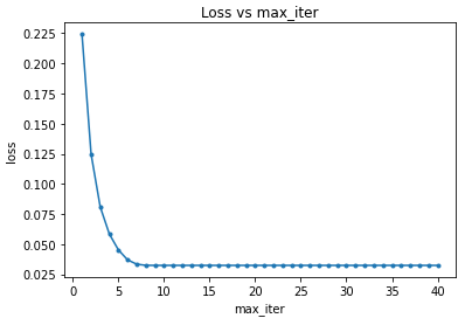
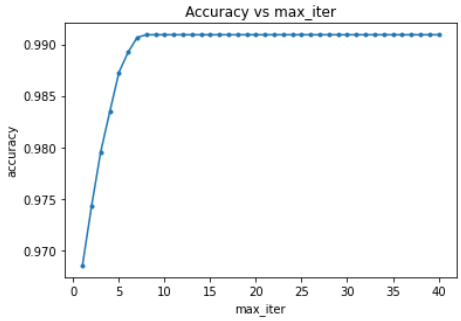
Hannah Doherty
CS135 – Project 01

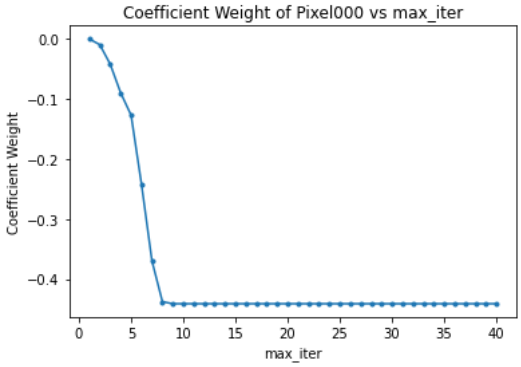
Part One: Logistic Regression for Digit Classification

1. In these plots we are showing the accuracy and loss of the training dataset when it is fit using logistic regression models with only the liblinear solver and varying values of max_iter. We can see for accuracy that as the number of iterations increase, the accuracy also increase. And we can see for loss that as the number of iterations increase, the loss decreases which makes sense because as the number of iterations increases we are likely to converge to a solution with decreased loss and better accuracy.





2. In this plot we have accessed the weight assigned to feature pixel000 in the dataset and plotted it for each of the max_iter models. We can see that as the number of iterations increases, the coefficient weight for Pixel000 decreases. This is because with each iteration we increase the coefficient weight until it reaches the optimal weight that will get the data to its desired classification, either 0 or 1. In the case of Pixel 000, it plateaued when its weight reached -0.5.



3. When a series of models with a range of values of C is used, then it is possible to find the best value of C which is 0.0316. The accuracy score of the model is 0.9909 (at that value of C). Below is the confusion matrix of that model on the test data.

Table: Confusion Matrix			
		Predicted	
		0	1
True	0	923	51
	1	53	956

4. Although this is the best model based on the results, there are some mistakes in it. Below are images of the false positives and false negatives. The mistakes shown in the results are the bottom hook on the 9's for some of the false negatives could be mistaken as an 8's especially

with some of the white dot pixels scattered in the background. For the false positives, some of the bottom circles of the 8's are thin or smaller than the top circle in the 8's which can confuse the model with thinking that they are 9's.

