ECE 174 Project 1 Report

Pranav Reddy
University of California, San Diego
p3reddy@ucsd.edu

Abstract

We built the multi-class classifiers as specificed by the writeup. In the first part, we construct the binary classifier and use that to build a one-versus-all classifier and a one-versus-one classifier. We note that the one-versus-one classifier has markedly better performance, but also takes much longer to train as well and is in general much larger. In the second part we utilize four randomized feature mappings: identity, sigmoid, sinusoid, and ReLU, and compare the performance of each on both classifiers. We find that the randomized feature mappings produce better performance with both types of classifiers, but do add to the training time.

1 Least Squares Classifier

In this section we develop an explanation of our code and the theory behind the classifier.

1.1 Building the Binary Classifier

We want to form the normal equations from the minimization problem that we were given. We are given the following least squares problem.

$$\min_{\beta,\alpha} \sum_{i=1}^{N} \left(y_i - \beta^\top \mathbf{x}_i + \alpha\right)^2 \tag{1}$$

Then, we can format 1 as

$$\begin{aligned}
\min_{\beta,\alpha} \left\| \begin{bmatrix} y_1 - \beta^\top \mathbf{x}_1 + \alpha \\ \vdots \\ y_N - \beta^\top \mathbf{x}_N + \alpha \end{bmatrix} \right\|^2 &= \min_{\beta,\alpha} \left\| \mathbf{y} - \begin{bmatrix} \beta^\top \mathbf{x}_1 + \alpha \\ \vdots \\ \beta^\top \mathbf{x}_N + \alpha \end{bmatrix} \right\|^2 \\
&= \min_{\beta,\alpha} \left\| \mathbf{y} - \begin{bmatrix} \mathbf{x}_1^\top \beta + \alpha \\ \vdots \\ \mathbf{x}_N^\top \beta + \alpha \end{bmatrix} \right\|^2 \\
&= \min_{\beta,\alpha} \left\| \mathbf{y} - \begin{bmatrix} 1 & \mathbf{x}_1^\top \\ \vdots & \vdots \\ 1 & \mathbf{x}_N^\top \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \end{bmatrix} \right\|^2 .
\end{aligned}$$

By relabeling the variables in the last equation, we obtain the normal equations

$$\begin{bmatrix} 1 & \mathbf{x}_1^\top \\ \vdots & \vdots \\ 1 & \mathbf{x}_N^\top \end{bmatrix}^\top \mathbf{y} = \begin{bmatrix} 1 & \mathbf{x}_1^\top \\ \vdots & \vdots \\ 1 & \mathbf{x}_N^\top \end{bmatrix}^\top \begin{bmatrix} 1 & \mathbf{x}_1^\top \\ \vdots & \vdots \\ 1 & \mathbf{x}_N^\top \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \end{bmatrix} \tag{2}$$

For brevity we will write this as

$$X^\top \mathbf{y} = X^\top X\theta \tag{3}$$

where θ in 3 provides the optimal α and β for 1.

1.2 One-Versus-All Classifier

The code implementing all the classifiers is contained in a GitHub repository that is linked in the references. For speed, we first precompute $(X^\top X)^\dagger X^\top$ and the multiply by every binary relabeling for speed.

```
XTX = np.matmul(trainX.transpose(), trainX)
XTX_pinv = np.linalg.pinv(XTX)
XTX_pinv_XT = np.matmul(XTX_pinv, trainX.transpose())

labels = np.arange(0, 10)

one_vs_all_model = []
for i in range(len(labels)):
    binaryY = np.copy(trainY)
    binaryY[binaryY != labels[i]] = -1
    binaryY[binaryY == labels[i]] = 1

    one_vs_all_model.append(np.matmul(XTX_pinv_XT, binaryY))
one_vs_all_model=np.concatenate(one_vs_all_model, axis=1)
```

For convenience, we stack the 10 binary classifiers into a single matrix, so that we can simply using matrix multiplication to generate predictions for the data.

