Making Better Decisions with Logistic Regression and the Softmax Classifier in Python

Jacky Dam
University of California San Diego
San Diego, California
jdam@ucsd.edu

Ali Zaidi
University of California San Diego
San Diego, California
mazaidi@ucsd.edu

Abstract

In this project, we utilized Stochastic gradient descent along with two different models: Logistic regression, and Softmax classifier to accurately name different types of clothing types given an image. With the help of a large data set of training images, we are able to create a digital neural network which learns how to make the correct clothing classification of images which the program has never encountered before. Through this process, we experienced many interesting bugs and results such as when the program had a low accuracy for recognizing sandals in which we discovered that the algorithm made reasonable guesses with predictions such as ankle boots and sneakers. After developing our algorithm, we found that with the overall best parameters which will be discussed later in our paper, the logistic regression model had an accuracy of about 97% when given sets of images containing only t-shirts and ankle boots, and an accuracy of about 72% when given sets of images containing only pullovers and coats. Finally, the Softmax classifier performed relatively well when given a set of images containing all types of clothing, and returned an accuracy of about 76%.

1 Introduction

For our algorithm, we utilized Python as our primary programming language in which we taught the program how to make adequate decisions, and utilized mathematical concepts derived from the fields of calculus, statistics, and linear algebra to train the program. Our process of creating a digital neural network to make decisions began with having to import a large data sets of images with their classification, and the need to modify the data to fit our algorithm. In our program, we utilized two different models: Logistic regression, and the Softmax classifier in which we feed the algorithm with sets of images we set aside for training. Through Stochastic gradient descent, the algorithm essentially learns how to make better decisions over time by learning from it's mistakes and updating some numerical parameter called weights to hopefully make the correct decision with the next training set.

2 Related Work

Cottrell, Gary 2020, Lecture 3, lecture slides, *Linear Regression and Perceptrons*, UCSD delivered Oct 7, 2020

Tanjim, Mehrab 2020, 151B Discussion, discussion slides, *Discussion ML Practices and PCA v2*, UCSD delivered Oct 6, 2020

Tanjim, Mehrab 2020, *151B Discussion, discussion slides, Discussion PA1*, UCSD delivered Oct 13, 2020

Lecture 3 was useful for understanding PCA and Gradient descent. The Discussion slides helped with Data Manipulation, Model Architecture, and Training Procedure.

2.1 Principle Components Analysis

Principle Components Analysis is essentially the process in which we break down the p number of principle features given a set of images, and reduce the dimensions of the images. In our algorithm, we implemented principle components analysis as a universal method which takes in an array of images, and p number of principle components as our parameters. Through the process of Eigen Value Decomposition and mean-centering the data, we return the first p number of eigenvectors according to the eigenvalues sorted in ascending order. With more principle components, the program is able to better understand the image processed. This idea can be best illustrated in figure (2) below.

In figure (4), we show the top 10 PCs for various different sets of classes. A difference between comparing (T-Shirts and Ankle Boots) and (T-Shirts and Shirt) is the overlap of the Principal components. The T-Shirt and Shirt PC's overlap significantly making the model have harder time distinguishing between them (bad Test Accuracy). When comparing the T-Shirts and Ankle Boots, the overlap is not has severe, so model has much easier time to distinguishing between them (very good Test Accuracy).

2.2 Gradient Descent

Gradient Descent is essentially the process in which the algorithm learns how to learn from it's decision making mistakes, and increases/decreases or keeps the weight vector which affects the next decision that the program makes. By taking in the parameters: weight, learning rate, target, output, and an image, the algorithm is able to progressively make better predictions given enough data. The weight vector is a d+1 dimensional vector in which d represents the dimensions of the inputted image. The learning rate is a constant which we could change to modify how heavy we want the changes in the weight vector to occur. The target y represents the actual classification of the inputted image, and the ouput $\hat{y}$ represents the output of the classification that the program decided on for a specific image labeled as x. Finally, as shown below in the formula, we update the weight vector N times which represents the number of sets of data that we want to pass through. Shown below is the update equations for the weight vector in gradient descent, by solving the gradient of the entropy-cost function for a specific model, formula (2) can be derived. Since this derivation is trivial, it is left as an exercise to the reader.

$$w_{t+1} = w_t - \alpha \sum_{n=1}^{N} \Delta E^n(w) \tag{1}$$

$$w_{t+1} = w_t + \alpha \sum_{n=1}^{N} (y_k^n - \hat{y}_k) \times x_j^n \tag{2}$$

3 Working with the Data Set

From the Fashion-MNIST dataset found below:
https://github.com/zalandoresearch/fashion-mnist
We imported two separate data sets: the training set and testing set. Both of these data sets are composed of two separate arrays: an array of images, and an array of their correct classification. Figure (1) shown below are examples of the images and the 10 different types of clothing which will later be utilized in our deep learning algorithm.

