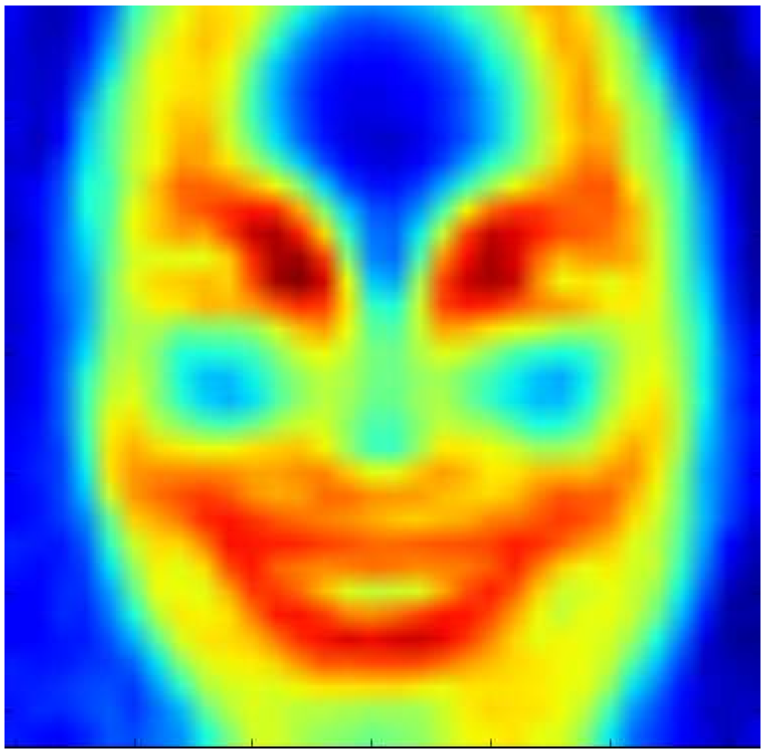
CSC411: Face Recognition and Gender Classification with Regression

Due on Wednesday, February 1, 2016


Funny image generated with a 16% accuracy on test set

Angus Fung

February 3, 2017

Problem 1

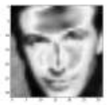
Dataset description

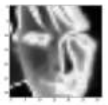
The dataset of faces comprises of the following actors and actresses: `['Fran Drescher', 'America Ferrera', 'Kristin Chenoweth', 'Alec Baldwin', 'Bill Hader', 'Steve Carell']`. The images seem to have been retrieved from random websites, and as such, there is much diversity in the images in terms of the environment (e.g lightning, scenery, etc.) and expression of the celebrities. This diversity in the dataset may provide a strong training set for the algorithm. Three examples illustrating this can be seen in Fig. 1(a)-(c). The corresponding cropped images are below them.

(a) Baldwin 1 (Uncropped)

(b) Baldwin 2 (Uncropped)

(c) Baldwin 3


(d) Baldwin 1 (Cropped)


(e) Baldwin 2 (Cropped)

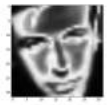
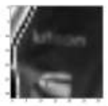
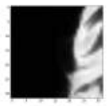

(f) Baldwin 3 (Cropped)

Figure 1: Cropped vs. Uncropped

Many issues arose downloading the images including (1) dead links [1] (2) missing images [2], (3) blank images [3], and (4) gray-scale downloaded pictures [4]. The resolution of these issues required using `try except` statements, and missing or blank images were removed from the dataset through code. For the most part, the bounding boxes were accurate, but there were incorrect bounding boxes from time to time, Fig.2 (a)-(b). Furthermore, some of the downloaded pictures were not celebrities but unidentified objects, Fig.2 (c). These errors in the dataset would negatively influence the results of the test set, so they were removed from the dataset by inspection.


(a) Chenoweth Bounding Box Error


(b) Chenoweth Bounding Box Error

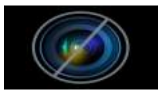

(c) Unidentified object

Figure 2: Errors in dataset

It is very unlikely cropped-out images can be precisely aligned with each other if we were to use the bounding boxes provided, and the images are randomly generated. That's because while the images can be rescaled, they must all have the same dimensionality, that is, $(32, 32)$. Therefore, within a dataset of the same dimensionality, people have different facial structure (some people have longer faces, etc.) and some faces are slanted due to their pose. This makes it very implausible that images can be aligned. If the constraint on the choice of bounding boxes is relaxed and to the discretion of the coder, it *may* possible to align the cropped images, but may not be time feasible. Images may have to be rotated, perfectly chosen bounding boxes must be found, and each image may have to be rescaled in different ratios (but with the same end

[1]Alec Baldwin 3209 1862
[2]Alec Baldwin 3214 1867
[3]Steve Carell 104735 53801
[4]Alec Baldwin 3246 1890

dimensionality) in order to ensure that the images align, irrespective of facial structure. As we will see, even with imperfect facial alignment, linear regression yields satisfactory results.

