to Figure 4(a) and Figure 3(a). However, the two images are obtained by different methods. Figure 3(b) was computed by using a training set of size two, with one image from each actor. So the theta follows the face of the actor in the one image.
Figure 4(b) was obtained by using a full size training set. What I changed in code in order to get the image is α. I made α really small which is $1e-14$. As mentioned in Part 3, when alpha is small enough, the gradient descent function diverges fast, so it won't be able to fit many images. Thus the theta will looks like the first few images processed.

Part 5

Size(training)	Size(validation)	Performance on Training set(%)	Performance on Validation Set(%)
70	10	100.0	93.3333333333
50	10	100.0	88.3333333333
30	10	100.0	83.3333333333
10	10	100.0	81.6666666667



The above graph is the performance of the classifiers on the training and validation sets vs the size of the training set. The performance of the classifier on the training set is always 100%. However, the performance of the classifier on the validation set varies according to the size of the training set. As the size of the training set increases, the accuracy of the classifier on validation set also increases.

The accuracy of the classifier on test set of the six actors not included in act is 88.3333333333%.
The six actor and actresses are ['Gerard Butler', 'Michael Vartan', 'Daniel Radcliffe', 'Fran Drescher', 'America Ferrera', 'Kristin Chenoweth'].

Part 6

a)

$$J(\boldsymbol{\theta}) = \sum_i \sum_j (\boldsymbol{\theta}_{i,j}^T \boldsymbol{x}_{ij} - \boldsymbol{y}_{ij})^2$$

$$\frac{\partial J}{\partial \theta_{pq}} = \partial \boldsymbol{\theta}_q (2 \sum_j (\boldsymbol{\theta}_{p,j}^T \boldsymbol{x}_{pj} - \boldsymbol{y}_{pj}) \boldsymbol{x}_{pj})$$

$$= 2(\theta_{p,q}^T x_{pq} - y_{pq}) x_{pq}$$

b)

$$\frac{\partial J}{\partial \boldsymbol{\theta}} = \sum_i \sum_j 2(\theta_{p,q}^T x_{pq} - y_{pq}) x_{pq}$$

$$= 2 \sum_i (\boldsymbol{\theta}_p^T \boldsymbol{x}_p - \boldsymbol{y}_p) \boldsymbol{x}_p$$

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

The dimension of $\boldsymbol{x}$ is (1025, 240). $\boldsymbol{x}$ is a matrix containing all the input training data and additional 1's at the top of each column. Each column represents an image.
The dimension of $\boldsymbol{\theta}$ is (1025, 4). $\boldsymbol{\theta}$ is the parameter linear regression. It is computed by the gradient descent function. $\boldsymbol{\theta}_0$ is the intercept, others are parameters aligned with each pixel.
The dimension of $\boldsymbol{y}$ is (4, 280). $\boldsymbol{y}$ is the observed result. Each column represents the result of an image with the corresponding row be 1. Other rows in that column are 0. There are six actors and actresses, so the number of rows is 4. The number of columns is 280 since there are 280 images in the training set (70 images per actor, 4 actors in total).

c)

```
def f_part6(x, y, theta):
    x = np.vstack( (ones((1, x.shape[1])), x))
    return np.sum(np.sum(y - np.dot(theta.T, x)) ** 2)

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

The function *f_part6* is the implemented cost function and *df_part6* is the implemented vectorized gradient function.

d)
The following function is the code I used to compute the derivative of the finite-difference approximation.
x and y are defined randomly with the corresponding size. The theta is also defined randomly.
theta_h is a matrix of size (1025, 4) containing all zeros except one of the points is changed to h.
I have changed (0, 0) and (1, 2) of the *theta_h* to h, respectively.

```
def testPart6():
    random.seed(0)
    y = np.reshape(np.random.rand(4 * 140), (4, 140))
    x = np.reshape(np.random.rand(1024 * 140), (1024, 140))
    theta = np.reshape(np.random.random(1025 * 4), (1025, 4))

    h = 0.00000001

    theta_h = np.zeros((1025, 4))
    theta_h[0,0] = h
    print((0,0))
    print((f_part6(x, y, theta + theta_h) - f_part6(x, y, theta - theta_h)) / (2 * h))
    print(df_part6(x, y, theta))

    theta_h = np.zeros((1025, 4))
    theta_h[1,2] = h
    print((1,2))
    print((f_part6(x, y, theta + theta_h) - f_part6(x, y, theta - theta_h)) / (2 * h))
    print(df_part6(x, y, theta))
```

The following is the result I got from the code above. We can see that for (0,0). The result of the finite-difference approximation is close to the (1, 1) of the result of the gradient function. And for (1, 2), the result of the finite-difference approximation is close to the (2, 3) of the result of the gradient function. Before I chose this h, I did the test among several coordinates and adjusted h to make sure the result from the finite-difference approximation is close to the corresponding result from the gradient function. Either the h is too large or too small would make the finite-difference approximation not accurate enough.

