LeMaur Kydd

Santander Bank Customer Transaction Prediction Report

The use of predictive modeling is a unique and recent venture for humanity. Specifically, the world of machine learning and artificial intelligence models is still very much in the beginning stages. One thing that has taken off due to these types of discoveries is the applications of these models in the real world ranging from simple business decisions to taking care of our Earth. One thing is for sure, the more these fields of study improve their products, the more people will notice their potential. Santander Bank, based out of Boston, MA, is taking full advantage of the improvements and capabilities of predictive modeling by hosting their own Kaggle challenge in which the task is to create a model to predict if a customer will make a purchase given data describing said customer behavior. This is the task taken on in this project.

To arrive at a working model that can make such predictions, a series of standard steps need to take place, starting with the raw data. An important characteristic of this data worth noting is that it has been anonymized for the sake of preserving the identities of individuals and their data. The data has been broken up into two subsets, a training dataset, and a testing dataset, both with 200,000 rows (customers) and 200 columns (features) with extra columns for customer ID’s and the target variable. Due to the size of these datasets, dimension reduction is the next necessary step.

**Chart, histogram

Description automatically generated** Exploring the descriptive statistics of the two datasets yielded favorable results. The histograms outlining the spread of the data **Fig. 1,2**

Figure Training Dataset Descriptive Statistics

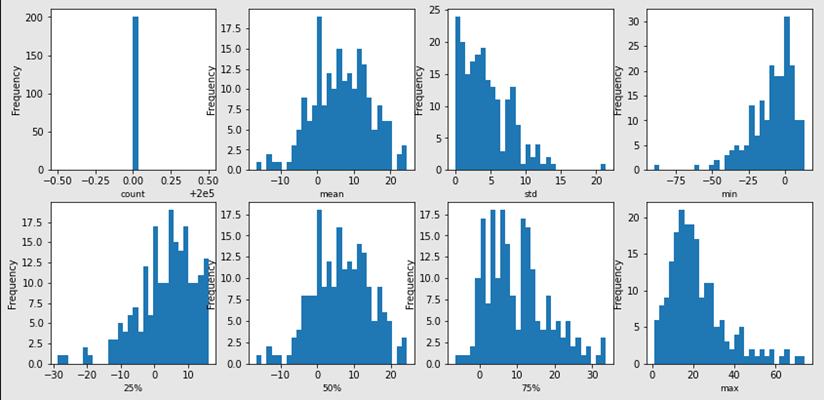
****show a balance of similar distributions. There were no NaN values in either dataset, and the spread of duplicate values in both datasets matched. Scaling/standardizing the data is the last step before being able to use

Figure Testing Dataset Descriptive Statistics

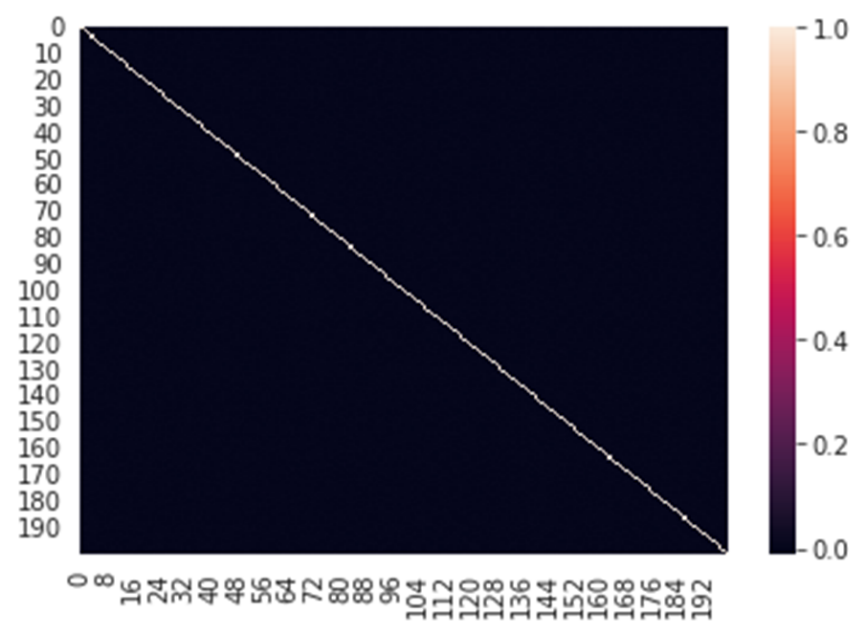
 Dimension reduction techniques are a way to reduce the dimensionality of your data for the primary purpose of making training of models easier. By employing these techniques storage space and computation times will decrease. This is essential for the Santander Bank dataset because there are 200+ columns that have the potential to “bog down” the training process. Three dimension reduction techniques were used: Principal Component Analysis (PCA), Random Forest, and Variance Inflation Factor (VIF). The hope of PCA is to rid the dataset of columns with high correlations while also compiling a set number of principal components that have a corresponding “variance explained” component. The issue with this dataset is, to start with, there are essentially no significant correlations between any of the 200 features. **Fig. 3.**

