ANALYTICS IN PRACTICE

Group 8

2042266 / 2082046 / 1980073 / 2086639 / 2092588 / 2092650

wb8consultancy

London • Frankfurt • Taipei • Moscow • Milan • Istanbul • Nashville

Contents

Introduction

wb8 is a data-driven consultancy founded in 2020 following a research project at the Warwick Business School.

In the fall of 2020, we were contacted Universal Plus, who requested us to develop and deploy a customer transaction prediction system, which we assumed was related to the mortgage sector of the bank. We proceeded by following the steps defined by the Cross-Industry Standard Process for Data Mining (CRISP-DM).

Business Understanding

Technology has enabled banks to better assists people with their financial needs, starting with ordinary transactions like cash deposits and ending with mortgage issuance. With the help of modern Machine Learning (ML) algorithms, banking challenges are managed more effectively, maximising the customer experience and minimising the bank's costs

The bank is seeking to implement a customer transaction prediction system, essential for forecasting which clients will make a particular transaction. Having such a technology in place means that the bank can take constructive measures to increase customers satisfaction and implement an effective customer retention strategy. Therefore, the main project goal is to determine which clients will make a specific transaction in the future.

Data Understanding

The dataset we were provided with contained 200 anonymised variables and 100,000 observations. Our variable of interest is the "target" variable, which indicates customers who made a specific transaction with “1” and "0" otherwise.

The data is highly imbalanced as "1" occupies approximately 10% of the total observations. The methodology used to tackle this problem was inspired by Pozzolo's et al. (2014) analysis of fraud detection data, which also was imbalanced and contained little information regarding each transaction. The article covers the problem of imbalanced data, sampling techniques, most effective supervised algorithms and model evaluation methods. Given the similarities between the structures of our datasets, we were directly able to emulate the techniques used in the sections about data balancing, supervised algorithms, and model evaluation to guide our modelling and reporting.

Data Preparation

We began by removing the variables not related to the target. For example, we omitted "ID_code" because it has no impact on the 'customers' decision-making. Moreover, we

removed all the NA values as they might have interfered with the results. Lastly, we converted the target variable into a factor, a necessary step for the modelling phase.

Furthermore, to balance the data, we utilised four sampling approaches: over-sampling, both-sampling, under-sampling, and stratified sampling. Also, different partitions of training and test sets were used, as they also allowed for different results.

Modelling

The models used include Support Vector Machine (SVM), Linear Discriminant Analysis (LDA), Decision Trees, Random Forrest (RF), Generalised Linear Model (GLM), Neural Network Model (NNM) and Naïve Bayes. Pivotal in the modelling phase was the work of Shaltout et al. (2014), who illustrates effective methods for handling and selecting variables. The Information gain (IG) determines the amount of knowledge gained about a target when the value of an attribute is known. The article also highlights ways to overcome time-consuming algorithms and enhance their classification efficiently, pivotal for our project. We calculated the IG for every variable, as it helped us gain insights on our training set. Successively, we tested our models with the top 10, 50, 100, or all variables with positive IG.

Our imbalanced data challenges were eased by Zhao's et al. article (2016), more specifically the parts about imbalanced data and RF models. One of the most common methods for dealing with imbalanced data is oversampling the minority class, leading to an overall balanced class distribution. Although such an approach often creates a model with a high degree of generalizability. The article offers a different approach: stratified over-sampling, which involves clustering training sets multiple times, grouping them by their entropies, and then taking a stratified sample from the clusters. As a result, the obtained sampled datasets are balanced, varied and suitable to train the RF model. This article relates to the issues we encountered with Universal Plus's data, which was also highly imbalanced. RF is a robust algorithm that outperformed many of our models, as it deals with overfitting better than most models.

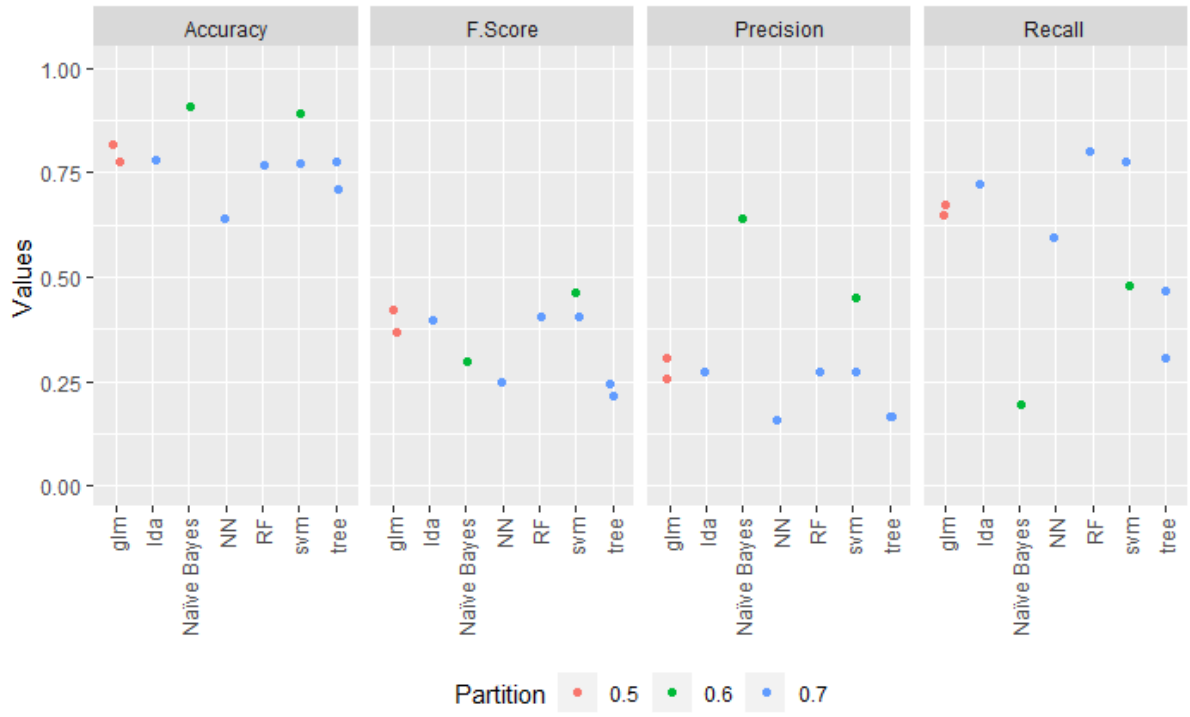
We improved our models using various techniques, such as K-fold cross-validation. We considered the study of Wong and Yeh (2020), aimed at questioning the repeated use of K-fold cross-validation to obtain reliable accuracy estimates. As opposed to the existing literature, the authors outline how repeating the K-fold cross-validation, in fact decreases the variance of the accuracy of the models. As a result, the authors find it inappropriate to say that the mean accuracy resulting from a larger number of replications would be more reliable. Moreover, the authors claim that their findings are particularly useful for imbalanced datasets. They conclude that the K-fold cross-validation should be adopted with a larger number of folds and a smaller number of replications. They also recommend using ten folds

as a first choice and five folds twice as a second choice. Therefore, given the article's insights, and considering the dimension of the dataset, we tried tuning our SVM with ten folds.

