CSC411 Project 1: Face Classification with Regression

Due on Friday, January 29, 2018

Hao Hui Tan

January 30, 2018

1 Problem 1

Important Note: For this project, I used **Python 2.7.14 Anaconda, Inc.** and my system specification development is MacOS Sierra Version 10.12.6. Everything is running fine on this system.

Note: I have uploaded the **cropped** folder to Markus that contains all the cropped images of the actors and actresses. Please put the cropped folder beside **faces.py** as the **get_data.py** script might not get all the images that I have due to time outs and other unforeseeable issues.

I have downloaded 1691 uncropped images of well-known actors and actresses from FaceScrub using the **get_data.py** script. Figures 1, 2 and 3 shows the uncropped images of Peri Gilpin, Steve Carell and Lorrain Bracco respectively. Figures 4, 5 and 6 show the cropped images of these individuals. In the **get_data.py** script I have transformed the raw images in the uncropped folder to grayscale, resized them to 32x32 and cropped them to the faces using the bounded box values.

Due to the unfortunate use of **imsave** function from the **pylab** library in the **get_data.py** script, the cropped images I got was of the shape (32, 32, 4) instead of (32, 32). As a result, when these images are opened using imread I got 4096 pixels instead of 1024 (32x32) due to the extra channel the cropped images has. To remedy the situation, whenever I read the cropped images, I always slice the images through [:, :, 0] in order to get rid of the extra channel and read the images as 32x32 for the purpose of this assignment.

As one can see, the cropped out images and the bounding boxes are fairly accurate in cropping out the faces. The bounding boxes values are slightly off in figure 6 because the person in the picture is not facing straight to the camera. The cropped out faces can be aligned with each other for figure 4 and figure 5. Figure 6, as mentioned, is slightly off to the left of the centre than those 2 images.

Figure 1: Peri Gilpin - uncropped

Figure 2: Steve Carell - uncropped

Figure 3: Lorraine Bracco - uncropped

Figure 4: Peri Gilpin - cropped

Figure 5: Steve Carell - cropped

Figure 6: Lorraine Bracco - cropped

2 Problem 2

Applying an algorithm: initial exploration.
I have used the shuffle_images function to separate the images into training set, validation set and test set. I have used dictionaries to store these images where the key is the actor or actress' last name and the value corresponding to the key is the list of his/her images. There are 70 images in the training set, 10 in validation set and 10 in the test set. My algorithm to achieve such a goal is to first put all the cropped image names into an array and then use **np.random.shuffle()** function to ensure they are out of order. Then I would iterate over all the actors/actresses, and for each actor/actress, I would iterate through all the images available and check if the actor/actress name is in each image's name. If it is, I would add the image to the list of images in the dictionaries with the actor/actress's last name being the dictionary key.

3 Problem 3

The cost function that I minimized

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

where θ is the vector of weights we seek to minimize x^i is the set of pixel features of the i^{th} image sample m is the number of image samples $h_\theta(x^i)$ is the hypothesis function with respect to x^i y^i is the label of the i^{th} image sample where $y^i = 1$ represents the image is Alec Baldwin and $y^i = 0$ represents the image is Steve Carell

The following values can be generated and seen in the **part_3_data_file.txt**:
The cost function value for the training set is 3.73786191455
The cost function value for the validation set is 2.41870583942
The performance of the classifier (percent accuracy) on the training set is 1.0
The performance of the classifier (percent accuracy) on the validation set is 0.85

