ENGG 680: Introduction to Digital Engineering
Assignment 3

Adam Smith

November 21, 2022

Data Preprocessing for the Logistic Model

We begin by loading the data into a Google Colab noteboook as a pandas DataFrame and checking for missing values. The data appears to contain no `null` values, so there is likely no need to impute any data. Next, we need to encode the categorical attributes. The first attribute we look at is "Community Name". This feature has 300 distinct values. In order to avoid introducing too many new features, I originally considered applying a frequency encoding or, as I did in assignment 2, a one-hot encoding where we keep the most common n communities and lump the rest into an "Other" category. However, I found that a one-hot encoding on the full feature provided the best results.

The next feature, "Group Category", is binary, so we map Disorder to 0 and Crime to 1.

I then plotted the "Resident Count" feature with a violin plot (fig.1) to examine its distribution. The

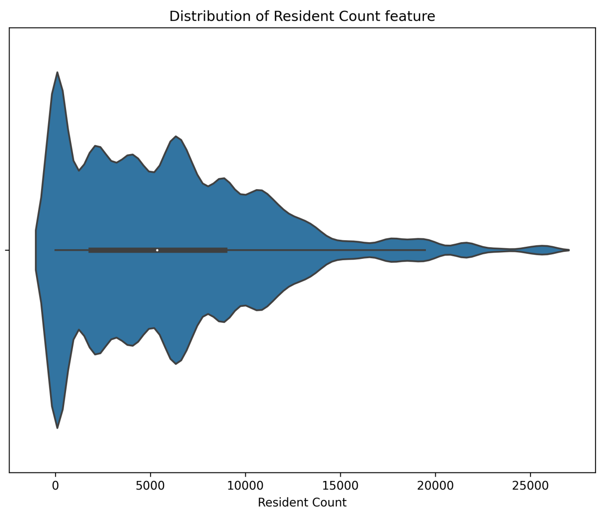

Figure 1: Violin plot showing the distribution of the Resident Count feature.

distribution is clearly skewed right, and has an unbiased skew value of 1.05. Unskewed features generally perform better in machine learning tasks, so I applied a square root transformation to this feature in an attempt to reduce the skewness. The resulting distribution (fig. 2) appeared less skewed, and had a better skew value of -0.22.

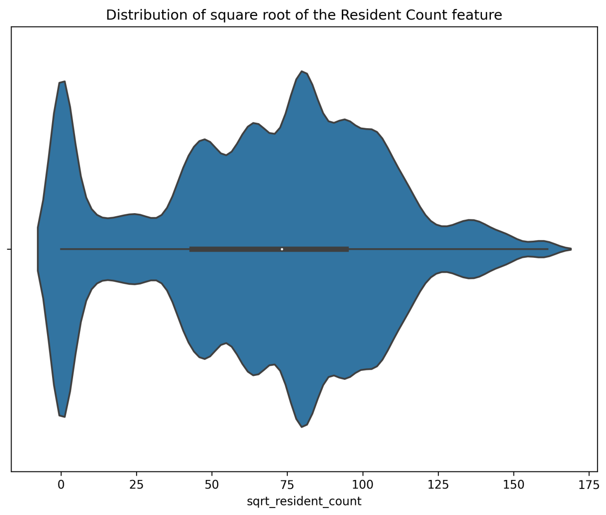

Figure 2: Violin plot showing the distribution of the square root of the Resident Count feature.