Moreover, we considered Cervantes's et al. (2020) article, outlining the theory behind the SVM algorithm, the different kernels (Gaussian, Linear, Radial), as well as SVM's weaknesses. Given that the training kernel matrix grows quadratically, the SVM works very slowly in large datasets. Also, the SVM tends to perform poorly on imbalanced data. Due to the difficulty of achieving the optimal separation hyperplane for SVM with the imbalanced data, predicting the minority class without using any sampling techniques can give wrong results. In conclusion, even though SVM has disadvantages, this method can provide excellent results. Therefore, before applying SVMs on the data, the dataset should be balanced by using a sampling technique.

Evaluation

We used different parameter settings for each model, including partitioning both the training and test sets, selecting various informative variables, and various sampling methods. Successively, we identified the best settings for each model and compared the best models by Accuracy, Recall, F-Score and Precision. The visual representation can be found below, and the R code of the best models in the Appendix.



The plots demonstrate how the measures vary between models. As per the visual representation, the partition represents the split ratio between the training and test data. For example, GLM used a 0.5 partition.

To decide the most beneficial model for Universal Plus, we created cost matrices for two scenarios and calculated the expected profit for each model. We assume that after identifying the promising customers through the model, Universal Plus will target these clients to offer them a mortgage.

Following our research on the UK housing market (Jones, 2020), we were able to estimate the average mortgage in the UK as £ 200,000. Based on the development of the ten-year fixed interest rates (Bank of England, 2019), we assume an average interest rate of 2.5%. Predicting, therefore, an additional annual revenue for identifying promising customer of (£200,000 × 2.5%) £ 5,000 per customer.

First Scenario - Low Cost

TP Profit	£	5,000.00
Cost	£	50.00

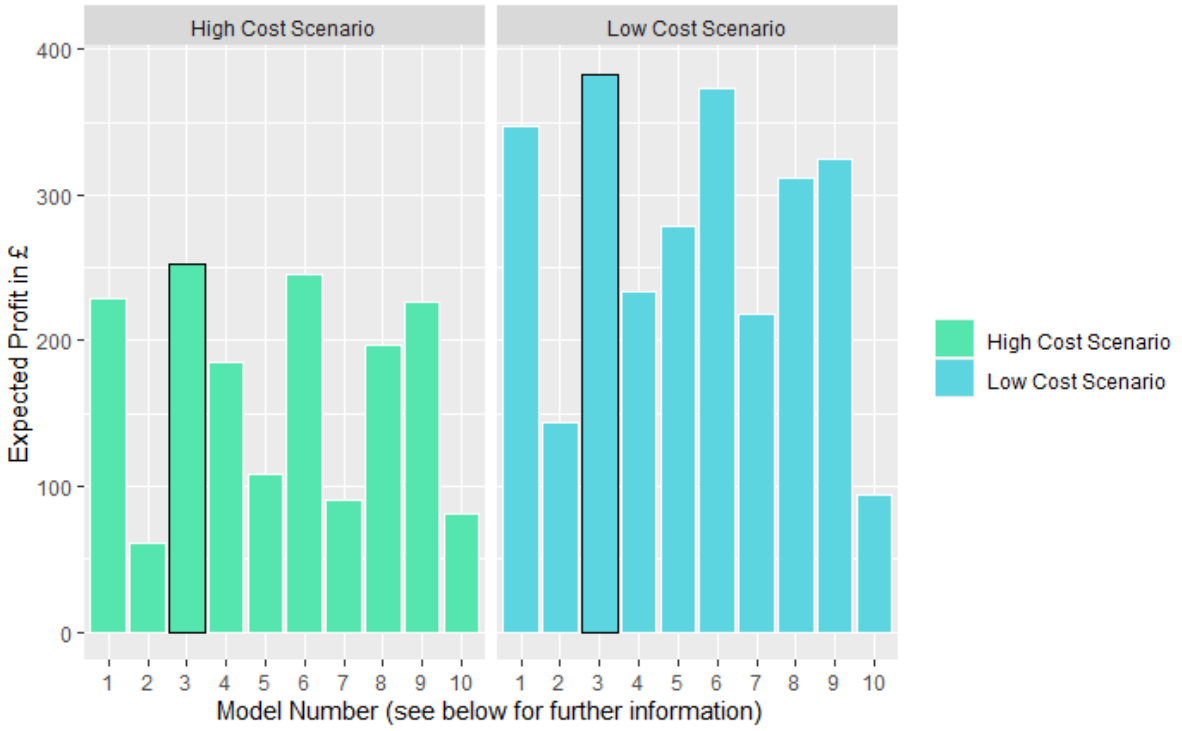
		Actual	
		0	1
Predicted	0	£ -	£ -
	1	-£ 50.00	£ 4,950.00

Second Scenario - High Cost

TP Profit	£	5,000.00
Cost	£	500.00

		Actual	
		0	1
Predicted	0	£ -	£ -
	1	-£ 500.00	£ 4,500.00

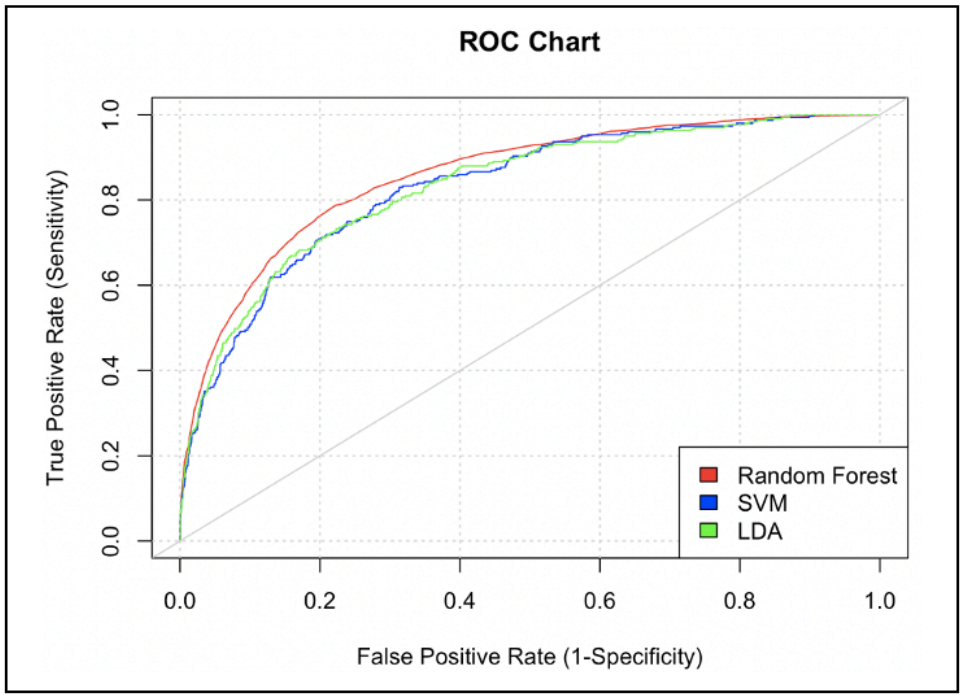
On the cost side, the promotion campaign that will be implemented to attract these clients will have two alternatives. The actual prices of the promotion campaign are unknown to us. Therefore, we created two scenarios outlining possible situations for the bank, demonstrating the effectiveness of our model with both scenarios. Having low promotional costs is our first scenario, and we estimated those to be 1% of the profit. The second scenario corresponds to the high promotional costs, where we estimated costs to be 10% of the profit. The graph below shows the expected payoff for all models in both scenarios.

