Bella Lin, Emilie Baek
Emily Bates, Makenzie Kalebick

Case Study: Customer Acquisition & Retention

Executive Summary

In our case study, we dove into the critical aspects of customer acquisition and retention, important components of effective customer relationship management. By exploring classification and regression tasks, we aimed to predict both the likelihood of customer acquisition and the duration of customer relationships. Our approach involves constructing predictive models using logistic regression, decision trees, and random forest algorithms. Additionally, we use variable importance analysis and hyperparameter optimization to refine our models and enhance their predictive accuracy.

Results from our model exploration highlight the performance of logistic regression, decision trees, and random forest models in predicting customer acquisition. Variable importance analysis reveals key predictors, while grid search optimization significantly enhances model accuracy, particularly in the case of predicting retention.

Our study concludes with a reflection on the challenges encountered, including limitations in model improvement beyond certain thresholds. Despite the effectiveness of grid search optimization, there remains a need for ongoing refinement and validation of predictive models. Recommendations for future research include exploring alternative methodologies, parameter settings, and data preprocessing techniques to enhance predictive capability further and address the complexities of customer acquisition and retention dynamics. Ultimately, our study highlights the evolving role of machine learning methodologies in optimizing customer relationship management strategies.

Problem

The tasks we were given for this case study focus on customer acquisition and retention, a topic that is crucial for effective customer relationship management. We will approach the problem by first addressing the classification task (*acquisition*) and then the regression task (*duration*). We will build models to predict which customers will be acquired and for how long. We will identify which models are most successful, comparing logistic regression, decision tree, and random forest. Additionally, variable importance analysis will be conducted to identify significant interactions and optimize hyperparameters for customer acquisition. Our goal is to produce models that accurately predict customer retention and acquisition to target the right customers, therefore decreasing the cost of marketing and increasing efficiency. This will ultimately increase profitability. The subsequent sections of this report will consist of a literature review, an exploration of the methodology used, specifics regarding data preparation and manipulation, and a comprehensive analysis of our models and findings.

Review of Related Literature

Customer acquisition and retention are common key performance indicators in marketing. Being able to accurately predict which prospects will become customers is essential for a company to minimize acquisition expenditures. Likewise, a company can improve its customer retention by predicting the customer's lifetime with the company. By doing so, the company can identify both high-probability prospects and at-risk customers to focus their marketing dollars more efficiently.

Using machine learning algorithms to predict customer acquisition and retention has shown to be an effective approach across many industries, B2B as well as B2C. Consequently, numerous studies are attempting to predict acquisition, retention, and related measures such as customer churn and customer lifetime value. A 2020 study by R. Singh[1] sought to predict the *acquisition* of customers to a term deposit given predictors such as demographic data, banking information, and campaign exposure. Singh employed and compared various classifiers, and found that XGBoost exhibited the best performance, while logistic regression, support vector machines, and random forest were close runner-ups. Another study by S. F. Sabbeh (2018)[2] focused on predicting customer churn (i.e. the opposite term of *retention)* based on telecommunication data. This study concluded that among the many methods assessed, random forest and ADA boost resulted in the best accuracy of predicting customer churn.

Methodology

As described earlier, this is a two-fold problem consisting of a classification task (*acquisition*) and a regression task (*duration*). Other than these two response variables, the data consists of an ID column (*customer*), 11 numeric predictors, and a binary predictor variable

[1] https://cocolevio.com/efficiently-identify-prospects-using-machine-learning/. Accessed 4/17/2024.
[2] https://thesai.org/Publications/ViewPaper?Volume=9&Issue=2&Code=ijacsa&SerialNo=38. Accessed 4/17/2024.

(*industry*). The numeric predictors include customer information such as CLV (*profit*), acquisition and retention expenditures *(acq_exp*, *ret_exp*), number of purchases (*freq*), number of product categories the customer purchased from (*crossbuy*), a share-of-wallet measure (*sow*), and squared terms (*acq_exp_sq*, *ret_exp_sq*, *freq_sq*). Seeing that many of these variables are 0 if *acquisition* = 0, we will check for perfect separation to ensure a realistic prediction of whether a customer will be acquired.

For the classification task, we will employ three supervised machine learning techniques: Logistic regression, decision tree, and random forest. These models will be evaluated on accuracy, while also considering the balance of specificity and sensitivity. Our sampling technique consists of a random train/test split of all 500 observations using a ratio of 80/20, leaving 400 observations for training and 100 for testing.

