CSC411: Project 1
Face Recognition and Gender Classification with Regression

Due on Friday, February 3, 2017

Michelle Leung

February 23, 2018

Part 1

Dataset description The dataset consists of 2359 grayscale-equivalent 32×32-pixel of images of six different celebrities (three female and three male) obtained from running a Python script. The `rgb2gray()` function should be reapplied to the saved images when used again to restore the true grayscale-norm for dataset manipulation since saving the picture removes the division by 255 from the picture pixel values. The faces appear taken at various angles and in various environments, resulting in considerable variation in the appearance of the faces. Upon examining the dataset, most the bounding boxes are accurate at isolating the faces from each photo, however, due to the fact that the pictures are taken at different angles, many of the cropped-out faces can only roughly be aligned with each other. Occasionally, obstructions to parts of the face, such has hands, hair and glasses are worth noting.

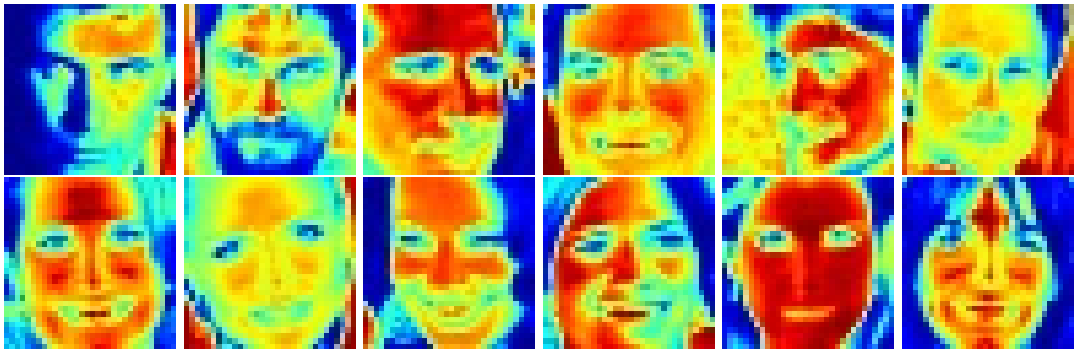

Figure 1: A selection of faces from the dataset and from the cropped faces.