The following function is the code that I used to compute the output of the classifier.

```
def binary_classify(performer_1_full, performer_2_full,  training_samples_num=70,
    alpha=0.0000010, init_theta_coefficient=0):
    """Inputs are 2 actors' last names to classify
       Returns the trained thetas array for the hypothesis and the cost function
           values
       for the training and validation sets
    """
    performer_1 = performer_1_full.split(" ")[1].lower()
    performer_2 = performer_2_full.split(" ")[1].lower()
    if performer_1_full in actors:
        path_1 = "cropped/actors/"
    else:
        path_1 = "cropped/actresses/"

    if performer_2_full in actors:
        path_2 = "cropped/actors/"
    else:
        path_2 = "cropped/actresses/"
    # Training images for alec baldwin and steve Carel
    performer_1_training_set = training_dictionary[performer_1]
    performer_2_training_set = training_dictionary[performer_2]
    performer_1_validation_set = validation_dictionary[performer_1]
    performer_2_validation_set = validation_dictionary[performer_2]
    training_set = []
    validation_set = []
    performer_1_num_images = 0
    performer_2_num_images = 0
    performer_1_num_images_validation = 0
    performer_2_num_images_validation = 0

    for image_name in performer_1_training_set:
        if performer_1_num_images > training_samples_num:
            break
        path = path_1 + image_name
```

```
        image_file = imread(path)[:, :, 0]
        # Get the flatten image of inputs
        flatten_image = image_file.flatten()
        flatten_image_processed = flatten_image / 255.0  # so that each input is
            between 0 and 1
        training_set.append(flatten_image_processed) # training set 2D array
        performer_1_num_images = performer_1_num_images + 1

    for image_name in performer_2_training_set:
        if performer_1_num_images > training_samples_num:
            break
        path = path_2 + image_name
        image_file = imread(path)[:, :, 0]
        # Get the flatten image of inputs
        flatten_image = image_file.flatten()
        flatten_image_processed = flatten_image / 255.0  # so that each input is
            between 0 and 1
        training_set.append(flatten_image_processed) # training set 2D array
        performer_2_num_images = performer_2_num_images + 1

    for image_name in performer_1_validation_set:
        if performer_1_num_images_validation > training_samples_num:
            break
        path = path_1 + image_name
        image_file = imread(path)[:, :, 0]
        # Get the flatten image of inputs
        flatten_image = image_file.flatten()
        flatten_image_processed = flatten_image / 255.0  # so that each input is
            between 0 and 1
        validation_set.append(flatten_image_processed) # training set 2D array
        performer_1_num_images_validation = performer_1_num_images_validation + 1

    for image_name in performer_2_validation_set:
        if performer_2_num_images_validation > training_samples_num:
            break
        path = path_1 + image_name
        image_file = imread(path)[:, :, 0]
        # Get the flatten image of inputs
        flatten_image = image_file.flatten()
        flatten_image_processed = flatten_image / 255.0  # so that each input is
            between 0 and 1
        validation_set.append(flatten_image_processed) # training set 2D array
        performer_2_num_images_validation = performer_2_num_images_validation + 1

    x_matrix = np.vstack(training_set) # input x matrix for gradient descent. 70 rows
        = 70 images. 4096 columns = pixels
    x_matrix_validation = np.vstack(validation_set)
    # print "sum of x_matrix is:", sum(x_matrix)
    # print "init_theta_coefficient", init_theta_coefficient
    initial_theta = init_theta_coefficient * np.ones(len(x_matrix[0])) # initial
        thetas for graident descent
    y_vector = np.ones(performer_1_num_images) # 1 for alec baldwin, 0 for Steve
        Carell
```

```
    y_vector = np.append(y_vector, np.zeros(performer_2_num_images))

    y_vector_validation = np.append(np.ones(performer_1_num_images_validation), np.
        zeros(performer_2_num_images_validation))
    result_theta = grad_descent(f, df, x_matrix.T, y_vector, initial_theta, alpha)
    result_cost_training = f(x_matrix.T, y_vector, result_theta)
    result_cost_validation = f(x_matrix_validation.T, y_vector_validation,
        result_theta)
    return result_theta, result_cost_training, result_cost_validation
```

Description of the process to get the code to work

I am using the python functions f, df and $grad_descent$ from the tutorial code. As I am building the classifier, there were dimension misalignment issues. I had to use the x_matrix transpose to solve the issue. At first, my alpha was 0.1. I run into the RuntimeWarning: overflow encountered in multiply error. It seems to suggest that the alpha is too large and as a result gradient descent overshoots the minima multiple times and as a result it could not converge. If the alpha is too small, it will terminate prematurely since this increases the chance for it to get stuck at a local minimum. I began to experiment with smaller alphas and eventually chose the alpha value of 0.0000010 as the default. I chose it because it is one of the smallest values for the code to not run forever and it gives a decent set of θ values for the prediction.

4 Problem 4

Part a)

Figure 7: Thetas obtained from training using the full training set

Figure 8: Thetas obtained from training using 2 samples from the training set

Figure 9: Thetas obtained from training using full training set

Figure 10: Thetas obtained from training using full training set but with different start thetas

In figure 10, I used a new set of initial θ values of 0.5 instead of zeros. In figure 10, I just followed the normal procedure to produce the image.

5 Problem 5

Performance of the gender classifiers on the training and validation sets vs the size of the training set.
Note: For this training problem, I use $y = 1$ to denote the performer in the image is male and $y = 0$ to denote the image is female.

