CSC411: Assignment 1

Due on Monday, January 29, 2018

Zhongtian Ouyang

Problem 1

Dataset description

The pictures are downloaded and mordified using the get_data.py from FaceScrub dataset. A ZIP of cropped pictures is included in the submission. Unzipped it and put it in the root folder for the codes to word. The FaceScrub dataset claims to have about 200 per person. In reality, for the 6 actors we choose to use, each of them has about 120 downloadable photos, except for Gilpin who only have about 90 good photos. For the downloadable pictures, a very few of them is either corrupted or invalid such as the carell39.jpg showed below. There are also one to two photoes that has inaccurate bounding boxes, demonstrated by the cropped and uncropped harmon50.jpg below. For most of the pictures, the bounding box is pretty accurate and the cropped version is able to aligned with each other, at least to some degree. See the examples below for Baldwin.

(a) baldwin0.jpg

(b) baldwin1.jpg

(c) baldwin2.jpg

(d) baldwin3.jpg

(e) baldwin4.jpg

(f) harmon50.jpg

Figure 1: Example cropped figures from dataset

(a) baldwin0.jpg

(b) carell39.jpg

(c) harmon50.jpg

Figure 2: Example uncropped figures from dataset

Problem 2

Seperate Dataset into Training Set, Validation Set and Test Set

The seperation of the dataset is done using the seperate_sets function in faces.py. The input of the function is the path of the directory where the cropped images are saved. The function would create a folder named "sets" in the current directory and three sub folders in the "sets" folder. The images will be readed as a list and shuffled. The first ten occurrences of each character in the shuffled list will be copied to the test_set folder, the next ten will be copied to the validation_set folder. The next seventy will be copied to the training_set folder. Notice that there is only 88 pictures of Gilpin, so there is only 68 of her photo in the training set.

Problem 3

Use Linear Regression to build a classifier of two Actors' photoes

For this question, I choose to minimized the quadratic cost function $\sum_{i=1}^{m}(\theta^T X^i - y^i)^2)$. The value of the cost function on the training set is 0.26931154634136567. The performance on the training set is 100% correctness.The value of the cost function on the validation set is 1.3211391164441706. The performance on the validation set is 95% correctness.

In order to get the system to work, the alpha value is at most 0.00002, a value greater than that would make the value of the cost function keep growing and eventually overflow. It is possibly becuase the large alpha make the theta move too much. If the theta is too small, it will need more iterations to reach a similar value of the cost function.

The tester_p3 load each image from the desired set and it use the classifier_p3 to find out who that person is and compare it to the correct answer. At the end, the function would print the correct rate and the cost function for that set.

```
def classifier_p3(x, theta):
    x = np.hstack((np.ones(1), x))
    h_x = np.dot(theta.T, x)
    if h_x > 0.5:
        return actors[3]
    else:
        return actors[5]

def tester_p3(directory, theta):
    images = os.listdir(directory)
    correct = 0.0
    total = 0.0
    x_temp = []
    y_temp = []
    for image in images:
        cur_name = re.search('[a-z]+', image).group(0)
        flag = -1
        if cur_name == actors[3]:
            flag = 1
        elif cur_name == actors[5]:
            flag = 0

        if flag != -1:
            im = imread(directory+"/"+image)
            im = im.reshape(1024)
            im = im.astype(float)
            im = im/255
            x_temp.append(im)
            y_temp.append(flag)
            total += 1
            result = classifier_p3(im, theta)
            if result == cur_name:
                correct += 1

    y_temp = np.array(y_temp)
    x_temp = np.array(x_temp)
    x_temp = np.vstack((np.ones((1, x_temp.T.shape[1])), x_temp.T))
    cost = f_p3(x_temp,y_temp,theta)
    print directory + ' tested'
    print "Cost function's value is " + str(cost)
    print str(correct) + "/" + str(total) + " is correct. The accuracy rate is " + str(
        correct/total*100) + "%."
```

Problem 4

Showing Thetas As Heatmap

Part A)
The left figure is the theta obtained from P3, the right figure is the theta from training using a training set that contains only two images of each actor. The left one doesn't looks like a face because while training in P3, the value of the cost function gets to pretty small so the theta is overfitted to the training set.