Figure Caption: False Positive Examples

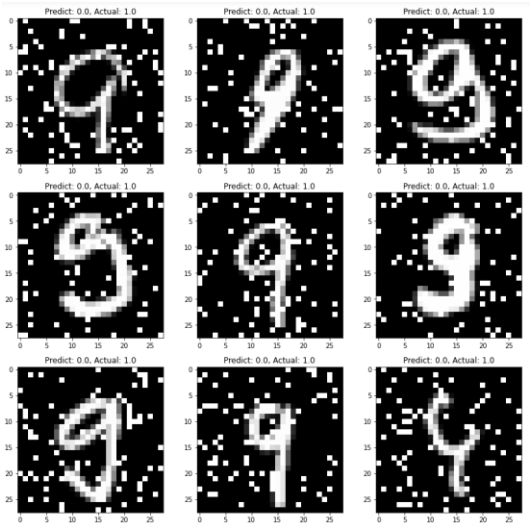


Figure Caption: False Negative Examples

5. This is an image of the reshaped weight coefficients. These show that the pixels that get updated in the positive direction go to the blue color and if the weights get updated in the negative direction then they go to the red color. The pixels that correspond to an 8 (have negative weights) are red. The pixels that correspond to a 9 (have positive weights) are blue. The red pixels are considered 8's because 8's are given the output value of 0 which means the weights will be more negative which are given the color red. The 8's can be classified easiest and the most in the quadrant that is red while the 9's can be classified easiest in the quadrant that is blue. Light colors mean neutral to classifier. Since 8's usually consists of a smaller top circle over a larger bottom circle, it makes sense the 9's have a strong blue area shaped like a concave region. It also makes sense that the 8's have a strong red region in the subsection of pixels (or features) that 9's usually lack.

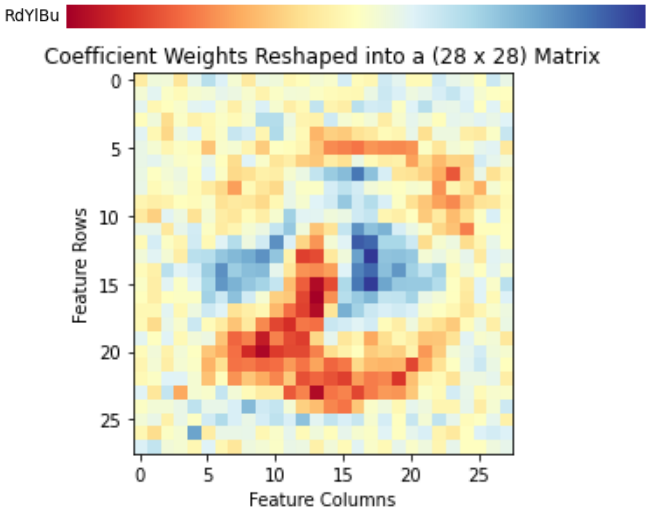


Figure: Heat Map of Weights of each Feature for Training Dataset

Part Two: Sneakers versus Sandals

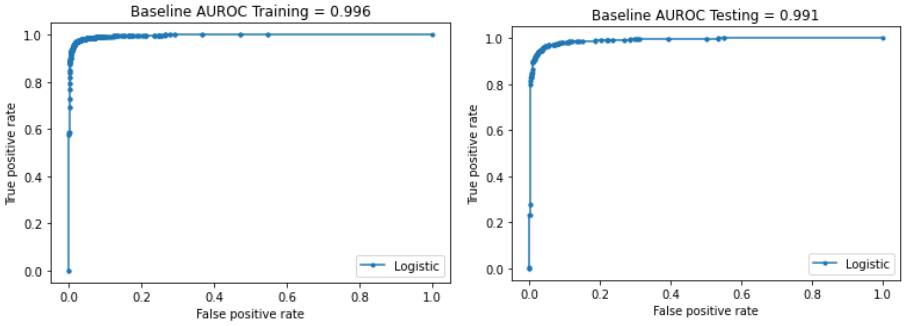
Overview

The provided image data was in the format where each row of the input data consists of pixel data from a (28 x 28) image with gray-scale values between 0.0 (black) and 1.0 (white). This pixel data is represented as a single feature-vector of length $28^2 = 784$ values. The output data is a binary label, with 0 representing a sneaker, and 1 representing a sandal. Given the input and output data for a training set, I was able to use a few supervised learning techniques to learn more about the dataset.

