# Case study 7

Monetary Loss

Balaji Avvaru, Apurv Mittal, Ravi Sivaraman

Quantifying The World

## Introduction

The goal of this project is to minimize the loss for the organization which loses money every time the prediction is inaccurate. The case study consists of anonymized dataset. There is no metadata available for the dataset of 50 features and a target variable.

The model is expected to reduce the losses due to incorrect predictions. If the prediction is class 1 and the actual is not class 1, then the organization loses $100 each incorrect prediction. Similarly, if the prediction is class 0 and actual is not class 0, then the organization loses $250 each incorrect prediction.

The goal is to create a model that predicts the class that minimizes financial losses.

## Data Analysis

The dataset contains 160,000 records with 50 features. The features are numbered from *x0* to *x49,* label. *"y"* is the target variable in the dataset.
There are very few missing values. Total of 1608 missing records in the dataset which is less than 0.5% per feature. This dataset doesn't require imputation for the features, the missing records are dropped from the final dataset.

Prior to continuing with data modeling, data cleanup is required. For example, *x32* has '*%*' in the data and *x37* has '*$*' in the data which makes them categorical variables. The *% and $* signs are removed from the dataset.

There are three other categorical variables:

*x24: Continent names*
*x29: Calendar Months*
*x30: Day of the week*

All the features above are one-hot encoded for the data modeling.

The correlation matrix below shows the collinearity among the variables. Data shows there is high collinearity for two sets of the attributes: *x2:x6, x38:x41*

Since the above features shows greater than 99% collinearity, the study drops *x6 and x41* from further analysis.