Figure Correlation Heatmap of all columns in the

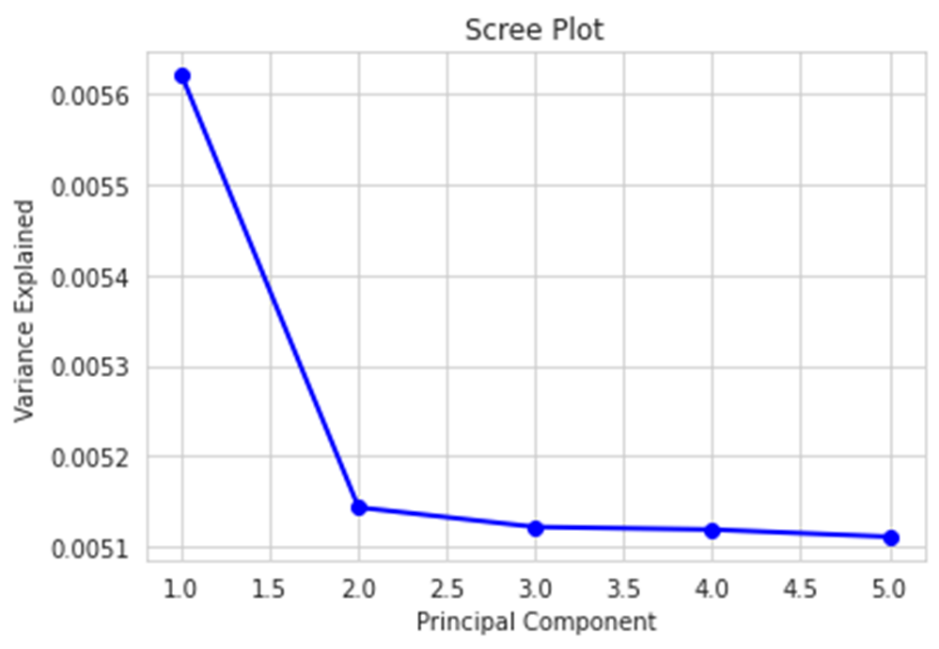
In a typical situation where dimension reduction is not necessary, having virtually no correlations between all the variables is a great thing to have. But if columns need to be removed, this becomes an issue solely because the main criterion for omitting a column is if the correlations are significant. The scree plot **Fig. 4** from the PCA strategy, expectedly, also yielded undesirable results with very low variance values, which meant these results were not used.

