ML 135 Project 1

Name: Chris Mitsopoulos \ Date: 10/24/2019

Part 1: Logistic Regression for Digit Classification

1.1: Logistic Regression with limited iterations

Allow iteration values 1 - 40, record accuracy and log loss.

Plot accuracy/loss versus max_iteration value

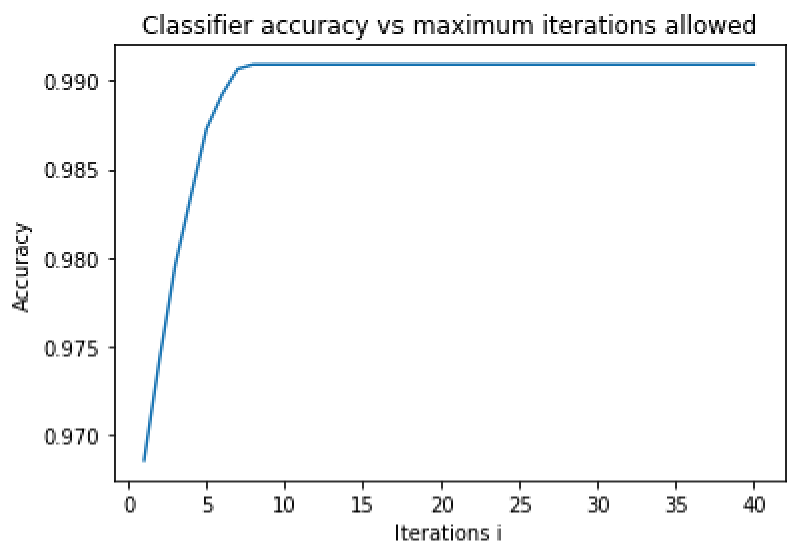
In [4]:

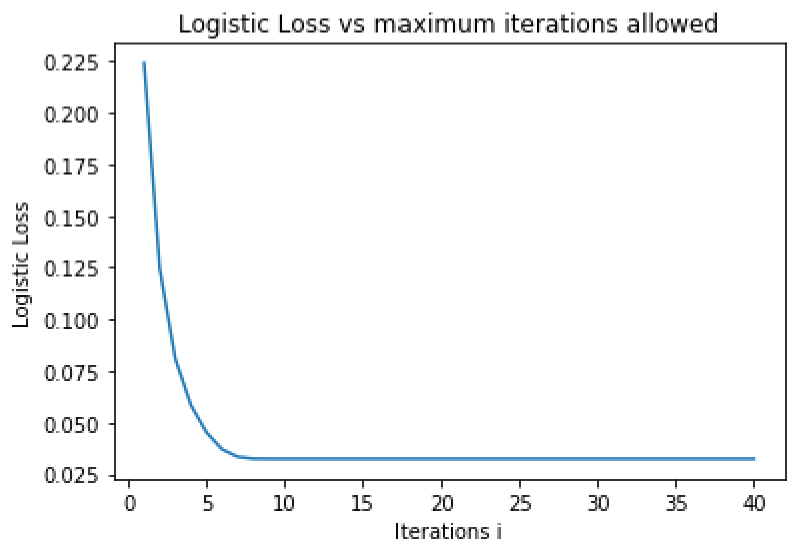
```

```

Out[4]:

```
[<matplotlib.lines.Line2D at 0x10fcfb510>]
```





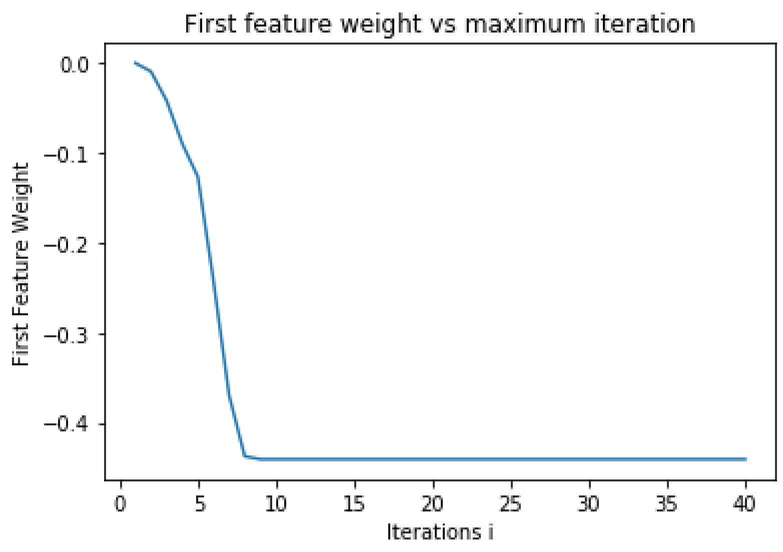
As you can see in the plot of classifier accuracy, for low maximum iteration values (i) the accuracy is low. Initially as the number of iterations increase, we see a rapid increase in accuracy (steep positive slope). At approximately i = 8 the slope decreases significantly and converges to 0 meaning that the classifier accuracy peaks. This happens because the number of iterations are enough for the solver to converge. What it means for a classifier to converge to a maximum accuracy is that internally, its loss function is minimized so weight updates stop. This is why for any value of i after the point of convergence, accuracy does not increase. Similarly, the logistic loss of the model (seen in the second figure) starts high for low iterations and decreases rapidly as iterations increase up to the point of convergence to a minimum loss value. Converging to a minimum logistic loss is equivalent to converging to a maximum classifier accuracy.

1.2: First feature weight inspection

```
In [5]:
```

```
Out[5]:

[<matplotlib.lines.Line2D at 0x1a3201eed0>]
```



Initially for low i values, the coefficient/weight of the first feature (pixel 000) is close to zero. As iterations increase, but before the solver is able to converge to a solution, we see that the feature weight is updated repeatedly by being decreased. When the model converges, or in other words weight updates fall below a certain threshold, we expect the feature weights to remain at a consistent value. That is what happens at the convergence point of about i = 8, where the slope of the curve flattens and falls to zero, and the weight of pixel000 ceases to change. Any classifiers that use max_iteration values greater than that at the convergence point will have the same feature weights after that point.

1.3: Exploring effect of inverse penalty strength

Explore the effect of C on the log loss and accuracy of the model. Record best model along with important metrics such as confusion matrix. For C, use a regularly-spaced grid of values.

```
In [6]:
```

```
In [7]:
```

```
Best C-value: 0.03162

Testing set log-loss at best C-value: 0.0897

Testing set accuracy at best C-value: 96.72%

Predicted    0    1
True
0          942   32
1           33  976
```