3.1 Shuffling the Data Set

The shuffling method takes in the images array, and targets array as parameters and zips each individual image with it's correct classification before randomly shuffling. We chose to zip each image with it's corresponding classification because our algorithm has to learn from it's mistakes, and

would struggle to progressively get better given bad data sets. As one of the first methods we utilized to clean up the data, we decided to return a deep copy, shuffled version of an array of images zipped with their respective target.

3.2 Augmenting the Data Set

Prior to passing the images through gradient descent in both logistic regression and softmax regression, we decided to implement a method to specifically augment each individual image by 1 to account for the bias w_0 in the weight vector. We implemented this method by taking the images array as the parameter and augmenting each individual image by 1, and returning the augmented images array.

3.3 One Hot Encoding

The targets array contains many individual targets which are composed of numbers on the range from 0 to 10 to signify the classification of it's corresponding image. The images with their classification number can be seen in figure (1). One hot encoding is a specific method of converting k number of classifications to a vector form of dimensions k, and contains values of 0's, and a 1. The value 1 in the one hot encoding vector appears the the index of the target classification value, and the 0's represent empty placeholders for later mathematical calculations. We implemented the one hot encoding method by taking in the targets array as the parameter, and appending vectors with the 1 located at the index of the target classification value to a separate array which we later return.

3.4 K Folds Split

The K folds split method segments the set of data into K equal partitions in which K represents the number of folds (not number of classifications). In our K folds split method, we took in the images array, target array, and number of K folds as parameters. We implemented the K folds split method by first shuffling the images and targets arrays and filling up an empty array up to the length of the data sets divided by K number of sets, and appending the full arrays to another array which we later return.

3.5 Normalization

For our data set, we utilized the min-max normalization to reduce distortion. In our implementation of normalization, we took in the images array as a parameter where we calculated the min and max per column to get values between one and zero, and return an array of normalized images:

$$\frac{(image - min[C_1, C_2, ..., C_d])}{max[C_1, C_2, ..., C_d] - min[C_1, C_2, ..., C_d]} \tag{3}$$

3.6 Image Projection

With the Principle Components Analysis calculation method shown in the beginning of the paper, we want to create a projection of the image for the algorithm to process into useful information. We implemented the projection method which takes in parameters: PCA calculation, and the images array. In our program, we iterated through every image in the images array and multiplied it by the PCA calculation before returning the modified array of images.

Data Set:	# of T-Shirts:	# of Trouser:	# Pullover:	# of Dress:	# of Coat
Test Set	1000	1000	1000	1000	1000
Training Set	6000	6000	6000	6000	6000

Data Set:	# of Sandal:	# of Shirt:	# Sneaker:	# of Bag:	# of Ankle Boot
Test Set	1000	1000	1000	1000	1000
Training Set	6000	6000	6000	6000	6000