Figure PCA Scree Plot

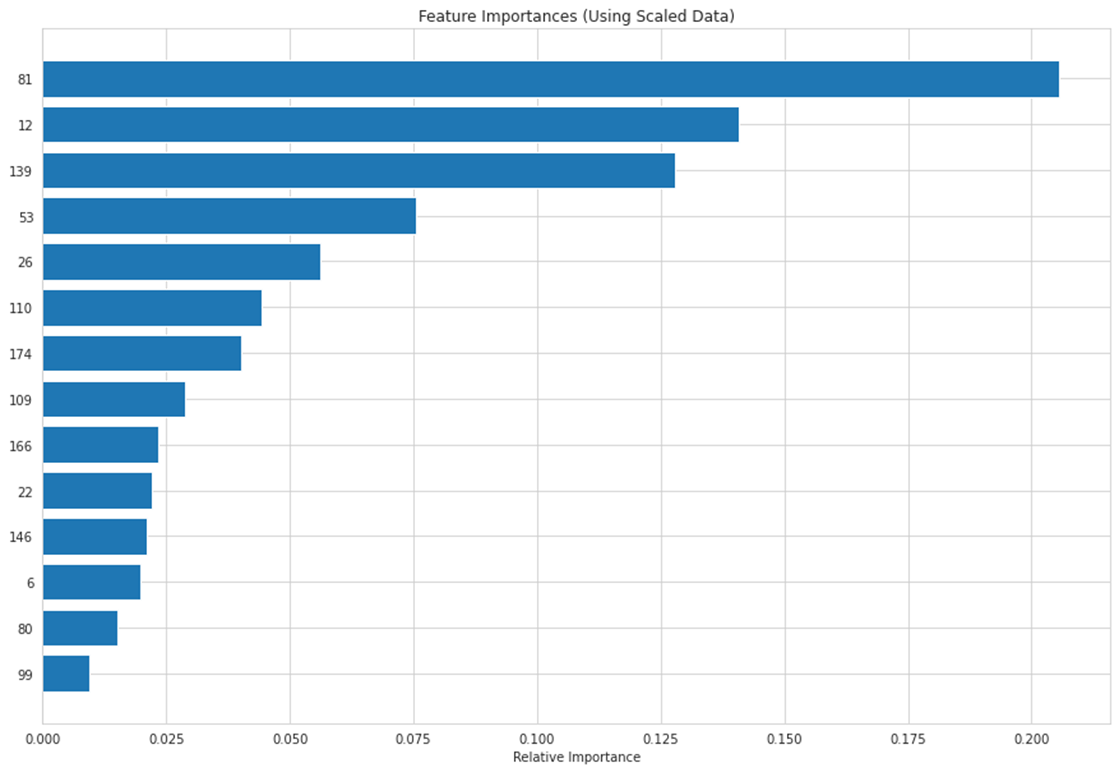
 The next attempt at dimension reduction was by using random forest to return a ranked list of feature importance’s which would narrow down the number of columns to include in training my model. The chart **Fig. 5** shows, in order from greatest to least importance, the column names along with their respective importance’s.

Figure Feature Importance's

This random forest was set with the max depth at 5 and 100 trees for a training time of thirty minutes. The results of random forest ended up not being used only because it seemed too limiting to only have ten columns. The though process was, the more column put into the model the better, but in hindsight perhaps using the top 25 features would not be such a bad thing.