1.2.1 Error Analysis

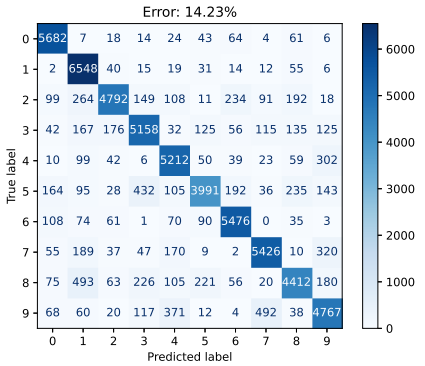


Figure 1: Confusion matrix for the training data.

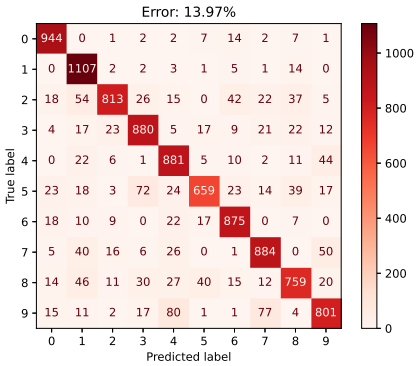


Figure 2: Confusion matrix for the test data.

We can see from the confusion matrix that the one-versus-all classifier has solid performance overall but confuses similar looking digits. The most common error was predicting 1 when the true answer was 8. On the other hand, 0 was very easily recognized, and 1 had the fewest false negatives.

For the test data performance, we see that the classifier performs well on the test data, even having a slight improvement in performance. The most common error was predicting 4 when the true answer was 9, and once again the model confuses similar looking digits, such as 3 and 5 or 7 and 9. Again, 0 was very easily recognized, and 1 had the fewest false negatives.

1.3 One-Versus-One Classifier

For the one-versus-one classifier, we arrange the $\binom{K}{2}$ classifiers in a 3D array in the following code.

```
    one_vs_one_model = np.empty((10,10,trainX.shape[1]))

    for i in range(len(labels)):
        for j in range(i+1, len(labels)):
            mask = ((trainY[:,0] == labels[i]) | (trainY[:,0] == labels[j
])))
            filtered_Y = trainY[mask]
            filtered_X = trainX[mask]

            filtered_Y[filtered_Y == labels[j]] = -1
            filtered_Y[filtered_Y == labels[i]] = 1
            model_ij = np.matmul(np.matmul(np.linalg.pinv(np.matmul(
filtered_X.transpose(),filtered_X)),filtered_X.transpose()),
filtered_Y)
            one_vs_one_model[i,j,:] = model_ij[:,0]
```

1.3.1 Error Analysis

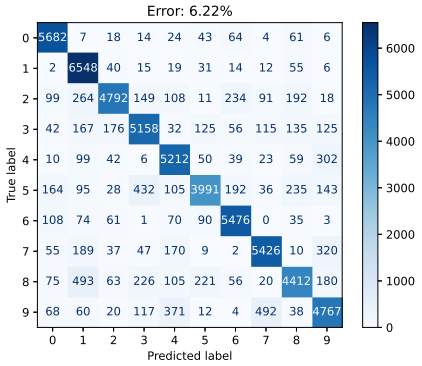


Figure 3: Confusion matrix for the training data.

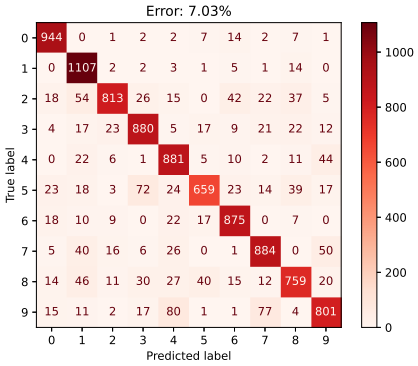


Figure 4: Confusion matrix for the test data.