1.4 Analyze FP and FN

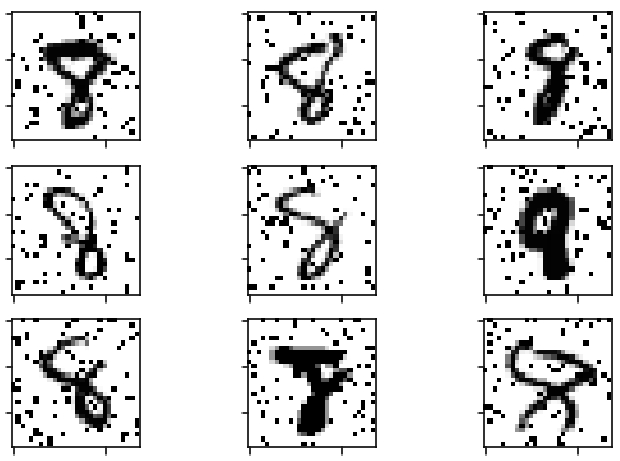
In [8]:

```
```

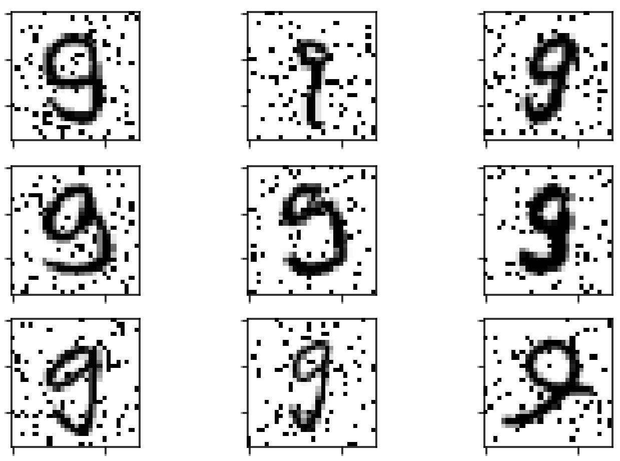
In [10]:

```
```

Examples of False Positives in test set



Examples of False Negatives in test set



Examples in the first figure correspond to false positives (8's that were wrongly classified as 9's). In most of these, the lower "loop" of the 8 digit is not a perfect circle (and in some of them the pixel values even resemble a vertical line rather than a loop). This makes it resemble the pixel values of the lower curve in an example of a 9 digit. Because of these deformities, the classifier believes that there is a higher probability that these belong to the nine class rather than the 8 class.

We can make a similar argument for the examples in the second figure which correspond to false negatives (9's that were wrongly classified as 8's). If one inspects a couple of correct examples of 9's in the dataset the lower region of a random 9 example is basically straight. In the examples seen in the second plot however the lower region is often curved, which according to the classifier, matches the pixel values in an 8 example leading to a misclassification.

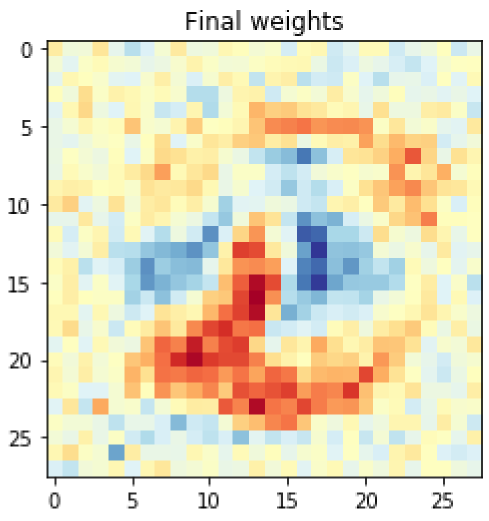
1.5 Final Model Weight Inspection

In [31]:

```
```

Out[31]:

```
Text(0.5, 1.0, 'Final weights')
```



The pixels that are shaded blue are the ones that correspond to a 9 and have positive weights. The pixels with an orange/yellow shade correspond to an 8. Most of the orange/yellow shaded pixels (negative weights-8's) are clustered at the lower-central region, which is where the lower loop of an 8 would be represented by pixel values. As you can see the color shade is more intense at that region because the magnitude of those weights are high, since this region is very significant for the shape of the 8. Similarly, I predict that the region with the darkest shade of blue corresponds to the vertical "body" part of a 9 figure, so the model weighs these pixels highly in magnitude as training pixel values for 9's in this region were consistently high.

In [ ]:

```
```

Part 2: Sneakers vs Sandals

In this part, we will perform various data exploration and feature selection techniques to build a binary classifier using logistic regression, that predicts whether an input image is a sandal (1 if true). We will start with a base model containing all features, and lead an exploration with the goal of minimizing the error rate of the classifier through feature selection and incremental model improvement.

Base Model

We start by building a Logistic Regression model for all features (pixels). We will call this model the base model, and its accuracy will be considered a lower bound for all future models. Here are some metrics of the base model for the training set and testing set:

ACC	0.973
TPR	0.971
TNR	0.976
PPV	0.976
NPV	0.971

Figure 1: Base Model Metrics for training set.

Predicted	**0**	**1**
True		
0	5857	143
1	175	5825

Figure 2: Confusion Matrix for training set

AUROC	**ERROR RATE**	**SCORE**
0.9932	0.0425	9.372

Figure 3: Classifier results for testing set

As you can see in Figure 3, the performance of the base model classifier is actually pretty good. What we can do to improve the model, is either employ data analysis techniques to add extra meaningful features to our training and testing sets that might decrease the error rate, or we can use techniques such as recursive feature elimination or L1 regularization to decrease the number of features used for classification, which will make the model faster to run and work with, or we can try adding more examples to our training set. We will start by performing data augmentation.