Plotting the "Crime Count" feature (fig. 3) we see that it, too, is skewed, although much more so with a skew value of 14.8. I found that a logarithmic transformation followed by square root transformation worked

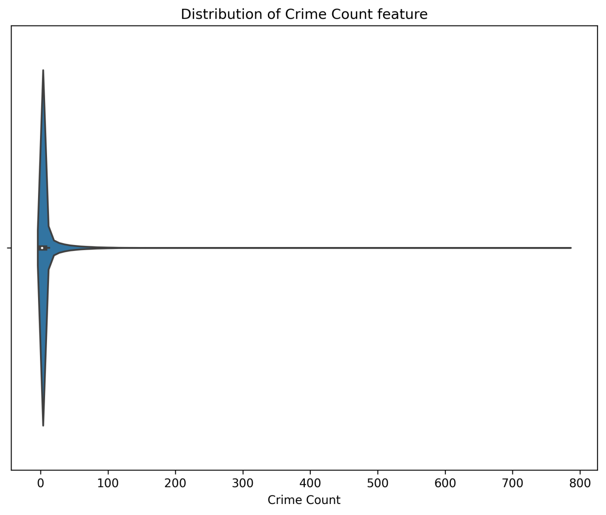

Figure 3: Violin plot showing the distribution of the Crime Count variable.

best to reduce the skew, and the results are shown in fig. 4. Applying this transformation reduced the skew to -0.00044. This is the same transformation I used to train the linear classifier in assignment 2 and it showed good results, so I will stick with it here.

The "Year" feature looks fine, so we will not apply any modifications to this variable except for standardization, which will come later.

The remaining two features, "Sector" and "Month", are cyclic in nature, so we apply a sin-cos encoding. Here, we map the values to the unit circle and apply the sin and cos functions to produce two new numeric

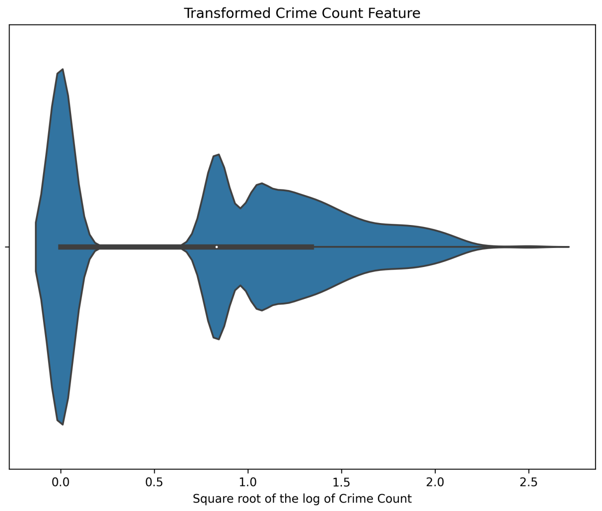

Figure 4: Violin plot showing the distribution of the transformed Crime Count variable.

features. This did not work well for the linear model in the last assignment, but it seems to work at least as well as other encodings in this model.

Next, we look at the target variable "Category". Printing the value counts, we see that there are 10 classes with many values, and one class called 1320.131 with only a single sample. Keeping this class as an output class for our model is rather pointless. We cannot do any meaningful training with only a single sample and the classifier will simply learn to ignore this class as it is rarely seen in training. Thus, we choose to remove this class and focus on classifying the data into the other classes. On the Criminal Code of Canada's website, we can find that 1320.131 corresponds to "Dangerous operation of a motor vehicle". We will consider this similar to violence and lump this sample in with the "Violence Other (Non-domestic)" class.

Next, we split the data into training and testing sets and take a look at the distribution of class labels in the train set (fig. 5). From this bar plot, we see that the classes are quite unbalanced. With unbalanced labels, the classifier will learn to classify the samples corresponding to the higher frequency classes more often simply because there is more data available for these samples and the classifier has 'seen' more of them in the past. To remedy this, we can compute class weights and pass them into scikit-learn's `LogisticRegression` class before training. The weights w_i for class i are computed as

$$w_i = \frac{N}{n_i \cdot C}$$

where N is the size of the dataset, n_i is the number of samples in class i, and C is the number of classes. The weights are used such that, for example, if $w_1 = 1$ and $w_2 = 10$, an instance of class 2 is treated like 10 instances of class 1 by weighting the loss function accordingly.