We see that the one-versus-one classifier has improved performance on the training data when compared to the one-versus-all classifier. The most common error was predicting 1 when the true answer was 8, and once again the model confuses similar looking digits, such as 3 and 5 or 7 and 9. Again, 0 was very easily recognized, and 1 had the fewest false negatives.

The performance on the test data is slightly worse. The most common error was predicting 4 when the true answer was 9, and once again the model confuses similar looking digits, such as 3 and 5 or 7 and 9. Again, 0 was very easily recognized, and 1 had the fewest false negatives.

Overall, we can see that the one-versus-one classifier is a definite improvement over the one-versus-all classifier, but still makes very similar mistakes and errors.

2 Randomized Feature Least Squares Classifier

2.1 One-Versus-All Classifier

2.1.1 One-Versus-All Classifier Implementation

I implemented the randomized feature mappings with the following Python code: The functions take in randomized matrices W and b and apply the given mappings to the sum.

```
    def identity(x, W, b):
        n = (x.shape)[0]
        return np.hstack((np.ones((n,1)), np.matmul(x, W)+b[:n, :W.shape
[1]]))
    def sin(x, W, b):
```

```
        n = (x.shape)[0]
        x = np.matmul(x, W)+b[:n, :W.shape[1]]
        x = np.sin(2 * np.pi * x / 180)
        return np.hstack((np.ones((n,1)), x))
    def relu(x, W, b):
        n = (x.shape)[0]
        x = np.matmul(x, W)+b[:n, :W.shape[1]]
        x = x * (x > 0)
        return np.hstack((np.ones((n,1)), x))
    def sigmoid(x, W, b):
        n = (x.shape)[0]
        x = np.matmul(x, W)+b[:n, :W.shape[1]]
        x = scipy.special.expit(x)
        return np.hstack((np.ones((n,1)), x))
```

After that, I extracted the code to create the classifier into a few different methods, shown here:

```
    def create_model(x, y, labels):
        XTX = np.matmul(x.transpose(), x)
        XTX_pinv = np.linalg.pinv(XTX)
        XTX_pinv_XT = np.matmul(XTX_pinv,x.transpose())
        model = []
        for i in range(len(labels)):
            binary = np.copy(y)
            binary[binary != labels[i]] = -1
            binary[binary == labels[i]] = 1
            model.append(np.matmul(XTX_pinv_XT,binary))
        model=np.concatenate(model,axis=1)
        return model

    def create_model_error_analysis(trueY, predictions):
        error = 0
        for i in range(len(predictions)):
            if(predictions[i] != trueY[i]):
                error = error + 1
        error = (error / len(trueY)) * 100
        return error

    def predict_one_vs_all(x, model):
        predictions = np.matmul(x, model)
        predictions = np.argmax(predictions, axis = 1)
        return predictions

    def plot_confusion_matrix(trueY, predictions, labels, colors,
    feature_type, data_type):
        display=metrics.ConfusionMatrixDisplay.from_predictions(trueY,
    predictions,labels=labels, cmap=colors)
        plt.title(f'{feature_type} Feature Mapping {data_type} Data
    Confusion Matrix - Error: {create_model_error_analysis(trueY,
    predictions):.2f}%')
        plt.show()
```

2.1.2 Identity Feature Mapping

We can see that the identity feature mapping produces similar results to the one-versus-one classifier without the randomization. These results are subject to some fluctuations due to the non-deterministic nature of the mapping.

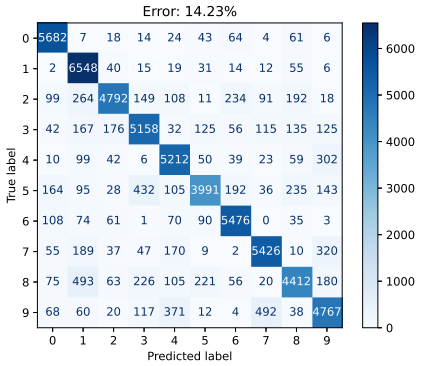


Figure 5: Confusion matrix for training data.

