Introduction

The problem of direct marketing campaigns of banks has been around for a long time. From calling their clients about loan deals to selling insurance, Bank workers often call clients who do not buy into the campaign they're selling. Hence, given the data about client personal information, bank's previous experience with the clients about the campaigns and current economic index, we are tasked to create a model that predicts if the client would react positively or negatively to the campaign.

Dataset

Brief Description

The dataset is straight forward with 21 features, 20 predictors and 1 label column. The label is a binary. This is therefore a binary classification problem. The 20 predictors are divided into 4 categories.

1. Bank client variables: age, job, marital, education, default, housing, loan.
2. Related with the last contact of the current campaign: contact, month, day_of_week, duration.
    a. Because duration is like a predictor which is unknown beforehand - we will not be using this at all.
3. Other attributes: campaign, pdays, previous, outcome
4. Social and economic context attributes: emp.var.rate, cons.price.idx, cons.conf.idx, euribor3m, nr.employed

Data Analysis and Preprocessing

Initial Data Imputation

Following columns contain "unknown" values which are taken as NAN values. Below section will detail how many lines each.
Job has 330 rows with "unknown" values.
Marital has 80 rows with "unknown" values.
Education has 1731 rows with "unknown" values.
Default has 8597 rows with "unknown" values.
Housing has 990 rows with "unknown" values.
Loan has 990 rows with "unknown" values.

Columns with less than 1000 rows with unknown values will have these rows imputed. This reduces the total number of rows from 41188 to 39803. This is less than 10% which is very good.

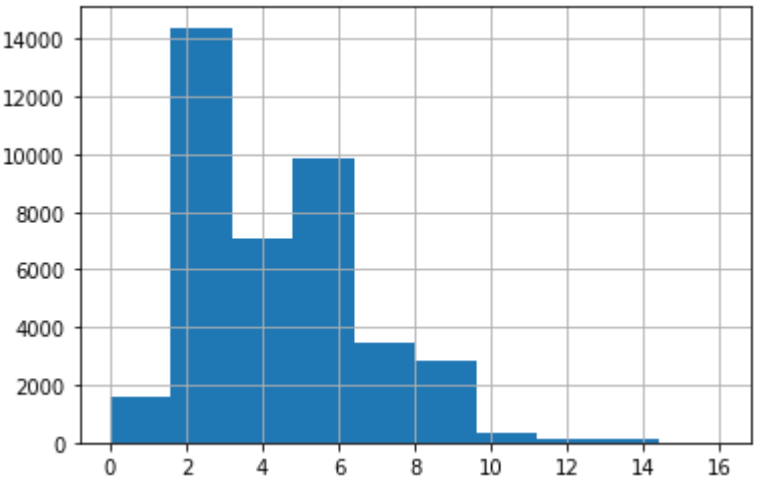
Meanwhile Education and Default will be one-hot encoded including the unknown value. This is because it is a large number which may have some predictive power when combined with other columns.

Data Preprocessing

Following detail new columns created and what were done to each column.
All columns are subjected to preprocessing by StandardScaler which follows the following formula.

> Standardize features by removing the mean and scaling to unit variance.
>
> The standard score of a sample `x` is calculated as:
>
> z = (x - u) / s
>
> where `u` is the mean of the training samples or zero if `with_mean=False`, and `s` is the standard deviation of the training samples or one if `with_std=False`.

Figure 1 shows the standardisation method from https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html

1. Bank-Client columns
    a. Age: Binned into a range of 5 from 15 to 100, a total of 17 bins.
        i. 

        ii. Figure 2 shows the distribution after binning
    b. Job: One-hot encoded
    c. Marital: One-hot encoded
    d. Education: One-hot encoded
    e. Default: One-hot encoded
    f. Housing: One-hot encoded
    g. Loan: one-hot encoded
2. Last Contact columns
    a. Contact: one-hot encoded
    b. Month: one-hot encoded although it could be argued that we can just encode it into numbers.
    c. Day_of_week: one-hot encoded although it is also arguable
    d. Duration: ignored

3. Other attributes columns
    a. Campaign: none
    b. Pdays: create new column call ‘is_contacted_pdays' where if pdays != 999 then 0 else 1.
    c. Previous: none
    d. poutcome: one hot encoding
4. Economic context attributes
    a. All variables are left as is since they are numeric but still are subjected to regular standardisation.

