CSC411: Assignment #1

Due on Friday, January 29, 2018

Lukas Zhornyak

February 21, 2018

Environment

The code used in this assignment is split across three file: main.py, get_data.py, and classifier.py. main.py is generally where the program should be run from, and will produce all the outputs used in this report. get_data.py has all function associated with downloading, saving, and manipulating the images for use in processing. Finally, classifier.py includes the code for gradient descent, the classifier, and all associated functions.

This project was created with Python 2.7.14 with numpy 1.14.0, scipy 1.0.0, scikit-image 0.13.1, and matplotlib 2.1.1, as well as all associated dependencies.

Part 1

The images were downloaded using the provided FaceScrub dataset. Only those images whose SHA-256 hash matched the one provided in the aforementioned dataset were kept to help reduce the problem of incorrect bounding boxes. A total of 1651 images were downloaded. Figure 1 provides three examples. While most images are of the actor with either a neutral expression or a small smile, some images have decidedly more extreme expressions. This likely increases the difficulty in correctly classifying this particular dataset.

After being downloaded, the images were cropped according to the bounding boxes provided and converted to grey-scale. This produced a total of 1569 cropped images, less than the number of raw images downloaded. Unfortunately, due to issues with the bounding boxes, not all images could be cropped without error. Three example images are provided in fig. 2.

As can be seen in the example images, the major facial features (i.e. nose, eyes, etc.) generally align pretty well with one another. This suggests that the bounding boxes are generally pretty accurate, at least for those images that were successfully processed. Much of the differences between the cropped images can be ascribed to differences in the pose and expression of the actor in question. One common such difference is when the actor does not face directly at the camera, but rather at an angle to it. Since these differences are not controlled and may vary in some systematic way between the actors, it may be possible for a learning algorithm to incorrectly focus on the expressions of the actor rather than facial features.

The number of cropped images available for each actor is:

Alec Baldwin: 137 Images

Lorraine Bracco: 107 Images

Gerard Butler: 123 Images

Steve Carell: 138 Images

Kristin Chenoweth: 173 Images

Fran Drescher: 143 Images

America Ferrera: 153 Images

Peri Gilpin: 86 Images

Bill Hader: 140 Images

Angie Harmon: 101 Images

Daniel Radcliffe: 139 Images

Michael Vartan: 129 Images

(a) America Ferrera (b) Bill Hader (c) Steve Carell

Figure 1: Three examples of raw images in the dataset.

(a) America Ferrera (b) Bill Hader (c) Steve Carell

Figure 2: Three examples of cropped images in the dataset, matching the examples provided in fig. 1.

Part 2

Creating the random training, validation, and testing sets was done in two steps. First, a random non-overlapping subset of the available images is selected. This is done directly through NumPy's choose function, but can also be implemented very simply without it. One example of an alternative method would be to randomly select then remove elements from the list of valid images. Since the order of the elements selected is randomly determined, there should be no correlation between an image and its position in the list of selected images. Thus, these elements are distributed between the three desired sets based on their index.

Part 3

In performing gradient descent for linear regression, the mean squared error was minimized:

$$J(\theta) = \frac{1}{m} \sum_i \left(\theta^\mathsf{T} x^{(i)} - y^{(i)} \right)^2$$

where m is the number of images. The mean squared error was chosen as opposed to the squared error to ensure that the magnitude of the gradient would not increase as a function of the size of the training set, thus allowing the same learning rate to be used regardless of the size of the training set.

The learning rate was selected to be as large as possible while maintaining the stability of the algorithm. If α is too large, then taking a step during gradient descent will place the current position further up the gradient on the other side, resulting in the runaway growth of the parameters. If α is too small (or slightly too large) however, the only issue may be that the algorithm would take a very long time to converge. Considering this, the learning rate was only selected to the nearest order of magnitude to minimize the validation error, settling on 10^{-3}.

The target epsilon for the desired change in θ before finishing descent was selected in a very similar manner, settling on 4×10^{-5}. A bound on the maximum number of iterations was set to 10^5 to ensure that the optimization concluded in a timely manner. As recommended in the assignment instructions, the images were converted to have values to values between 0 and 1, allowing for the learning rate and final θ to have reasonably sized values.

With these choices the optimization produced a final testing error of 0.01584 and a final validation error of 0.04005. These two errors are reasonably similar in magnitude, suggesting that overfitting is not a large problem. The parameters obtained produced a 100% accuracy on classifying the images in the validation set and a 95% accuracy on classifying images in the training set.