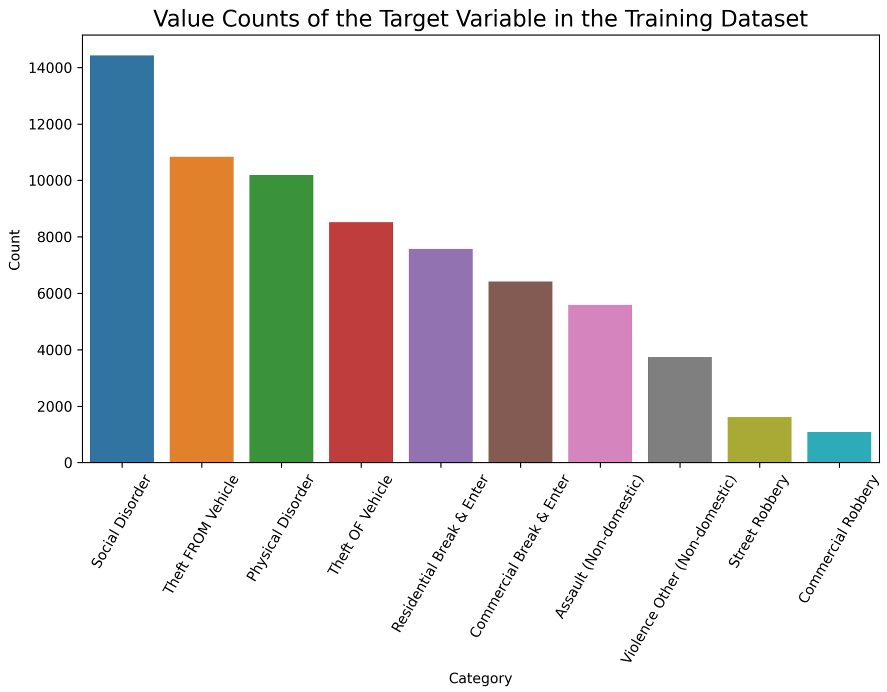

Figure 5: Value counts of the target variable “Category” in the training dataset.

Logistic Model Description

Now, we are ready to standardize the data and train our model. We use scikit-learn’s `StandardScaler` class to scale the training and testing input data based on the mean and standard deviation of the training data, and apply the logistic regression model using scikit-learn’s `LogisticRegression` class. The class weights are passed into the model here using the `class_weight` argument. We also increase the maximum number of iterations during fitting to 500 as the default value of 100 proved to be limiting performance. We then train the classifier on the training data and use the `predict` method to get predictions from the trained model.

Results of the Logistic Model

Computing the accuracy of the model yields a training accuracy of 51.83% and a testing accuracy of 50.13%. The confusion matrices for each of the datasets are shown in figs. 6 and 7. As is evident from these results, the model predicts very poorly and displays high bias. The only classes for which prediction is decent are the “Social Disorder” and “Physical Disorder” classes. This is likely because the binary “Group Category” variable separates out these two classes from the other eight, and the classifier is left with a much simpler binary decision from there. The remaining eight classes are not classified well. With some classes, such as “Assault (Non-domestic)” and “Violence Other (Non-domestic)”, the classifier classifies samples from these classes as other classes the vast majority of the time.