Correlation Matrix

From preprocessing steps from the previous 2 sections. Figure 3 shows the correlation values. However, due to one hot encoding, the result isn’t truly interpretable and the reader is advised to check it out in the notebook instead. Note that white columns imply empty columns which happens due to coding mishaps during preprocessing.

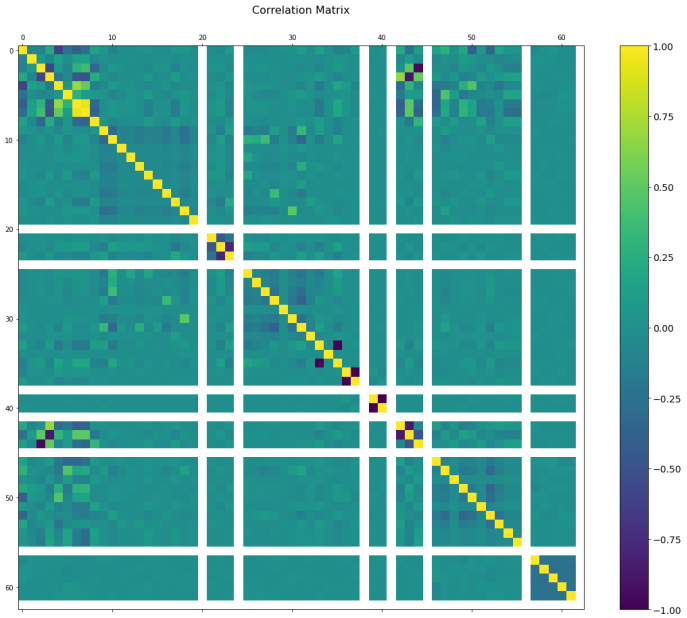


Figure 3

Modelling and Results

The modelling is divided into 4 rounds according to data used. Each round is subjected to its own modelling and only the best data will be shown per round.
We shall start with only bank client data and add more features as we go.

All models only have 3 dense layers, the dropout layers will depend on which round it is, usually only the later rounds of models have drop out layers.

All models have a learning rate reduction after certain epochs depending on which model it is.

Model visualisation is only shown in the final round.

Train Test Validation

Data is subjected to train, test, with split of 70%, 15%, 15%. The data is split in a stratified method which means that the split of labels are equal in proportion.

```
print(train_df[predict_column].value_counts(normalize=True))
print(test_df[predict_column].value_counts(normalize=True))
print(val_df[predict_column].value_counts(normalize=True))
```

```
0    0.887266
1    0.112734
Name: y, dtype: float64
0    0.88727
1    0.11273
Name: y, dtype: float64
0    0.887289
1    0.112711
Name: y, dtype: float64
```

This means that if we actually just predict only 0, we would be getting 88% accuracy, which I believe is not the point here. So in modelling, I attach AUC metric and RECALL metric to and also plot confusion matrices in order to see how well our models predict that the customer will opt into the campaign.

Round 1: Only Bank Client Columns

With only bank client columns, we only use 3 dense layers of 256, 256, and 64 nodes per respective layer. We achieve really high accuracy on both validation dataset and training data set but this is no different from predicting 0 only. Hence, we conclude that bank client data do not have very good prediction power.