## Correlation Analysis

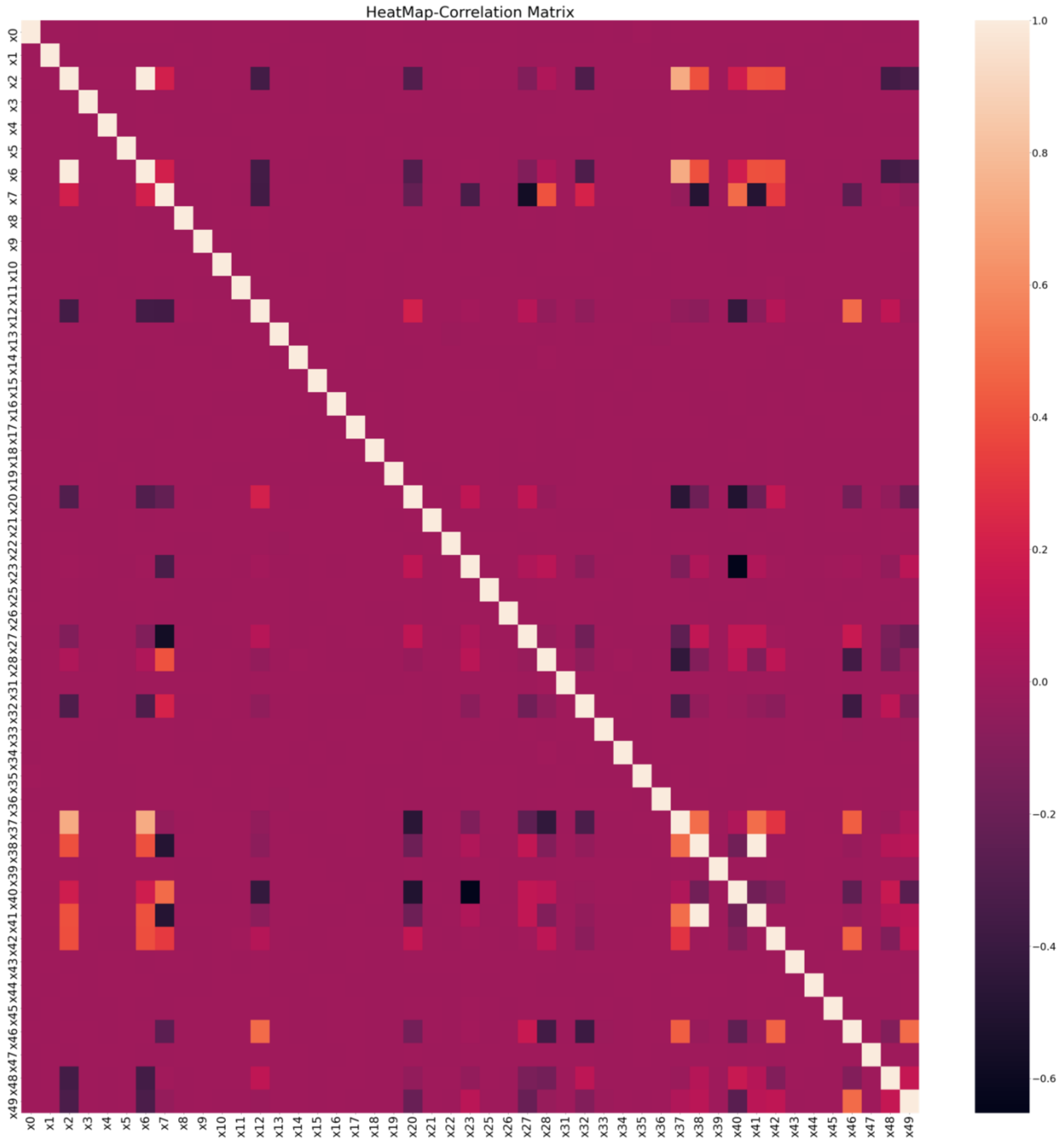

*Figure 1: Collinearity heatmap of Attributes*

### Target

Target is a binary variable called *'y'*, which has value either *0* or *1.*

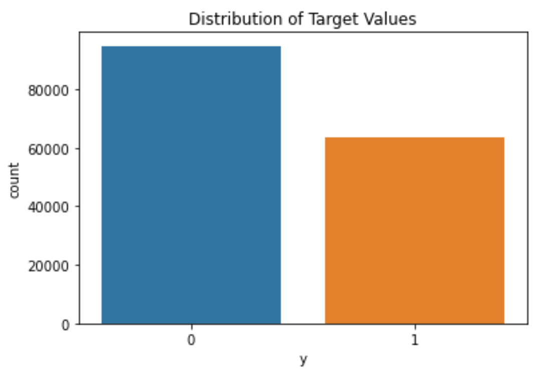

*Figure 2: Target variable distribution plot*

Target variable is not balanced, majority class is *"0"* with 60% of the data and minority class is *"1"* with 40%

## 2. Methods

`x24, x29, x30` are categorical variables, which are converted using one-hot encoding.

All the features are normalized using the `Standard Scaler` technique. After normalization, the data was split into train and test datasets by a factor of 80/20.

### Logistic Regression

One of the models used for this report is Logistic Regression, as the objective is a binary classification.

For classification model, *accuracy*, *precision* and *recall* are calculated.

The *precision* for the classification model represents the ratio of the predictions that the model correctly identifies as a positive out of all the predictions model identifies as positive. The *recall* (also referred to as *sensitivity*), represents the ability of the model to correctly identify true positive. The accuracy represents how well the model classifies correctly both positive and negative output.

To identify the most efficient model, *RandomizedSearch* algorithm is used. *RandomizedSearch* run with various parameters and then compares the *accuracy* (for this report) and chooses the best parameters for the *LogisticRegression*.

Parameters were used for calculating the best model for *accuracy*, the following values are used:

- `penalty:` used to specify the norm used in the penalization (values used: 'l1', 'l2')
- `C:` Inverse of regularization strength (0.001, 0.01, 0.1, 1, 10, 100, 1000)
- `class_weight:` The "balanced" mode uses the values of y to automatically adjust weights inversely proportional to class frequencies in the input data (value: balanced).
- `solver:` algorithm to use in the optimization problem ('lbfgs', 'liblinear')

The *RandomizeSearch* produced 200 models of which the model with highest accuracy had following parameters:

| **Parameter** | **Value** |
|---|---|
| **C** | 0.1 |
| **class weight** | Balanced |
| **Penalty** | L2 |
| **Solver** | lbfgs |

*Table 1: Logistic Regression best model*

## Classification report for Logistic Regression

| | **Precision** | **Recall** | **F1-Score** | **Support** |
|---|---|---|---|---|
| **0 (False)** | 0.77 | 0.73 | 0.75 | 18970 |
| **1 (True)** | 0.63 | 0.68 | 0.65 | 12709 |
| **Accuracy** | | | 0.71 | 31679 |
| **Macro avg** | 0.70 | 0.71 | 0.70 | 31679 |
| **Weighted avg** | 0.72 | 0.71 | 0.71 | 31679 |

*Table 2: Logistic Regression Classification Matrix*

## Random Forest

`Random forest` is an ensemble tree-based learning algorithm where it combines more than one algorithm of the same or different kinds for classifying objects. The `Random Forest Classifier` is a set of decision trees from a randomly selected subset of the training set. It aggregates the votes from different decision trees to decide the final class of the test object.

