CSC411: Assignment #1(run on macOS10.13.1)

Due on Monday, January 29, 2018

Yufeng Li

January 30, 2018

Part 1

The number of pictures I obtained are as follows:

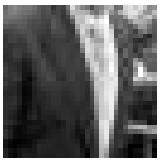
```
uncropped:
vartan 195
radcliffe 214
gilpin 189
drescher 239
hader 278
ferrera 246
chenoweth 257
bracco 216
butler 198
baldwin 266
carell 264
harmon 239

cropped:
vartan 189
radcliffe 196
gilpin 171
drescher 221
hader 236
ferrera 233
chenoweth 228
bracco 184
butler 184
baldwin 232
carell 236
harmon 195
```

On the lefthand side are the uncropped images(resized to better compare,the original size is bigger),on the righthand side are the cropped and been switched to greyscale.(32*32)

(a)

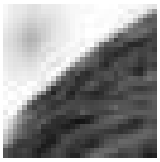
(b)

(c)



(d)

(e)



(f)

Some of the bounding boxes are not accurate thus creating uncertainty to our training process. However, some cropped picture can align with the uncropped counterpart.

On the lefthand side are the uncropped images(resized to better compare,the original size is bigger),on the righthand side are the cropped and been switched to greyscale.(32*32) *These are the pictures where the*

(a)

(b)

(c)

(d)

(e)

(f)

bounding boxes are actually accurate thus they can best represent their faces.

Part 2

Separating cropped pictures
The algorithm I used to separate:

```
def group():
    files = os.listdir("cropped")
    names_set = {}
    for file in files:
        name = ''.join([i for i in file if not i.isdigit()]).split('.')[0]
        if name not in names_set:
            names_set[name] = [file]
        else:
            names_set[name].append(file)
    training = []
    validating = []
    testing = []
    count = 0
    for name, files in names_set.items():
        for f in files:
            if count < 70:
                training.append(f)
                count += 1
            if 70 <= count < 80:
                validating.append(f)
                count += 1
            if 80 <= count < 90:
                testing.append(f)
                count += 1
        count = 0
    return names_set, training, validating, testing
```

At first, it reads "cropped" folder and parse each filename to get its actor name. Then I use that name as a key to my names_set dictionary followed by all its related filenames list. I used this dictionary to generate my training, validating and testing samples. In practice I use indexes of dictionary's value (which is the filename list) to access each sample set of a same actor.

Part 3

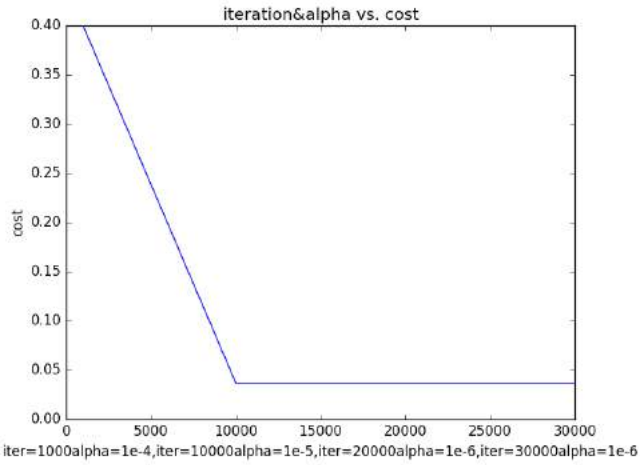
Building classifier to distinguish pictures of Baldwin from that of Carell
The cost function which I want to minimize is the quadratic loss, which is:(x is sample,y is target, theta is theta)

```
sum((y - dot(theta.T, x)) ** 2)
```

which then have the result of gradient of loss function like this:

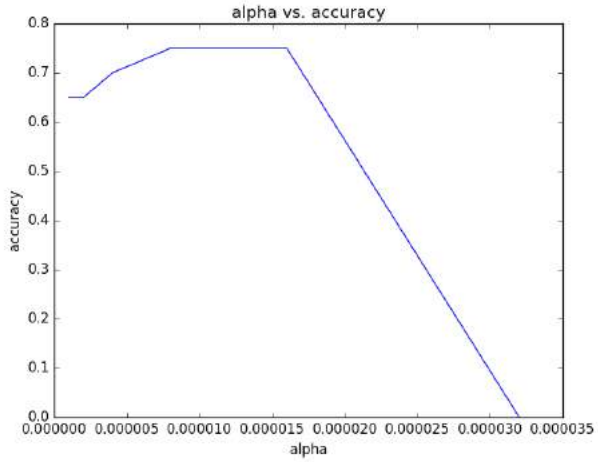
```
-2 * sum((y - dot(theta.T, x)) * x, 1)
```

To make the gradient descend works, I test out some max iterations and alpha first based on the tutorial code given, I've already strip away those result where it gets NaN or runtime warnings to make the graph looks more reasonable.



(a) Figure 1: Plot of cost function w.r.t. different alpha and max iteration test

Figure 3a demonstrates the effects of decreasing alpha and increase of max iterations. I use following max iteration and alpha: maxiter=1000 alpha=1e-4, maxiter=10000 alpha=1e-5, maxiter=20000 alpha=1e-6, maxiter=30000 alpha=1e-6. As we can see here there's a noticeable decrease in cost when max iteration increase and alpha decreases. The reason why I pick max iteration=30000 and alpha=1e-6 because it produce less error in coding later.



(b) Figure 2: Plot of accuracy rate w.r.t. different alpha and fixed max iteration=30000

Figure 3b demonstrates the effects of decreasing alpha and accuracy varies. This is tested on training set of both actors. As we can see here there's a best range of alphas, I expected that range to have a fluctuated accuracy rate, however for the sake of efficiency i only picked 6 points otherwise it takes too much time to generate. There's a trade off of alpha which is between efficiency and accuracy.

The result
The loss using thetas from running training set and validating set is as follows:

```
Loss for training:  46.3718173877
Loss for validating:  15.3119164676
```

The reason why validating has a smaller loss over training is because the sample size is smaller and possible more fined cropped images than training set. I used the training set's theta to compute the training set and validating set accuracy rate. Which is:

```
accuracy for training set:0.928571428571
accuracy for validating set:  0.65
```

Code I use to get prediction:(where pic is flattened picture)

```
result = dot(theta1.T, pic)
```

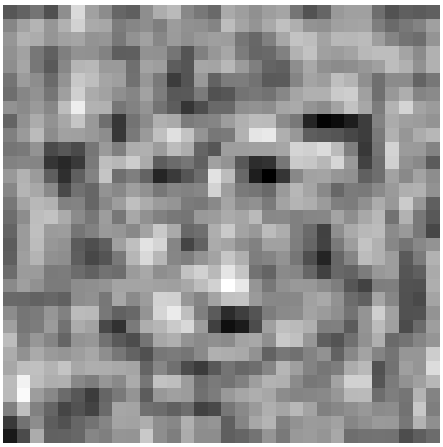
Code bellow is how I compute the result:

```
result_t = 0
for i in range(70):
    pic1 = imread('cropped/' + baldwin[i]).flatten() / 255.
    pic1 = np.insert(pic1, 0, 1)
    pic1 = pic1.T
    result1 = dot(theta1.T, pic1)
    pic1 = imread('cropped/' + carell[i]).flatten() / 255.
    pic1 = np.insert(pic1, 0, 1)
    pic1 = pic1.T
    result2 = dot(theta1.T, pic1)
    if result1 > 0:
        result_t += 1
    if result2 < 0:
        result_t += 1
accuracy = result_t / 140.
```

The number in **for i in range(70)** is the number of pictures of each actor, I compute **dot(theta.T,picture)** each time and see if the result is bigger or smaller than 0 (since I label them 1 and -1) if it is bigger than 0 and is Baldwin then it is correct, if it is smaller than 0 and is Carell then it is correct too. Finally I can get the accuracy using result/140.

Part 4

(a)Visualizing theta with different sample sizes
The theta pictures are generated with full training set and training set with each 2 pictures of a person. Since I use scipy.misc.imsave, the theta image appear to be gray.