Figure 1: All of the Different Classifications

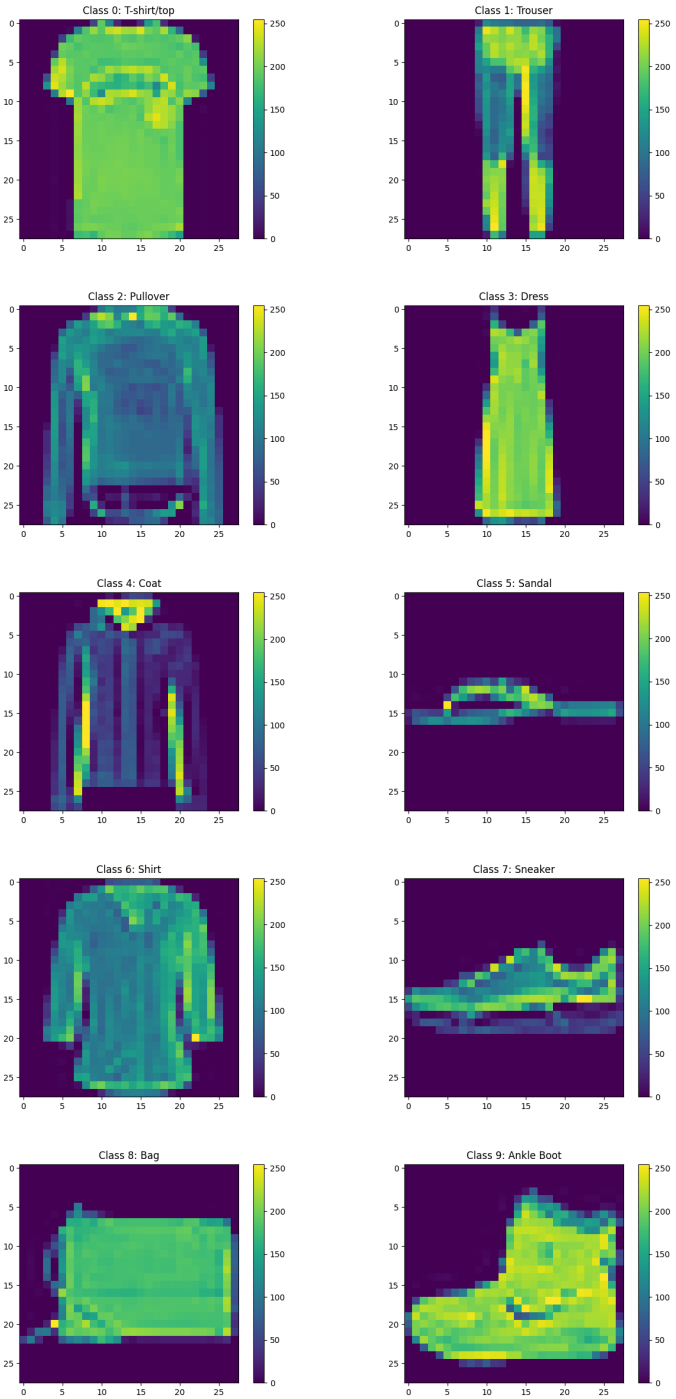
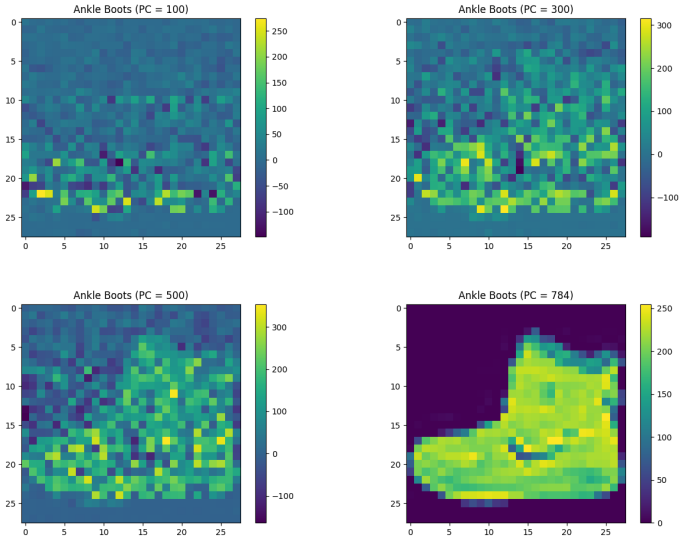


Figure 2: Visualizing the Change in Principle Components



4 Logistic Regression

4.1 Useful Equations

Below are the formulas for calculating the entropy loss E, and output $\hat{y}$.

$$E = -\sum_{n=1}^{N}(y^n ln(\hat{y}^n) + (1 - y^n) ln(1 - \hat{y}^n)) \tag{4}$$

$$\hat{y} = \frac{1}{1 + e^a} = P(C_1|x) \tag{5}$$

$$a = -w^T x \tag{6}$$

$$P(C_0|x) = 1 - P(C_1|x) \tag{7}$$

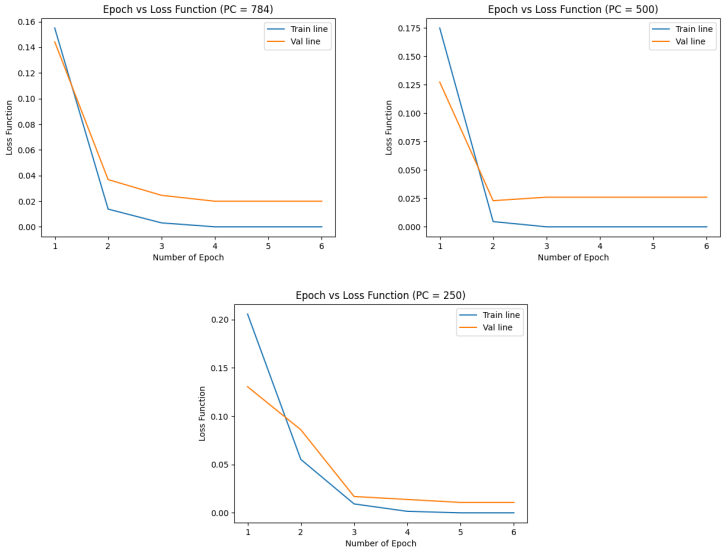
4.2 Logistic Regression Gradient Descent