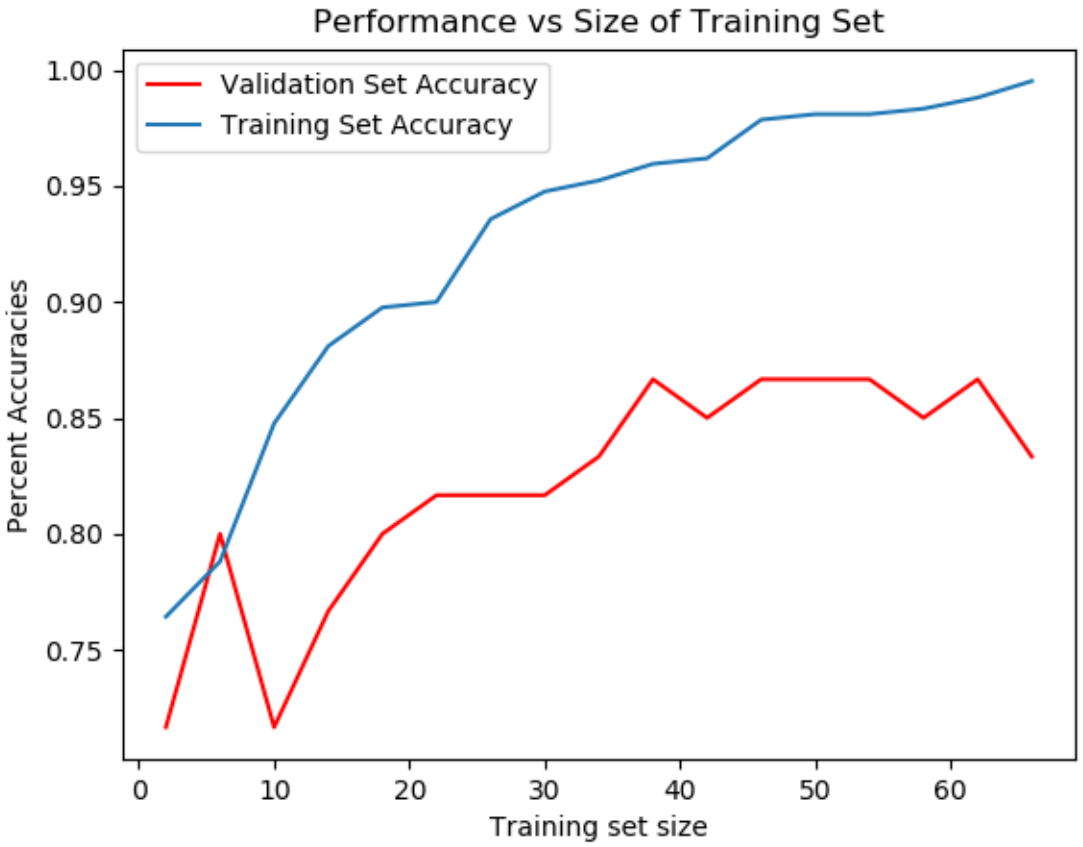

Figure 11: performance of the classifiers on the training and validation sets vs the size of the training set.

The following is the performance of the 6 actors who are not included in the original **act** set. The results are generated and seen in **part_5_different_performer_accuracies.txt** file.

Gerard Butler Accuracy is: 0.992857142857
Fran Drescher Accuracy is: 0.882352941176
Michael Vartan Accuracy is: 0.68085106383
America Ferrera Accuracy is: 0.955801104972
Daniel Radcliffe Accuracy is: 0.795918367347
Kristin Chenoweth Accuracy is: 0.766081871345

As one can see, there are clear signs of overfitting as the accuracies go down when one test the hypothesis on different actors and actresses that were not in the original actors and actresses sets for training.