Below is a reduced version of the function used to compute the output of the classifier. Reduced in this context means that several lines of code related to later parts of this assignment were removed as they are not relevant here, but there were no changes to the lines present. While technically this function is used to compute the labelling accuracy, by providing only one image for the data parameter and simply 1 for the real_labels parameter, the output of this function can be interpreted as a boolean. As described above and as implemented in main.py, the function will return True if this picture is Alec Baldwin and False if it is Steve Carell.

```
1 def labelling_accuracy(data, real_labels, parameters):
2     results = np.dot(data, parameters[1:]) + parameters[0]
3     return np.sum((results > 0.5) == real_labels) / len(real_labels)
```

Part 4

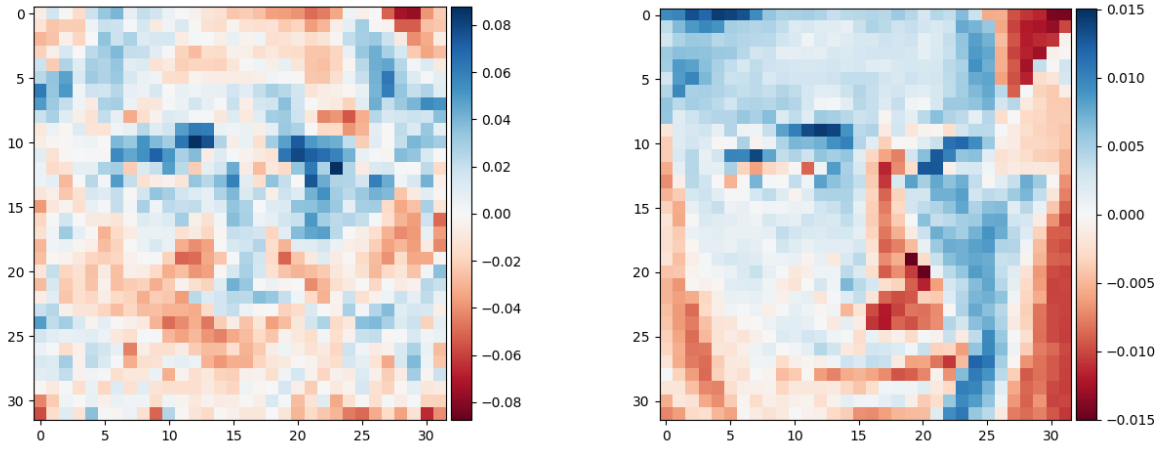
4 (a)

The two sets of parameters are visualized in fig. 3.

4 (b)

The two kinds of images produced are show in fig. 4. The more human like image was obtained by running gradient descent for only a small number of iterations (100), while the more random looking image was obtained by running for a very large number of iterations (100 000).

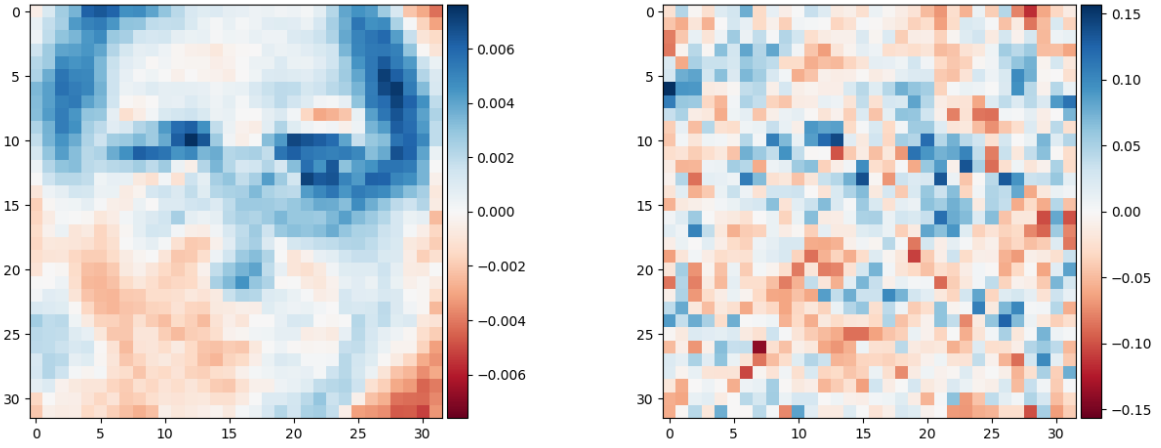
Often in image recognition, the parameters end up highlighting specific features that are being used to differentiate. When they look random and have very sharp changes in weight, it may be an example of overfitting as the optimization focuses on specific pixels. Meanwhile, after only a small number of generations the parameter values will tend to move in a generally constant direction. The amount they move is scaled by the intensity of the image, so it will tend to produce a visualization of the "average" picture.