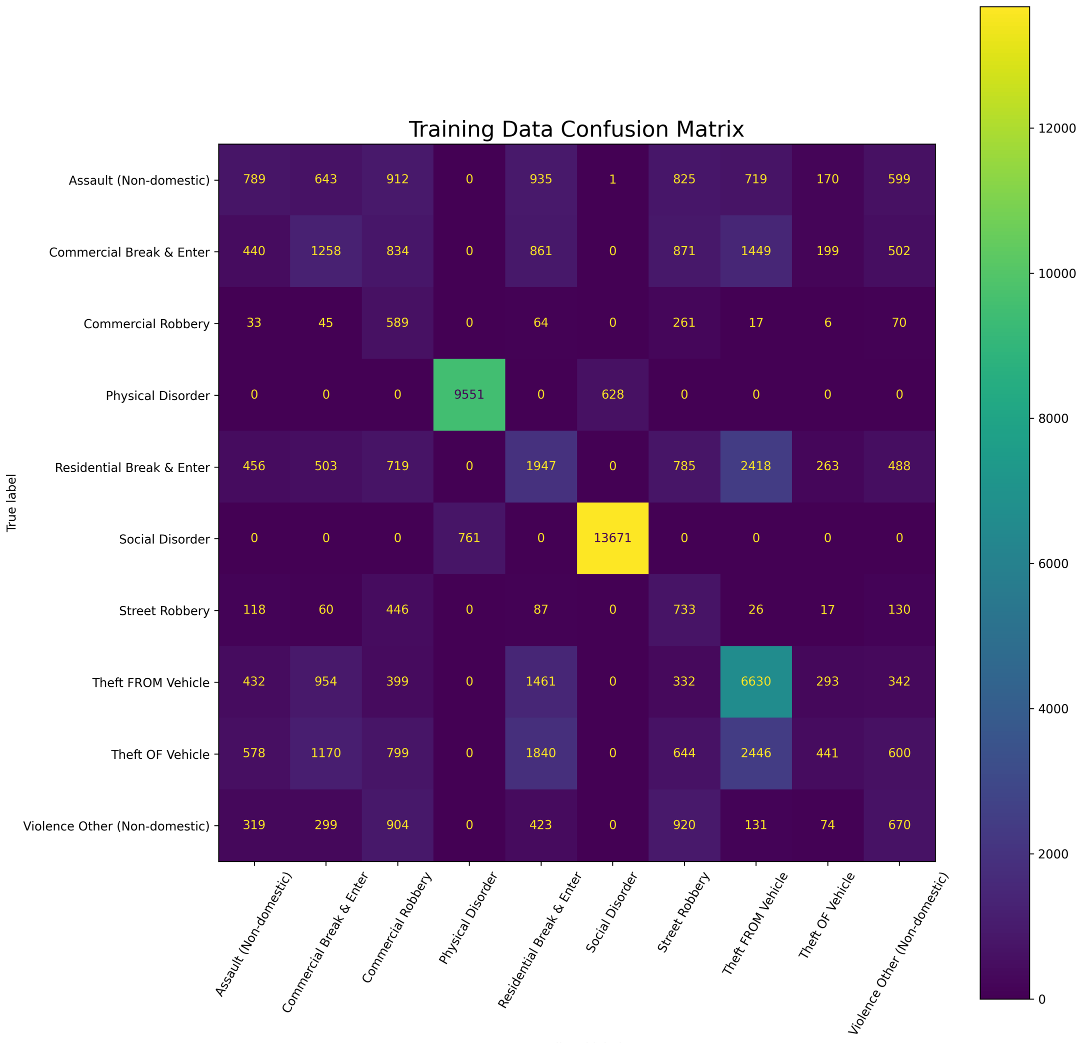

Figure 6: Confusion matrix for the logistic regression model on the training dataset.