### *Parameters:*

- `n_estimators:` number of trees in the forest

- `max_depth:` max number of levels in each decision tree

- `criterion`: The function to measure the quality of a split. Supported criteria are *gini* for the Gini impurity and *entropy* for the information gain. Note: this parameter is tree-specific

- `min_samples_split` = min number of data points placed in a node before the node is split

- *min_samples_leaf* = min number of data points allowed in a leaf node

- *class_weight*: The *balanced* mode uses the values of y to automatically adjust weights inversely proportional to class frequencies in the input data as

```
a. n_samples / (n_classes * np.bincount(y))
```

Random Forest is the basic unsupervised method to classify using Entropy/Gini. In theory, Random Forest gives a benchmark upon which any sufficient advanced algorithm must beat. In general, any algorithm that is using deep learning principles must have a better outcome than Random Forest.

For *Random Forest* below hyper tuning parameters were used:

*max_depth = [5, 7, 8, 10, 12]*
*n_estimators = [100]*
*criterion = ['entropy']*
*min_samples_leaf = [3, 4, 5]*
*min_samples_split = [8, 10, 12]*
*class_weight = ['balanced']*

Parameters listed above were executed with Cross Validation of 10.

The *RandomizedSearch* produced 200 models of which the model with highest accuracy had following parameters:

| Parameter | Value |
|---|---|
| *n_estimators* | 100 |
| *criterion* | entropy |
| *min_samples_leaf* | 3 |
| *min_samples_split* | 10 |
| *class_weight* | Balanced |
| max_depth | 12 |

*Table 3: Random Forest Best Model*

## Classification report for Random Forest

| | **Precision** | **Recall** | **F1-Score** | **Support** |
|---|---|---|---|---|
| **0 (False)** | 0.92 | 0.90 | 0.91 | 18970 |
| **1 (True)** | 0.86 | 0.88 | 0.87 | 12709 |
| **Accuracy** | | | 0.89 | 31679 |
| **Macro avg** | 0.89 | 0.89 | 0.89 | 31679 |
| **Weighted avg** | 0.89 | 0.89 | 0.89 | 31679 |

*Table 4: Random Forest Classification Matrix*

## XGBoost

XGBoost stands for eXtreme Gradient Boosting, which provides a gradient boosting, and often achieves higher accuracy simple Decision Trees, but at a cost of simple interpretability, as XGBoost follows paths of hundreds or thousands of trees, makes it harder to interpret, but provides an accuracy boost.

The hyper-parameters (tunable parameters) are:

- *learning_rate*: The learning rate. In each boosting step, this value shrinks the weight of new features, preventing overfitting or a local minimum. This value must be between 0 and 1. The default value is 0.3.

- *max_depth*: The maximum depth of a tree. Be careful, greater the depth, greater the complexity of the model, and easier to overfit. This value must be an integer greater than 0 and have 6 as default.

- *n_estimators*: The number of trees in ensemble.

- *gamma*: A regularization term and it’s related to the complexity of the model. It’s the minimum loss necessary to occur a - -split in a leaf. It can be any value greater than zero and has a default value of 0.

- *colsample_bytree*: Represents the fraction of columns to be subsampled. It’s related to the speed of the algorithm and preventing overfitting. The default value is 1 but it can be any number between 0 and 1.

- *lambda*: L2 regularization on the weights. This encourages smaller weights. Default is 1 but it can be any value.

For *XGBoost* below hyper tuning parameters were used:

*n_estimators = [100]*
*learning_rate = [0.0001, 0.001, 0.01, 0.1, 0.2, 0.3, 0.4, 0.5]*
*max_depth = range (3, 15)*
*colsample_bytree = [i/10.0 for i in range (1, 3)]*
*gamma = [0.01, 0.05, 0.1, 0.2, 0.3]*
*reg_lambda = [0.01, 0.05, 0.1, 1.0, 5.0, 10.0, 50.0, 100.0]*
*min_child_weight = [0.1, 0.9, 0.95,1, 2, 3]*

Parameters listed above were executed with Cross Validation of 10.

The GridSearch produced several models of which the model with highest accuracy had following parameters:

| Parameter | Value |
|---|---|
| ***n_estimators*** | 100 |
| ***Learning_rate*** | 0.5 |
| ***Max_depth*** | 9 |
| ***Colsample_bytree*** | 0.2 |
| ***gamma*** | 0.3 |
| ***Reg_lambda*** | 5.0 |

*Table 5: XGBoost Best Model*

### Classification report for RXGBoost

| | Precision | Recall | F1-Score | Support |
|---|---|---|---|---|
| **0 (False)** | 0.91 | 0.94 | 0.92 | 18970 |
| **1 (True)** | 0.90 | 0.86 | 0.89 | 12709 |
| **Accuracy** | | | 0.91 | 31679 |
| **Macro avg** | 0.90 | 0.90 | 0.90 | 31679 |
| **Weighted avg** | 0.91 | 0.91 | 0.91 | 31679 |

*Table 6: XGBoost Classification Matrix*

## ANN

Artificial neural networks (ANNs) are models inspired by humans (or biological) neural networks.