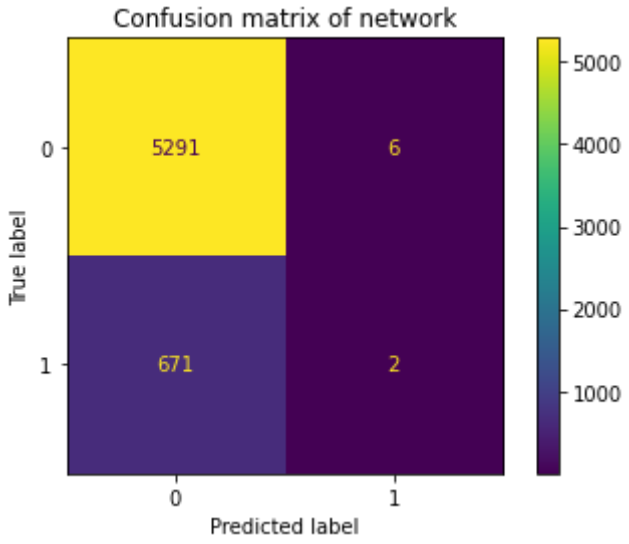


Figure 4 confusion matrix of network prediction on test dataset.

As we can see, we can barely recall the result. The recall value is almost close to 0. The result of this model is as follows:

1. val_accuracy: 0.8870
2. val_recall: 0.0045
3. val_auc: 0.6362

Round 2: Bank Client Columns + Other Attributes

Due to the increase in feature columns, we increased the nodes of the first layer to 512 but kept the others 256 and 64 respectively. We also add regularisation of 0.01. We gain substantially better recall values up to 0.2 in some epochs. However, the result is not very stable as can be seen in figure 5 and 6. This is still an improvement.

The result is as follows:

1. val_accuracy: 0.8890
2. val_recall: 0.0357
3. val_auc: 0.6877.

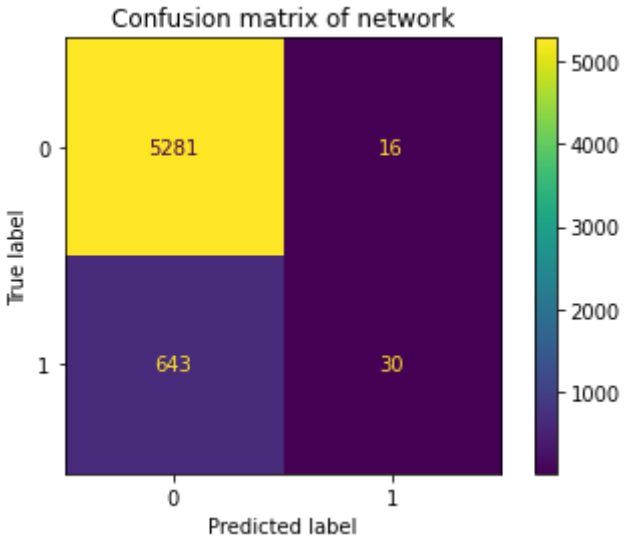


Figure 5

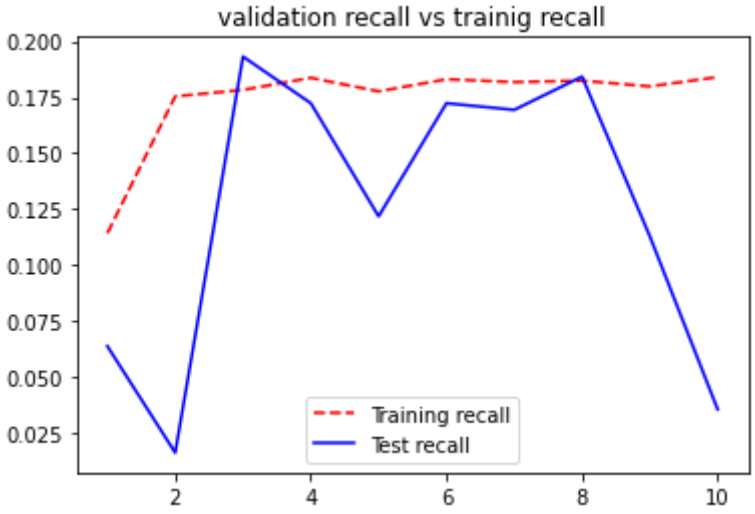


Figure 6

Round 3: Add economic attributes

We leave the same model as before and got the following results:

1. val_accuracy: 0.8982
2. val_recall: 0.1694
3. val_auc: 0.7701

As we can see in Figure 7 and Figure 8, we got a create improvement with some recall reaching above 20% and never go below 10%.