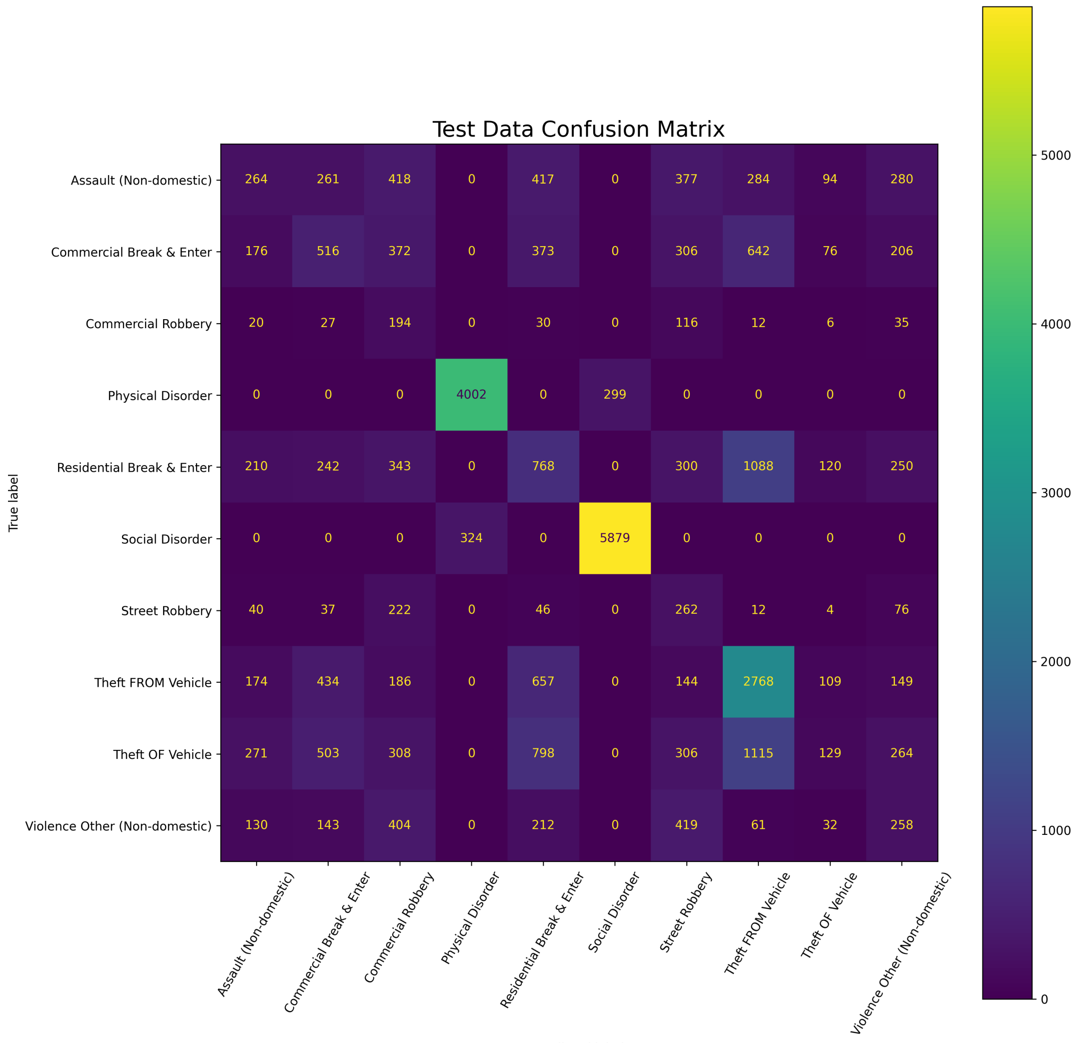

Figure 7: Confusion matrix for the logistic regression model on the testing dataset.

Data Preprocessing for the Neural Network Model

We performed the same data preprocessing for this model as for the logistic regression, but added in a few more features as input. Namely, I included both the original and transformed values of the “Resident Count” and “Crime Count” features. Also, for the “Community Name” variable, I included the full one-hot encoding as well as a frequency encoding. The frequency encoding simply maps each community name to its frequency of occurrence in the dataset. These extra feature encodings were included as their addition to the input data slightly improved the accuracy of the final classifier. Before defining the model, I performed the same training and testing split and standardized the data as before, and computed class weights to pass into keras’s `fit` method. In order to pass class weights into the model, keras requires the weight labels and the classes to be represented as integers instead of strings, so we switch them to integers here.