Problem 2

Separation of dataset.
At first, the training set, validation set, and the test set were formed by manually dragging the pictures into different folders. For example, the first 100 pictures of Alec Baldwin were dragged into the training set, the next 10 into the validation set, and the subsequent 10 into the test set. As such, there are no overlaps in the datasets. Initially, randomness was not used here to separate the pictures as the pictures in its current state are more or less random. For even if the pictures themselves were generated from the top searches of a search engine, the origins thereof (e.g articles, blogs, etc.) would all have different and varied contexts, and therefore sufficiently random.

However, in compliance with the assignment stipulations, a function `nameset` was created. `nameset` takes as parameter the file `cropped` and is used to separate it into three directories using randomness: training set, validation set, and testing set. Randomness here is seeded at 0, `random.seed(0)`, so that the results herein are fully reproducible. The usage of `nameset` is explained in the Appendix.

Problem 3

Use linear regression to build a classifier.
To distinguish between pictures of Bill Hader and Steve Carell, the problem of classification was reframed as a regression problem, where the decision boundary between the two actors is a linear equation lying on a 1025 dimensional hyperplane. Since the images themselves were 32 by 32, they had to be flattened into column vectors of size 1024, $\mathbf{x}^{(i)} = (x_1^{(i)}, x_2^{(i)}, ..., x_{1024}^{(i)})^T$. Including the bias term, $x_0^{(i)} = 1$, $\mathbf{x}^{(i)} = (1, x_1^{(i)}, x_2^{(i)}, ..., x_{1024}^{(i)})^T$.
The cost function that was minimized:

$$J(\theta_0, \theta_1, ..., \theta_{1024}) = \frac{1}{2m} \sum_{i=1}^{200} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2, \tag{1}$$

where $m = 200$, $\theta = (\theta_0, \theta_1, ..., \theta_{1024})$ are the parameters that minimizes J on the hyperplane and $\mathbf{y}$ contains the exact labels of the training set. For this part, Hader corresponds to $y = 1$ and Carell corresponds to $y = 0$; therefore, the decision boundary is of the form $\theta^T \mathbf{x} = 0.5$. For $\theta^T \mathbf{x} > 0.5$, $y = 1$, and for $\mathbf{x} < 0.5$, $y = 1$. This was minimized by gradient descent[5]. The implementation of the cost function and its derivative, in code, is the same as in the spiral galaxy linear regression example. After training with the training set, the following performance scores were obtained:

	f_{min}	Hader (out of 10)	Carell (out of 10)	Percentage (%)
Training Set	0.247	10	10	100
Validation Set	0.247	8	7	75
Test Set	0.247	9	7	80

Since there weren't any hyperparameters, the validation set functions exactly like the test set in this project. These results make sense - the training set did extremely well due to over-fitting. The parameters θ were ultimately chosen produce the best results for the validation and test set. The procedure and choice of parameters will be elaborated later in the report.

[5]Michael Guerzhoy's implementation

The code implementation of equation (1) is formatted slightly different on python.

$$J(\theta_0, \theta_1, ..., \theta_{1024}) = \text{sum}\left(\boldsymbol{\theta}^T\mathbf{X} - \mathbf{y}\right)^2, \tag{2}$$

where $\mathbf{X} = [x_{ij}] \in \mathbb{R}^{n\times m}$, where $x_{ij} = x_j^{(i)}$, and sum() adds up the all the elements in the column vector. More specifically, each row of the column vector,

$$\left(\boldsymbol{\theta}^T\mathbf{X} - \mathbf{y}\right)^2, \tag{3}$$

corresponds to one summation index of equation (1). (i.e row i in (3) is summation index i in (1).) Note that for (3), the square is an *element-wise square*, which is how Python handles squaring a vector or matrix. (Not to be confused with $(\boldsymbol{\theta}^T\mathbf{X} - \mathbf{y})^2 = (\boldsymbol{\theta}^T\mathbf{X} - \mathbf{y})^T(\boldsymbol{\theta}^T\mathbf{X} - \mathbf{y})$.)

The $\mathbf{X}$ matrix is generated by looping through all the picture files while simultaneously generating the correct labels of $\mathbf{y}$. Below are the implementations of $h = \theta^T x$ and the classifier function `testset`.