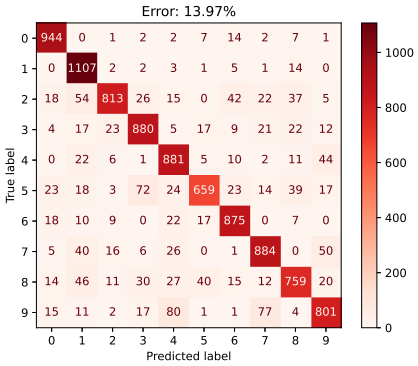


Figure 6: Confusion matrix for test data.

2.1.3 Sigmoid Feature Mapping

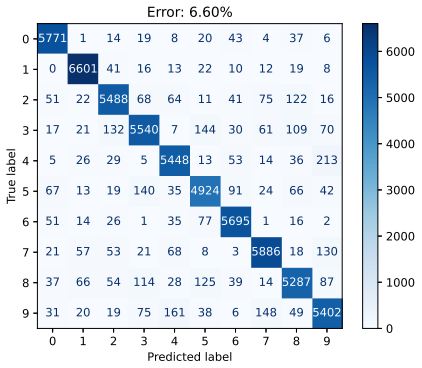


Figure 7: Confusion matrix for sigmoid training data.

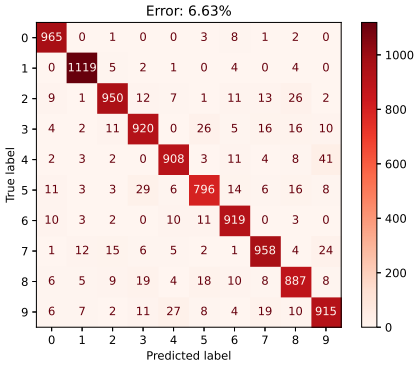
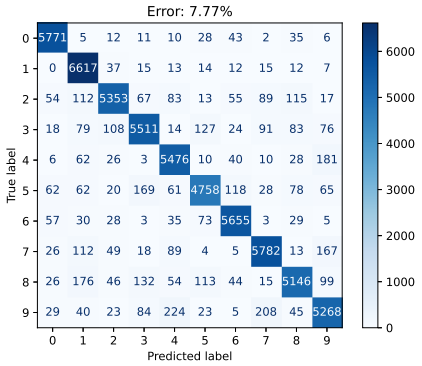


Figure 8: Confusion matrix for sigmoid test data.

The sigmoid feature mapping produces better results than the one-versus-one model without the feature mapping, although the same digit pairs (3-5, 2-7, 4-9) producee significant error. The generalization from the training data to test data is worse, as the increase in error is more than the other three feature mappings.

2.1.4 Sinusoid Feature Mapping

Once again, the same pairs produce error, but the increase in error from the training data to the test data is not as sharp as the increase for sigmoid, but it is more than the ReLU.



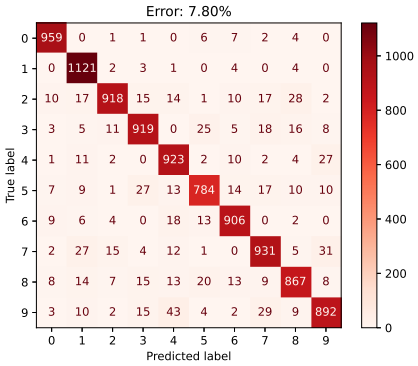
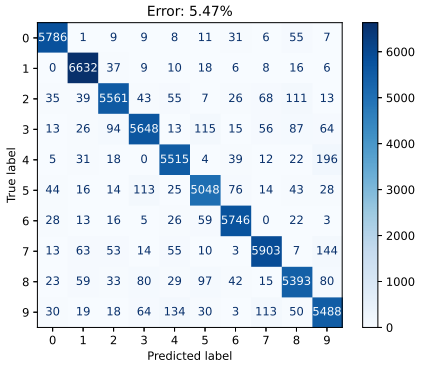


Figure 9: Confusion matrix for sinusoid train data.Figure 10: Confusion matrix for sinusoid test data.