Neural Network Description

In the final model, the input data first runs though a fully-connected layer with 256 neurons, followed by a tanh activation and batch normalization. The next layer is a fully-connected layer with 128 neurons, a ReLU activation function, and batch normalization. Finally, the data runs though a fully-connected layer with 10 neurons to produce logits. For regularization, the two batch normalization operations are followed by dropout layers with $p = 0.3$ and $p = 0.2$, respectively.

We use a cross-entropy loss function computed by keras’s `SparseCategoricalCrossentropy` class. This class allows us to compute the loss without needing to one-hot encode our class labels before training. Moreover, we can pass in the argument `from_logits=True` to compute the loss based on logits and avoid applying the softmax activation function to the last layer of out network. This is favourable as it allows keras to combine the softmax operation and loss computation together using the log-sum-exp trick for numerical stability.

The data is batched with a batch size of 1024 and loss is minimized with the Adam optimizer over 40 epochs with a learning rate of 0.001. Several combinations of all aforementioned hyperparameters were tested, and the ones reported are those which produced the best performance, or those which produced equivalent performance with lower model complexity.

Results of the Neural Network

The final accuracy of the classifier on the training dataset is 51.68% and it is 49.25% for the training dataset. In order to get predictions from the model, we concatenate the model with a softmax operation so that we obtain probability distributions over the classes as output. To match our data format, we take the argmax of the probability distribution to revert the one-hot-like vectors back into our numerical class labels. After doing this, we can compute the confusion matrices shown in figs. 8 and 9.

Like the logistic regression model, this model performs quite poorly on both datasets, with the accuracies being slightly lower than those from the logistic model. Unfortunately, this is the best performance I was able to achieve on the test set without vastly overfitting the training set.

With regard to the “Physical Disorder” and “Social Disorder” classes (classes 3 and 5, respectively), we can see from the confusion matrices that the classifier can almost perfectly distinguish these two classes from the other eight, and does a decent job of distinguishing between them. This, again, is due to the presence of the “Group Category” variable in the training set. However, on the other eight classes, the NN performs very poorly.