Code

```
def h(x, theta):
    sum=0
    x = hstack((1,x))
    for i in range(len(theta)):
        sum += x[i]*theta[i]
    return sum
    act = ["hader", "carell"]

def testset(filename):
    hader = 0
    carell = 0
    for a in act:
        for picture in os.listdir(filename):
            if a in picture:
                im = imread(filename + "/" +picture, True)
                im = reshape(im, (1024))
                sum=h(im,theta[0])
                print(sum, picture)
                if (sum>0.5) & ("hader" in picture):
                    hader +=1
                elif (sum<0.5) & ("carell" in picture):
                    carell +=1
    return (hader, carell)
```

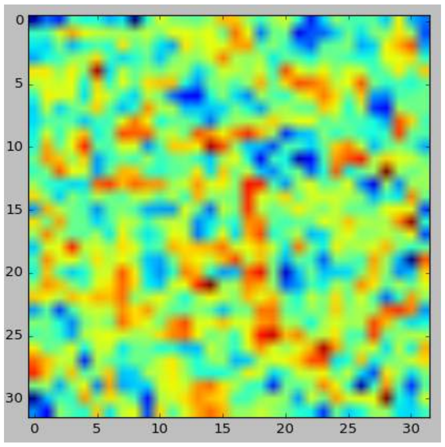
In order to find the optimal theta's using gradient descent, four key parameters needed to be varied: (1) α (2) initial conditions (3) ϵ, the threshold and (4), number of iterations. Initially, the initial condition was set to $[0, ..., 0] \in \mathbb{R}^{1025}$, the threshold at 10^{-5}, and $\alpha = 10^{-6}$. Upon initial testing, it was found out that gradient descent diverged, almost immediately. This most likely meant that the step, α was too large and the function was overshooting, passing the minimum. Hence, α was adjusted to 10^{-10} which led to steady function minimization. This value was chosen by decreasing α until it converged; however, too small an α would also lead to divergence.

Upon testing the resulting optimized θ, the results were disappointing: less than 50% accuracy on the training, validation, and test sets. It was noticed that the threshold being too high and the number of iterations being too low caused the optimization to exit prematurely before a minimum was actually achieved. These values were adjusted until the function converged to a minimum. Testing these θ parameters once again yielded a 100% accuracy rate for the training set. This was because of overfitting. The initial condition was fiddled around and tested until optimal results for the training and validation sets were achieved, as seen in the table above (with initial condition -0.04). Although randomized initial theta was considered, along with code that would run gradient descent n times with randomized theta's and performance testing, the initial theta consisting of only -0.04 worked reasonably well. While these sets performed decently, due to a lack of hyperparameters, it may very likely be that the test and validation sets were overfitted as well.

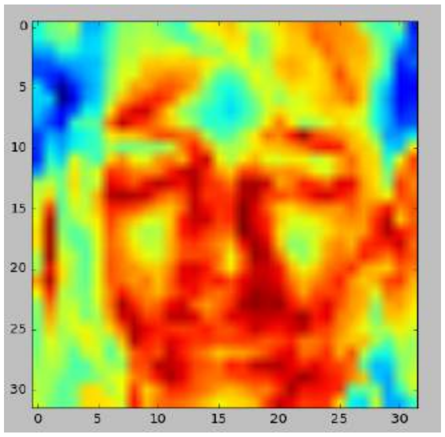
Problem 4

Image of $(\theta_1, ..., \theta_{1024})$

The image of $(\theta_1, ..., \theta_{1024})$ as generated by the training set of 100 images of Hader and Carell, and 2 images of Hader and Carell, are shown in Fig. 3(a) and 3(b), respectively.



(a) training set of 100 images per actor



(b) training set of 2 images per actor

Figure 3: Image of $(\theta_1, ..., \theta_{1024})$

Problem 5

Demonstrating Overfitting

```
act =['Fran Drescher', 'America Ferrera', 'Kristin Chenoweth', 'Alec Baldwin',
'Bill Hader', 'Steve Carell']
```

Below, in Fig 4. and 5., are the table and graph of the performance of the validation set and training set on `act`. The specifics of the parameters used in gradient descent is outlined in the Appendix. These parameters (including initial conditions) were kept the same for all training sizes, from 10 images of each actor in `act` up to 100 images of each. This is so that an accurate measure of overfitting is seen. If that weren't the case and the algorithm was trained with different parameters for different training sizes, bias may be introduced to the performance, as each sizes would be trained to different extents. "Size" refers to the *number* of images of each actor in the training set, and VS and TS refer to the validation set and training set, respectively.