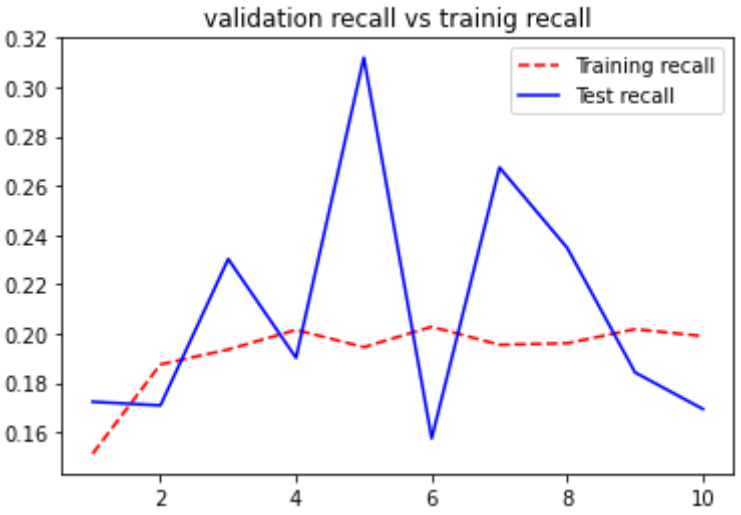


Figure 7

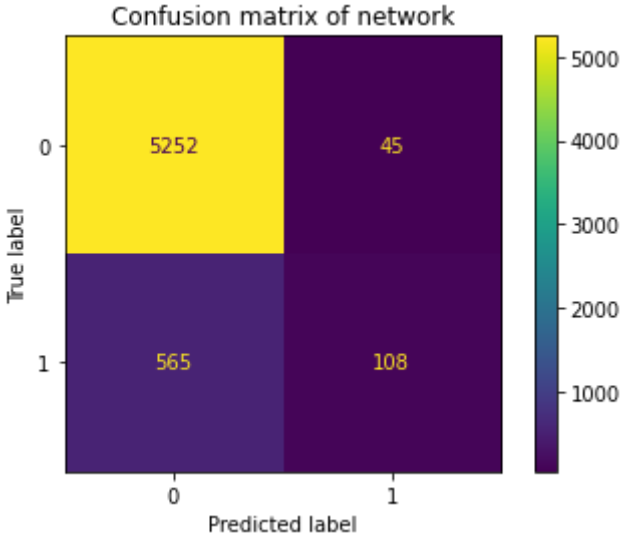


Figure 8

Round 4: Add last contact

We gain more recall but nothing substantial and did not gain stability and hence choose to omit graphs and result and move on to the final round

Round 5: Oversampling SMOTE

Due to low positive labels in the training dataset, we generate fake data based on positive labels in the training dataset via SMOTE technique and add these fake data for training. This means that we now have way more training data points and we have a balanced training dataset. The size of validation and testing dataset REMAIN as is because they represent the real world data.
We also reduce the model layers to 256, 256, 64 and add dropout layers of 0.1 per layer. Once again we kept regularisation of l2 to 0.1. Figure 14 shows our final model layers.
We are able to achieve a decent recall (60%+) with expected accuracy of 80% and above in our final model.
The result of final model is as follow:
The result is as follows: (max 1, basically multiply by 100 for percentage)

1. loss: 0.4821
2. accuracy: 0.8053
3. recall: 0.7827
4. auc: 0.8836
5. val_loss: 0.4682
6. val_accuracy: 0.8486
4. val_recall: 0.6077
5. val_auc: 0.7977

As we can see, most of the time, the validation accuracy is higher than training accuracy. This is not an indicator of non-overfitting models since they contain different class distributions. Despite the great result of achieving over 80% accuracy and over 60% recall, in figure 13, we can see that our model is slightly overfitting because the AUC result is not very good, over 9% difference from training and testing. However, since we achieve our objective of creating a relatively generalizable model, we stop our project here.