6 Problem 6

Part a

$$J(\theta) = \sum_{i=1}^{m}(\sum_{j=1}^{k}(\theta^T x^i - y^i)_j^2)$$

By chain rule,

$$\frac{\partial J}{\partial \theta_{pq}} = (\sum_{i=1}^{m}(\sum_{j=1}^{k} 2(\theta^T x^i - y^i)_j \frac{\partial}{\partial \theta_{pq}}(\theta^T x^i)))$$

$$\frac{\partial J}{\partial \theta_{pq}} = 2(\sum_{i=1}^{m}(\sum_{j=1}^{k}(\theta^T x^i - y^i)_j x^i))$$

Part b

From **part a**, we see that the partial derivative of the cost function with respect to individual θ_{pq} is equal to $2(\sum_{i=1}^{m}(\sum_{j=1}^{k}(\theta^T x^i - y^i)_j)x^i)$. Then when you organize and put all the sample derivatives w.r.t to each θ together, you get

Note: I omitted the steps for taking the scalars out as they are trivial.

$$J(\begin{bmatrix} \theta_{11} & \theta_{12} & \theta_{13} & \dots & \theta_{1k} \\ \theta_{21} & \theta_{22} & \theta_{23} & \dots & \theta_{2k} \\ \dots & \dots & \dots & \dots & \dots \\ \theta_{n1} & \theta_{n2} & \theta_{n3} & \dots & \theta_{nk} \end{bmatrix}) = 2(\begin{bmatrix} x_{11} & x_{12} & x_{13} & \dots & x_{1m} \\ x_{21} & x_{22} & x_{23} & \dots & x_{2m} \\ \dots & \dots & \dots & \dots & \dots \\ x_{n1} & x_{n2} & x_{n3} & \dots & x_{nm} \end{bmatrix})[(\begin{bmatrix} \theta_{11} & \theta_{21} & \theta_{31} & \dots & \theta_{k1} \\ \theta_{12} & \theta_{22} & \theta_{32} & \dots & \theta_{k2} \\ \dots & \dots & \dots & \dots & \dots \\ \theta_{1n} & \theta_{2n} & \theta_{3n} & \dots & \theta_{kn} \end{bmatrix})$$

$$(\begin{bmatrix} x_{11} & x_{12} & x_{13} & \dots & x_{1m} \\ x_{21} & x_{22} & x_{23} & \dots & x_{2m} \\ \dots & \dots & \dots & \dots & \dots \\ x_{n1} & x_{m2} & x_{n3} & \dots & x_{nm} \end{bmatrix} - \begin{bmatrix} y_{11} & y_{12} & y_{13} & \dots & y_{1n} \\ y_{21} & y_{22} & x_{23} & \dots & y_{2n} \\ \dots & \dots & \dots & \dots & \dots \\ x_{k1} & y_{k2} & y_{k3} & \dots & y_{kn} \end{bmatrix})]$$

Now, we can simplify this to,

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

Let m be the number of sample images.
Let n be the number of features.
Let k be the number of ways that a label can be different.
Let X be the nxm matrix that contains all the input data. Each number is a pixel.
Let Y be the kxm matrix of labels where each row is the label vector. The label vector uniquely identifies the actor/actress
Let θ be a nxk matrix
Let θ^T be a kxn matrix

Part c

```
The following is my implementation of the cost function and the vectorized form of the
    gradient. \textbf{f_multiclass} is the cost function and \textbf{df_multiclass} is
    the gradient.
```

```
def f_multiclass(x, y, theta):
    """The multiclass cost function"""
    print "theta.T", theta.T
    print "x", x
    print "dot(theta.T,x)", dot(theta.T,x)
    return sum( (y - dot(theta.T,x)) ** 2)

def df_multiclass(x, y, theta):
    """The multiclass Gradient"""
    return -2*dot(x, (y-dot(theta.T, x)).T)
```