Part B)
To make theta contains a face, I stop the gradient descent fairly early at 1000 iteration when the cost function is still around 7. And for my own interest, I tested the performance of this theta on the training set is about 97%, while on the validation set, its performance is 100%

(a) Full set (b) Two each

Figure 3: Part4 (a)

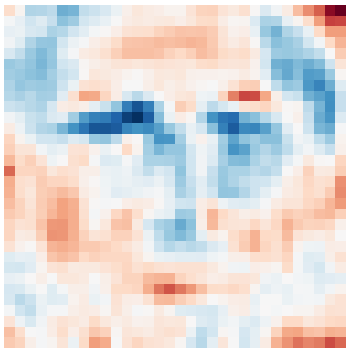


Figure 4: Part4 (b)

Problem 5

Demonstration of overfit

Below is the plotting for performance of the classifiers on the training and validation sets vs the size of the training set. To demonstrate the overfitting, I run gradient descent with training sets with various numbers of photoes of the 6 actors in act for 100000 iteration. Then use all the photoes of actors not in act as the validation set. First when the training set increase from 100 to 400, we can see a increase in the performance on the validation set. Because as the size of training set gets bigger, it is harder for the model to overfit those irregularities because the only contribute to a smaller and smaller percentage of the data. However, we can see that when the size of the training set increase from 400 to 500, the performance on the training set stays 100, while the performance on the test set drops slightly, this could be some spurious regularities in the training set builds up enough to affect the calculation of θ.

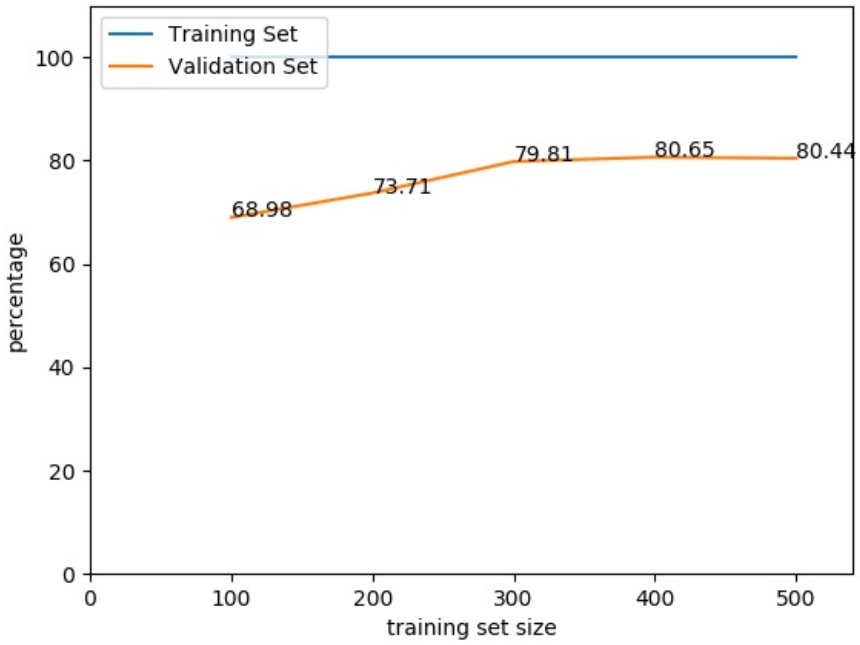


Figure 5: Demonstration of Overfitting

Problem 6

Multi-classification problem
Part A)

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

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

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

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

$$= 2 \sum_i x_{pq}^{(i)} \left(\sum_j \left(\theta^T x^{(i)} \right)_j - \sum_j y_j^{(i)} \right)$$

Figure 6: Derivative

Part B)

$$\frac{\partial J}{\partial \theta} = \left(\begin{bmatrix} \frac{\partial J}{\partial \theta_{11}} & \cdots & \frac{\partial J}{\partial \theta_{1k}} \\ & \vdots & \\ \frac{\partial J}{\partial \theta_{n1}} & \cdots & \frac{\partial J}{\partial \theta_{nk}} \end{bmatrix} \right)$$