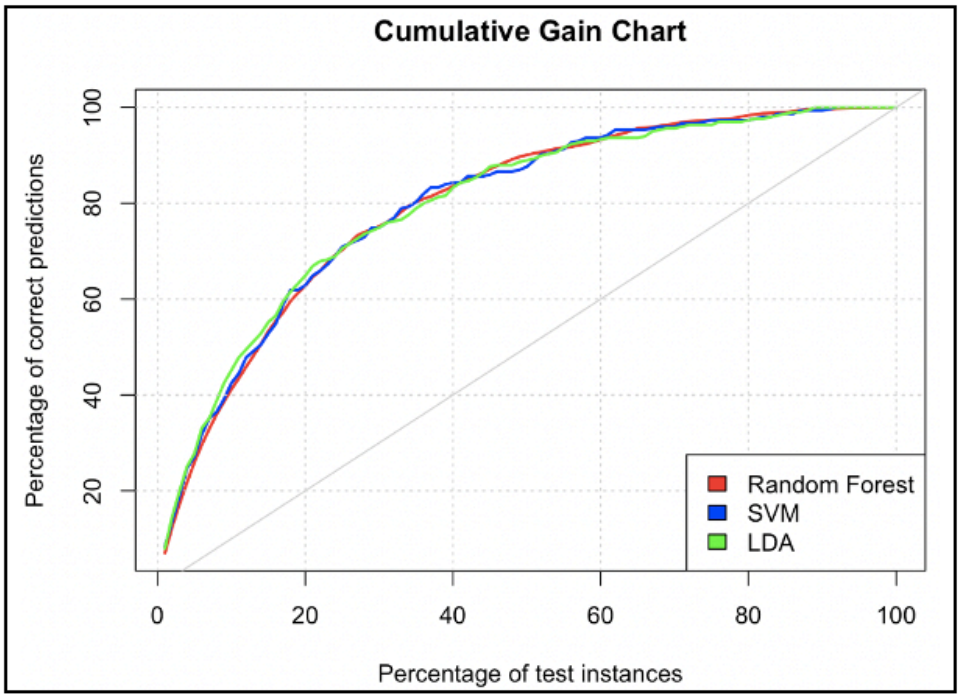


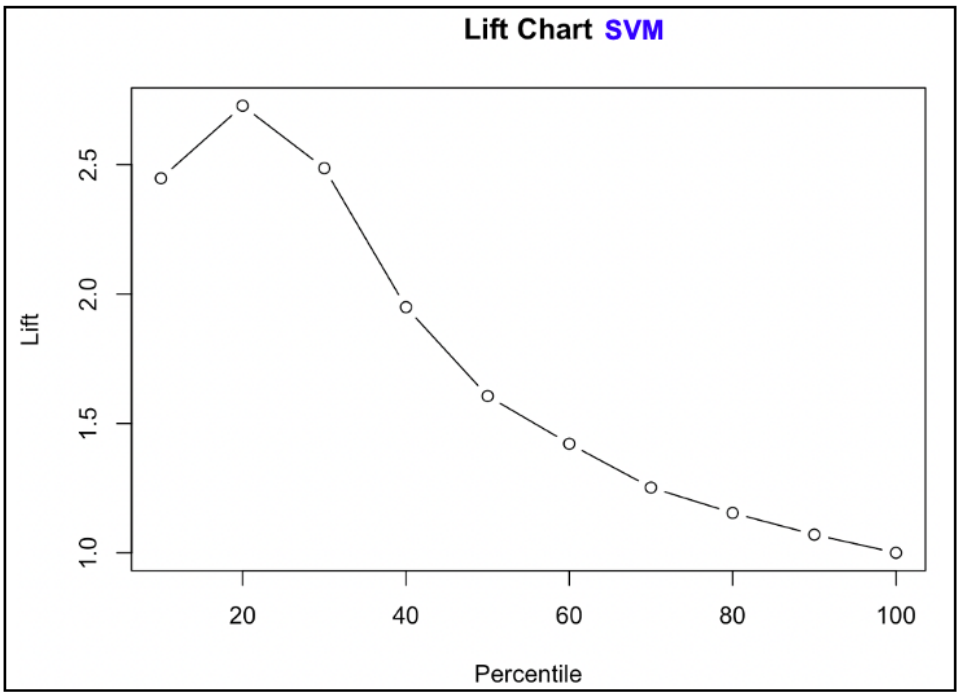
Model Number	Description
1	Partition of 70/30 for LDA on an oversampled dataset (all variables).
2	Partition of 70/30 for the decision tree (type ctree) on an oversampled dataset from which we have selected 50 variables with the highest IG.
3	Partition of 70/30 and used a combination of stratified sampling and undersampling and selected the 100 variables with the highest IG for a RF model.
4	Partition of 60/40 and used both over and undersampling on all variables with IG greater than zero for a radial SVM.
5	Partition of 70/30 and used SMOTE sampling on all variables for NNM.
6	Partition of 70/30 and used oversampling on all variables with IG greater than zero for a linear SVM.
7	Partition of 70/30 for the decision tree (type tree) on an oversampled dataset from which we have selected 50 variables with the highest IG.
8	Partition of 50/50 and used undersampling on all variables for a GLM.
9	Partition of 50/50 and used oversampling on all variables for a GLM.
10	Partition of 60/40 and used oversampling on all variables for a Naive Bayes Model.