2.1.5 Recitfied Linear Unit (ReLU) Feature Mapping



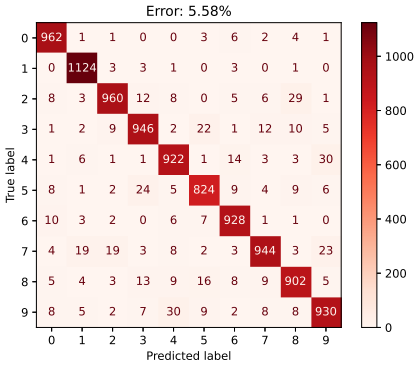
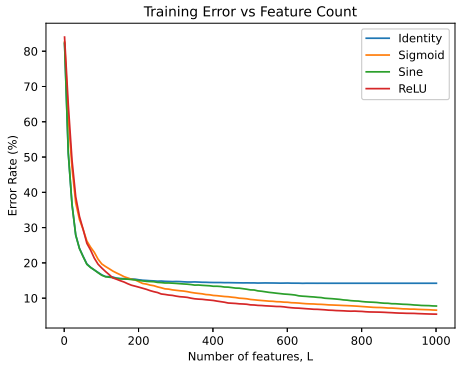


Figure 11: Confusion matrix for ReLU train data. Figure 12: Confusion matrix for ReLU test data.

The ReLU dataset still has the same error-prone pairs of numbers, but it performs the best out of all the feature mappings, and also has the least increase in error from training to test.

2.1.6 Effect of Features on Performance of One-Versus-All Classifier



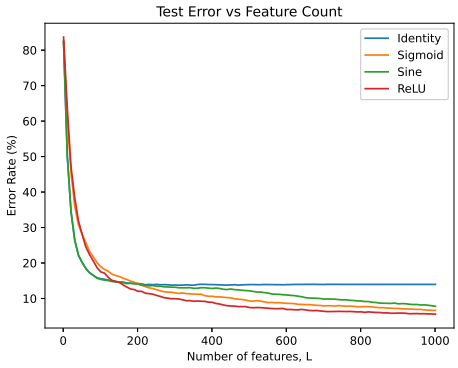


Figure 13: Training error rate vs feature count. Figure 14: Test error rate vs feature count

Table 1: Error Rates of Different Feature Mappings with One-Versus-One Classifier

Mapping	Training Error	Test Error
Identity	14.23%	13.97%
Sigmoid	6.60%	6.63%
Sinusoid	7.77%	7.80%
ReLU	5.47%	5.58%

I plotted 100 equally space values of L from 1 to 1001. We can clearly see that after a certain number of features, ~170, the rate of decrease in error sharply decreases, and we see diminishing returns on accuracy.