Data Augmentation: Increase training size

Let's attempt to increase the training size, to see if we have any major improvements. To do this, we perform horizontal flip on all images on the training set, and we append them to the end of the training set. Note, that we also have to append elements to the classifier output file y_train. Let's print two examples in dataset to see if horizontal flip occured properly:

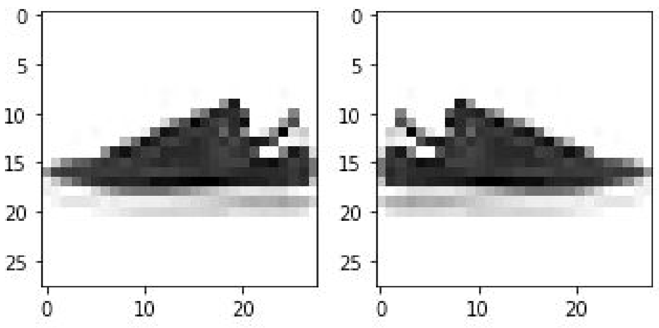


Figure 4: Example of horizontally flipped image in dataset

Since, horizontal flip functions as wanted, lets fit the model. Here are the confusion matrix and accuracy results:

Predicted	**0**	**1**
True		
0	8395	3605
1	1626	10374

Figure 5: Confusion Matrix for training set

AUROC	**ERROR RATE**	**SCORE**
0.9839	0.0555	6.126

Figure 6: Classifier results for testing set

Doubling the training size increased the error rate of the classifier. We can see from the confusion matrix in Figure 6 that the model is subject to a high False Positive rate, signifying that the model often misclassified sneakers as sandals. It turns out that the extra examples lead to misclassifications with a large number of FP and FN, so other methods need to be tested.

Data Augmentation: Add non-zero count as feature

If someone inspects the training and testing datasets, he will see that most of the pixel values are usually 0, except at the center of the image, where the shoe usually resides. So, it might be helpful for the classifier to add an extra feature to the training sets, that keeps track of the number of pixels that are non-zero in each image. Fitting a baseline model with this extra feature, leads to the following results:

AUROC	ERROR RATE	SCORE
0.9946	0.0375	10

Figure 7: Classifier results for testing set

As you can see, the error rate of the classifier decreased and the area under the ROC curve increased, making this the best model yet.

Feature Elimination using RFE

Because the images in the dataset are of size 28x28 pixels, the datasets consist of examples with 784 features. So many features lead to bottlenecks in classifier performance, so finding ways to minimize the number of features used, is prudent in minimizing the time taken for the classifier to run. I tried many methods in the hopes of reducing the number of features, starting with Recursive Feature Elimination through cross-validation. This validator, ranked all 784 features by performing stratifiedKFold for 5 features at a time with the scoring of accuracy. Through this method, 520/784 features were kept for the classifier. The results were not that bad but worse than the accuracy achieved by adding the extra feature. However the drawback of this method is that for a large number of features it is very computationally expensive as it goes through various feature combinations, so I didn’t choose this method alone to get the best classifier.

Feature Selection using L1 Regularization

Another method I found to eliminate some features was L1 regularization (or Lasso regularization, which shrinks some weights to zero in order to eliminate some features). I built this classifier using the selectFromModel object from sklearn, which shrunk the weights of 474 features to zero. The results on the testing set were as follows:

AUROC	ERROR RATE	SCORE
0.993	0.0405	9.877

Figure 8: Classifier results for testing set

As you can see the classifier performs worse than that which uses all features and has extra feature which counts the number of nonzero pixels.