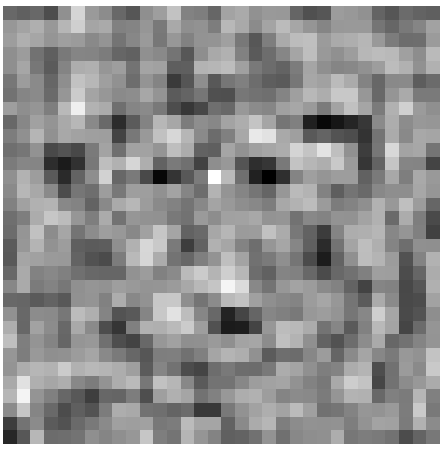


(a) full training set thetas

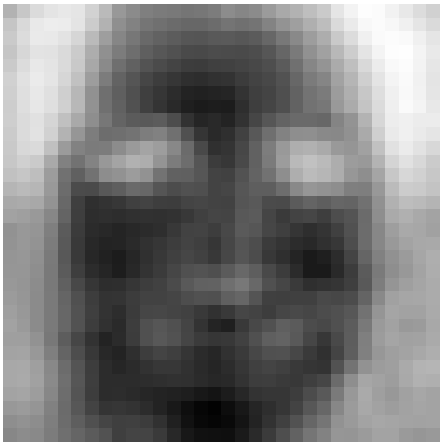
(b) partial training set thetas

We can observe that the thetas get from partial training set which only contains 2 pictures from each person has a clearer face on it. This demonstrates overfit. If we use full training set to get thetas, it will generalize all data so it will look less likely to see a face.

(b)Visualizing theta with different max iterations given full training set



(a) Figure1:max iterations=30000



(b) Figure2:max iterations=10

I change different max iterations to control how deep will the gradient descend goes. If it stops early result in Figure2 which looks like a face. With more iterations, the gradient descend will go deeper and created a less face-like image like Figure1. So we can conclude that with more iterations the gradient descend will get more accurate and generalized thetas.

Part 5

Classifying actors as male or female
The actors I built classifier on is:

```
act = ['Lorraine Bracco', 'Peri Gilpin', 'Angie Harmon',
'Alec Baldwin', 'Bill Hader', 'Steve Carell']
```

And the final performance test is from these actors:

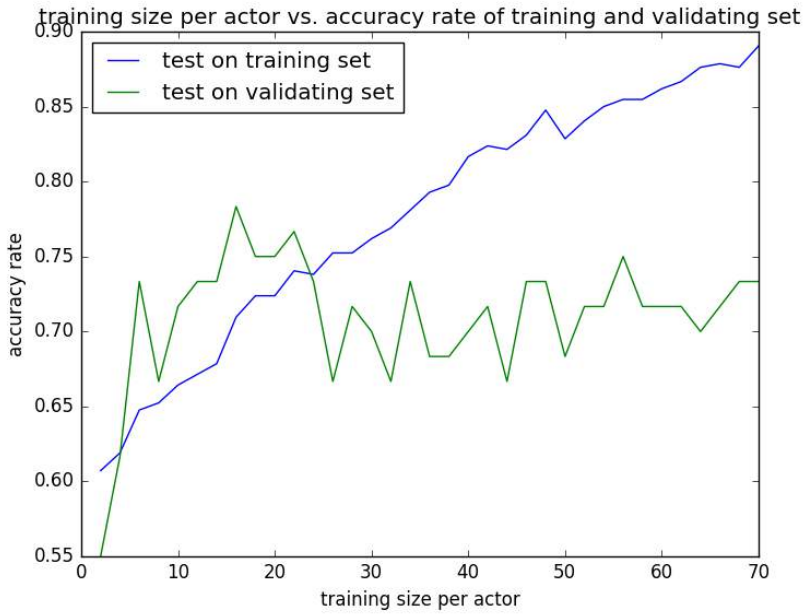
```
other_act = ['Daniel Radcliffe', 'Gerard Butler', 'Michael Vartan',
'Kristin Chenoweth', 'Fran Drescher','America Ferrera']
```

Following the final result:

```
performance on this 6 actors: 0.890476190476
performance on other 6 actors:  0.640476190476
```

We can see that the performance on this 6 actors which built this classifier is much higher than the performance tested on other 6 actors. This demonstrate an over fitting which is because this classifier is optimized and generalized and best matches this 6 actors. However it is to well designed that if it test gender recognition on other people, it will perform worse.