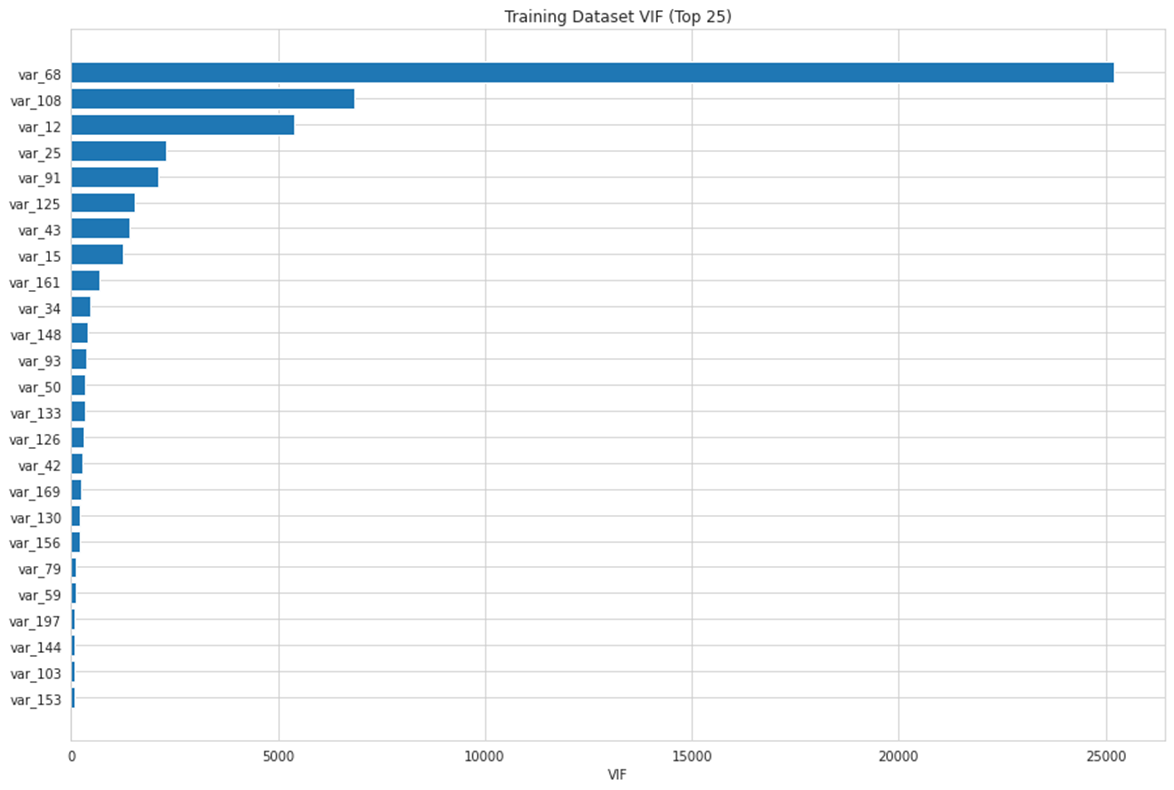
 Lastly, for dimension reduction, VIF was employed to return a list of features with the greatest VIF. The top 25 resulting columns of this test **Fig. 6** are shown in the table below.

Figure Training Dataset VIF's

The greatest VIF came in at over 25,000 which is an extremely large value for this test, where typically a VIF of greater than ten is considered high. Mostly all the variables reported back a VIF greater than 25, but for the sake of being able to move on, the variables displayed were removed from both the training and testing datasets.

After all the attempts at reducing the dimensionality of the data, the resulting training and testing datasets had 200,000 rows x 177 columns and 200,000 rows x 176 columns respectively. With this portion of the task completed, the next step is to begin the process of choosing an appropriate binary classification model for the data. Three options were employed for modeling: Logistic Regression, Deep Neural Networks, and K-Nearest Neighbors.

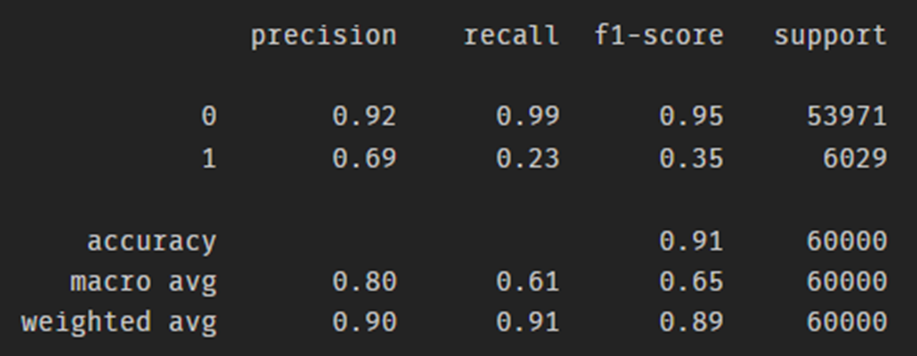
The logistic regression task proved to be one of the more reliable options considering the nature of the classification problem is binary. A testing split of 30% was used to ensure there was ample amount of data to test on and to help avoid overfitting. The test yielded a 91% accuracy with the other details of the evaluation metrics in the table **Fig. 7** below.