(a) Full training set

(b) Two images per actor

Figure 3: Visualization of the parameters obtained for the linear regression performed on a full training set and performed on a training set containing only two images per actor. The results from the smaller training have several sharp boundaries and odd features on an otherwise human looking face, suggesting that there is overfitting to specific features present in this smalller set.



(a) 100 iterations

(b) 100 000 iterations

Figure 4: Visualization of the parameters obtained for the linear regression for a varying number of generations.

Part 5

The performance of the classifier versus the size of the training set is shown in fig. 5. Note that, as expected the performance on the training and validation sets increases as the size of the training set increases. This increase is fast at first, but slows down as the size of the training set increase; each additional image added does less to improve performance that the previous image.

On the full set, the individual actors were correctly classified as male or female with almost perfect accuracy:

- Lorraine Bracco classified with 1.0 accuracy
- Peri Gilpin classified with 0.98 accuracy
- Angie Harmon classified with 1.0 accuracy
- Alec Baldwin classified with 1.0 accuracy
- Steve Carell classified with 1.0 accuracy
- Bill Hader classified with 0.96 accuracy

Meanwhile, for the actors not in the training set, the accuracy ranged from great to not very good:

- Kristin Chenoweth classified with 0.64 accuracy
- America Ferrera classified with 0.92 accuracy
- Fran Drescher classified with 0.92 accuracy
- Gerard Butler classified with 0.96 accuracy
- Michael Vartan classified with 0.74 accuracy
- Daniel Radcliffe classified with 0.7 accuracy

This is likely the result of the overfitting of the classifier on the training set, focusing on features common to the specific actors rather than the features distinguishing male from female.

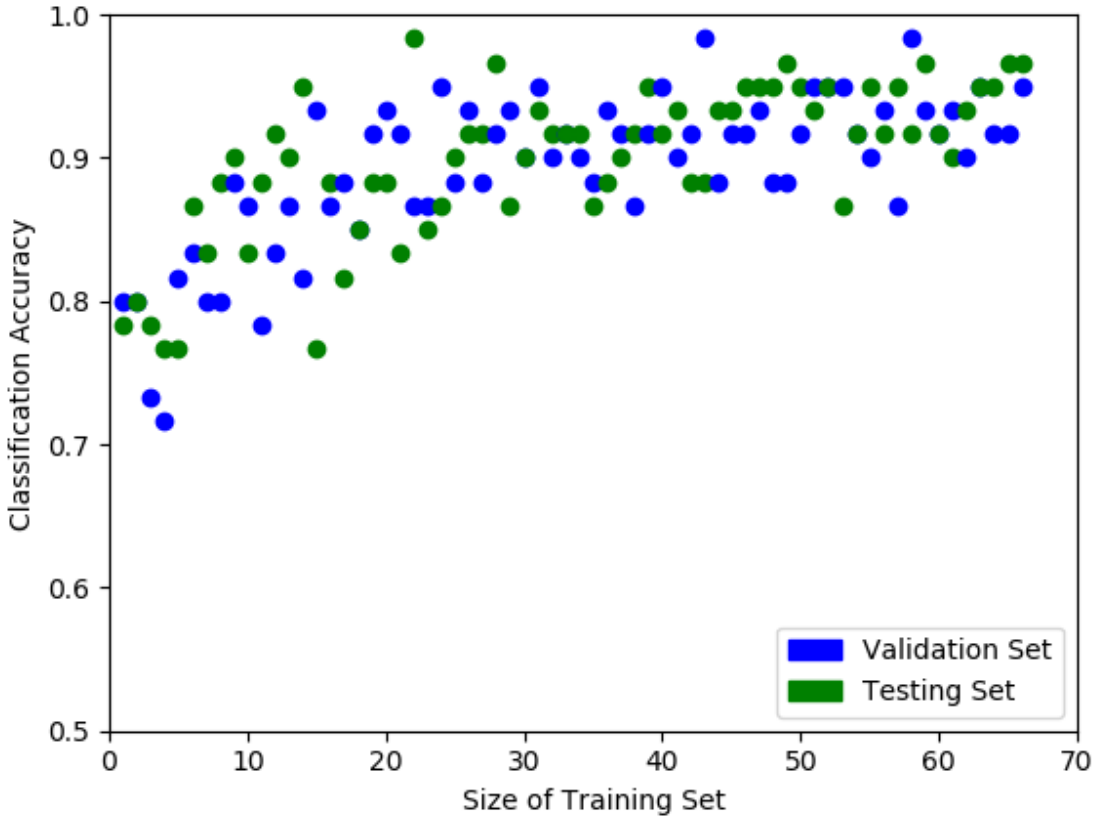


Figure 5: Performance of classifier versus size of training set on both the validation and testing set. Note that this figure was obtained with a random seed for each set size.