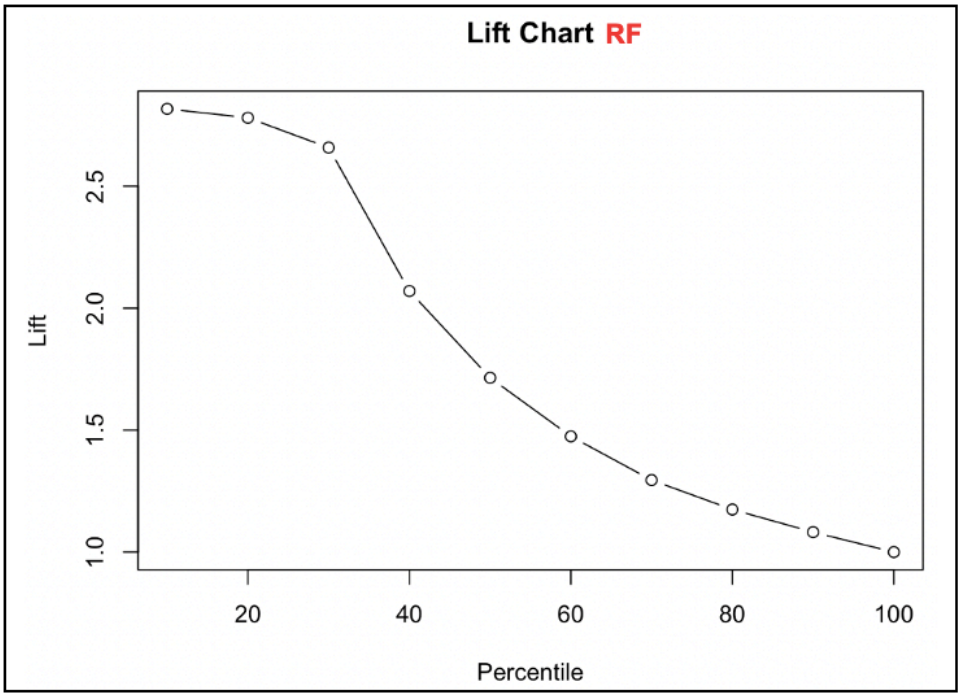
Based on the cost matrices and a comparison graph above, we were able to select the three best performing models in terms of expected profit. These models were the RF ran on the stratified sample with under-sampling and top 100 IG variables. Secondly, the SVM model with oversampled data and with all the variables included. Lastly, the LDA model with oversampled data and with all of the variables included.

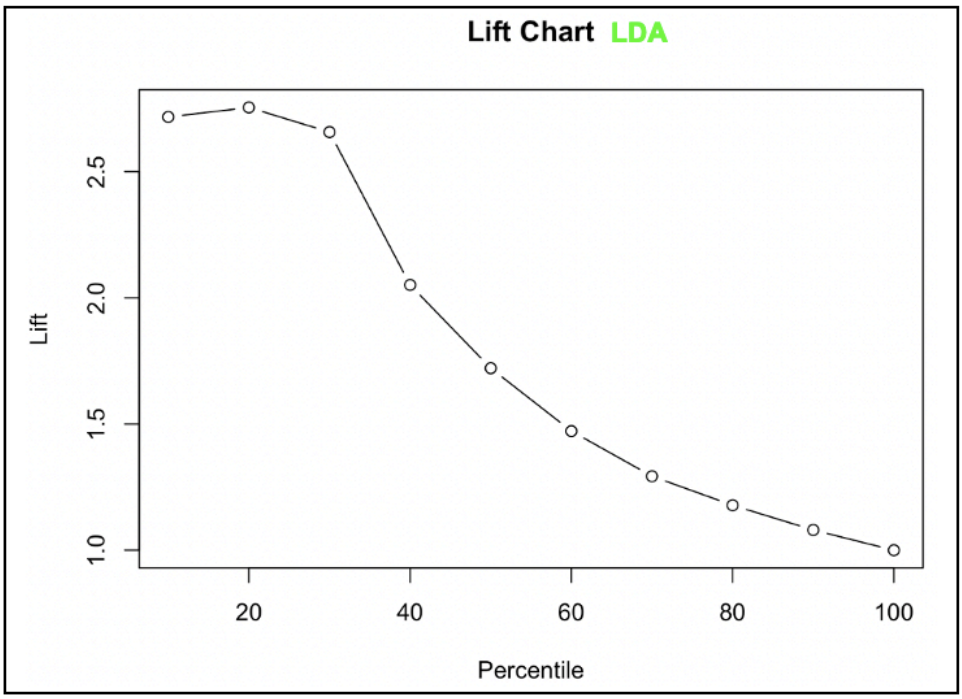
To assess and identify the best performing model, we produced a ROC, Cumulative Gain and Lift charts, along with the AUC comparison table.






Lift Chart SVM
Lift
1.0
1.5
2.0
2.5
20
40
60
80
100
Percentile

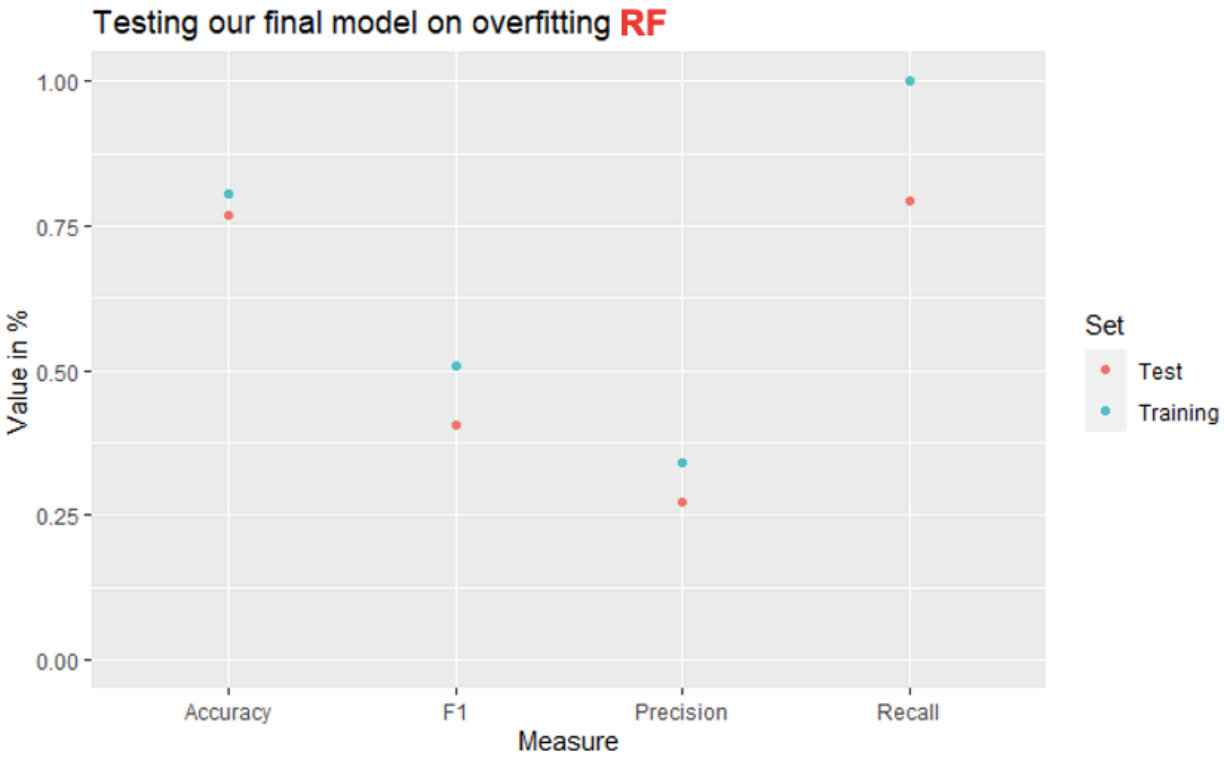

Lift Chart RF
Lift
1.0
1.5
2.0
2.5
20
40
60
80
100
Percentile



Model	AUC
Random Forest	0.8574767
Support Vector Machine	0.8310219
Linear Discriminant Analysis	0.8323706