2.2 One-Versus-One Classifier

Similar to the one-versus-all classifier, I wrote some helper methods to assist in creating the models for each feature mapping.

```
    def create_model(x, y, labels):
        model = np.empty((10,10, x.shape[1]))

        for i in range(len(labels)):
            for j in range(i+1, len(labels)):
                mask = ((y[:,0] == labels[i]) | (y[:,0] == labels[j]))
                filtered_Y = y[mask]
                filtered_X = x[mask]

                filtered_Y[filtered_Y == labels[j]] = -1
                filtered_Y[filtered_Y == labels[i]] = 1
                model_ij = np.matmul(np.matmul(np.linalg.pinv(np.matmul(
    filtered_X.transpose(),filtered_X)),filtered_X.transpose()),
    filtered_Y)
                model[i,j,:] = model_ij[:,0]
        return model

    def create_model_error_analysis(trueY, predictions):
        error = 0
        for i in range(len(predictions)):
            if(predictions[i] != trueY[i]):
                error = error + 1
        error = (error / len(trueY)) * 100
        return error
    def predict_one_vs_one(x, model, labels):
        votes = np.zeros((x.shape[0], labels.shape[0]))
        for i in range(len(labels)):
            for j in range(i+1, len(labels)):
                prediction_ij = np.matmul(x, model[i,j,:])
                votes[prediction_ij > 0,i] = votes[prediction_ij > 0,i] +
     1
                votes[prediction_ij < 0,j] = votes[prediction_ij < 0,j] +
     1
        predictions = np.argmax(votes, axis=1)
        return predictions
    def plot_confusion_matrix(trueY, predictions, labels, colors,
    feature_type, data_type):
        display=metrics.ConfusionMatrixDisplay.from_predictions(trueY,
    predictions,labels=labels, cmap=colors)
        plt.title(f'Error: {create_model_error_analysis(trueY,
    predictions):.2f}%')
        display.figure_.savefig(f'/Users/pranavreddy/Desktop/ECE 174/
    Project1/images/one_vs_one_{data_type.lower()}_confusion_matrix_{
    feature_type}.eps')
        plt.show()
```

2.2.1 Identity Feature Mapping

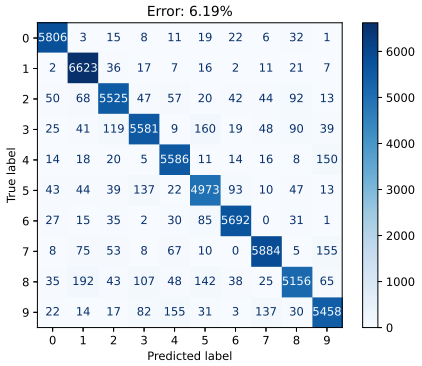


Figure 15: Confusion matrix for training data.

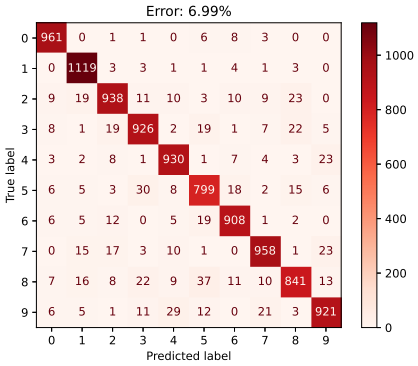


Figure 16: Confusion matrix for test data.

We can see that the identity feature mapping produces similar results to the one-versus-one classifier without the randomization. These results are subject to some fluctuations due to the non-deterministic nature of the mapping.

2.2.2 Sigmoid Feature Mapping

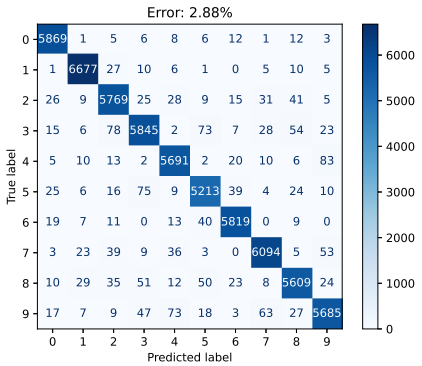


Figure 17: Confusion matrix for training data.

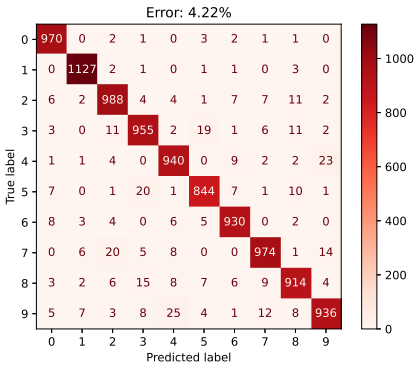


Figure 18: Confusion matrix for test data.

The sigmoid feature mapping produces better results than the one-versus-one model without the feature mapping, although the same digit pairs (3-5, 2-7, 4-9) producee significant error. The generalization from the training data to test data is worse, as the increase in error is more than the other three feature mappings.