Size	Female (VS)	Male (VS)	Percentage (%)	Female (TS)	Male (TS)	Percentage (%)
10	15/30	21/30	60	30/30	30/30	100
20	14/30	19/30	55	60/60	60/60	100
30	24/30	19/30	72	90/90	90/90	100
40	25/30	16/30	68	120/120	120/120	100
50	27/30	22/30	82	150/150	150/150	100
60	26/30	22/30	80	180/180	180/180	100
70	25/30	21/30	77	207/210	209/210	99
80	24/30	23/30	78	238/240	239/240	99
90	25/30	22/30	78	267/270	267/270	99
100	24/30	22/30	77	295/300	297/300	99

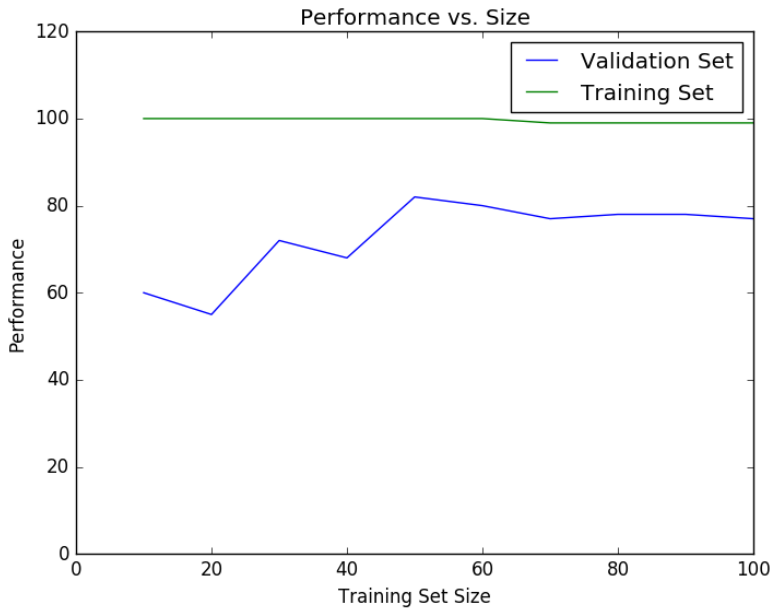


Figure 4: Performance vs. Size

From Fig. 4, a few features are immediately apparent. The performance of the training set is consistently at 100% for the sizes less than 60, and then around 99% for larger sizes. This would be expected, as it would be difficult to obtain a hyperplane that would perfectly classify such a large and diverse collection of images. This high percentage results from overfitting. The validation set however, is much more interesting. It starts off with poor and inconsistent performance, increasing to a maximum performance at size 50, before leveling off and slowly declining in performance.

At small sizes, the training set is also small. For sizes 10, and 20, the training set comprises of 60 and 120 images. This would explain why the initial validation sets score relatively low. Furthermore, since the images in the training set is small, the presence of image "noise" in the set is more noticeable or impactful in the results. These noises could include images that were incorrectly cropped due to incorrect bounding boxes or irregularity in a specific picture (e.g actor wearing glasses that do not generally wear glasses, slanted faces, etc.). These noises may explain inconsistency in performance in small training sizes, such as size 30 performing better than size 40.

Performance is at its highest at size 50. At this size, the training set is at a modest 300 images. As the size increases pass 50, there is a steady decline in performance. This may be a result of overfitting. That is, the algorithm is becoming more and more specific to the training set or sets that are very similar to it, and so will not perform as well on other sets. Overfitting defeats the purpose of facial or gender recognition, as the algorithm now performs very well on *seen* images, but very poorly on *unseen* images. The algorithm may be trained to recognize random features in the training set, which may have nothing to do with the intended goal.

The performance on actors who are not in `act`:
```
act_test = ['Gerard Butler', 'Daniel Radcliffe', 'Michael Vartan',
'Lorraine Bracco', 'Peri Gilpin', 'Angie Harmon']
```

Running the optimized $\boldsymbol{\theta}$ obtained from 100 images from each actor in `act` on a validation set of different actors (10 image from each), the performance turned out to be 82%. A score of 22/30 of accurately identifying the gender of the females, and a score of 27/30 of accurately identifying the gender of the males.