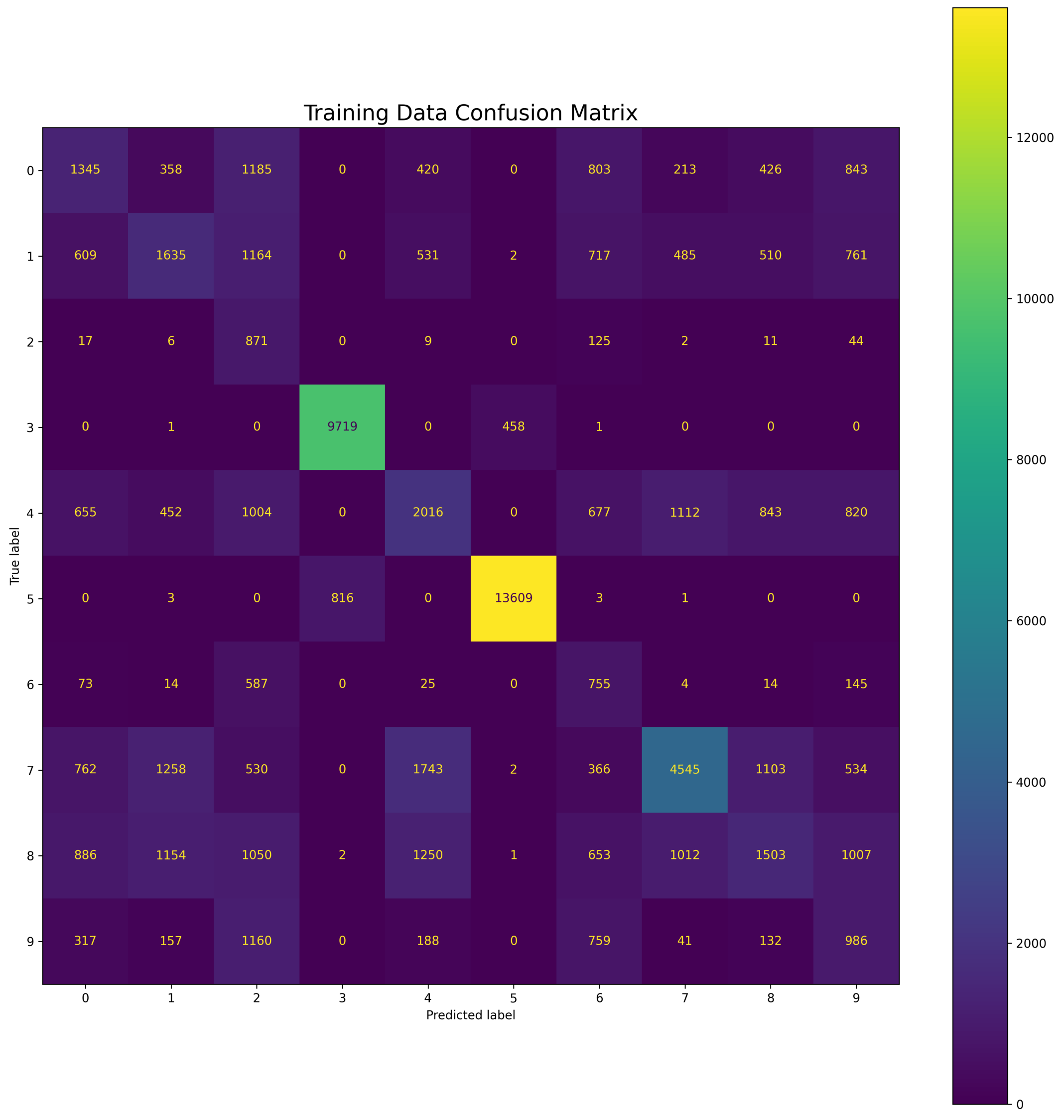

Figure 8: Confusion matrix for the fully-connected neural network on the training dataset. The integer-label mapping is as follows: [0: Assault (Non-domestic), 1: Commercial Break & Enter, 2: Commercial Robbery, 3: Physical Disorder, 4: Residential Break & Enter, 5: Social Disorder, 6: Street Robbery, 7: Theft FROM Vehicle, 8: Theft OF Vehicle, 9: Violence Other (Non-domestic)]