Each value in $\frac{\partial J}{\partial \theta}$ correspond to one such $\frac{\partial J}{\partial \theta_{pq}}$ solved in 6(a). Each value in$(\theta^T X - Y)$ correspond to the $(\theta^T X^i - y^i)_j$ for some i, j. Transpose the whole part and dot product it by $2X$. Each entry in $2X(\theta^T X - Y)^T$ correspond to $2X_{pq}^i(\theta^T X^i - y^i)_j$ value for one of the k columns of the theta with one of the n sample images. Define k as the number of labels, m as the number of training example, n as the number of pixels in one example plus 1:
The dimension of θ is $n * k$
The dimension of X is $n * m$
The dimension of Y is $k * m$

Part C)
The f_p6 below is the cost function, df_p6 below is the gradient function. I have already transposed X and add a row of 1s before passing it to the function to avoid repeating the process during the gradient descent.

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

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

Part D)
The following code generates a random x with size 5x10, a random y with size 3x10 and a random theta with size 5x3. Then calculate and print the gradient on five directions. And in the end calculate and print the gradient using df_p6. We can check whether the values are close to the result. The result are shown below. It is easy to see that the results are almost identical.I choose a small enough h too demonstrate that when h gets to really small, the result from the two methods is almost identical

```
def part_6():
    np.random.seed(0)
    x = np.random.rand(5,10)
    y = np.random.rand(3,10)
    theta = np.random.rand(5,3)
    h = 0.0000001
    h_array = np.zeros((5,3))
    for i in range(5):
        h_array[i][1] = h
        print 'changing theta at [' + str(i)+', ' +  '1]'
        print (f_p6(x, y, theta+np.array(h_array)) - f_p6(x, y, theta-np.array(h_array)))
            /(2*h)
        h_array[i][1] = 0.0
    print df_p6(x, y, theta)
```

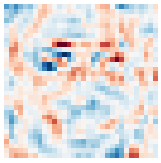
```
Doing part6
changing theta at [0, 1]
9.38697874986
changing theta at [1, 1]
9.87473635305
changing theta at [2, 1]
10.1618410397
changing theta at [3, 1]
8.29621136234
changing theta at [4, 1]
5.53991927887
[[ 14.009767     9.38697875   7.73695701]
 [ 14.29292737   9.87473635   7.21595527]
 [ 14.41686725  10.16184104   7.59067318]
 [ 13.31166437   8.29621136   8.07369466]
 [  6.73157401   5.53991927   3.35461769]]
```

Problem 7

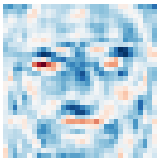
Implement Multi-classification The performance on the validation set is 95.6937799043%. The performance on the validation set is 95.6937799043%. I choose to use 0.000001 as alpha because that is the greatest alpha that would not induce an overflow when calculating the gradient. I stop the gradient descent early at 20000 iteration to avoid overfitting. Stopping it earlier would make the performance on validation set decrease. The output of the model will be a array of 6 number representing the probabilities of this images being each person. Find out the index of the max of the array and who that index respects to. For example, if output is [0.01, 0.03, 0.23, 0.85, 0.32, 0.05] and y for Baldwin is [0,0,0,1,0,0], then we say this is a picture of Baldwin.

Problem 8

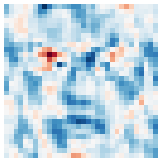
Images of thetas for Multi-classification Below are the 6 images for the theta



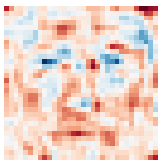
(a) bracco



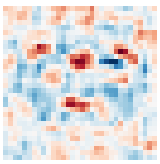
(b) gilpin



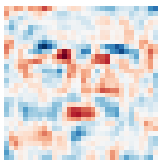
(c) harmon



(d) baldwin



(e) hader



(f) carell

Figure 7: Images of theta