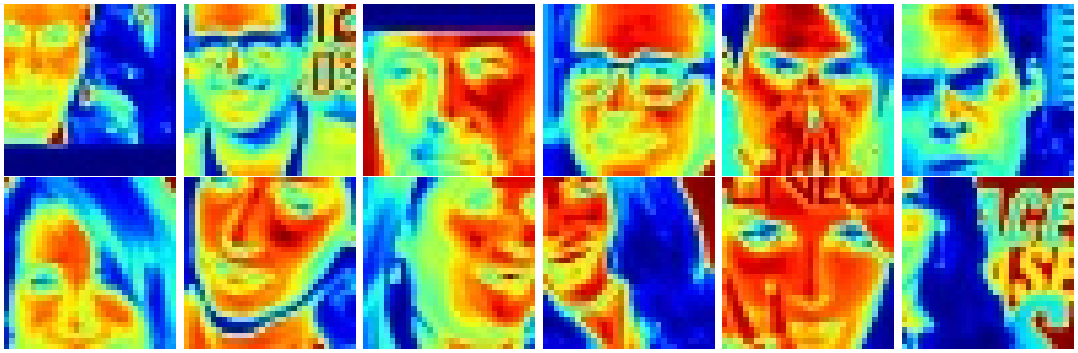

Figure 2: Examples of bad bounding boxes and partial obstructions

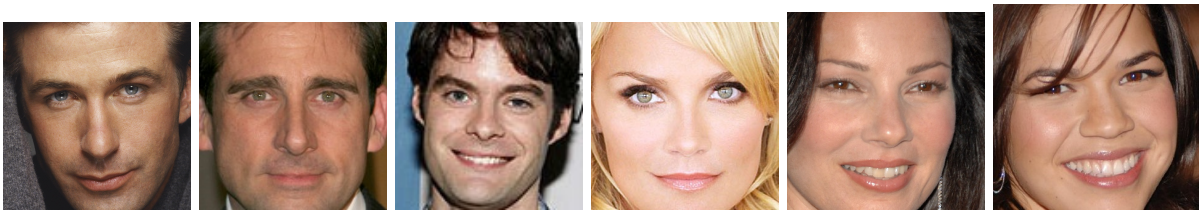

Figure 3: Examples of the cropped out faces

Part 2

Separating the Dataset
A list of integers from 0 to 200 was generated and then shuffled using `np.random.seed(0)` followed by `np.random.shuffle(list)`. The list of integers can easily be adjusted to accomodate for larger or smaller datasets, however 200 was chosen because each actor/actress had about 160 pictures. File names for each actor/actress would sequentially be generated as per `np.random.shuffle(list)`, and checked with the respective directories to see if the file exists. If the file exists, the filename will be appended to another list. Then, the first 100 files for each actor/actress in the randomized list of filenames would be separated into the trainingset, the next ten would go to the validation set, and another ten would be placed into the test set.
The following is the function used to generate the randomized, non-overlapping sets:

```
def part_2():
    """Randomizes the set of pictures in act and act_test and sorts them into
    training (100 pictures), validation(10 pictures), and test sets (10
    pictures) for each person in act, along with 10 pictures from each person
    in act_test
    """

    numlist = list(range(0,200))
    np.random.seed(0)
    np.random.shuffle(numlist)

    for person in act:
        current = []
        a = person.split()[1].lower()
        g = gender(a)
        for i in numlist:
            if os.path.exists(os.getcwd() +"/cropped_"+str(g)+"/"+str(a)+str(i)+".jpg"):
                current.append(str(a)+str(i)+".jpg")
        traininglist.append(current[:trainsize])
        validatinglist.append(current[trainsize+1:trainsize+1+valsize])
        testlist.append(current[trainsize+valsize+2:trainsize+valsize+testsize+2])

    for person in act_test:
        current = []
        a = person.split()[1].lower()
        g = gender(a)
        for i in numlist:
            if not os.path.exists(os.getcwd() +"/cropped_"+str(g)+"/"+str(a)+str(i)+".jpg"
                continue
            else:
                current.append(str(a)+str(i)+".jpg")
        acttestlist.append(current[30:30+valsize])
```

Part 3

Using Linear Regression to build a classifier to distinguish pictures of Bill Hader form Steve Carell
The cost function minimized was the sum of least squares function:

$$\sum((y - \theta^T x)^2) \tag{1}$$

```
def f(x, y, theta):
    """
    Cost function to be minimized.
    """
    x = vstack( (ones((1, x.shape[1])), x))
    return sum( (y - dot(theta.T,x)) ** 2)
```

The value of the cost function on the training set was 5.139589. The value of the cost function on the validation set was 1.753300.

On the training set, the classifier correctly identifies 92 percent of the images of Bill Hader and 93 percent for Steve Carell, with an overall performance of 92.5 percent.

On the validation set, the classifier correctly identifies 80 percent of the images of Bill Hader and 90 percent for Steve Carell, with an overall performance of 85 percent.