*Performance on training set and validating set of actors in **act** versus classifiers built from different sizes of training set*

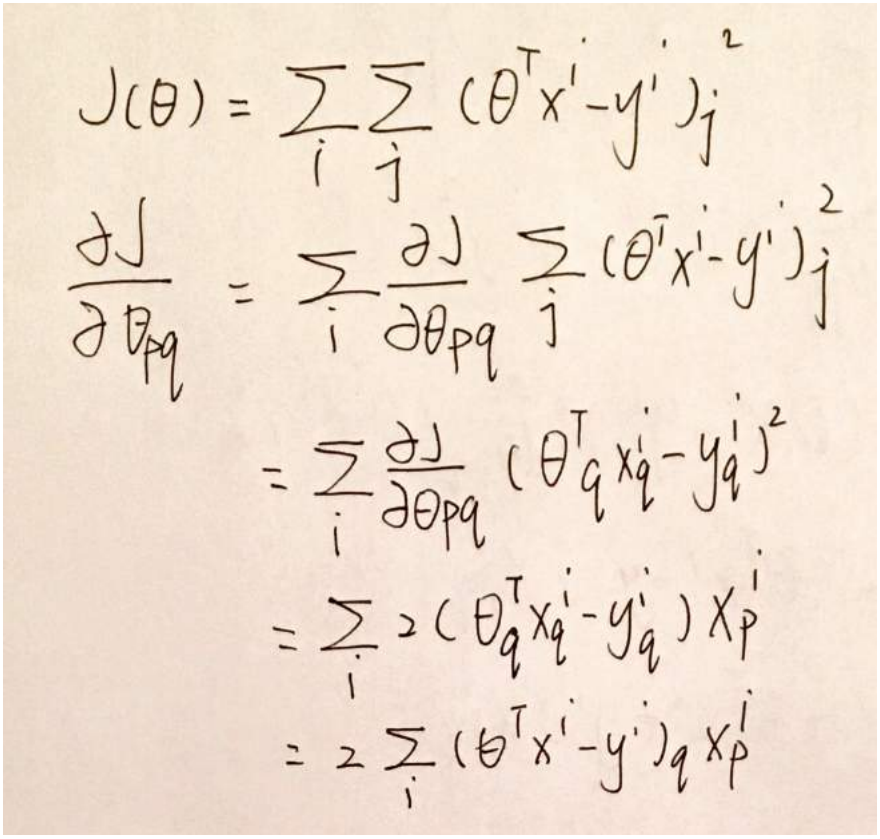


(a) Figure1:performance vs. set size per person

We can observe that the performance on training set is increasing as the training set size is increasing, however, as for validating set, after reaching 70% it starts to fluctuated and won't increase anymore. Indeed, increasing training size will result in increase in accuracy rate, however, the effect will not that great due to overfitting. The difference between after 40 pictures per person is because the thetas are so well fitted into training set itself but it performs unchanged for validating set.