Part 6

6 (a)

$$\begin{aligned}\frac{\partial J(\theta)}{\partial \theta_{pq}} &= \frac{\partial}{\partial \theta_{pq}} \sum_i \left(\sum_j \left(\theta^\mathsf{T} x^{(i)} - y^{(i)} \right)_j^2 \right) \\ &= \frac{\partial}{\partial \theta_{pq}} \sum_i \left(\theta_{pq} x_p^{(i)} - y_q^{(i)} \right)^2 \\ &= 2 \sum_i x_p^{(i)} \left(\theta_{pq} x_p^{(i)} - y_q^{(i)} \right) \end{aligned} \tag{1}$$

6 (b)

Let $X \in \mathbb{R}^{1025 \times m}$ represent the training set with additional ones, where each column represents one image and m is the number of images. Let $Y \in \mathbb{R}^{n \times m}$ represent the labels assigned to each image, with each column denoting one label and n being the number of actors to classify. Let $\theta \in \mathbb{R}^{1025 \times n}$ represent the parameters used for the linear regression. By the definition of the gradient,

$$\nabla J(\theta) = \frac{\partial J(\theta)}{\partial \theta} = \begin{bmatrix} \frac{\partial J(\theta)}{\partial \theta_{11}} & \cdots & \frac{\partial J(\theta)}{\partial \theta_{1q}} \\ \vdots & \ddots & \vdots \\ \frac{\partial J(\theta)}{\partial \theta_{p1}} & \cdots & \frac{\partial J(\theta)}{\partial \theta_{pq}} \end{bmatrix}$$

Let $\chi_p \in \mathbb{R}^{1 \times m}$ represent the p-th row of X and let $\gamma_q \in \mathbb{R}^{1 \times m}$ represent the q-th row of Y. Equation (1) can thus be rewritten as:

$$\frac{\partial J(\theta)}{\partial \theta_{pq}} = 2\chi_p(\theta_{pq}\chi_p - \gamma_q)^\mathsf{T} \tag{2}$$

Let $\theta^{(q)} \in \mathbb{R}^{1025 \times 1}$ represent the q-th column of θ. Then, using eq. (2),

$$\begin{aligned}\frac{\partial J(\theta)}{\partial \theta^{(q)}} &= \begin{bmatrix} \frac{\partial J(\theta)}{\partial \theta_{1q}} & \cdots & \frac{\partial J(\theta)}{\partial \theta_{pq}} \end{bmatrix}^\mathsf{T} \\ &= 2X \left(\left(\theta^{(q)} \right)^\mathsf{T} X - \gamma_q \right)^\mathsf{T} \end{aligned}$$

By the same process, the matrix is expanded along q, obtaining the desired final expression:

$$\nabla J(\theta) = \frac{\partial J(\theta)}{\partial \theta} = 2X \left(\theta^\mathsf{T} X - Y \right)^\mathsf{T} \tag{3}$$

6 (c)

For the reasons described in Part 3, the mean squared error is preferred. Additionally, it is more convenient to express the images as row vectors rather than column vectors. For these reasons, eq. (3) is modified by dividing by the number of images (m) and taking its transpose. Thus, the equation implemented is:

$$\frac{\partial J(\theta)}{\partial \theta} = \frac{2}{m} X^\mathsf{T} (X\theta - Y) \tag{4}$$

This is implemented below,, along with the cost function. Note that in this implementation, X is not extended with additional ones but rather only multiplied with the relevant part of θ.

```python
def cost_function(data, labels, parameters):
    cost = np.linalg.norm(np.dot(data, parameters[1:]) + parameters[0] - labels)
    return cost ** 2 / len(data)

def cost_gradient(data, labels, parameters):
    temp = np.dot(data, parameters[1:]) + parameters[0] - labels
    gradients = np.empty_like(parameters)
    gradients[0] = np.sum(temp)
    gradients[1:] = np.dot(data.T, temp)
    return 2 * gradients / len(data)
```

6 (d)