Problem 6

Compute $\partial J/\partial\theta_{pq}$

$$\begin{aligned}
J(\boldsymbol{\theta}) &= \sum_{i=1}^{m}\left(\sum_{j=1}^{k}(\boldsymbol{\theta}^T\mathbf{x}^{(\mathbf{i})}-\mathbf{y}^{(\mathbf{i})})_j^2\right) \\
&= \sum_{i=1}^{m}\sum_{j=1}^{k}\left(\theta_{1j}x_1^{(i)}+\theta_{2j}x_2^{(i)}+\ldots+\theta_{nj}x_n^{(i)}-y_j^{(i)}\right)^2 \\
\frac{\partial J}{\partial\theta_{pq}} &= \sum_{i=1}^{m}\frac{\partial}{\partial\theta_{pq}}\sum_{j=1}^{k}\left(\theta_{1j}x_1^{(i)}+\theta_{2j}x_2^{(i)}+\ldots+\theta_{nj}x_n^{(i)}-y_j^{(i)}\right)^2 \\
&\overset{*}{=} 2\sum_{i=1}^{m}\left(\theta_{1q}x_1^{(i)}+\theta_{2q}x_2^{(i)}+\ldots+\theta_{pq}x_p^{(i)}+\ldots+\theta_{nq}x_n^{(i)}-y_q^{(i)}\right)\frac{\partial}{\partial\theta_{pq}}\left(\theta_{pq}x_p^{(i)}\right) \\
&= 2\sum_{i=1}^{m}\left(\boldsymbol{\theta}_q^T\mathbf{x}^{(\mathbf{i})}-y_q^{(i)}\right)x_p^{(i)}
\end{aligned}$$

The asterisk (*) in the third line denotes the fact that all columns except the qth column vanishes.

Furthermore, $\boldsymbol{\theta}_q^T$ is the qth column of $\boldsymbol{\theta}$, m is the number of images in the training set, k is the number of possible labels, $\boldsymbol{\theta}$ is an n by k matrix, with each column corresponding to a different label k. Lastly, $\mathbf{x}^{(i)}$ and $\mathbf{y}^{(i)}$ correspond to the ith image and label, respectively.

Derivative of $J(\boldsymbol{\theta})$ with respect to $\boldsymbol{\theta}$

Let

$$\boldsymbol{\theta}^T = \begin{bmatrix} \boldsymbol{\theta}_1^T \\ \boldsymbol{\theta}_2^T \\ \vdots \\ \boldsymbol{\theta}_k^T \end{bmatrix} \in \mathbb{R}^{k\times n}$$

where $\boldsymbol{\theta}_i$ is the ith column of $\boldsymbol{\theta}$, and

$$\boldsymbol{X} = \begin{bmatrix} x_1^{(1)} & \dots & x_1^{(m)} \\ x_2^{(1)} & \ddots & \vdots \\ \vdots & & \\ x_n^{(1)} & \dots & x_n^{(m)} \end{bmatrix} = \begin{bmatrix} \mathbf{x^{(1)}} & \mathbf{x^{(2)}} & \dots & \mathbf{x^{(m)}} \end{bmatrix} \in \mathbb{R}^{n\times m},$$

$$\boldsymbol{Y} = \begin{bmatrix} y_1^{(1)} & \dots & y_1^{(m)} \\ y_2^{(1)} & \ddots & \vdots \\ \vdots & & \\ y_k^{(1)} & \dots & y_k^{(m)} \end{bmatrix} \in \mathbb{R}^{k\times m},$$

Then,

$$\boldsymbol{\theta}^T\mathbf{X} - \mathbf{Y} = \begin{bmatrix} \boldsymbol{\theta}_1^T\mathbf{x^{(1)}} - y_1^{(1)} & \dots & \boldsymbol{\theta}_1^T\mathbf{x^{(m)}} - y_1^{(m)} \\ \boldsymbol{\theta}_2^T\mathbf{x^{(1)}} - y_2^{(1)} & \ddots & \vdots \\ \vdots & & \\ \boldsymbol{\theta}_k^T\mathbf{x^{(1)}} - y_k^{(1)} & \dots & \boldsymbol{\theta}_k^T\mathbf{x^{(m)}} - y_k^{(m)} \end{bmatrix}$$

For brevity, it is sufficient to analyse only the i, jth component, using the shortform matrix notation $A = [a_{pq}]$. Therefore, the above can be rewritten as follows:

Proof.