To get an idea of what the data looks like I started by plotting several of the data points to see what the images looked like and give a visual. The image to the left is of the training data set reshaped into an (28 x 28) Matrix. The first task will be to build the optimal logistic regression classifier to use on the testing dataset. We begin by looking at the weight coefficients and use cross validation to look at the hyperparameters.

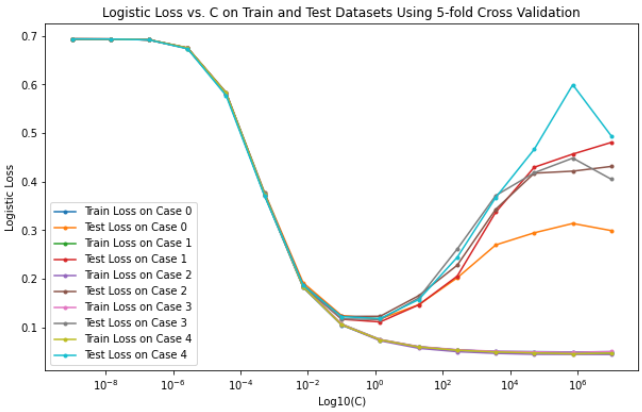
Analysis of Logistic Regression Parameters

In order to get uniform results on the 5-fold cross validation for the baseline model, I set the shuffle and random_state parameters in order to stay as fair and consistent throughout all the tests I'd be doing. Then I created a 5-fold cross validation baseline model for logistic regression with only the liblinear parameter set to see what the loss is on average for the training set and validation set. This allowed me to get an average fit on the training model and an accurate representation of accuracy and loss to use as the baseline model. Prediction probability was used to calculate the logistic loss. Gathering baseline data showed that changing the max iterations from 100 (the default) to 10,000 does not affect the logistic loss, accuracy, and AUROC for the average training and validation datasets. There was also no convergence warning but despite this, I decided to run all the tests in max_iter = 10,000 incase other tests ended up requiring more time to converge.



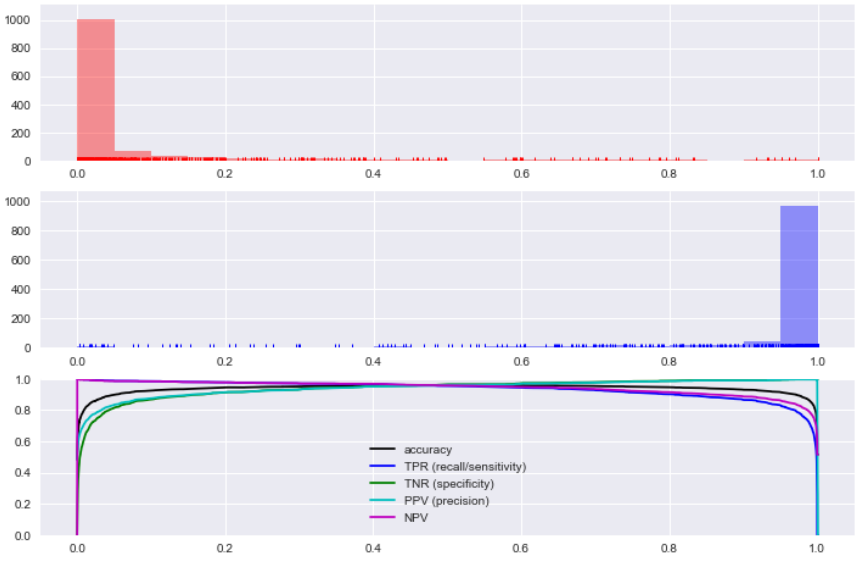
Since logistic regression has regularization set by default where the penalty = l2 and C = 1, I ran some tests to see what the best value for C would be and if the penalty could improve on the loss and accuracy. To do this, I ran another 5-fold cross validation test with a range of C values (on the log scale) from 10^{-6} to 10^{9}. I stored the loss and accuracy for each model of C. This showed a typical plot where the validation data's loss was of convex shape and the training data continued to decrease it's loss as it was becoming overfit due to the decrease in the regularization strength. The values for C = 1, C = 10, C =