The ROC chart above demonstrates that RF has a better performance compared to LDA and SVM, as its curve is closer to the top left corner, meaning that given a fixed False Positive Rate RF has the highest True Positive Rate. The latter is also highlighted in the AUC table, with a value of 0.8574767. The Cumulative Gain Chart indicates that all three models have similar performance in terms of identifying a particular outcome. For example, the top 20% of customers with the highest predicted probabilities contains approximately 60% of the correct predictions for all models. By considering the Cumulative Gain Chart alone, it cannot be inferred which model performs the best. Moreover, looking at the Lift Charts, all three models have similar patterns with slight differences in the 0-30 percentile range. Nevertheless, all models show considerable improvements for this range, for example, by contacting 20% of the customers the number of respondents reached would be more than 2.5 times higher than if no models were used.

Overall, the RF yields the highest expected profit for both scenarios and has the highest Recall value. Moreover, it is the best ROC based model and one of the best Cumulative Gain and Lift chart models, meaning that considering these criteria outlined it appears that RF generally has a better performance. (See Appendix for R-code)



However, before finalising the model we had to make sure it did not overfit. Hawkins (2003) defines overfitting as the use of models that violates parsimony, meaning, the use of over-complicated approaches to describe a relationship between two or more variables. The critical problem with overfitting arises from the structure of the ML task. The goal of a learning algorithm is to accurately guess the values of unseen data rather than the data with which it was trained. Overfitting occurs when too much tweaking is put into achieving a forced high accuracy prediction for the training data, it leads to the model to memorise the training data instead of identifying influencing variables (Dietterich,1995).

Thus, to test if our models were overfitting, we reran our models on the training set and compared the results. As visualised in the graph above, the accuracy of our model in the training set is only slightly higher than in the test set and was therefore considered fit to be deployed.

Deployment

Our model would be deployed into the bank's internal IT system, facilitating the implementation with the existing systems and allowing the bank to identify and consequently target the promising customers easily. Moreover, the employees will not have to understand the model as we will provide a dashboard based on the algorithm of the RF model. This would provide better visualisation of the potential clients and would be easy to use. Moreover, this dashboard would not only promote efficiency but also would increase employee satisfaction, as it would reduce the number of rejects significantly, while of course

increasing the success rate. Our solution would reduce the costs and increase the profitability and efficiency of the bank.

Conclusion

We addressed the problem by applying the CRISP-DM showing that the RF model with stratified sampling, 70% partition, and top 100 IG selected provides the best solution for Universal Plus, providing the highest expected profits for both low-cost and high-cost scenarios.

Compared to other models, the RF has the highest Recall rate of 79.45%, which means that the model can detect promising customers accurately. Considering the nature of the data we were provided with, we believe this to be an excellent result, that will improve as the quality of the data improves. Comprehensively examining the results related to model performance is critical, therefore even considering the relatively low accuracy rate (76%) of the RF model, we believe it is less crucial for the task, as the bank might fail to communicate with promising customers in case accuracy was prioritised over recall. In conclusion, RF is the best model to meet Universal Plus's requirement and it correctly predicts future transactions.

References

Cervantes, J. & Garcia-Lamont, F. & Rodriguez-Mazahua, L. & Lopez, A. (2020) A comprehensive survey on support vector machine classification: Applications, challenges and trends. *Neurocomputing,* vol. 408: 189-215.

Hawkins, D. (2003) The problem of Overfitting. *Journal of Chemical Information and Computer Sciences*, *44* (1): 1-12.

Pozzolo, A. & Caelen, O. & Borgne, Y. L. & Waterschoot, S. & Bontempi, G. (2014) Learned Lessons in Credit Card Fraud Detection from a Practitioner Perspective. *Expert Systems with Applications*, vol. 41, no. 10: 4915–4928.

Shaltout, N. & Elhefnawi, M. & Rafea, A. & Moustafa, A. (2014) *Information Gain as a Feature Selection Method for the Efficient Classification of Influenza Based on Viral Hosts*. Lecture Notes in Engineering and Computer Science.

Wong, T. & Yeh, P. (2020) Reliable Accuracy Estimates from k-Fold Cross Validation. *IEEE Transactions on Knowledge and Data Engineering*, vol. 32, no. 8: 1586-1594.

Zhao, H. & Chen, X & Nguyen, T. & Huang, J. Z. & Williams, G. & Chen, H. (2016) *Stratified Over-Sampling Bagging Method for Random Forests on Imbalanced Data*. Pacific-Asia Workshop on Intelligence and Security Informatics.

Web sources:

Jones, N., 2020. *UK House Price Index - Office For National Statistics*. [online] Ons.gov.uk. Available from: https://www.ons.gov.uk/economy/inflationandpriceindices/bulletins/housepriceindex/august2020 [Accessed 8 December 2020].

Statista. 2020. *UK: Average Mortgage Rate 2019 | Statista*. [online] Available at: https://www.statista.com/statistics/386301/uk-average-mortgage-interest-rates/ [Accessed 8 December 2020].

Appendix 1: Calculation of Cost Matrices

Expected profit per customer to account for different samplings

First Scenario - Low Cost

TP Profit	£	5,000.00
Cost	£	50.00

	Low Cost	
	0	1
0	0	0
1	-£ 50.00	£ 4,950.00

[1]

	Low Cost	
	0	1
0	2152	84
1	581	219

Profit £ 347.50

[2]

	Low Cost	
	0	1
0	2239	207
1	459	92

Profit £ 144.29

[3]

	Low Cost	
	0	1
0	14461	429
1	4429	1668

Profit £ 382.86

[4]

	Low Cost	
	0	1
0	3364	208
1	234	191

Profit £ 233.61

[5]

	Low Cost	
	0	1
0	1745	121
1	953	178

Profit £ 278.09

[6]

	Low Cost	
	0	1
0	2080	67
1	618	232

Profit £ 372.87

[7]

	Low Cost	
	0	1
0	1990	160
1	708	139

Profit £ 217.77

[8]

	Low Cost	
	0	1
0	3554	175
1	943	324

Profit £ 311.58

[9]

	Low Cost	
	0	1
0	2993	131
1	605	268

Profit £ 324.33

[10]

	Low Cost	
	0	1
0	3555	322
1	43	77

Profit £ 94.82

Second Scenario - High Cost

TP Profit	£	5,000.00
Cost	£	500.00

	High Cost	
	0	1
0	0	0
1	-£ 500.00	£ 4,500.00

[1]

	Low Cost	
	0	1
0	2152	84
1	581	219

Profit £ 228.92

[6]

	Low Cost	
	0	1
0	2239	207
1	459	92

Profit £ 61.56

[3]

	Low Cost	
	0	1
0	14461	429
1	4429	1668

£ 252.13

[4]