$$\begin{aligned}
\boldsymbol{\theta}^T\mathbf{X} - \mathbf{Y} &= [\boldsymbol{\theta}_p^T\mathbf{x^{(q)}} - y_p^{(q)}] \\
(\boldsymbol{\theta}^T\mathbf{X} - \mathbf{Y})^T &= [\boldsymbol{\theta}_q^T\mathbf{x^{(p)}} - y_q^{(p)}] \\
\mathbf{X}(\boldsymbol{\theta}^T\mathbf{X} - \mathbf{Y})^T &= \mathbf{X}\cdot[\boldsymbol{\theta}_q^T\mathbf{x^{(p)}} - y_q^{(p)}] \\
&= [x_p^{(1)}(\boldsymbol{\theta}_q^T\mathbf{x^{(1)}} - y_p^{(1)}) + x_p^{(2)}(\boldsymbol{\theta}_q^T\mathbf{x^{(2)}} - y_p^{(2)}) + ... + x_p^{(m)}(\boldsymbol{\theta}_q^T\mathbf{x^{(m)}} - y_p^{(m)})] \\
&= \left[\sum_{i=1}^{m}\left(\boldsymbol{\theta}_q^T\mathbf{x^{(i)}} - y_q^{(i)}\right)x_p^{(i)}\right] \\
2\mathbf{X}(\boldsymbol{\theta}^T\mathbf{X} - \mathbf{Y})^T &= \left[2\sum_{i=1}^{m}\left(\boldsymbol{\theta}_q^T\mathbf{x^{(i)}} - y_q^{(i)}\right)x_p^{(i)}\right] \\
&\overset{*}{=} \left[\frac{\partial J}{\partial\theta_{pq}}\right]
\end{aligned}$$

which is by definition the derivative of a function with respect to a matrix: an element-wise partial differentiation. The asterisk (*) uses the result from 6(a), applied to every element in the matrix. Since the indices p and q are arbitrary and apply to every element in the matrix, the steps hold in both directions, then the proof is complete.

■

Cost function and vectorized gradient function

Code

```
def fvector(x, y, theta):
    x = vstack( (ones((1, x.shape[1])), x))
    return sum(sum( (y - dot(theta.T,x)) ** 2,0))

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

`fvector` is the cost function exactly, but with all the x pixels concatenated in one matrix $\mathbf{X}$ as columns. The inner `sum` performs element-wise summation (over j) along each column, whereas the outer sum performs element-wise summation (over i) along the rows.

Finite Differences

Code

```
   random.seed(0)
   y = reshape(random.rand(800), (4,200))
   h = 0.000000001
   theta0 = reshape(random.rand(4100),(1025,4))
5  dtheta = zeros((1025,4))
   dtheta[0,0]=h #partial derivative with respect to element(1,1)
   #dtheta[0,1]=h #partial derivative with respect to the element (1,2)
   #dtheta[1,1]=h #partial derivative with respect to the element (2,2)
   print (fvector(x, y, theta0+dtheta) - fvector(x, y, theta0-dtheta))/(2*h)
10 print dfvector(x, y, theta0)
```

The vector x is generated earlier in the code, and `y`, `theta0` were randomly chosen to demonstrate the functionality of the code. Also, `dtheta` is a zero matrix less one element which is h. By defintion, this calculates the partial derivative, a small change in h along one direction, which is dictated by line 6.

```
In: dtheta[0,0]=h
print (fvector(x, y, theta0+dtheta) - fvector(x, y, theta0-dtheta))/(2*h)
print dfvector(x, y, theta0)

Out: 30273437.5
[[  3.02906024e+07   3.00667913e+07   2.95583392e+07   2.96159407e+07]
 [  2.40552290e+09   2.38692648e+09   2.34459534e+09   2.35053357e+09]
 [  2.29817665e+09   2.28179369e+09   2.23801354e+09   2.24554318e+09]
 ...,
 [  3.57350004e+09   3.54954667e+09   3.48894366e+09   3.49610092e+09]
 [  3.37405342e+09   3.35171982e+09   3.29359779e+09   3.29979592e+09]
 [  3.26462735e+09   3.24428678e+09   3.18840374e+09   3.19353674e+09]]
```

The first output and the first element of the matrix are similar, as expected.

```
In: dtheta[1,1]=h
print (fvector(x, y, theta0+dtheta) - fvector(x, y, theta0-dtheta))/(2*h)
print dfvector(x, y, theta0)
```

```
Out: 2386718750.0
[[  3.02906024e+07   3.00667913e+07   2.95583392e+07   2.96159407e+07]
 [  2.40552290e+09   2.38692648e+09   2.34459534e+09   2.35053357e+09]
 [  2.29817665e+09   2.28179369e+09   2.23801354e+09   2.24554318e+09]
 ...,
 [  3.57350004e+09   3.54954667e+09   3.48894366e+09   3.49610092e+09]
 [  3.37405342e+09   3.35171982e+09   3.29359779e+09   3.29979592e+09]
 [  3.26462735e+09   3.24428678e+09   3.18840374e+09   3.19353674e+09]]
```

The first output is similar to the array element (2,2).

Problem 7

Run gradient descent on the set of six actors
The method of one-hot-encoding was used to classify the actors, with the states:

```
[1,0,0,0,0,0] = 'drescher'
[0,1,0,0,0,0] = 'ferrera'
[0,0,1,0,0,0] = 'chenoweth'
[0,0,0,1,0,0] = 'baldwin'
[0,0,0,0,1,0] = 'hader'
[0,0,0,0,0,1] = 'carell'
```

Of course, the output of $\mathbf{h} = \boldsymbol{\theta}^T\mathbf{x}$ will not normally result in one of above discrete states. However, rewritting $\boldsymbol{\theta}^T = [\boldsymbol{\theta}_1^T\boldsymbol{\theta}_2^T...\boldsymbol{\theta}_6^T]$, where θ_i^T is a column matrix ($\mathbb{R}^{1025\times1}$) corresponding to the ith label, then $\mathbf{h} = [\boldsymbol{\theta}_1^T\mathbf{x}^{(\mathbf{i})}\ \boldsymbol{\theta}_2^T\mathbf{x}^{(\mathbf{i})}...\boldsymbol{\theta}_6^T\mathbf{x}^{(\mathbf{i})}]$, where $\mathbf{x}^{(\mathbf{i})}$ is the ith image of the test set. Intuitively, $\theta_i^T\mathbf{x}^{(\mathbf{i})}$ yields the cosine similarity of the image $\mathbf{x}^{(\mathbf{i})}$ with the aggregate-trained image $\boldsymbol{\theta}_i$ of actor i. In a given output, each index i corresponds to the similarity of the test image with actor i. Thus, the index j corresponding to the largest output (i.e $j = \arg\max \mathbf{h}$) means that actor j is most similar to the test image, which is the criteria used to classify the images.

The training set and validation set resulted in 89% and 82%, respectively. Again, gradient descent was tuned using four (4) parameters: (1) initial theta, (2) threshold (ϵ), (3) step size (α), and (4) number of iterations. For the most part, the initial theta was kept to be all 0, with the justification for that mentioned earlier in Problem 3. Since, it was found that randomizing theta, even for very small values, were always found to converge slower than if theta were to be all zeros. Since the process of tuning the parameters was elaborated in Problem 3, for brevity, the discussion here will be limited in regards to that.

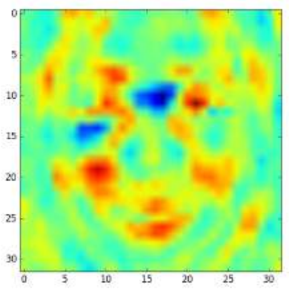
The mindset and goal of tuning here however, is to tune in such a way that gradient descent *does* find a minimum, but it doesn't converge to the point that it *overfits*. This delicate balance required many combinations of $\alpha = 10^{-8}, 10^{-9}, 10^{-10}, 10^{-11}$ and threshold values of $10^{-7}, 10^{-8}, 10^{-9}, 10^{-10}$. The number of iterations was capped relatively low, at 60000 as any more iterations would most probably lead to overfitting. As can be seen, the performance on the training set was only 89%, and not 100%, which is a good indicator that overfitting may not be as prevalent of a problem.

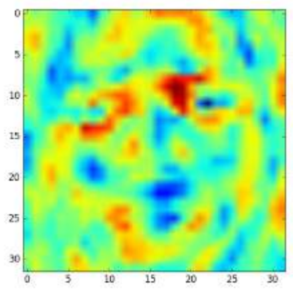
The final tuned parameters were $\boldsymbol{\theta}_0 = [0\ 0...0] \in \mathbb{R}^{1025}$, $\epsilon = 10^{-10}$, $\alpha = 10^{-12}$, and threshold $= 50000$. These make sense, as the threshold is not too high as to introduce much overfitting, ϵ not too large so gradient descent doesn't end too early and reaches the minimum, and α not too large so that the algorithm doesn't overshoot the minimum and diverge immediately. These parameters were sought to minimize the effects of overfitting, yet still find a point close to the minimum.

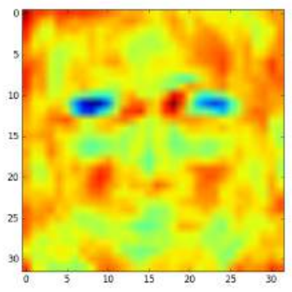
Problem 8

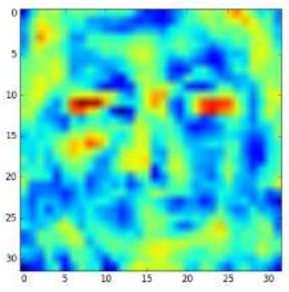
Images of Actors

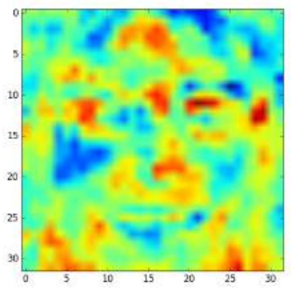
While the images do resemble faces, it is definitely difficult to identify the corresponding actor. This may be because the validation set yielded only a performance of 82%, which may be increased with more sophisticated techniques. Although, it would seem very difficult to produce θ images with striking resemblance due to the variable nature of the images.


(a) Fran Drescher


(b) America Ferrera


(c) Kristen Chenoweth


(d) Alec Baldwin


(e) Bill Hader

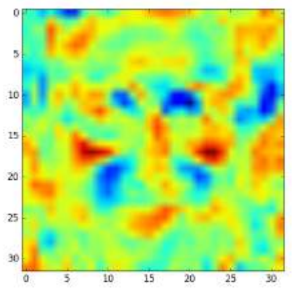

(f) Steve Carell

Figure 5: Visualizing θ

Appendix

How to run the code

Note: Following these instructions, the code is completely reproducible. It requires running the code, section by section. [6]

Running Part 2 Create the training, testing and validation set directories

Comment everything out except Part 2.