0.774, and C = 0.1 are in Table 1. I chose a value smaller and larger than ideal just to see how well the loss and accuracy would perform. The ideal value for C showed to be 1. For C = 0.1 and C = 1, I uploaded these models to Gradescope to see how the test data would perform. The C = 0.1 model performed better based on the Error Rate and AUROC values even though my own cross validation analysis said otherwise. The only explanation for this would be if the test data is very different compared to the training data due to data that the training dataset could not predict previously. Because of this, I chose to upload results into Gradescope using C = 0.1 while using C = 1 for the cross validation tests. This is because the regularization I found to be sufficient in the cross validation (C = 1) of the training set model did not regularize the data well enough.

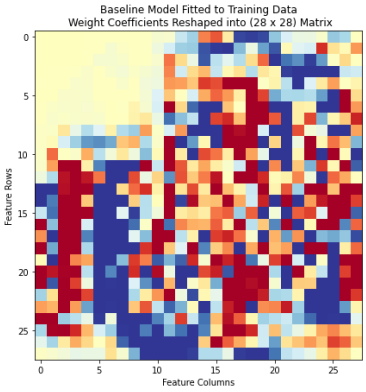


After playing around with the C parameter in the logistic regression model, I started to look at penalty. I changed the penalty from l2 to l1. The l1 had slightly worse loss and accuracy for both the training and testing sets so I decided to stick with the default l2 value. These tests on the hyperparameters for the regression model taught me that the default values do an excellent job of fitting the training model.

Once the ideal parameters were set for the logistic regression model, I looked into the right threshold. We can look at the y training dataset and see that it's perfect split between sandals and sneakers by calculating the TPR, TNR, PPV, NPV and Accuracy over a range of predict probabilities because then we can know for sure if the threshold should be set to 0.5. The figures below are connected because they all correspond to each other. Their x-axes are all the predicted probability of the data's classification. The top histogram (red) is the number of class-0 entries grouped by the probabilities assigned by the predictor. The second histogram (blue) shows the number of class-1 entries. When the threshold is set to 0.5, the bottom plot shows that our TPR, TNR, PPV, and NPV have all been maxed out. The accuracy is 0.961, TPR = 0.956, TNR = 0.965, PPV = 0.963, and NPV = 0.959.

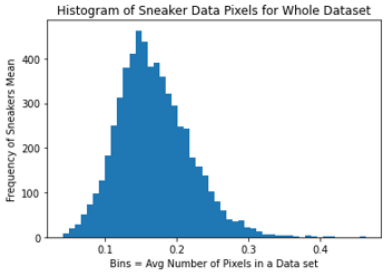


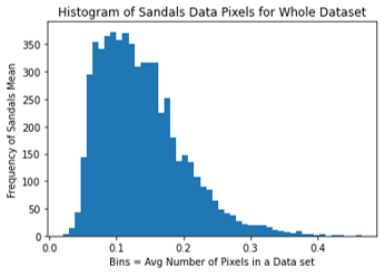
Feature Transformation



I was hoping to find hyperparameters that would improve loss and accuracy but I am content that the whole analysis showed that the default values are the best options. After figuring out the best possible hyperparameters for the logistic regression model, I started looking into creating the best possible add-on feature for this classifier. In my mind, it made sense that in addition to the given dataset, I would want to add more information that would help the model classify the data. The input data is on a scale of values from 0 to 1 representing a gray scale, and the output data is a label of 0 or 1. One way to classify data would be to find the features (or pixels) that on average give big disparities in data. In other words, features that are heavily weighted will be the ones definitively categorized as one class or the other. The way I started to do this was by looking at a colormap of the weight coefficients. The colormap can be used to determine which features have been updated negatively (going towards sneakers) or positively (towards sandals). The range of colors show how the model prefers certain classifications based on the shade of the color.