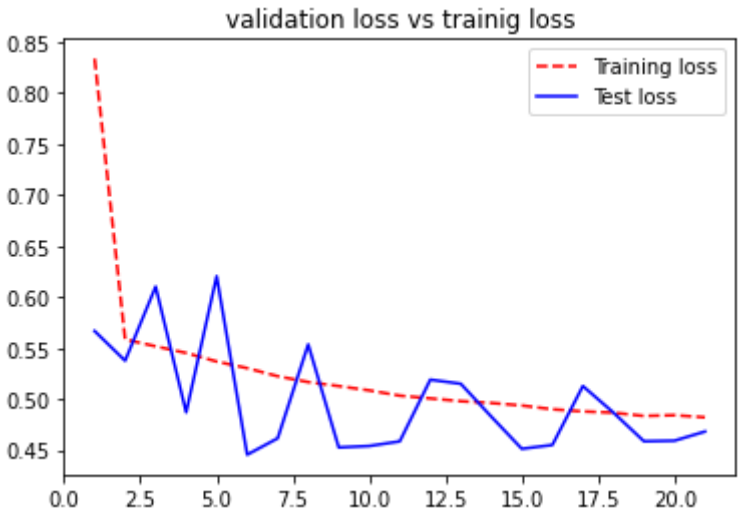


Figure 9 Loss of the final model

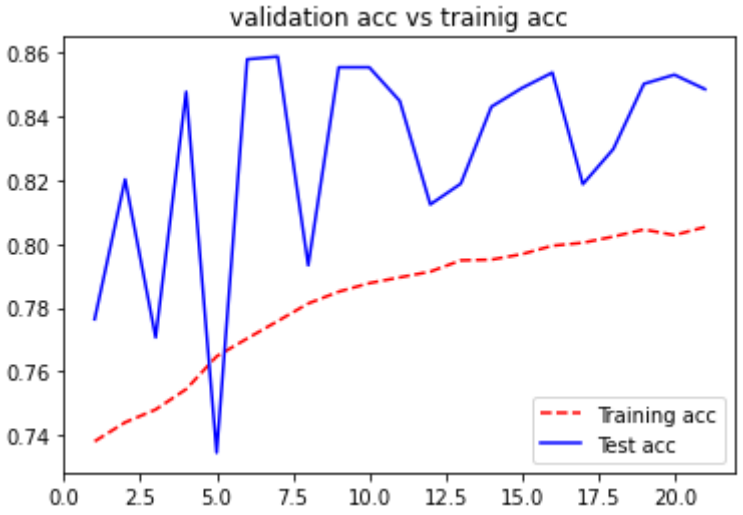


Figure 10 Accuracy of the final model

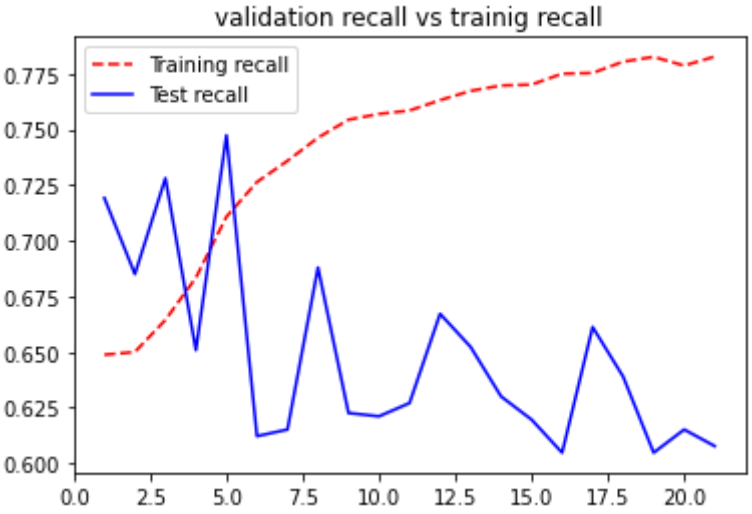


Figure 11 recall of the final model

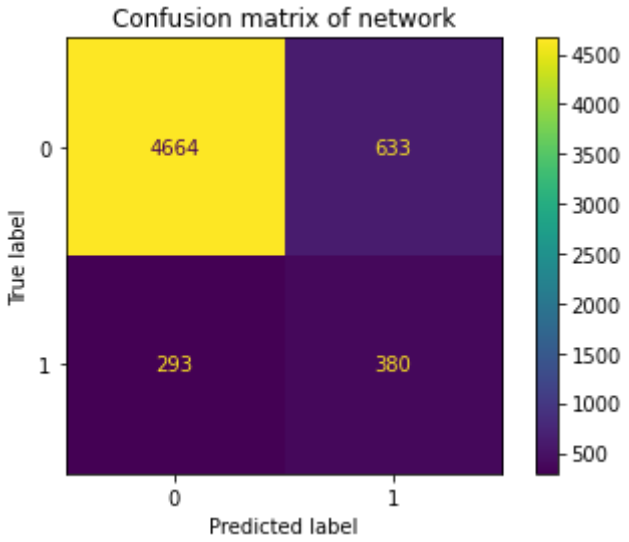


Figure 12 confusion matrix of the final model

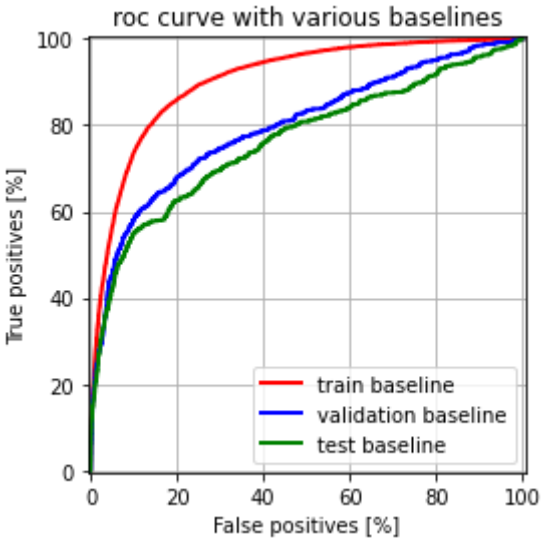


Figure 13 ROC of the final model showing some overfitting but still give good result

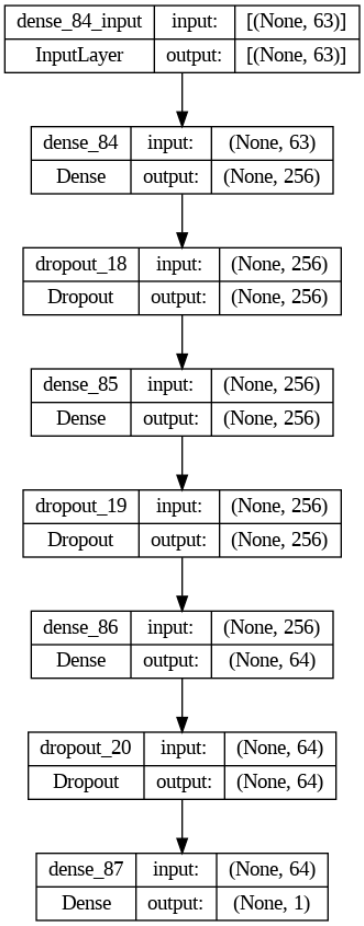


Figure 14 Network Architecture Visualization

Conclusion

Overall, I believe that I have achieved the goal of the project since the start which is to build a model to predict if client calls would respond positively to the campaign. Given the challenge of class imbalance favouring the negative response. It was easy to create a model that would predict purely negative predictions which would give an above 80% accuracy. However, this is simply not the aim of the project and as a result, the aim is to increase the recall of the model to above 50% while keeping the model accuracy to be above 80%. This, as a result, leads to the usage of SMOTE technique to oversample the training dataset which as a result yields a testing and validation accuracy of more than 80% while still keeping the recall result to be above 60%.

What I could have done better might be to compare the model if SMOTE is used to generate data for validation and testing too in order to balance the class dataset. However, I believe that this would not reflect the real world use of incoming data distribution and therefore, did not experiment with it. The lack of time does play a part in this decision.