Logistic regression is a parametric technique that is highly interpretable as it outputs beta coefficients that can explain each predictor's effect on the response variable. However, for this analysis, we utilize logistic regression for prediction rather than inference. The logit model returns a probability of an event occurring (*acquisition* = 1), making the output between 0 and 1. This is an advantage because it allows for insights into the likelihood of a prospect becoming a customer. Having that said, logistic regression is limited by its parametric assumptions such as linearity in the logit function for continuous variables, no highly influential outliers, and no multicollinearity among the variables. We will assess the latter by using the `VIF()` function in R and remove any highly correlated predictors in our logit model.

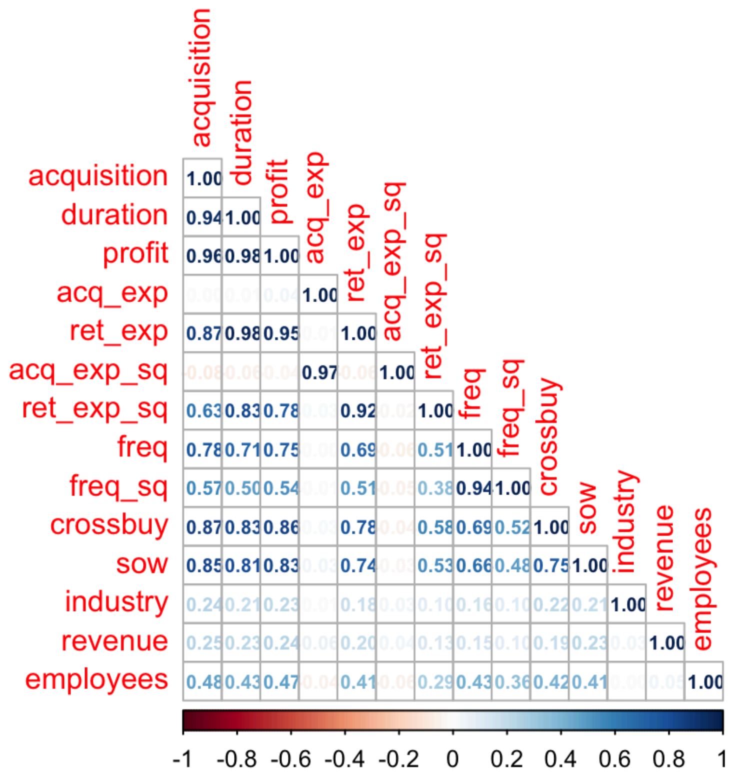
A decision tree is an algorithm that can be used for both classification and regression tasks. A classification tree ultimately predicts which class an observation is most likely to belong to based on recursive binary splitting. Three criteria can be used as decision-making for these binary splits: Classification error rate, Gini coefficient, and cross-entropy. In our analysis, we will use the Gini coefficient criterion, which is the default for the `tree()` function. Like logistic regression, decision trees offer high interpretability, allowing insights on the importance of individual variables. Additionally, it is a non-parametric technique that is not limited by the distribution of the data. Some of the disadvantages of decision trees, is its lack of robustness against small changes in the data and the likelihood of overfitting on the training data. While methods like pruning can generally decrease the level of overfitting, more efficient approaches include ensemble methods such as bagging, boosting, and random forest.

A random forest builds multiple decision trees during training and outputs the mode of the classes (for classification) or the average prediction (for regression) of the individual trees. It introduces randomness by bootstrapping the data and randomly selecting a subset of features for each tree. Some of the hyperparameters to consider when tuning a random forest include the number of trees in the forest (*ntree*), the maximum depth of the trees (*nodedepth*), the minimum number of samples required to split a node (*nodesize*), and the maximum number of features to consider when looking for the best split (*mtry*). We will seek to optimize the performance by tuning these parameters using grid search to find the settings that produce the minimal out-of-bag error. Random forests are generally very effective in handling outliers, missing data, and datasets with imbalanced classes. A potential limitation of a random forest is overfitting to noisy data.