	Low Cost	
	0	1
0	3364	208
1	234	191

£ 185.76

[11]

	Low Cost	
	0	1
0	1745	121
1	953	178

£ 108.27

[6]

	Low Cost	
	0	1
0	2080	67
1	618	232

£ 245.25

[7]

	Low Cost	
	0	1
0	1990	160
1	708	139

£ 90.59

[8]

	Low Cost	
	0	1
0	3554	175
1	943	324

£ 197.46

[9]

	Low Cost	
	0	1
0	2993	131
1	605	268

£ 226.04

[10]

	Low Cost	
	0	1
0	3555	322
1	43	77

£ 81.31

Model Legend

Model Number	Description
[1]	We have used a partition of 70/30 for our LDA on an oversampled dataset (all variables).
[2]	Partition of 70/30 for the decision tree (type ctree) on an oversampled dataset from which we have selected the 50 variables with the highest IG.
[3]	Partition of 70/30 and used a combination of stratified sampling and undersampling and selected the 100 variables with the highest IG for a RF model.
[4]	Partition of 60/40 and used both over and undersampling on all variables with IG greater than zero for a radial SVM.
[5]	Partition of 70/30 and used SMOTE sampling on all variables for NNM.
[6]	Partition of 70/30 and used oversampling on all variables with IG greater than zero for a linear SVM.
[7]	Partition of 70/30 for the decision tree (type tree) on an oversampled dataset from which we have selected the 50 variables with the highest IG.
[8]	Partition of 50/50 and used undersampling on all variables for a GLM.
[9]	Partition of 50/50 and used oversampling on all variables for a GLM.
[10]	Partition of 60/40 and used oversampling on all variables for a Naive Bayes Model.

Appendix 2

WB8 Consultancy

09/12/2020

Contents

Import necessary libraries

```
library(dplyr)
library(ggplot2)
library(tidyverse)
library(FSelector)
library(e1071)
library(caTools)
library(vctrs)
library(ROSE)
library(caret)
library(randomForest)
library(pROC)
library(partykit)
library(CustomerScoringMetrics)
library(plyr)
library(tree)
library(MASS)
```

Read the CSV files

```
#data to be used in models later
datasmall <- read_csv("datafile_small.csv")

data <- read_csv("datafile_full.csv")

#We tested models that are more time consuming on a smaller dataset
```

1 Data Preparation

```r
#convert target into factor
data$target <- as.factor(data$target)

#Check levels of the target variable
levels(data$target)

#remove id
data$ID_code <- NULL

#remove na
data <- na.omit(data)

#Repeat the same with small dataset
datasmall$target <- as.factor(datasmall$target)

#Check levels of the target variable
levels(datasmall$target)

#remove id
datasmall$ID_code <- NULL

#remove na
datasmall <- na.omit(datasmall)
```

1.1 Apply Stratified Sampling

```r
#Use stratified sampling for large dataset
set.seed(10)
stratified_data <- splitstackshape::stratified(data, "target", 0.7)
```

1.2 Splitting

```r
set.seed(123)

# Split large stratified dataset into training and test sets
split= caTools:: sample.split(stratified_data$target, SplitRatio =0.70)

training_set = subset(stratified_data, split==TRUE)
test_set =subset(stratified_data, split==FALSE)

#Split small dataset into training and test sets
split2= caTools:: sample.split(datasmall$target, SplitRatio =0.70)

training_set_small = subset(datasmall, split2==TRUE)
test_set_small =subset(datasmall, split2==FALSE)
```

2 Code for Top 3 best performing models

2.1 Random Forest

2.1.1 Undersampling

```
#Apply undersampling to the large dataset to be used with Random Forest
set.seed(123)
undersample_data <- ROSE:: ovun.sample(target~. , data = training_set, method = "under",
                                                                    p = 0.5, seed = 1)$data
```

2.1.2 Selecting top 100 attributes based on their Information Gain

```
attribute_weights_under <- FSelector::information.gain(target~., undersample_data)
attribute_weights_under <- attribute_weights_under %>% arrange(desc(attr_importance))
selected_attribute_under <- FSelector:: cutoff.k(attribute_weights_under, 100)

undersampledata <- undersample_data[selected_attribute_under]
undersampledata$target=undersample_data$target
```

2.1.3 Model

```
#Random Forest tuning
training_features_under <- subset(undersampledata, select = -target)

set.seed(111)
tuneRF(training_features_under, undersampledata$target, mtryStart = 3, ntree = 500,
       stepFactor = 1.5, improve = 0.01)

RF_model_under<- randomForest(target~., undersampledata, mtry = 3, ntree = 500)

prediction_RF_under<- predict(RF_model_under, test_set)

(cf_random_under_70_100IG <- caret:: confusionMatrix(prediction_RF_under,
                test_set$target, positive = '1', mode = "prec_recall"))
```

2.1.4 Check RF for overfitting

```
prediction_RF_under_train <- predict(RF_model_under, training_set)

(cf_random_under_70_100IG_train <- caret:: confusionMatrix(prediction_RF_under_train,
                        training_set$target, positive = '1', mode = "prec_recall"))
```

2.2 SVM

2.2.1 Oversampling

```
set.seed(123)

oversample_data_70 <- ROSE:: ovun.sample(target~. , data = training_set_small,
                                      method = "over", p = 0.5, seed = 1)$data
```

2.2.2 Selecting all attributes with positive Information Gain

```
attribute_weights_over_70_pos <- FSelector:: information.gain(target~., oversample_data_70)

selected_attribute_over_70_pos <- attribute_weights_over_70_pos
%>% filter(attr_importance > 0) %>%
mutate(indexNames = row.names.data.frame(.)) %>% pull(., indexNames)

oversample_data_all_70_pos <- oversample_data_70
oversample_data_70_pos <- oversample_data_all_70_pos[selected_attribute_over_70_pos]
oversample_data_70_pos$target <- oversample_data_all_70_pos$target
```

2.2.3 Model

```
svm_model_over_linear_70_pos <- e1071:: svm(target~., data = oversample_data_70_pos,
                                    kernel="linear", scale=TRUE, probability = TRUE)


svm_prediction_over_linear_70_pos <- predict(svm_model_over_linear_70_pos, test_set_small)

(cf_svm_over_linear_70_pos <- caret:: confusionMatrix(svm_prediction_over_linear_70_pos,
                           test_set_small$target, positive = "1", mode = "prec_recall"))
```

2.2.4 Check SVM for overfitting

```
svm_prediction_over_linear_70_pos_fitting <- predict(svm_model_over_linear_70_pos,
                                                     oversample_data_70_pos)

(cf_svm_over_linear_70_pos_fitting <- caret::
confusionMatrix(svm_prediction_over_linear_70_pos_fitting,
oversample_data_70_pos$target, positive = "1", mode = "prec_recall"))
```

2.3 LDA

2.3.1 Oversampling

```
set.seed(123)
ov_training <- ovun.sample(target~., data = training_set_small, method = "over",
                                                    p = .5, seed = 10)$data
```