2.2.3 Sinusoid Feature Mapping

Once again, the same pairs produce error, but the increase in error from the training data to the test data is not as sharp as the increase for sigmoid, but it is more than the ReLU.

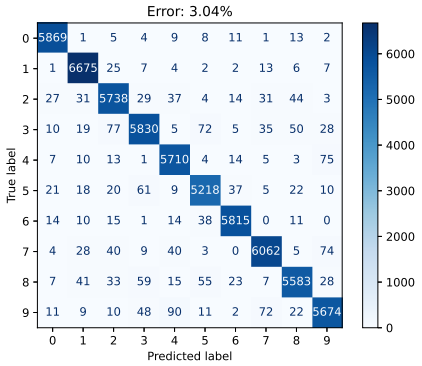


Figure 19: Confusion matrix for training data.

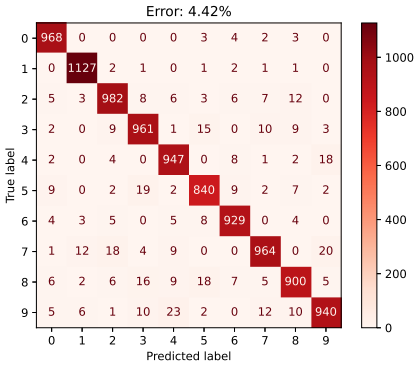


Figure 20: Confusion matrix for test data.

2.2.4 Recitfied Linear Unit (ReLU) Feature Mapping

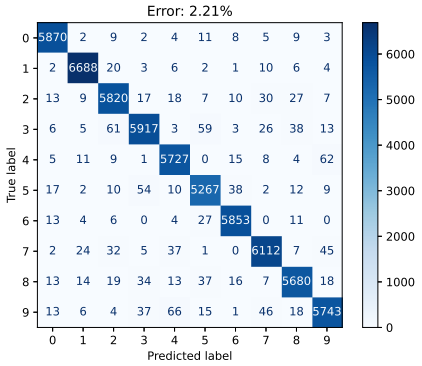


Figure 21: Confusion matrix for training data.

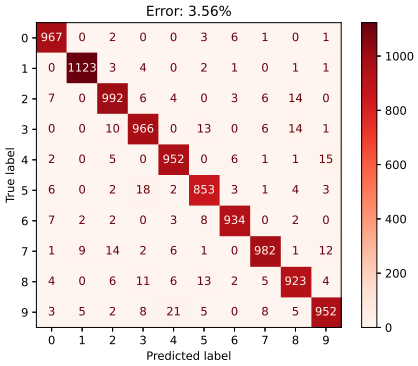


Figure 22: Confusion matrix for test data.

The ReLU dataset still has the same error-prone pairs of numbers, but it performs the best out of all the feature mappings, and also has the least increase in error from training to test.

2.2.5 Effect of Features on Performance of One-Versus-One Classifier

Figure 23: Training error rate vs feature count.

Figure 24: Test error rate vs feature count

Table 2: Error Rates of Different Feature Mappings

Mapping	Training Error	Test Error
Identity	6.19%	6.99%
Sigmoid	2.88%	4.22%
Sinusoid	3.04%	4.42%
ReLU	2.21%	3.56%

For the sake of time, I only plotted 10 equally space values of L from 1 to 1001. We can clearly see that after a certain number of features, ~170, the rate of decrease in error sharply decreases, and we see diminishing returns on accuracy.

References

[1] Api reference. URL `https://matplotlib.org/stable/api/index.html`.

[2] Numpy reference. URL `https://numpy.org/doc/1.26/reference/index.html#reference`.

[3] Scipy api. URL `https://docs.scipy.org/doc/scipy/reference/`.

[4] Api reference. URL `https://scikit-learn.org/stable/modules/classes.html`.

[5] S. Boyd and L. Vandenberghe. *Introduction to Applied Linear Algebra*. Cambridge University Press, 2018. ISBN 9781108583664.