Although both logistic regression and softmax regression utilize the concept of gradient descent, we decided to create separate implementation for logistic regression because we're updating a single weight vector unlike softmax regression. Following the gradient descent formula shown in the beginning, we increase the values of the weight vector given a false negative, decrease the values of the weight vector given a false positive, and keep the weights if the output matched the target. We keep updating the weights in gradient descent for M number of epochs, and for all the examples in the data set in steps of batch sizes, finally returning the weights. Additionally, we included a specific method inside the M epoch for loop called early stopping, which stops updating the weight vector when it appears that the accuracy isn't getting much better.

4.3 Logistic Regression Loss Function

The Cross-Entropy cost function shown in equation (4) represents the loss given a target and output as parameters. We used this method to help us in the early stopping section part of gradient descent, and the loss over epochs graph can be seen in figure (3).

Figure 3: Logistic Regression Loss Curves



4.4 Target

After utilizing the one hot encoding method earlier in our code, we realized that we required a method to tell the program whether the target is the same as the classification that we want to train the algorithm on. If the target is the right classification we should we working on, we return 1, otherwise we return 0.

4.5 Prediction

To calculate the model prediction ($\hat{y}^n$), we use Eq. 4. We first transpose the weights and take the dot product with the input image. That result we multiple by a negative one and then exponentiate with e. Finally we add a one and take the reciprocal to get our model prediction ($\hat{y}^n$).

4.6 Logistic Regression Accuracy

After running many the algorithm with many different parameters shown in sections 5, 6, 7, 8, 9, we found that the accuracy of the predictions increased when we lowered batch size, learning rate, and increased the number of principle components. Although it's compelling to maximize the accuracy by finding the extremes of these parameters, we recognize that there was a trade off in computing speed when attempting to achieve the best accuracy. When we tested the accuracy of the algorithm for the classification of t-shirts and ankle boots, we found out that our program had a high accuracy of above 95%. However, testing the accuracy for pullover and coats, we found out that the accuracy was 70% which was drastically lower than the other two classifications. We hypothesize that the similiarity between classes drastically affected our accuracy, and we found the same pattern occurring in similar images such as ankle boots vs. sneakers.

5 Modifying the Batch Size

Parameters:
Batch_size = 200
M_Epochs = 100
Learning_Rate=0.1
K_Folds=10
PC=784

Trials:	T-Shirts and Ankle Boots:	Pullover and Coats:
1	98.10%	69.75%
2	96.05%	69.60%
3	97.50%	72.40%

Parameters:
Batch_size = 100
M_Epochs = 100
Learning_Rate=0.1
K_Folds=10
PC=784

Trials:	T-Shirts and Ankle Boots:	Pullover and Coats:
1	97.95%	77.55%
2	98.85%	77.75%
3	98.00%	74.60%

Parameters:
Batch_size = 50
M_Epochs = 100
Learning_Rate=0.1
K_Folds=10
PC=784

Trials:	T-Shirts and Ankle Boots:	Pullover and Coats:
1	99.75%	76.50%
2	99.50%	72.85%
3	99.65%	77.45%

6 Modifying the Learning Rate

Parameters:
Batch_size = 200
M_Epochs = 100
Learning_Rate=0.1
K_Folds=10
PC=784

Trials:	T-Shirts and Ankle Boots:	Pullover and Coats:
1	98.10%	69.75%
2	96.05%	69.60%
3	97.50%	72.40%

Parameters:
Batch_size = 200
M_Epochs = 100
Learning_Rate=0.01
K_Folds=10
PC=784

Trials:	T-Shirts and Ankle Boots:	Pullover and Coats:
1	97.2%	76.80%
2	98.55%	72.00%
3	95.5%	68.05%

Parameters:

Batch_size = 200
M_Epochs = 100
Learning_Rate=0.001
K_Folds=10
PC=784

Trials:	T-Shirts and Ankle Boots:	Pullover and Coats:
1	99.4%	75.05%
2	99.40%	74.85%
3	98.5%	73.90%

7 Modifying the Principle Components

Parameters:
Batch_size = 200
M_Epochs = 100
Learning_Rate=0.1
K_Folds=10
PC=784

Trials:	T-Shirts and Ankle Boots:	Pullover and Coats:
1	98.10%	69.75%
2	96.05%	69.60%
3	97.50%	72.40%

Parameters:
Batch_size = 200
M_Epochs = 100
Learning_Rate=0.1
K_Folds=10
PC=500

Trials:	T-Shirts and Ankle Boots:	Pullover and Coats:
1	95.45%	60.35%
2	99.05%	72.30%
3	99.35%	74.46%

Parameters:
Batch_size = 200
M_Epochs = 100
Learning_Rate=0.1
K_Folds=10
PC=250

Trials:	T-Shirts and Ankle Boots:	Pullover and Coats:
1	91.45%	71.40%
2	98.05%	72.15%
3	94.25%	70.00%

8 Modifying the Number of Epochs

Parameters:
Batch_size = 200
M_Epochs = 100
Learning_Rate=0.1
K_Folds=10
PC=784

Trials:	T-Shirts and Ankle Boots:	Pullover and Coats:
1	98.10%	69.75%
2	96.05%	69.60%
3	97.50%	72.40%

Parameters:

Batch_size = 200
M_Epochs = 200
Learning_Rate=0.1
K_Folds=10
PC=784

Trials:	T-Shirts and Ankle Boots:	Pullover and Coats:
1	94.60%	68.41%
2	97.80%	67.45%
3	97.65%	76.95%

Parameters:
Batch_size = 200
M_Epochs = 400
Learning_Rate=0.1
K_Folds=10
PC=784

Trials:	T-Shirts and Ankle Boots:	Pullover and Coats:
1	96.55%	70.50%
2	97.85%	77.40%
3	98.50%	71.35%

9 Summarizing the Best Parameters (Logistic Regression)

Parameters:
Batch_size = 16
M_Epochs = 100
Learning_Rate=0.001
K_Folds=10
PC=784

Trials:	T-Shirts and Ankle Boots:	Pullover and Coats:
1	99.75%	74.95%
2	99.40%	69.70%
3	99.75%	78.90%

10 Summarizing the Best Parameters (Softmax Regression)

Parameters:
Batch_size = 100
M_Epochs = 100
Learning_Rate=0.001
K_Folds=10
PC=784