2.3.2 Model

```
lda_ov <- lda(target~., ov_training)
predict_lda <- predict(lda_ov, test_set_small, type="class")$class
(cf_lda_over_70=confusionMatrix(predict_lda, test_set_small$target,
                                positive="1",mode="prec_recall"))
```

2.3.3 Check LDA for overfitting

```
predict_lda_overfitting <- predict(lda_ov, ov_training, type="class")$class
(cf_lda_over_70_fitting=confusionMatrix(predict_lda_overfitting, ov_training$target,
                                                    positive="1",mode="prec_recall"))
```

3 Plots for top 3 models

3.1 ROC plots

```
RF_model_under_prob=predict(RF_model_under, test_set, type ="prob")[,2]
SVMpred <- predict(svm_model_over_linear_70_pos , test_set_small, probability = TRUE)
SVM_prob <- attr(SVMpred, "probabilities")[,2]
LDA_prob <- predict(lda_ov, test_set_small, type="prob")$posterior[,2]
```

```
roc_RF=roc(test_set$target, RF_model_under_prob)
roc_SVM <- roc(test_set_small$target, SVM_prob)
roc_LDA <- roc(test_set_small$target, LDA_prob)


df_RF = data.frame((1-roc_RF$specificities), roc_RF$sensitivities)
df_SVM = data.frame((1-roc_SVM$specificities), roc_SVM$sensitivities)
df_LDA = data.frame((1-roc_LDA$specificities), roc_LDA$sensitivities)
```

```
plot(df_RF, col="red", type="l",
xlab="False Positive Rate (1-Specificity)",
ylab="True Positive Rate (Sensitivity)", main="ROC Chart")
lines(df_SVM, col="blue")
lines(df_LDA, col="green")
grid(NULL, lwd = 1)

abline(a = 0, b = 1, col = "lightgray")

legend("bottomright",
c("Random Forest","SVM", "LDA"),
fill=c("red","blue", "green"))
```

3.2 AUC Table

```
auctable=tibble("model"=c("RF", "SVM", "LDA"),"AUC"=c(auc(roc_RF),
                                    auc(roc_SVM), auc(roc_LDA)))
auctable
```

3.3 Cumulative Gain Charts

```
GainTable_RF <- cumGainsTable(RF_model_under_prob, test_set$target, resolution = 1/100)
GainTable_SVM <- cumGainsTable(SVM_prob, test_set_small$target, resolution = 1/100)
GainTable_LDA <- cumGainsTable(LDA_prob, test_set_small$target, resolution = 1/100)
```

```
plot(GainTable_RF[,4], col="red", type="l", lwd = 2,
xlab="Percentage of test instances",
ylab="Percentage of correct predictions", main="Cumulative Gain Chart")
lines(GainTable_SVM[,4], col="blue", type ="l",lwd = 2)
lines(GainTable_LDA[,4], col="green", type ="l",lwd = 2)
grid(NULL, lwd = 1)

legend("bottomright",
c("Random Forest","SVM", "LDA"),
fill=c("red","blue","green"))
```

```r
abline(a = 0, b = 1, col = "lightgray")
```

3.4 Lift Charts

```r
liftChart(predict_lda , test_set_small$target)
liftChart(svm_prediction_over_linear_70_pos , test_set_small$target)
liftChart(prediction_RF_under, test_set$target)
```

4 Other model types used in report

4.1 GLM

4.1.1 Oversampling

```
data.over <- ovun.sample(target~., data=training_set_small, method="over", seed=10)$data
```

4.1.2 GLM Model 1

```
log_reg <- glm(target ~.,data.over, family = "binomial")
log_reg_predict <- predict(log_reg, test_set_small, type = "response")
log_reg_class <- ifelse(log_reg_predict >0.6, "1", "0")
log_reg_class <- as.factor(log_reg_class)
cf_glm_over_70 <- confusionMatrix(log_reg_class, test_set_small$target, positive = "1",
                                                                    mode = "prec_recall")
```

4.1.3 Undersampling

```
data.under <- ovun.sample(target~., data=training_set_small, method="under", seed=10)$data
```

4.1.4 GLM Model 2

```
log_reg2U <- glm(target ~.,data.under, family = "binomial")
log_reg_predict2U <- predict(log_reg2U, test_set_small, type = "response")
log_reg_class2U <- ifelse(log_reg_predict2U >0.65, "1", "0")
log_reg_class2U <- as.factor(log_reg_class2U)
cf_glm_under_70 <- confusionMatrix(log_reg_class2U, test_set_small$target, positive = "1",
                                                                      mode = "prec_recall")
```

4.2 Decision trees

4.2.1 Oversampling

```
oversample_data_70 <- ROSE:: ovun.sample(target~. , data = training_set_small,
                                    method = "over", p = 0.5, seed = 1)$data
```

4.2.2 Selecting top 100 attributes based on their Information Gain

```
attrweights_over_70 <- FSelector:: information.gain(target~., oversample_data_70)
attrweights_over_70 <- attrweights_over_70 %>% arrange(desc(attr_importance))
selectedattr_over_70 <- FSelector:: cutoff.k(attrweights_over_70, 100)


oversample_70 <- oversample_data_70[selectedattr_over_70]
oversample_70$target <- oversample_data_70$target
```

4.2.3 Tree() Model

```
regtree_70_over <- tree(target~., data=oversample_70)
summary(regtree_70_over)
print(regtree_70_over)
```

```
plot(regtree_70_over)
text(regtree_70_over)

regtree_70_over_pre <- predict(regtree_70_over, test_set_small, type="class")
correct_regtree_70_over <- which(test_set_small$target == regtree_70_over_pre)
length(correct_regtree_70_over)

cf_tree_over_70_100IG <- confusionMatrix(regtree_70_over_pre,
test_set_small$target, positive='1', mode = "prec_recall")
```

4.2.4 Ctree() Model

```
DeTree_70_over <- ctree(target~., data=oversample_70)

DeTree_70_over_pre <- predict(DeTree_70_over, test_set_small)
correct_DeTree_70_over <- which(test_set_small$target == DeTree_70_over_pre)
length(correct_DeTree_70_over)
(accuracy_DeTree_70_over <- length(correct_DeTree_70_over)/nrow(test_set_small)*100)

cf_ctree_70_100IG <- confusionMatrix(DeTree_70_over_pre,
test_set_small$target, positive='1', mode = "prec_recall")
```

4.3 Other SVMs used

4.3.1 Splitting using 60/40 ratio

```
split_60 <- caTools:: sample.split(datasmall$target, SplitRatio =0.6)

training_set_60 <- subset(datasmall, split_60 ==TRUE)
test_set_60 <-subset(datasmall, split_60 ==FALSE)
```

4.3.2 Both sampling

```
set.seed(123)

bothsample_data_60 <- ROSE:: ovun.sample(target~. , data = training_set_60,
                                  method = "both", p = 0.5, seed = 1)$data
```