In order to make the system work, α was chosen to be 0.000001 (six decimal places) since experimentation with fewer decimal places would result in fewer iterations due to the α being too large and jumping over critical areas or resulting in a theta array of *nan*, however with a smaller α (seven decimal places), the gradient descent function would again result in fewer iterations due to the fact that the change in f is now smaller than the epsilon chosen.

```
    #Generate the Randomized Training, Validation, and Test Sets
    part_2()

    #Initialize Variables
    Xset = []
    Yset = []
    imflat1 = []
    alpha = 0.000001
    theta_i = zeros(1025)

    #Generate the X and Y arrays of the Training Set for Gradient Descent
    for person in training:
        a = person.split()[1].lower()
        g = gender(a)
        if a == "hader":
            index = 4
            ynum = 0
        elif a == "carell":
            index = 5
            ynum = 1

        i = 0
```

```
        while i < trainsize:
            for picture in traininglist[index]:
                im = Image.open(os.getcwd() + "/cropped_"+ str(g) + "/" + picture)
                im = np.asarray(im)
                im = rgb2gray(im)           #Reapply Grayscale
                imflat = im.flatten()       #Flattens the 32x32 pixels into 1x1024
                Xset.append(imflat)         #Add the picture data to X
                Yset.append(ynum)           #Add the person data to Y
                i = i + 1

    #Calculate the Gradient Descent
    theta = grad_descent(f, df, array(Xset).T , array(Yset).T, array(theta_i), alpha)

#Testing the Performance of the Classifier on the Training Set
    hright_t = 0
    cright_t = 0
    totalright_t = 0

    for person in training:
        a = person.split()[1].lower()
        g = gender(a)
        if a == "hader":
            index = 4
        elif a == "carell":
            index = 5

        i = 0
        while i < trainsize:
            for picture in traininglist[index]:
                im = Image.open(os.getcwd() + "/cropped_"+ str(g) + "/" + picture)
                im = np.asarray(im)
                im = rgb2gray(im)
                imflat = im.flatten()
                Xcheck = hstack((array(imflat),1))
                Guess = dot(array(theta),array(Xcheck).T)
                if Guess <0.5:
                    print "I think %s is Bill Hader" %picture
                    if str(a) == "hader":
                        hright_t = hright_t + 1
                        totalright_t = totalright_t + 1
                elif Guess >0.5:
                    print "I think %s is Steve Carell" %picture
                    if str(a) == "carell":
                        cright_t = cright_t + 1
                        totalright_t = totalright_t + 1
                i = i + 1
```

Part 4

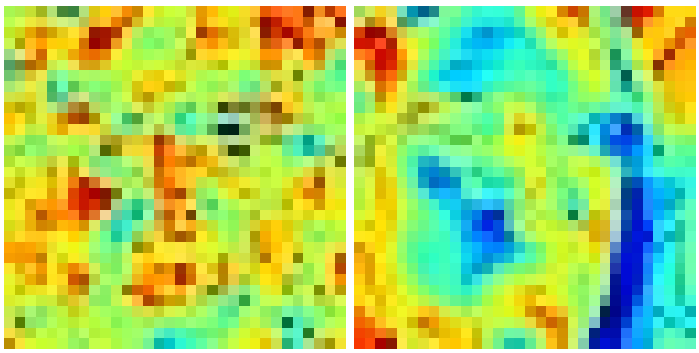

Figure 4: Left: Display of the θ's from training 100 Hader and 100 Carell images. Right: Display of the 's from training 2 Hader and 2 Carell images

Part 5

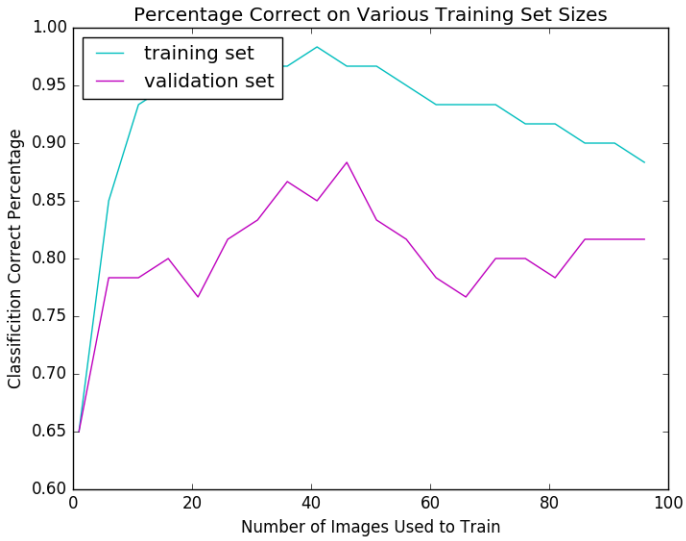

Figure 5: Performance of running gradient descent with one-hot encoding on various sizes of the training set

Performance of the classifer on act_test with 100 training images:
Correctly Identifies FEMALES: 63.3333333333
Correctly Identifies MALES: 86.6666666667
Percent Correctly Identified: 0.75

The above demonstrates over-fitting since the classifier trains too well on the faces in the training set and as a result, does not perform as well on the validation set or with the faces in the act_test, similar to if a student were to only train their knowledge of a course using only past exams, they will be less likely to perform as well on a new exam with a different stype and a variety of different questions.