Part d

I used the following code to compute the finite difference gradient matrix using finite difference approximations. This function returns the gradient matrix on the initial thetas calculated using df, finite difference matrix calculated using finite difference with h=0.000000000001 for **each individual** θ (I append each finite difference value to this finite difference 2D-array matrix) and the comparison matrix which is the difference of the two. The x_input matrix is the image data, y_vector label is the same for all images since I used only 5 images of Alec Baldwin. The comparison matrix stores the difference between the gradient matrix and the finite difference matrix. If the gradient is correctly implemented, the values in the comparison matrix should be approaching 0 since the comparison matrix is equal to gradient - finite difference matrix.

```
def part_6_finite_difference(f_multiclass, df_multiclass, h=0.000000000001, k=4,
    init_theta_coefficient=0, alpha=0.0000010):
    """Returns the gradient in matrix form, finite difference in matrix form, and
        their """
    training_set = []
    # Choose 5 images
    path = "cropped/actors/"

    for i in range(5):
        image_name = training_dictionary["carell"][i]
        path_image = path + image_name
        image_file = imread(path_image)[:, :, 0]
        # Get the flatten image of inputs
        flatten_image = image_file.flatten()
        flatten_image_processed = flatten_image / 255.0
        training_set.append(flatten_image_processed)

    x_matrix = np.vstack(training_set)
    initial_theta = []
    for i in range(k):
        initial_theta_row = init_theta_coefficient * np.ones(len(x_matrix[0]))
        initial_theta.append(initial_theta_row)
    initial_theta = np.vstack(initial_theta)
    initial_theta = initial_theta.T
    initial_theta_copy = initial_theta.copy()
    y_vector = np.array([[0, 1, 0, 0] for i in range(len(x_matrix))])
    y_vector = np.vstack(y_vector)
    y_vector = y_vector.T
    gradient = df_multiclass(x_matrix.T, y_vector, initial_theta)
    lst_finite_difference = []
```

```
    ## Calculate the finite difference
    for row_idx in range(len(initial_theta)):
        lst_row = []
        for column_idx in range(len(initial_theta[0])):
            cost_function_original = f_multiclass(x_matrix.T, y_vector,
                initial_theta_copy)
            initial_theta_copy[row_idx][column_idx] = initial_theta_copy[row_idx][
                column_idx] + h
            cost_function_added_h = f_multiclass(x_matrix.T, y_vector,
                initial_theta_copy)
            finite_difference = (cost_function_added_h - cost_function_original) / h
            print "finite_difference is", finite_difference
            lst_row.append(finite_difference)
        lst_finite_difference.append(lst_row)
    return gradient, lst_finite_difference, gradient - lst_finite_difference
```

As you can see, the comparison matrix values are approaching 0, which means the values in the gradient matrix and the finite difference matrix are very similar. For the gradient and finite difference matrices, you can find the values in the part_6d_file.txt

```
comparison matrix is [[ -0.00000000e+00  -1.35560006e-04  -0.00000000e+00  -0.00000000
    e+00]
 [ -0.00000000e+00   2.75801237e-07  -0.00000000e+00  -0.00000000e+00]
 [ -0.00000000e+00  -8.33791115e-05  -0.00000000e+00  -0.00000000e+00]
 ...,
 [ -0.00000000e+00  -8.91768193e-04  -0.00000000e+00  -0.00000000e+00]
 [ -0.00000000e+00   1.02980605e-04  -0.00000000e+00  -0.00000000e+00]
 [ -0.00000000e+00   7.37151759e-05  -0.00000000e+00  -0.00000000e+00]]
```

I used a very small h since the derivative is defined as h approaching 0. Picking a very small h makes sense in this case for getting a finite difference matrix that is as close as possible to the gradient matrix.

7 Problem 7

The following is the gradient descent code for part 7.:

I used the default alpha value of **alpha=0.00000005** to run the multi-class gradient descent. I chose this value because values greater than this make the code run for a long time while values less than this causes the gradient descent to terminate very quickly.

The labels I used for the classifier is

```
six_performer_dict = ({'Lorraine Bracco':     [0, 0, 0, 0, 0, 1],
                       'Peri Gilpin':         [0, 0, 0, 0, 1, 0],
                       'Angie Harmon':        [0, 0, 0, 1, 0, 0],
                       'Alec Baldwin':        [0, 0, 1, 0, 0, 0],
                       'Bill Hader':          [0, 1, 0, 0, 0, 0],
                       'Steve Carell':        [1, 0, 0, 0, 0, 0]})
```

The hypothesis should return an array where the index of its maximum element should correspond to the index of 1 in the corresponding actor's label. In other words, the classifier uses the index of the max element in the hypothesis array to identify the actors and actresses in act.

The performance of this multiclass classifier is the following:

```
The accuracy of the multiclass classifier on the validation set is 0.7
The accuracy of the multiclass classifier on the training set is 0.757142857143
```

You can generate and see these findings in **part_7_file.txt**

8 Problem 8

The following images are composed by the thetas matrix gotten from the multi-class gradient descent. As you can see, they look similar to the actor/actress label that they are corresponding to.

Baldwin.jpg 

Figure 12: Theta Images - Alec Baldwin

Harmon.jpg 

Figure 13: Theta Images - Angie Harmon

Hader.jpg 

Figure 14: Theta Images - Bill Hader

Bracco.jpg 

Figure 15: Theta Images - Lorraine Bracco

Gilpin.jpg 

Figure 16: Theta Images - Peri Gilpin

Carell.jpg 

Figure 17: Theta Images - Steve Carell