The color map shows that the dark red areas are where pixels (or features) are very obviously sneakers. While the dark blue areas are sandals. As you can see, the darkest reds and blues are heavily disbursed in all four quadrants except the top left. By calculating the average value of the pixels in the dark-red and dark-blue areas then the model will have an easier time classifying what is a sneaker and what is a sandal. I chose to focus on the center third section of the image due to the cluster of largest weights for both classes.



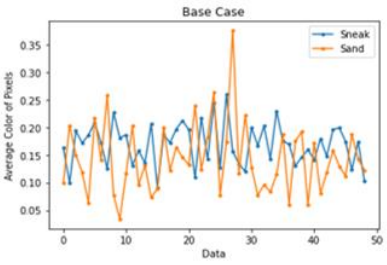


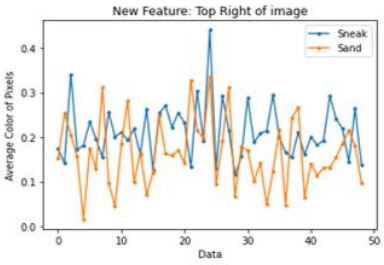
In addition to the colormap, the plots shown above are of the average value of the pixels for all of the datapoints and plotted in a histogram. It shows that sneaker values can be consistently distributed around a slightly higher mean while sandals data is skewed to the left around a lower mean. There's so much crossover that the data is not easily separated by the mean but could be separated by heavily weighted features.

Using the information from the colormap, I had the idea that if I added a feature that combined the most distinguishing features from the original dataset then it would increase the accuracy and decrease the logistic loss. With this logic, I created a feature that consisted of the center third section of the images that has the easiest time classifying data. Since the input, output, and new feature are all in the range of (0,1), I decided not to scale it since it would not be adding anything except time to the process.

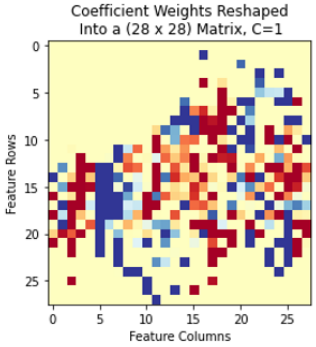
To help show how certain groups of features (a subset of pixels) can on average show differences in data, shown below are plots of the average value of the pixels (or features) for the whole image as well as the best performing quadrant. These plots are of the training set data of the first 50 images. In the table you can see that the top right section has the best and biggest disparity between average values.

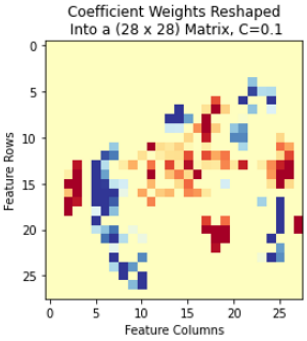
At first I thought I could add a feature that consisted of a subset of the image with the average values calculated to help the testing data predict it's classification. My logic came from thinking if this subset mean is added as a feature then it would package all the most important features together to help the classifier determine how the image should be classified.





Instead of using a quadrant, I chose a bigger area as the feature which was was the center third of the image since it encompasses the majority of the largest weighted features. After I added the new feature, I ran the 5-fold cross validation (with default penalty, default solver, C = 1, and max_iter = 10k) and acquired the results shown in Table 3.
Unfortunately, this additional feature did not improve the loss and accuracy. To understand why this might be, I tried another new feature. The next add-on feature that I tried was a repeat of the features 5 times (increasing to 5488 features total). It improved the loss for training and testing data but worsened the accuracy data.