Trials:	All Classes:
1	72.87%
2	72.25%
3	74.66%

11 Softmax Regression

11.1 Useful Equations:

$$E = -\sum_{n}\sum_{k=1}^{c} y_k^n ln(\hat{y}_k^n) \tag{8}$$

$$\hat{y}_k^n = \frac{e^{a_k^n}}{\sum_{k'} e^{a_{k'}}} \tag{9}$$

$$a_k^n = w_k^T x^n \tag{10}$$

Figure 4: Images based on Top 10 Principle Components

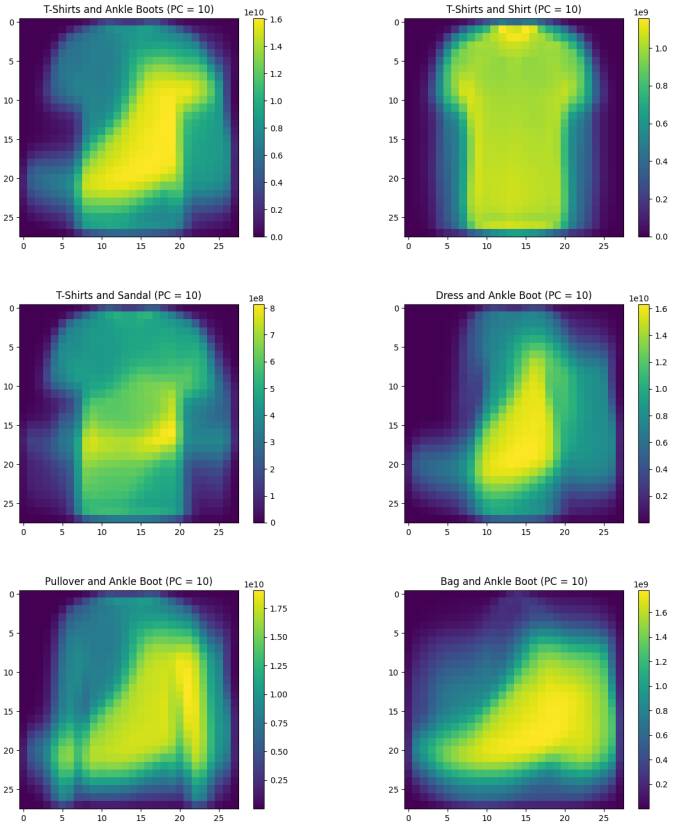
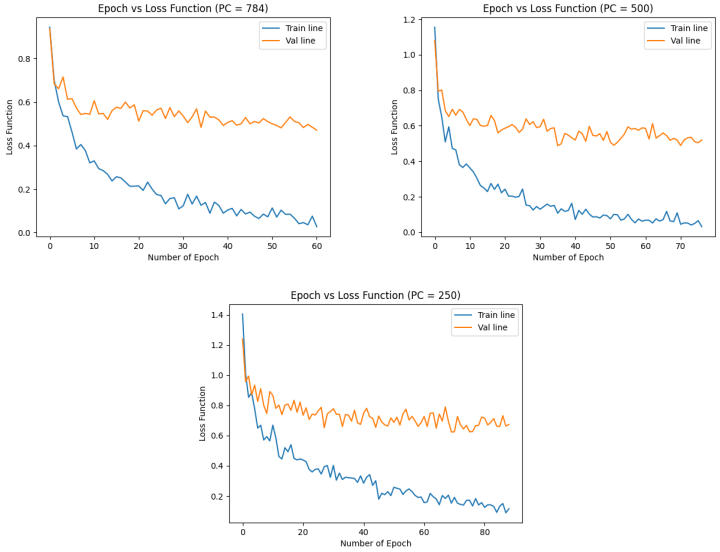


Figure 5: Softmax Regression Loss Curves



11.2 Softmax Prediction

Our implementation of the Softmax prediction method involved taking in the k weights and individual image as parameters. By inputting an image into this method, the algorithm attempts to find where to update the k weights for all the k weights which gave a false positive, false negative, or keep the same k weights.

11.3 Softmax Decision

Equation (9) and Equation (10) shows the Softmax Decision we used our project. We first transpose the k_weights and take the dot product with the input image (Equation (10)). Then we exponentiate Equation (10) and sum over all the classes, to get the denominator of Equation (9). The Softmax Decision will output a probability vector, with the probabilities that a certain image is of a label.

11.4 Softmax Regression Loss Function

Similar to the logistic regression loss function, Softmax regression records the losses which occur over M epochs and displays it like shown in figure (5). Although they are similar, Softmax regression utilizes a different entropy loss function to calculate the losses which occur. Additionally, the graph on Softmax appears to have most data points than the logistic regression loss graph. We hypothesize that since Softmax regression is a beefy algorithm which has a large runtime, it's able to collect more data overall and display a more complete curve. Finally, similar to logistic regression, we found that the only parameters which increased accuracy was lowering the learning rate, lowering batch size, and having a high number of principle components.

11.5 Softmax Regression Accuracy

After implementing the Softmax regression method, we found out that our program had an accuracy of about 75% with optimal parameters shown in section 10. Additionally, we discovered that the runtime of Softmax when attempting to find the most optimal parameters was drastically slower than logistic regression, and we hypothesize that this is the case because of the need to update k different weight classes. When attempting to debug the Softmax classifier, we thought it was interesting to see that the different weights in the k_weights vector had unique accuracies which contributed to the overall accuracy of all the images.

11.6 Softmax Regression Gradient descent

For the Softmax Gradient descent, we used k_weights since Softmax is generalization of logistic regression, over multiple classes. Each individual classification will have corresponding weight, to create the k_weights matrix. As you feed the algorithm more data sets of the images and their classification, the computer selectively modifies the k_weights matrix and selectively modify the weights which were misclassified. The learning rule for Softmax Gradient descent is the same for LR.

12 Team contributions

12.1 Jacky Dam

In the project, I primarily focused on the mathematical derivation of the different equations that we used in the algorithm, and focused on learning why mathematical operations make sense in the implementation. Additionally, I debugged numerical errors and miscalculations which came up from the program.

12.2 Ali Zaidi

In the project, I worked on coding parts of the Softmax/LR cross validation and Gradient Descent. Also worked on plotting the graphs and calculating the test accuracy. I worked with Jacky to debug our program.