Alonso Salcido, Chris Haub

Final Project, Case 7

**Introduction**

This project focused on analyzing data to come up with the most cost savings method to make a binary decision given a set of attributes. We were given a goal, to save the company as much money as possible. With cost in mind, we set our goal to build the most cost-effective method and not the most accurate one. False Positives are more costly than False Negatives, $35 and $15 respectively. This is something that is applicable to most companies no matter the industry. Saving money is always top of mind.

**Methods and Results**

This data set appeared to be a collection of numerical features representing various attributes of some unknown real-world system, alongside a binary target variable, "y", which indicated whether a particular instance belonged to class 0 or 1. The features were labeled as x0 through x49, and there were 50 columns in total, including the target variable. It was not clear what the specific domain of this data set was or what the individual features represent, but it is likely that they were obtained through some sort of measurement or observation process.

The data set provided was imbalanced with 60% of dependent variable was “0” and 40% was “1”. It contained NaN values that were addressed. NaN values were replaced with mode. Null values were imputed to median and numerical values were imputed to mean. There were also typos in the data set, however they did not affect the data. Create dummies was used on categorical variables to make them numerical.

An 80/20 training/test split was used, and the data was scaled using StandardScaler. StandardScaler was used to scale each feature to zero mean and unit variance.

SGD

The first model built was an SGD classifier. It didn’t perform as expected. It had an accuracy in the high 60’s and instead of trying to tune it by adjusting different parameters, we decided to try a more robust model.

SGD Results:

Chart, treemap chart

Description automatically generated Table

Description automatically generated

**Class Label Prediction**

Neural Network

The second model built was a neural network model. The Keras library in TensorFlow was used to define the neural network model. The model consisted of several 7 hidden layers, and an input and output layer. The input layer expected input data with a shape of (67,), while the output layer had a single neuron with a sigmoid activation function, which is appropriate for binary classification problems. Each dense layer had a different number of neurons, with the first layer having 50 neurons, and the subsequent layers having 80 and 100 neurons respectively. All the dense layers used the ReLU activation function, which helps introduce non-linearity into the model. The kernel regularizer argument was set to l2(0.001), which applied L2 regularization to the kernel weights of each dense layer, helping to prevent overfitting. Stochastic gradient descent optimizer was used with binary cross-entropy loss function, the drawback to this is time is takes to run SGD.

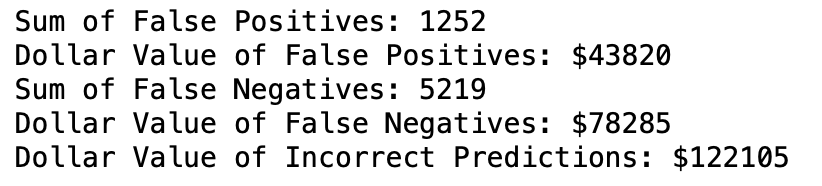
Since we had a very specific goal in mind, we set it to run an extremely large number of epochs (1,000) with a batch size of 15. This was done to make sure that the early stopping criteria is used. The model was set to stop once the validation loss stopped improving. The patience argument was set to 1 for EarlyStopping. Once the loss stopped decreasing the model stopped running.

This model performed a lot better and minor tunings were done before the final model was built.

After building a few variations of the initial neural network model, a final model was found. The final model used class weights to compensate for the slightly imbalance of the response variable. Early stopping was changed from loss to accuracy, meaning that the model was focused on accuracy and as soon as accuracy stopped improving the model stopped running. The last adjustment made to the final model was increasing the threshold. By default, the classifier outputs probabilities of each class and turns them into a binary classification by using a threshold of 0.5. Since, false positives are more expensive than false positives, we increased our threshold from 0.5 to 0.7. This adjustment reduces the number of False Positives, but int turn, creates more False Negatives.

In order to get an accurate score while predicting the full dataset, a 5-fold cross validation was used. The classifier split the dataset into 5 parts to create 5 different train-test splits of 80/20. The final model had an accuracy of 95.96%, specificity of 0.979, and sensitivity of 0.947. The total dollar cost of False predictions created with this model predicting 160,000 lines of data was $122,105. It’s worth to mention, that this model was not the most accurate model built, but it was the most cost-efficient model.

Chart, treemap chart

Description automatically generated 

**Confusion Matrix for Multiple Cross-Validation**

**Index**Text

Description automatically generated Text

Description automatically generated Text

Description automatically generated Text

Description automatically generated Text

Description automatically generated Text

Description automatically generated Text

Description automatically generated Text

Description automatically generated Text

Description automatically generated Text

Description automatically generated Text

Description automatically generated Text

Description automatically generated Text

Description automatically generated Text

Description automatically generated Text

Description automatically generated Text

Description automatically generated Text

Description automatically generated Text

Description automatically generated