Figure Logistic Regression Evaluation Metrics

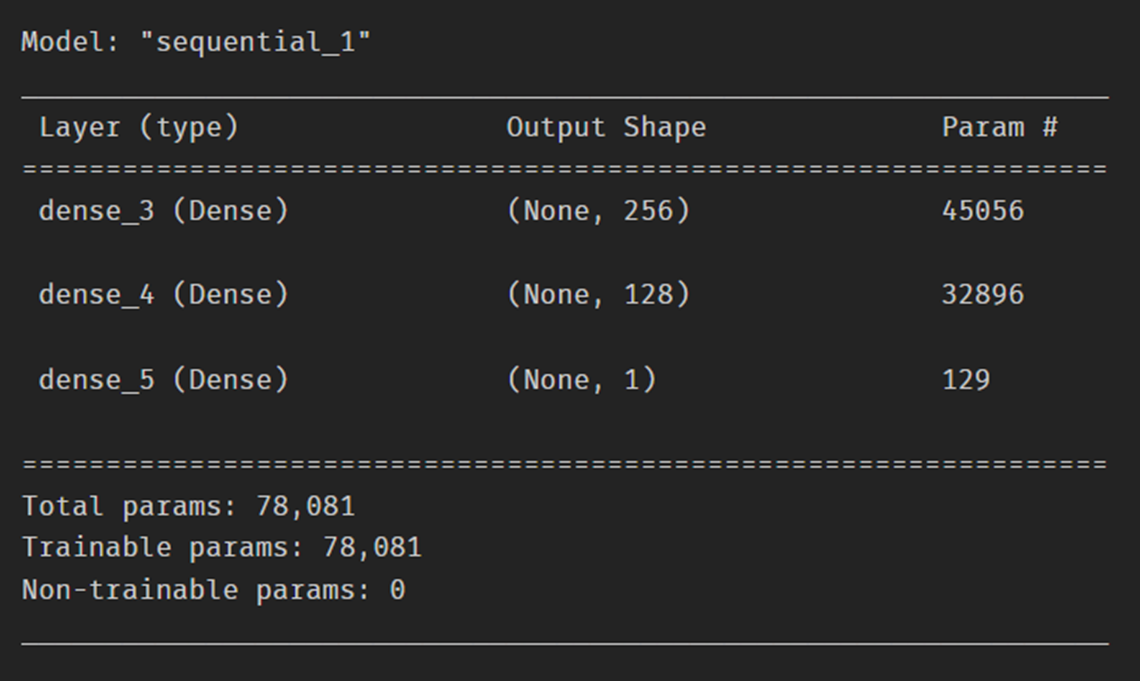
When running the testing dataset through the logistic regression, a distribution of 96% zeros and 4% ones resulted. After creating a submission file with the predicted values and submitting it to Kaggle, it received a score of 0.61049, meaning ~61% of the predictions are correct. The option for modeling the classification was a Deep Neural Network with a basic architecture **Fig. 8** that has a logistic regression as the output variable.

Figure Deep Neural Network Architecture

There were two variations of NN models used. The only difference is the first model was trained using 100 epochs and the second model was trained using only 10 epochs. The 100 epoch (more robust) model accuracy **Fig. 10** and model loss **Fig. 9** plots were not ideal and they seem to illustrate overfitting.