Part 6

(a)

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

Expanding the summation over j terms in the inner most bracket gives us:

$$J(\theta) = \sum_i ((\theta_1^T x_1^{(i)} - y_1^{(i)}) + (\theta_2^T x_2^{(i)} - y_2^{(i)}) + ... + (\theta_{j-1}^T x_{j-1}^{(i)} - y_{j-1}^{(i)}) + (\theta_j x_j^{(i)} - y_j^{(i)}))^2 \tag{3}$$

Taking the derivative with respect to θ

$$\frac{dJ}{d\theta} = \frac{d}{d\theta} \sum_i ((\theta_1^T x_1^{(i)} - y_1^{(i)}) + (\theta_2 x_2^{(i)} - y_2^{(i)}) + ... + (\theta_{j-1} x_{j-1}^{(i)} - y_{j-1}^{(i)}) + (\theta_j x_j^{(i)} - y_j^{(i)}))^2 \tag{4}$$

$$\frac{dJ}{d\theta} = \sum_i \frac{d}{d\theta} ((\theta_{1^T} x_1^{(i)} - y_1^{(i)}) + (\theta_2 x_2^{(i)} - y_2^{(i)}) + ... + (\theta_{j-1} x_{j-1}^{(i)} - y_{j-1}^{(i)}) + (\theta_j x_j^{(i)} - y_j^{(i)}))^2 \tag{5}$$

$$\frac{dJ}{d\theta_k} = 2 \cdot \sum_i \frac{d}{d\theta_k} ((\theta_{1,k}^T x_1^{(i)} - y_1^{(i)}) + ... + (\theta_{j,k} x_j^{(i)} - y_k^{(i)})) \cdot \frac{d}{d\theta_k} (\theta_k^T x^{(i)}) \tag{6}$$

As discussed in lecture, everything except the k^{th} column will go to zero, and thus we can write:

$$\frac{dJ}{d\theta} = 2 \cdot \sum_i (\theta^T x^{(i)} - y^{(i)}) x^{(i)} \tag{7}$$

(b) Let:
θ be n x k
x be n x m
y be k x m

where k is the number of possible labels, m is the number of pictures in the dataset, and n is the number of pixels.

Using the result from (a) we can write:

$$\frac{dJ}{d\theta} = 2 \cdot \sum_i (\theta_{(n,k)}^T x_{(n,m)}^{(i)} - y_{(k,m)}^{(i)}) x_{(n,m)}^{(i)} \tag{8}$$

Taking the transpose of theta, we have:

$$\frac{dJ}{d\theta} = 2 \cdot \sum_i (\theta_{(k,n)} x_{(n,m)}^{(i)} - y_{(k,m)}^{(i)}) x_{(n,m)}^{(i)} \tag{9}$$

Now, the dimensions of the inner bracket when matrix multiplication and subtraction are carried through is kxm, and taking the transpose of that would result in a mxk matrix. Thus, using matrix multiplication axioms and carrying out the summation over i, this can be written as:

$$\frac{dJ}{d\theta} = 2 \cdot \sum_i (\theta_{(k,n)} x_{(n,m)}^{(i)} - y_{(k,m)}^{(i)})^T x_{(n,m)}^{(i)} \tag{10}$$

$$\frac{dJ}{d\theta} = 2 \cdot X_{n,m} (\theta_{(k,n)} X_{(n,m)} - Y_{(k,m)})^T \tag{11}$$

(c)

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

#Derivative of the Cost Function
def df(x, y, theta):
    x = vstack( (ones((1, x.shape[1])), x))
    return 2*(dot(x, (dot(theta.T,x)-y).T))

#Gradient Descent Function (modified to accommodate for 2D array input)
def grad_descent_onehot(f, df, x, y, init_t, alpha):
    EPS = 1e-5   #EPS = 10**(-5)
    prev_t = init_t-10*EPS
    t = init_t.copy()
    max_iter = 30000
    iter  = 0
    while norm(t - prev_t) >  EPS and iter < max_iter:
        prev_t = t.copy()
        t -= alpha*df(x, y, t)
        if iter % 500 == 0:
            print "Iter", iter
            print " f(x) = %.2f" %  f(x, y, t)
            print "Gradient: ", df(x, y, t), "\n"
        iter += 1
    return t
```