```
    In: act =['drescher', 'ferrera', 'chenoweth', 'baldwin', 'hader', 'carell']
    In: makeset(act)
```

Running Part 3

Comment out all code after Part 3. Ensure that the number of iterations for gradient descent is set to 100000. Compile the code, and:

```
    In  [1]: theta0 = array([-0.04]*1025)
        theta = grad_descent(f, df, x, y, theta0, 0.0000000001)
    Out [1]: theta
(array([-0.03973726, -0.00403923, -0.00246078, ..., -0.00139349,
       -0.00150549, -0.00084913]), 0.2466809438183149)
    In  [2]: testset("trainingset")
    In  [3]: testset("validationset")
    In  [4]: testset("tesset")
```

The output of [2], [3], and [4] is a tuple corresponding to the score of Hader and Carell, respectively. The output of [2] is out of 100, the output of [3] and [4] are out of 10. Since there are two actors, summing the tuple up and dividing by the total number of actors will give the percentage score.

Running Part 4

Uncomment section "Part 4a" and run it. This will display the image created using 100 training sets of each actor. Uncomment section "Part 4b" and run the initial conditions:

```
    In: theta0 = array([0.0]*1025)
    theta = grad_descent(f, df, x1, y1, theta0, 0.0000000001)
```

Then using this theta value, run the code from "Part 4a". This will generate the image created using 2 training sets of each actor.

Running Part 5

Uncomment all of Part 5. Set `size=10`, incrementing by 10 each time until `size=100`. Set the number of iterations of gradient descent to 60000.

```
    In: theta0 = array([0.0]*1025)
        theta = grad_descent(f, df, x1, y1, theta0, 0.00000000001)
        validationset("validationset")
        trainingset("trainingset")
```

[6]Allowed by Piazza Post #266

The results of `validationset` is out of `size` multiplied by 6 (number of actors), but both `validationset` and `trainingset` return a tuple, wherein the first and second elements of the tuple are the performances of the female and male actors, respectively. These gender performances are out of `size` multipled by 3 (number of male actors or number of female actors).

To generate the plot, uncomment the section "Plot". Then, run that section and type the following into the shell:

```
In: Plot
In: Show()
```

To test the performance on actors not in `act`. First run the code from "Part 2", replacing `act` with `act_test`. Then, in the function `makeset`, change the name of the folder that will be created to "validation1". Ensure that `i` is set to 10. Now, this will create a folder "validation1" with randomized images of each actor in `act_test`.
Run the code "Part 5" with `size` = 100, then:

```
In: validationset1("validationset1")
```

The output is a tuple whereby the first number is the performance of females (out of 30) and the second number is the performance of males (out of 30).

Running Part 6

This was shown in detail in the report.

Running Part 7

First, need to create a validation set comprising of 50 random images of each actor in `act`. Go to the code from "Part 2" and ensure that `i` is set to 50. Then, change the name of the folder that will be created to "validationset7". Ensure that `act` comprises of the actors we want. Then, run this code. Use the parameters: $\alpha = 0.000000000001$, $\epsilon = 1e^{10}$, and number of iterations = 50000.

```
In: accuracy("trainingset")
    accuracy("validationset7")
```

The output should be a number between 0 and 1, which represents the accuracy in percentage.

Running Part 8

Using the θ obtained from Part 7, run code from section "Part 8". Then run `imshow()` followed by `show()`, where the parameters of `imshow()` is "imX", where X is the number corresponding to the actor. This number is mentioned in the report. .