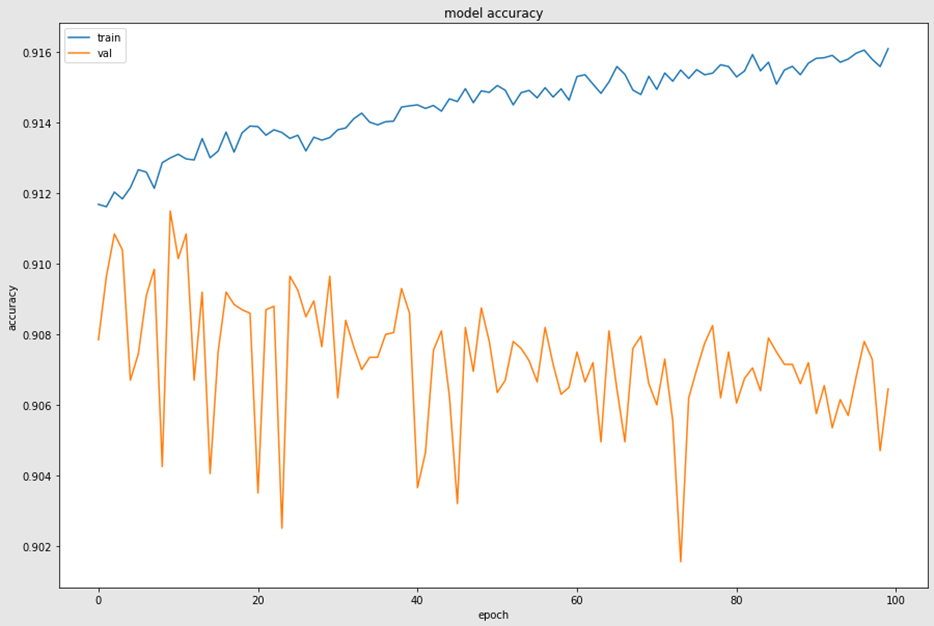
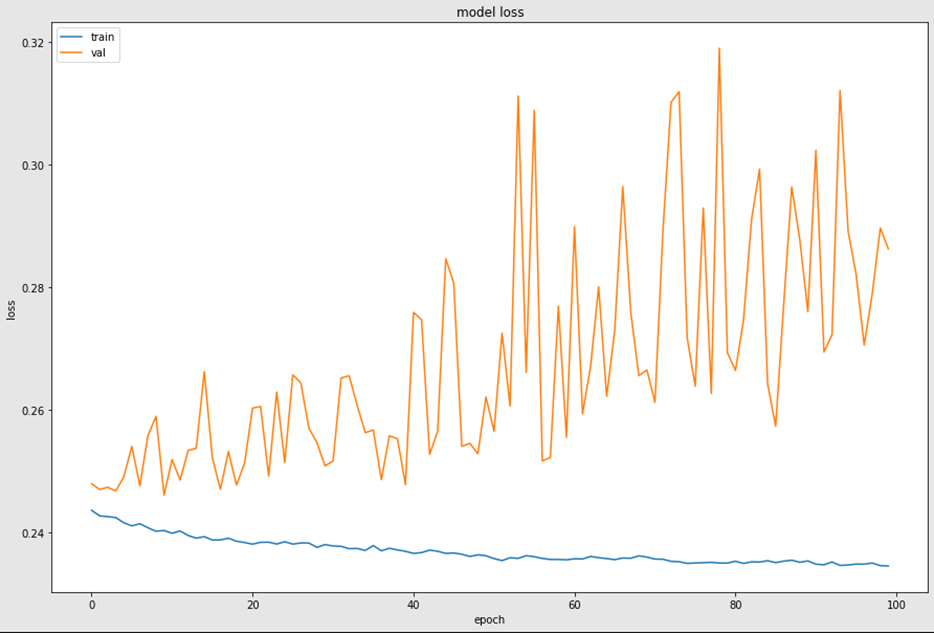
This model received a Kaggle score of 0.62945 which is comparable to the DNN model with only ten epochs which received a Kaggle score of 0.62382. The results from both models were included in the report only because it seems interesting that the increase in epochs did not directly translate into an increase in the Kaggle score. Also, the accuracy **Fig. 12** and loss **Fig. 11** plots for the less robust model also show an overfitting problem.