An ANN is a collection of neurons (nodes) loosely based on the biological brain interconnected in a dense layer. Each neural is a linear regression created from each feature. The neurons send a signal once it crosses a certain threshold. Usually, they are aggregated into several layers (sequential or dense). The final layer depends on the type of classification required. For binary classification, sigmoid functions are used. The last layer uses a sigmoid function for this study, as the target is a binary classification.

The training data was further split into training and validation set by 80/20. The validation set is used to validate the model performance during training.

The test data is unseen data and is used to calculate the final accuracy of the best model and confusion matrix. The data from the confusion matrix is used for the monetary loss calculations.

## Layers of Neural Network

In this study, 5 layers neural network with the following layers were created:

| Layer | Activation Function | Number of Neurons | Input to Layer |
|---|---|---|---|
| **1** | swish | 256 | Features |
| **2** | swish | 128 | Layer1 |
| **3** | swish | 256 | Layer2 |
| **4** | swish | 128 | Layer3 |
| **5** | sigmoid | 2 | Layer4 |

*Table 7: Layers of Neural Network*

### *Parameters:*

*batchsize:* Constant that multiplies the regularization term; the higher the value, the stronger the regularization. Also used to compute the learning rate when *learning_rate* is set to '*optimal.*'

*Optimizer:* Adam(lr=1e-2).

*loss*: Sparse Categorical Cross-Entropy

*safety:* Early Stopping (using *val_loss*)

*patience*: 4

*min_delta*: 2e-4

The models are compiled to run for 50 epochs. Each epoch runs the entire dataset once. With 50 epochs, the model runs each record in the dataset 50 times or unless early stopping condition is met (when loss stops reducing further). If the accuracy or val_loss doesn't improve, then the model doesn't run the entire 50 epochs and stops early. The model is configured to stop if 4 (hyper-param patience) consecutive epochs did not improve val_loss. This prevents running neural networks when the metrics don't improve.

### Classification report for ANN

| | Precision | Recall | F1-Score | Support |
|---|---|---|---|---|
| 0 (False) | 0.98 | 0.98 | 0.98 | 18970 |
| 1 (True) | 0.97 | 0.96 | 0.96 | 12709 |
| Accuracy | | | 0.97 | 31679 |
| Macro avg | 0.97 | 0.97 | 0.97 | 31679 |
| Weighted avg | 0.97 | 0.97 | 0.97 | 31679 |

*Table 8: ANN Classification Matrix*

### Ensemble Model

Ensemble model was created with the outputs from Logistic regression, Random Forest, XG boost and Artificial Neural Network models. Calculated the probability predictions from each of the 4 models using cross_val_predict method with a cross validation of 10 on entire dataset. These probability predictions were given to Linear Regression/ ANN as input to find final predictions. Final prediction used in calculating the loss per prediction

Final loss per prediction with Linear Regression: $4.86
Final loss per prediction with Linear Artificial Neural Network: $4.66

## 3. Results

Since there is a cost associated to incorrect predictions. The real test of the results is by calculating the total loss due to incorrect predictions.

Here are the results for each model:

| Model | Accuracy | Precision | Recall | F1 Score | Loss per prediction |
|---|---|---|---|---|---|
| Logistic Regression | 0.71 | 0.70 | 0.71 | 0.70 | $48.25 |
| Random Forest | 0.89 | 0.89 | 0.89 | 0.89 | $18.18 |
| XGBoost | 0.91 | 0.90 | 0.90 | 0.90 | $17.82 |
| Neural Network | 0.97 | 0.97 | 0.97 | 0.97 | $4.99 |
| Ensemble (Linear Regression) | | | | | $4.86 |
| Ensemble (ANN) | | | | | $4.66 |

*Table 9: Performance vs Monetary Loss across models*

## 4. Conclusion

The neural networks model gives the model with highest accuracy, precision, and recall. But the task was to create a model which minimizes the monetary loss associated with wrong predictions, with False Positive or classifying the data as true when its false costs the client 100 dollars and False Negative or classifying the data as false when it is true costs the client 250 dollars, the Ensemble model provides the model with lowest loss. The loss per prediction for Ensemble model comes out to be $4.66.

Since model explain-ability is not a requirement, recommendation is to use the Ensemble model to minimize the loss.

## Appendix – Code

NB Viewer Link: Jupyter Notebook Viewer (nbviewer.org)
NB Viewer Link for Ensemble Model : Jupyter Notebook Viewer (nbviewer.org)