The code used to compare gradient descent algorithm above to the results obtained by finite differences is provided below:

```python
# initialize some data to be used and find gradient
images = get_data.load_data('bracco', [40])
param = np.random.random((1025, 6)) * 1e-2
labels = np.array([[1, 0, 0, 0, 0, 0]] * 40)
grad = classifier.cost_gradient(images, labels, param)
h = 1e-6
# compare against finite differences
np.random.seed(17)
for _ in range(5):
    p = np.random.randint(0, 1025)
    q = np.random.randint(0, 6)
    param_mod = param.copy()
    param_mod[p, q] += h
    estimate = (classifier.cost_function(images, labels, param_mod) - classifier.cost_function(images, labels, param)) / h
    print('(p,q) =', (p, q), '-> function:', '{:f}'.format(grad[p, q]), '\t', 'estimate:', '{:f}'.format(estimate))
```

By incrementing the element of the parameters at (p, q) by h and then using the finite difference approximation, the gradient of the element at (p, q) is approximated. Thus, by comparing this result to the value at the position (p, q) of the result from from the gradient descent function function, we can attempt to see if the approximation matches what would be expected from the derived result.

In this case, with an h of 10^{-6} the values are accurate to at least seven significant figures, suggesting that the formula for the gradient descent found is correct. A smaller h would lead to even more accurate results, up to a certain limit of machine precision. The value above was selected since it is the smallest order of magnitude for which the the two values completely match in the standard python formatted float output.

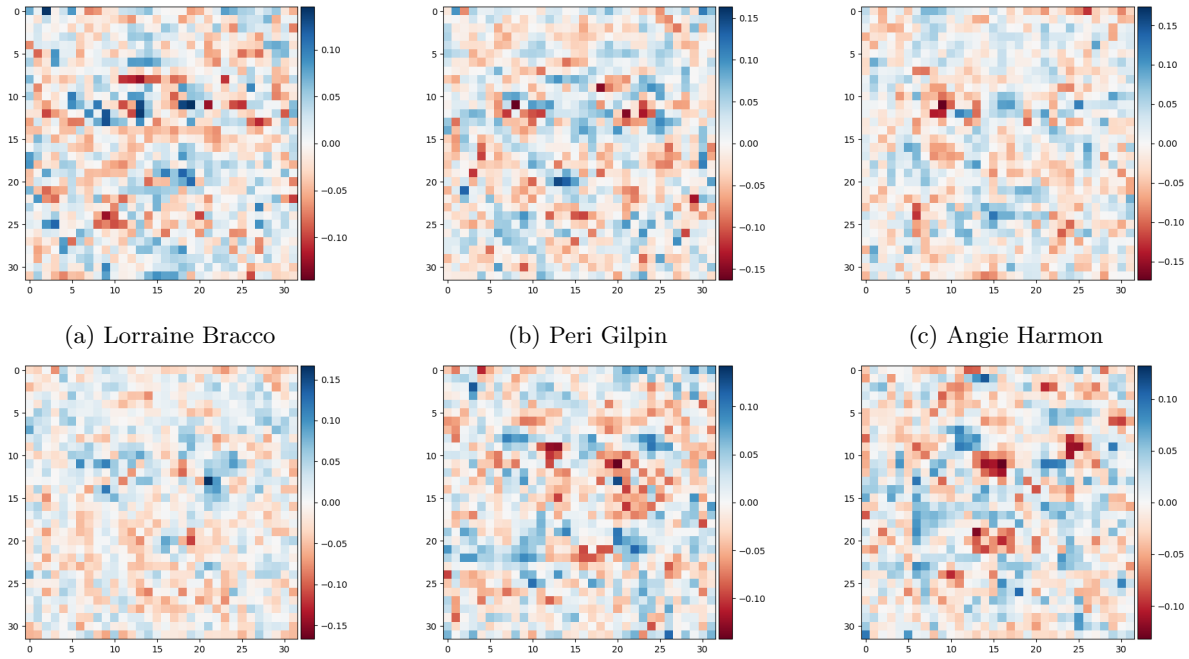
Part 7

The parameters chosen for the gradient descent were the same as those described in Part 3, for the same reasons. Since the only practical difference is the number of images and since a error per image is used, it would make sense that there would not be any change needed in the parameters. With these choices, the classifier achieved a 0.8167 accuracy on both the validation and training set for selecting the right actor out of the six possible choices.

To find the label associated with the output of the classifier, the argmax of each row is found. The actor whose label has a one in this position and a zero in the other positions is then the corresponding actor associated with this output.

Part 8

The parameters are visualized in fig. 6.



(a) Lorraine Bracco (b) Peri Gilpin (c) Angie Harmon

(d) Alec Baldwin (e) Steve Carell (f) Alec Baldwin

Figure 6: Visualization of the parameters obtained for classification between the six different actors.