```
def part_6():

    part_2()            #Generate the randomized sets and Initialize Variables
    Xset = []
    Yset = []
    imflat1 = []
    alpha = 0.000001
    theta_i = zeros((1025,6))

    for person in act:
        a = person.split()[1].lower()
        g = gender(a)

        if a == "drescher":
            ynum = [1, 0, 0, 0, 0, 0]
        elif a == "ferrera":
            ynum = [0, 1, 0, 0, 0, 0]
        elif a == "chenoweth":
            ynum = [0, 0, 1, 0, 0, 0]
        elif a == "baldwin":
            ynum = [0, 0, 0, 1, 0, 0]
        elif a == "hader":
            ynum = [0, 0, 0, 0, 1, 0]
        elif a == "carell":
            ynum = [0, 0, 0, 0, 0, 1]
        #while i < 100:
        #for filename in os.listdir(os.getcwd() +"/" + path):

        i = 0
        j = index_act(a)
        while i < trainsize:
            for picture in traininglist[j]:
                im = Image.open(os.getcwd() + "/cropped_"+ str(g) + "/" + picture)
                im = np.asarray(im)
                im = rgb2gray(im)
                imflat = im.flatten()
                Xset.append(imflat)
                Yset.append(ynum)
                i = i + 1

#Calculate the Gradient Descent using One-Hot Encoding
    theta = []
    theta = grad_descent_onehot(f, df, array(Xset).T , array(Yset).T, array(theta_i),
        alpha)
```

(d) Continuing from the code in in (c) to calculate and compare Finite Differences ("dcompute") vs Gradient Descent:

```
for i in range (1, 1024, 100):
    for j in range (0, 6):
        h = 0.000001
        theta_copy = theta.copy()
        theta_copy[i][j] = theta_copy[i][j]+h
        dcompute = (f(array(Xset).T, array(Yset).T, theta_copy) -
            f(array(Xset).T, array(Yset).T, theta))/h
        print str(i)+", "+str(j)
        print (str(dcompute) + "\tFinite Differences")
        print (str(df(array(Xset).T, array(Yset).T, theta)[i][j]) +
            "\tGradient Function")
```

Sample outputs of the Finite Differences vs Gradient Descent (indices of the theta matrix displayed):

801, 1
0.0185871158465 Finite Differences
0.0184434373414 Gradient Function

801, 2
0.137270163236 Finite Differences
0.137126482798 Gradient Function

801, 3
-0.763509760304 Finite Differences
-0.763653447886 Gradient Function

801, 4
-0.276544867006 Finite Differences
-0.276688550382 Gradient Function

801, 5
0.591014696738 Finite Differences
0.590871009056 Gradient Function

901, 0
0.246773296908 Finite Differences
0.246590839052 Gradient Function

901, 1
-0.182649287694 Finite Differences
-0.18283174587 Gradient Function

901, 2
-0.055770868812 Finite Differences
-0.0559533266035 Gradient Function

Part 7

On the training set, the classifier correctly identifies 76.3 percent of the overall images in the dataset. The following is a breakdown of the classifier's performance for each actor/actress:

Correctly Identifies Drescher: 0.84
Correctly Identifies Ferrera: 0.68
Correctly Identifies Chenoweth: 0.92
Correctly Identifies Baldwin: 0.6
Correctly Identifies Hader: 0.71
Correctly Identifies Carell: 0.83

On the validation set, the classifier correctly identifies 68.33 percent of the overall images in the dataset. The following is a breakdown of the classifier's performance for each actor/actress:

Correctly Identifies Drescher: 0.8
Correctly Identifies Ferrera: 0.8
Correctly Identifies Chenoweth: 0.4
Correctly Identifies Baldwin: 0.6
Correctly Identifies Hader: 0.6
Correctly Identifies Carell: 0.9

Overfitting was a big issue, as demonstrated by the Chenoweth images: in training, the classifier correctly identifies 92 percent of the Chenoweth pictures, however in the validation set, it was only able to identify 40 percent of her pictures. Here, the value of α is 0.000001. Various values for the maximum number of iterations run and α were experimented with, but no visible improvements were observed, however with different datasets (ie sequentially ordered index) the performance of the classifier was noted to be much higher, averaging at about 92 percent.

Part 8

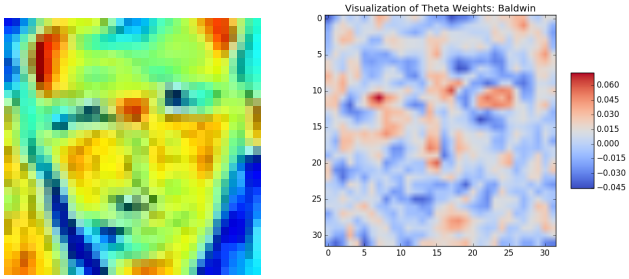

Figure 6: Baldwin: 2 training images on the left, 100 on the right

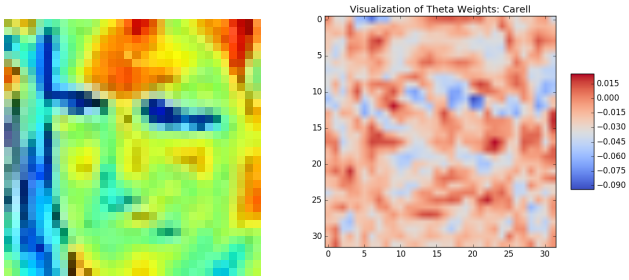

Figure 7: Carell: 2 training images on the left, 100 on the right

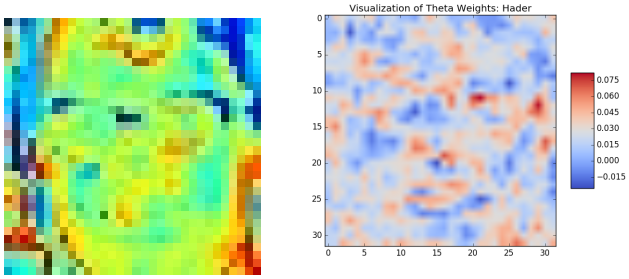

Figure 8: Hader: 2 training images on the left, 100 on the right

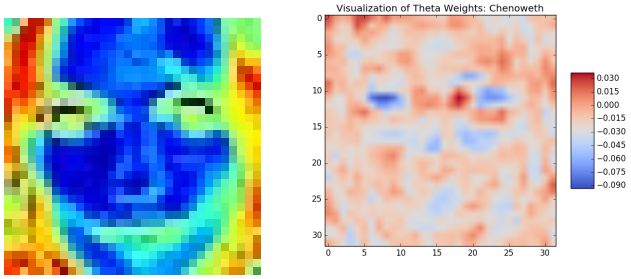

Figure 9: Chenoweth: 2 training images on the left, 100 on the right

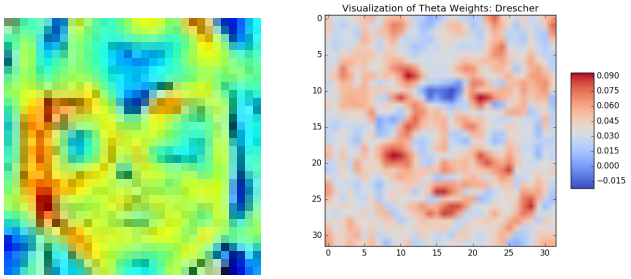

Figure 10: Drescher: 2 training images on the left, 100 on the right

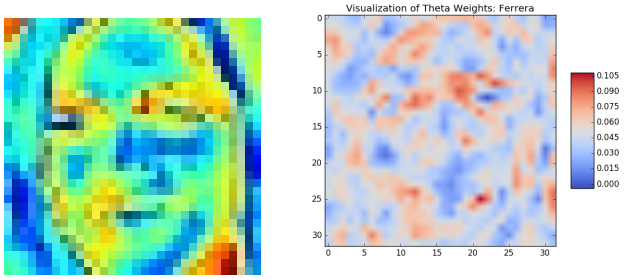

Figure 11: Ferrera: 2 training images on the left, 100 on the right