Figure DNN (more robust) Model Loss

Figure DNN (more robust) Model Accuracy

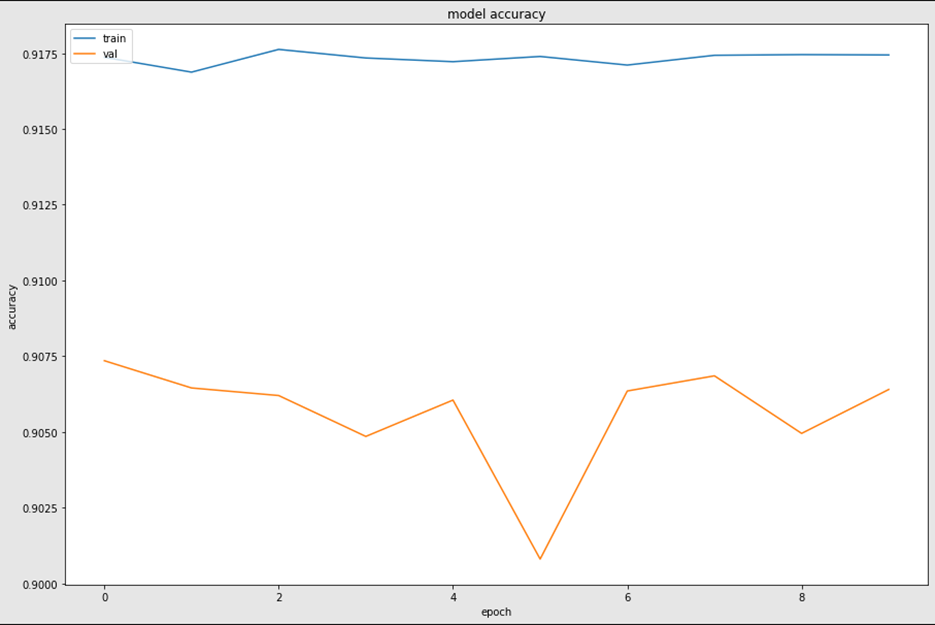
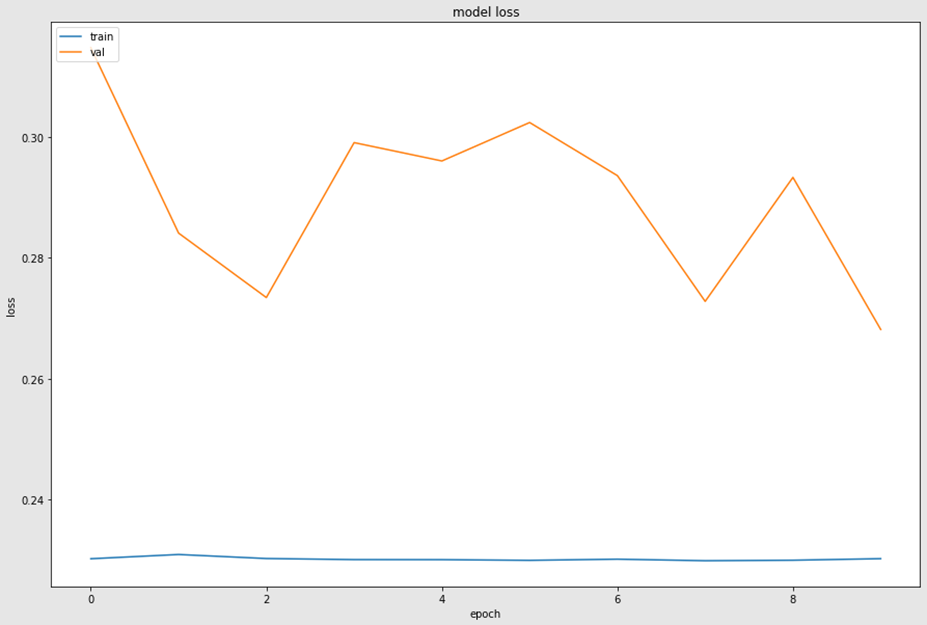
The last option used for modeling was a K-Nearest Neighbor algorithm. The hyperparameters assigned was to make 2 clusters (representing 0’s and 1’s), Euclidian Distance, and the Minkowski metric. This model produced a target variable prediction distribution from the testing dataset of 99% zeros and 1% ones, with 89% testing accuracy. The resulting confusion matrix **Fig. 13** showed a very high false negative rate of 6017 out of 60,000 and a low true positive rate of 12 out of 60,000. This model received a Kaggle score of 0.50165 which is lower than the others.

Figure DNN (less robust) Model Loss

Figure DNN (less robust) Model Accuracy

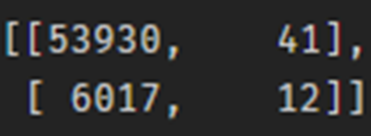
Coming to a decision of which model served this dataset the best has to be split based on two metrics, the model testing accuracy and the resulting Kaggle score. Logistic regression returned the greatest testing accuracy of all options and the more robust DNN model gave the greatest resulting Kaggle score. So the final conclusion is that Logistic regression and NN offered the best results out of the models tried for this project.

Figure K-Nearest Neighbors Confusion Matrix