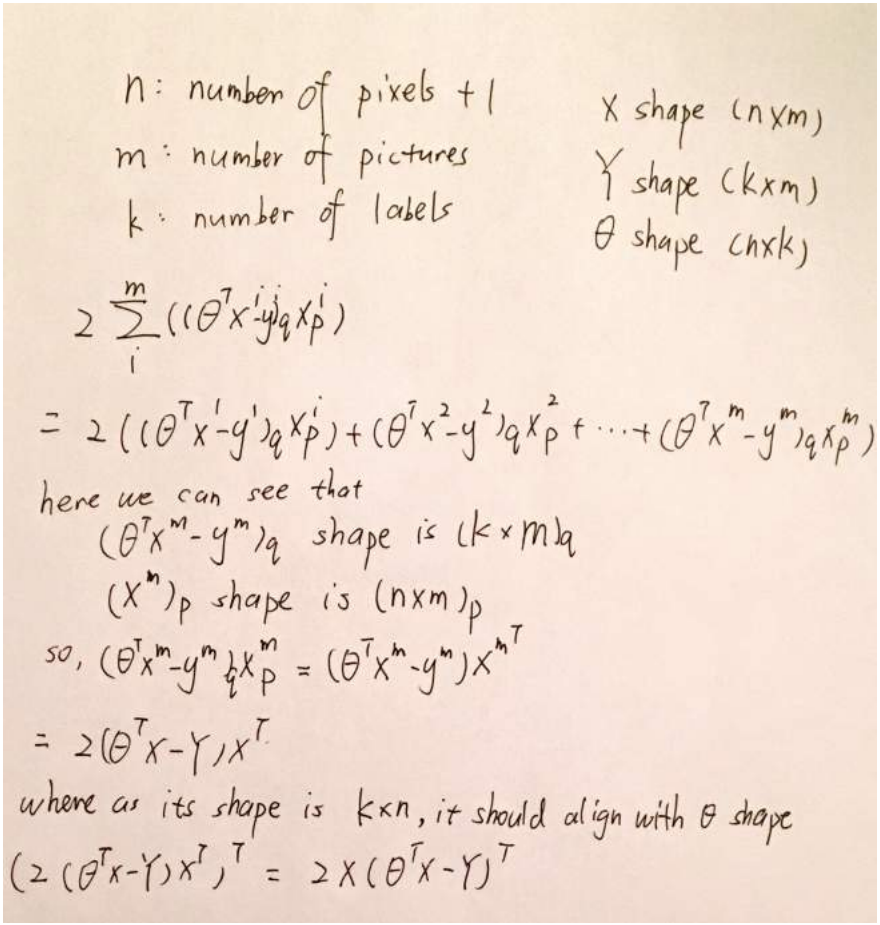
Part 6

(a)compute gradient



(a) Figure1:compute gradient

(b)proof



(a) Figure1:proof

(c)code for cost function and gradient
Code I use to compute cost:

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

Code I use to compute gradient:

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

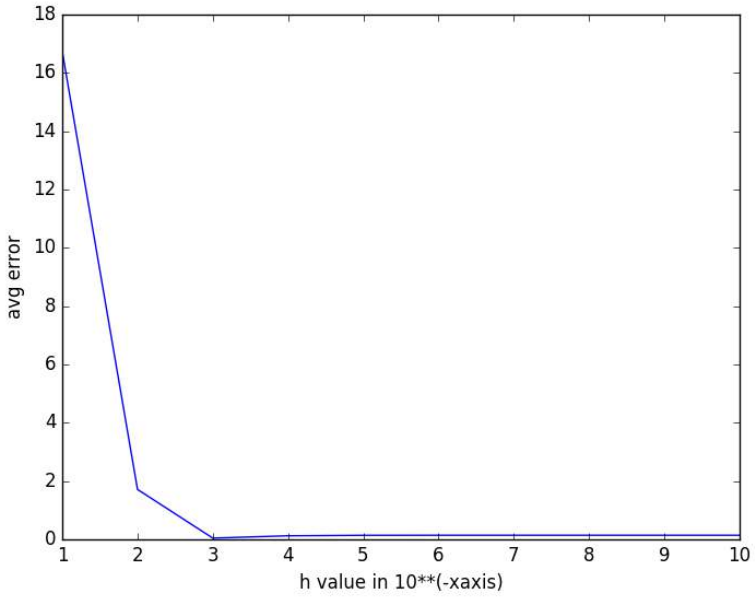
Suppose I have N pictures and M labels, here I use x which is the flatten picture stacks with shape (1025,N),each column represents a picture, y which is the target labels with shape (M,N), theta with shape (1025,M)

(d)experimenting h

Code I use to get finite differences and compare to the gradient function we produce is:

```
def finite_difference(x, y, theta, h):
    origin_t = multi_f(x, y, theta)
    theta = theta + np.full((theta.shape[0], theta.shape[1]), h)
    after_t = multi_f(x, y, theta)
    finite_diff = (after_t - origin_t)/h
    total_error = sum(finite_diff - multi_f(x, y, theta))
    return abs(total_error)/(1025*6*1.0)
```

I first get an original cost function and then add **h** to every theta array's entry to produce a cost function with augmented theta. I subtract them and divide it by h to get the finite difference. Then I use finite difference to subtract the derivative we get from our gradient function **multi_df** and calculate the average error.