```
(0, 0)
71171.6711521
[[ 71174.1400375   75619.89282615  72285.0143558   70610.09942562]
 [ 35596.50689779  37852.13636156  36150.33834077  35326.06103187]
 [ 34594.25558401  36741.78933463  35144.65254695  34303.55726553]
 ...,
 [ 33216.59964796  35306.8493485   33720.35340743  32889.14475364]
 [ 32277.94772054  34293.783001    32803.47980477  32036.19881797]
 [ 33265.41909505  35348.36815007  33769.67890899  32976.42957597]]
(1, 2)
36150.2170563
[[ 71174.1400375   75619.89282615  72285.0143558   70610.09942562]
 [ 35596.50689779  37852.13636156  36150.33834077  35326.06103187]
 [ 34594.25558401  36741.78933463  35144.65254695  34303.55726553]
 ...,
 [ 33216.59964796  35306.8493485   33720.35340743  32889.14475364]
 [ 32277.94772054  34293.783001    32803.47980477  32036.19881797]
 [ 33265.41909505  35348.36815007  33769.67890899  32976.42957597]]
```

Part 7

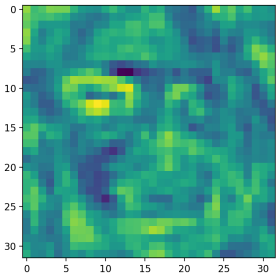
The performances of the classifier on training set and validation are 92.1428571429% and 85.0%, respectively. For gradient descent, the theta I choose is an array of zeros of size (1025, 6), the α I chose is 0.0000005. The parameters does make sense because the gradient descent function does converge and finds a minimum cost function. Moreover, the performances of the classifier on training set and validation set show that it was not overfitted.

Output values are assigned like this:

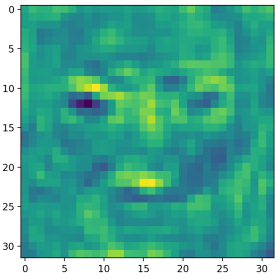
```
'bracco':  [1,0,0,0,0,0]
'gilpin':  [0,1,0,0,0,0]
'harmon':  [0,0,1,0,0,0]
'baldwin': [0,0,0,1,0,0]
'hader':   [0,0,0,0,1,0]
'carell':  [0,0,0,0,0,1]
```

The result I used to classify the image is $\boldsymbol{\theta}^T \boldsymbol{x}$, where θ is computed by the classifier and x is the pixels of the image. More specifically, the result is computed by $[\theta_1^T x_i, \theta_2^T x_i, \theta_3^T x_i, \theta_4^T x_i, \theta_5^T x_i, \theta_6^T x_i]$, since 1 is added to x. Therefore, the result would be a matrix of size $(6, 1)$. As shown above, the six indexes would be correspond to each actor. The value of each row in the matrix corresponds to the similarity of the image to the actor. So if row i has the highest value among the six, that means the image is most similar to actor i.

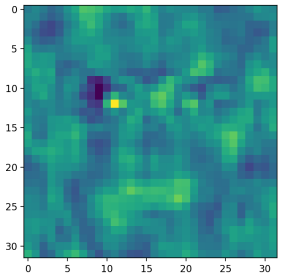
Part 8



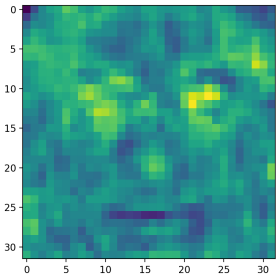
(a) Lorraine Bracco



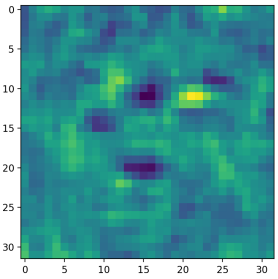
(b) Peri Gilpin



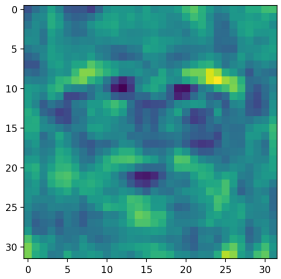
(c) Angie Harmon



(d) Alec Baldwin



(e) Bill Hader



(f) Steve Carell

Figure 5