4.3.3 Selecting all attributes with positive Information Gain

```
attribute_weights_both_60_pos <- FSelector:: information.gain(target~., bothsample_data_60)
selected_attribute_both_60_pos <- attribute_weights_both_60_pos %>%
filter(attr_importance > 0) %>% mutate(indexNames = row.names.data.frame(.)) %>%
                                                        pull(.,indexNames)


bothsample_data_all_60_pos <- bothsample_data_60
bothsample_data_60_pos <- bothsample_data_all_60_pos[selected_attribute_both_60_pos]
bothsample_data_60_pos$target <- bothsample_data_all_60_pos$target
```

4.3.4 SVM Model 1

```
svm_model_both_radial_60_pos <- e1071:: svm(target~., data = bothsample_data_60_pos,
svm_prediction_both_radial_60_pos <- predict(svm_model_both_radial_60_pos, test_set_60)

(cf_svm_both_radial_60_pos <- caret:: confusionMatrix(svm_prediction_both_radial_60_pos,
                              test_set_60$target, positive = "1", mode = "prec_recall"))
```

4.3.5 Additional SVM Model with tuning

```
svm_for_tuning_no_sampling_60 <-e1071:: svm(target~., data = training_set_60,
                                                        kernel = "radial")

tune_out_60 = e1071:: tune(method = "svm",  target~. , data = training_set_60,
             kernel = "radial", tunecontrol = e1071::tune.control(cross = 10))

svm_prediction_tune_60 <- predict(tune_out_60$best.model, test_set_60)
cf_svm_prediction_tune_60 <- caret:: confusionMatrix(svm_prediction_tune_60,
                  test_set_60$target, positive = "1", mode = "prec_recall")
```

4.4 Naive Bayes

4.4.1 Oversampling

```
oversampledtraining60 <- ROSE::ovun.sample(target~.,
data = training_set_60, method = "over", p = 0.5, seed = 111)$data
```

4.4.2 Tuning

```
search_grid <- expand.grid(
  usekernel = c(TRUE, FALSE),
  fL = 0:5,
  adjust = seq(0, 5, by = 1)
)
```

4.4.3 Model

```
#install.packages("klaR")
library(klaR)

features <- setdiff(names(training_set_60), "target")
x <- training_set_60[, features]
y <- training_set_60$target

train_control <- trainControl(
method = "cv",
number = 10
)

nb.m2 <- train(
  x = x,
  y = y,
```

```
  method = "nb",
  trControl = train_control,
  tuneGrid = search_grid,
  preProc = c("center", "scale", "pca")
  )

nb.m2$results %>%
  top_n(5, wt = Accuracy) %>%
  arrange(desc(Accuracy))


pred <- predict(nb.m2, newdata = test_set_60)
(cf_nb_60 <- confusionMatrix(pred, test_set_60$target,
                  positive='1', mode = "prec_recall"))
```

4.5 Neural Network

4.5.1 SMOTE

```
#install.packages("DMwR")
library(DMwR)
smotedt <- as.data.frame(training_set_small)
smotedt$target <- as.factor(smotedt$target)

smote_round1 <-
  DMwR::SMOTE(
    form = target ~ .,
    data = smotedt,
    perc.over = 100,
    perc.under = 200
  )
```

4.5.2 Model

```
#install.packages("neuralnet")
library(neuralnet)
nn=neuralnet(target~ ., data=smote_round1, hidden=100,act.fct = "logistic",
                linear.output = FALSE)

Predict=compute(nn,test_set_small)
prob <- Predict$net.result
pred <- ifelse(prob>0.5, "1", "0")
pred <- as.factor(pred[,2])
class(pred)

(cf_nn_70 <- confusionMatrix(pred, test_set_small$target,
                    positive='1', mode = "prec_recall"))
```

5 Plots used in the report

5.1 Exporting the outputs of the confusion matrices

```
tocsv <- data.frame()

#Named all variables with "cf_", so that we can easily
#find all matrices and export them to csv
for (file in ls()){
  print(paste("testing file", file))
  if (stringr::str_detect(file, "cf_")){
    cm <- get(file)
    print(cm)
    tocsv2 <- data.frame(cbind(file, t(cm$overall),t(cm$byClass)))
    tocsv <- rbind(tocsv, tocsv2)
  }}


write.csv(tocsv,file="final_models.csv")
```

5.2 Model comparsion plot

```
library(ggplot2)
library(stringr)
library(gridExtra)
library(readxl)
library(tidyverse)

plotting_data <- read_excel("report_plot_models.xlsx")
plotting_data$Sample <- NULL
plotting_data$type <- as.factor(plotting_data$type)
plotting_data$Partition <- as.factor(plotting_data$Partition)
plotting_data$TP <- NULL

plotting_data_longer <- plotting_data %>% pivot_longer(5:8)

ggplot(plotting_data_longer) + geom_jitter(aes(x= type, y= value, color = Partition),
width = 0.1) + theme(axis.text.x = element_text(angle = 90, hjust = 1, vjust = 0.5)) +
theme(legend.position = "bottom") + ylim(0, 1) + labs(x = NULL, y="Values") +
                                                    facet_grid(~name)
```

5.3 Expected profit model comparison chart

```
cost_confusion_path <- "Cost Confusion Matrix.xlsx"

cost_confusion_data <- read_excel(cost_confusion_path,sheet = "Plotting_Data")

cost_confusion_longer <- cost_confusion_data %>% pivot_longer(cols = c(Low_Cost,
High_Cost), names_to = "Cost_Type") %>% mutate(Cost_Type = factor(Cost_Type),
`Model_No_[]` = factor(`Model_No_[]`), Model_No = factor(Model_No))

levels(cost_confusion_longer$Cost_Type) <- c("High Cost Scenario", "Low Cost Scenario")
```

```
ggplot(cost_confusion_longer, aes(x= Model_No,y = value, fill = Cost_Type,
color = factor(ifelse(Name=="RF 70% Stratified + Under 100 IG", "Yes", "No")))) +
geom_col()+ facet_grid(~Cost_Type) + scale_fill_manual(values = c("#55e6b0", "#5cd5e0")) +
                            scale_color_manual( values=c("white", "black"), guide=FALSE) +
labs(y = "Expected Profit in £", fill = "Scenario",
x = "Model Number (see below for further information)") +
                       theme(legend.title=element_blank())
```

5.4 Testing final model for overfitting plot

```
plotting_data_final <- read.csv("final_models.csv")

plotting_data_final <- plotting_data_final %>% mutate(categorie =
ifelse(str_detect(file, "train"), "Training", "Test"), type = "RF")

plotting_data_final_longer <- plotting_data_final %>% select("file",
"Accuracy", "Precision", "Recall","F1" , "categorie","type")

plotting_data_final_longer <- plotting_data_final_longer %>%
pivot_longer(cols = c("Accuracy", "Precision", "Recall", "F1"))

ggplot(plotting_data_final_longer) + geom_point(aes(x= name, y= value,
color =categorie)) + ylim(0, 1) +
labs(y = "Value in %", x = "Measure", col = "Set",
 title = "Testing our final model on overfitting")
```