(a) Figure1:average error vs. h

We can observe that with a rather small h, the average error is far smaller than a big h, but not too small or else it will cause a slightly more error. It also suggests that our gradient function is correct.

Part 7

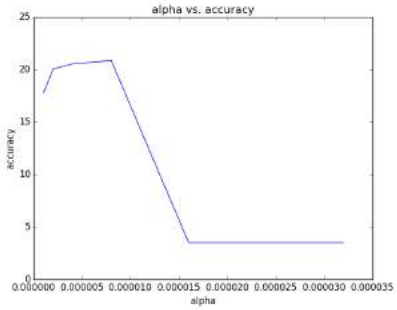
The gradient descend will perform on following actors:

```
act = ['Lorraine Bracco', 'Peri Gilpin', 'Angie Harmon',
'Alec Baldwin', 'Bill Hader', 'Steve Carell']
```

And I labeled them:

```
act_labels = {'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]}
```

First I tested the right alpha, still, I picked a fairly large max iteration



(a) Figure1:max iterations fixed to 30000

The graph shows some similarity with the previous alpha vs. accuracy graph in part3 and the alpha selection is alright for this question too. So I choose the same alpha and max iteration as before.
Code I use to compute the output and accuracy:

```
result_t = 0
for name, files in names_set.items():
    if name in act_nickname:
        for i in files[:70]:
            pic1 = imread('cropped/' + i).flatten() / 255.
            pic1 = np.insert(pic1, 0, 1)
            pic1 = pic1.T
            result1 = dot(theta.T, pic1)
            if np.argmax(result1) == act_labels[name].index(1):
                result_t += 1
accuracy=result_t/420.
```

Code I use to get predicted label is:

```
result = dot(theta.T, pic1)
```

Suppose result for baldwin may look like [0.367739812 0.452342234 0.56352423 0.234134134 0.87981623 0.412423412]

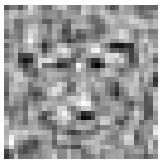
Here I use **np.argmax(result1) ==act_labels[name].index(1)** to compare the prediction and the target. What it does is get the maximum number's index in the prediction and to see if it matches the number 1's index in its label. In this case, baldwin's label is [0, 0, 0, 1, 0, 0] and its 1's index is 3 and the maximum number's index in the prediction is 4, thus this picture is not a correct guess.
The result:

```
accuracy for training_set:  0.847619047619
accuracy for validation set:  0.516666666667
```

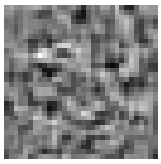
We can observe that there's a decreasing in accuracy compared to just distinguish one person from others. This is because the thetas are more look like a face and thus more optimized for the training set. This will cause a not so generalized theta and decrease in accuracy in validation.

Part 8

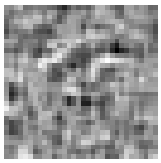
Visualizing thetas
The theta pictures are generated using scipy.misc.imsave so the theta image appear to be gray.



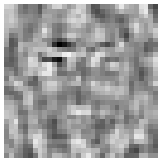
(a) baldwin thetas



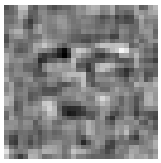
(b) gilpin thetas



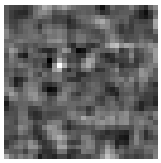
(c) hader thetas



(d) bracco thetas



(e) carell thetas



(f) harmon thetas

The algorithm works and produces 6 thetas for each person. The thetas have some resemblance with the actors which is perfectly fine because the prediction is basically **dot(theta.T,picture)**, with a clearer face, the dot product is bigger at the same index its label has 1.