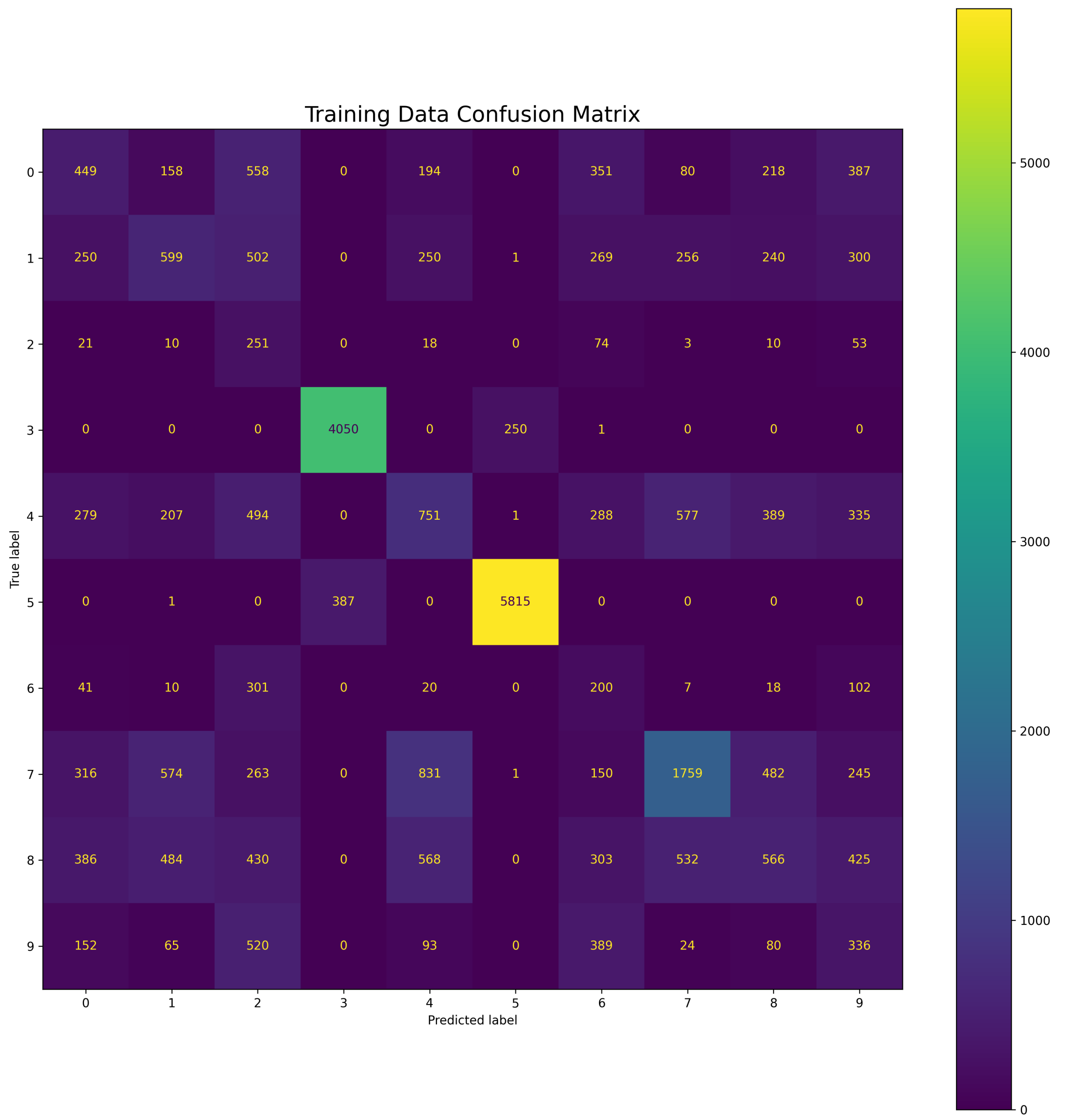

Figure 9: Confusion matrix for the fully-connected neural network on the testing dataset. The integer-label mapping is as follows: [0: Assault (Non-domestic), 1: Commercial Break & Enter, 2: Commercial Robbery, 3: Physical Disorder, 4: Residential Break & Enter, 5: Social Disorder, 6: Street Robbery, 7: Theft FROM Vehicle, 8: Theft OF Vehicle, 9: Violence Other (Non-domestic)]

Why are the models so bad?

Loosely, the universal approximation theorem for neural networks says that a neural network with enough complexity should be able to approximate any “sufficiently nice” function. However, I was unable to get the network to accurately model the training data even with a much larger network, more training time, and neglecting all regularization. Why is this? I suspected this was happening because there are identical samples in our input dataset which correspond to different class labels. If this is the case, then the “function” our model is trying to predict is not actually a function at all as it is multi-valued, so the universal approximation theorem does not apply. We can check this by computing the number of classes corresponding to each *unique* input, and we find this value to be 13743. If we compute the *total* number of values for which their is another identical sample corresponding to a different class, we find this value to be 31867. Thus, nearly 32% of the dataset has at least one other identical sample in the input dataset for which the class labels contradict each other. If we exclude the “Crime Count” variable from the dataset and redo the latter computation, we find that this value jumps to 88467. This means that, for more than 88% of the data, the classifiers have only the “Crime Count” feature to distinguish between two or more class labels, which is most definitely not enough data for accurate classification. Ergo, it is no surprise that the models perform poorly. To achieve an accurate model, we need data which correlates more strongly with the target variable and does not contain so many contradicting samples.