Then I tried data transformations. I tried focusing on the areas that were heavily red and blue and deleting the others. Ideally if I had coded a function that was able to find the best subset of features then that would give the best logistic loss and accuracy and then I will have found the optimal solution. There are two ways that could be time efficient. First, I thought I might be able to improve on the loss and accuracy if I found the optimal features to use in a subset of the size 580. When I delete all features with low values that are less than or equal to 0.078 (using absolute value), then I end up with a total of 580 features left in the dataset. Of these 580 features, if the worst performing 10 were removed and then the best performing 10 were added then this strategy could possibly find a more optimal solution. The second option to finding the best subset is by changing the penalty from the default l2 to l1. By changing the penalty from l2 to l1 it allows the model to zero out certain features with low weights that are not adding anything to the process. Because as C increases, so does the regularization which removes any low weight features. But as previously stated in the "Analysis of Logistic Regression Parameters" section, changing the penalty from l2 to l1 did not improve the results unfortunately.

Conclusion

The final results show that the loss, accuracy, AUROC for training were optimal when using the default values given by logistic regression. By looking at the weights in the colormap and the average feature values in a histogram, patterns in the pixels can be seen between the sneakers and sandals data. Adding features did not end up improving the testing data's performance but deleting data and creating subsets did.

When I applied the "delete features" model to the testing data and uploaded it onto the leaderboard on Gradescope I noticed that the AUROC performed better than my cross-validation results but the accuracy performed much worse. This made me think that the test data is very different than the training set data that I had been using for the cross-validation predictions. I tested this by changing C=0.1 to strengthen regularization and then uploaded the results to the leaderboard. This resulted in the highest accuracy yet and a AUROC that was about the same. This shows that I had been relying on the training set data too much and probably overfit it. After learning the analysis wrongly shows this is the ideal C, I figured I needed to strengthen C before uploading it to the leaderboard in case my data is too fitted to the training set. In conclusion, I learned that even though cross-validation is a great way to find the best value for C, I still need to keep in mind that overfitting is a potential possibility and be weary of it.

Appendix

Table 1: Comparisons Between Logistic Regression Parameters

(On Average for Cross Val.)	Penalty = l2 Max_iter = 10k C = default, 1	Penalty = l2 Max_iter = 10k C = 0.774	Penalty = l2 Max_iter = 10k C = 0.1	Penalty = l2 Max_iter = 10k C = 10	Penalty = l1 Max_iter = 10k C = 1
Train Loss	0.07658	0.07891 ↑	0.10538 ↑	0.06170 ↓	0.07862 ↑
Valid. Loss	0.11490	0.11386 ↓	0.12029 ↑	0.14193 ↑	0.11540 ↑
Train Accuracy	0.97394	0.97306 ↓	0.96571 ↓	0.97860 ↑	0.97317 ↓
Valid. Accuracy	0.96067	0.96042 ↓	0.95850 ↓	0.95617 ↓	0.96067 ↓
Train AUROC	0.996	0.996	0.994	0.998	0.996
Valid. AUROC	0.991	0.991	0.991	0.990	0.991

Table 2: Analysis of Mean Feature Values in Base Case and Quadrants

	Difference between Average Sneaker and Sandal values for first 50 datapoints in training set
Baseline Case: Full image	0.026
Top Left	0.043
Top Right (best section)	0.048
Bottom Left	0.001
Bottom Right	0.006

Table 3: Analysis of New Features and Deleted Features Compared to Base Case

	Penalty = l2 Max_iter = 10k, C = 1	New Feature (center third)	New Feature (x5 features)	Delete all features with values <=0.078
Train Loss	0.07658	0.07578	0.06347	0.07712
Valid. Loss	0.11490	0.11491	0.13537	0.11005
Train Accuracy	0.97394	0.97389	0.97802	0.97423
Valid. Accuracy	0.96067	0.9605	0.95667	0.96158
Train AUROC	0.996	0.996	0.997	0.996
Valid. AUROC	0.991	0.991	0.990	0.992