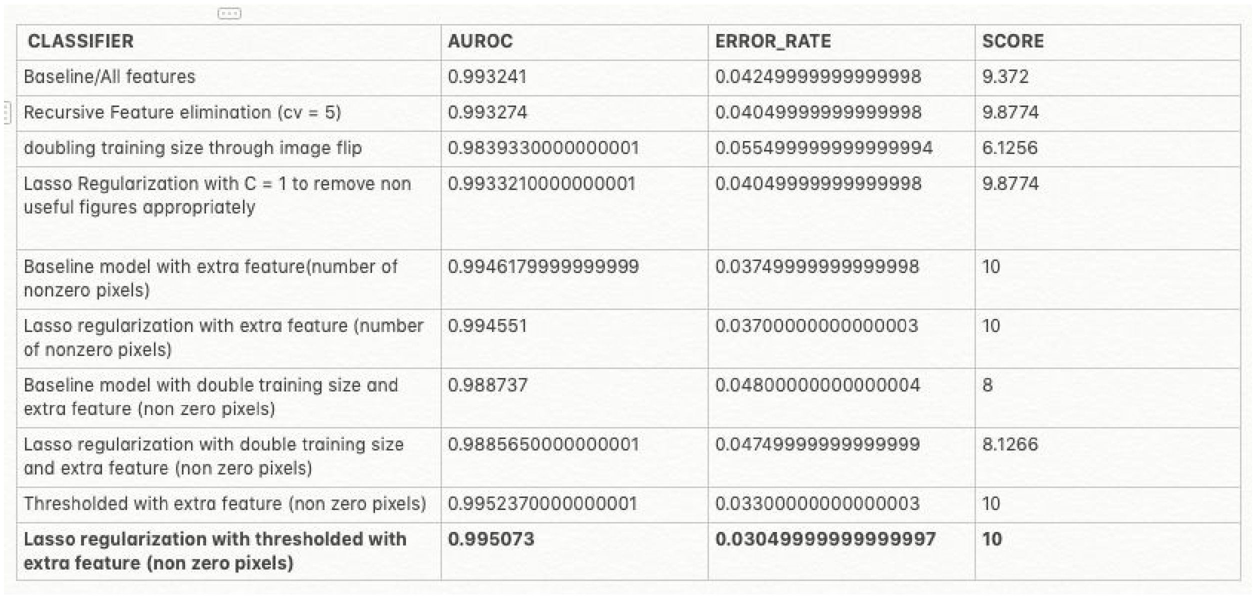
Data Augmentation: Threshold values

I thought it would be cool to see what results we would get from our classifiers, if I went through all the testing and training x_values, and thresholded values to be 1 if non-zero and 0 else. Our examples are now all binary, except for the last feature which counted the number of nonzero pixels in an example, and I performed logistic regression on them. The results were really good, and the best model to date, as there was a significant reduction in the error rate:

AUROC	**ERROR RATE**	**SCORE**
0.995	0.033	10

Figure 9: Classifier results for testing set

The rest of the work done to find the best model was just trying to combine the best ideas from all the above combinations, and choosing the one with the greatest area under the ROC curve and least error rate. Thus, this process was one of trial and error. I have organized my results in the table below:



CLASSIFIER	AUROC	ERROR_RATE	SCORE
Baseline/All features	0.993241	0.04249999999999998	9.372
Recursive Feature elimination (cv = 5)	0.993274	0.04049999999999998	9.8774
doubling training size through image flip	0.9839330000000001	0.05549999999999994	6.1256
Lasso Regularization with C = 1 to remove non useful figures appropriately	0.9933210000000001	0.04049999999999998	9.8774
Baseline model with extra feature(number of nonzero pixels)	0.9946179999999999	0.03749999999999998	10
Lasso regularization with extra feature (number of nonzero pixels)	0.994551	0.03700000000000003	10
Baseline model with double training size and extra feature (non zero pixels)	0.988737	0.04800000000000004	8
Lasso regularization with double training size and extra feature (non zero pixels)	0.9885650000000001	0.04749999999999999	8.1266
Thresholded with extra feature (non zero pixels)	0.9952370000000001	0.03300000000000003	10
Lasso regularization with thresholded with extra feature (non zero pixels)	**0.995073**	**0.03049999999999997**	**10**

Figure 10: Final table comparing results of all models tested

The “best” model that I was able to design through trial and error, uses many of the techniques described above together. More specifically, an extra feature was added which represented the count of non-zero pixels. The values of all the other features were then thresholded to 0 and 1, with the process described above. After this, lasso regularization was employed to eliminate a number of features that were not critical in classification. Through all these combinations, the error rate decreased to 0.0305 with an AUROC of 0.995, metrics which seem satisfactory for the task at hand.