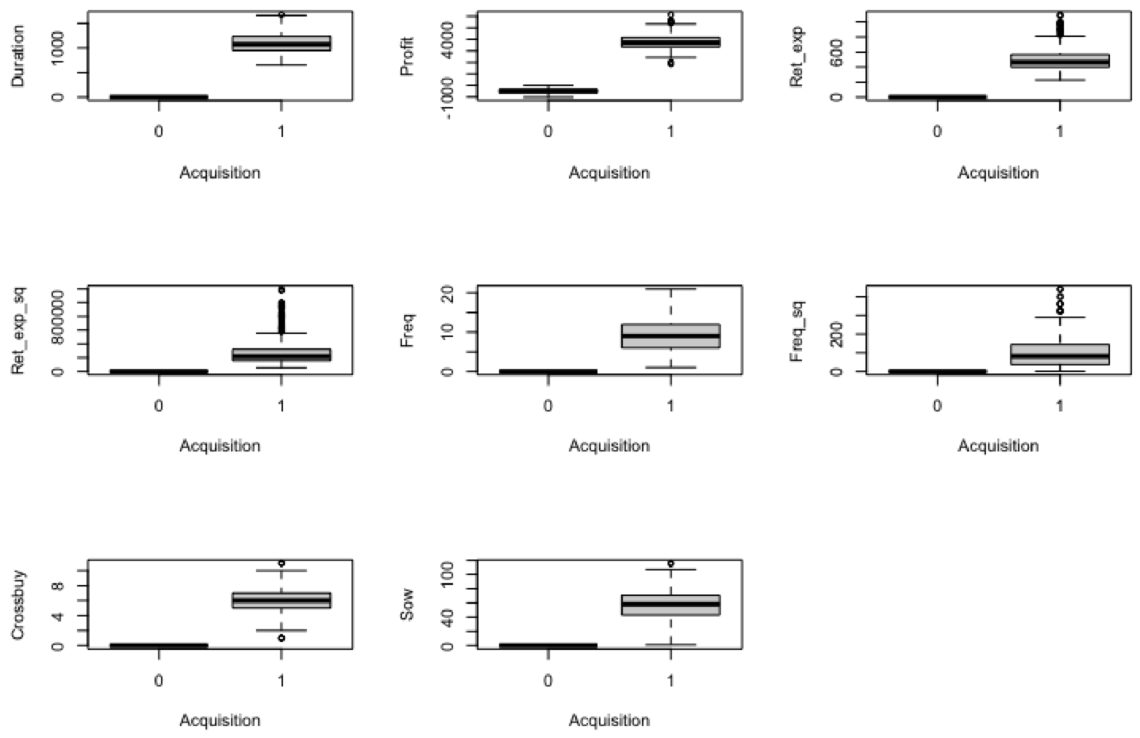
For the regression task, we will simply focus our efforts on fitting and optimizing a random forest. Like the previous task, we randomly split the data using an 80/20 ratio. However, the total number of observations for this task is reduced to 247 in total. We collected the customers who were predicted as acquired by our random forest (*rf1.preds* = 1) *and* who were indeed acquired (*acquisition* = 1). This selection is crucial because customers who were predicted to be acquired but were not acquired have a duration of 0, which would distort the modeling for the duration. Once again, we optimize the random forest using grid search to tune hyperparameters. Different from the classification task, the untuned and tuned models for *duration* are evaluated and compared using the mean squared error (MSE) metric.

Data

The dataset provided for this case study contains 15 variables describing customers with 500 observations. First, we checked for NA values and found that there were none in the dataset. We removed the *customer* column since it functions as an ID rather than a meaningful data point. Next, we prepared our dataset for the classification task. First, we examined the relationships between the numeric variables.



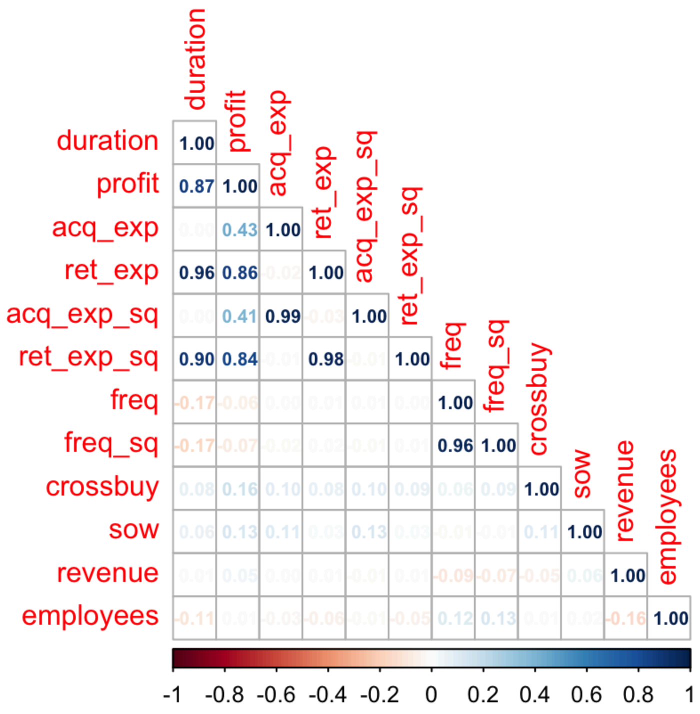
Quite a few variables are highly correlated with our target variable, *acquisition*, for the classification task. Highly correlated variables include *duration*, *profit*, *ret_exp*, *freq*, *crossbuy*, and *sow*. To examine these variables further, we created box plots.

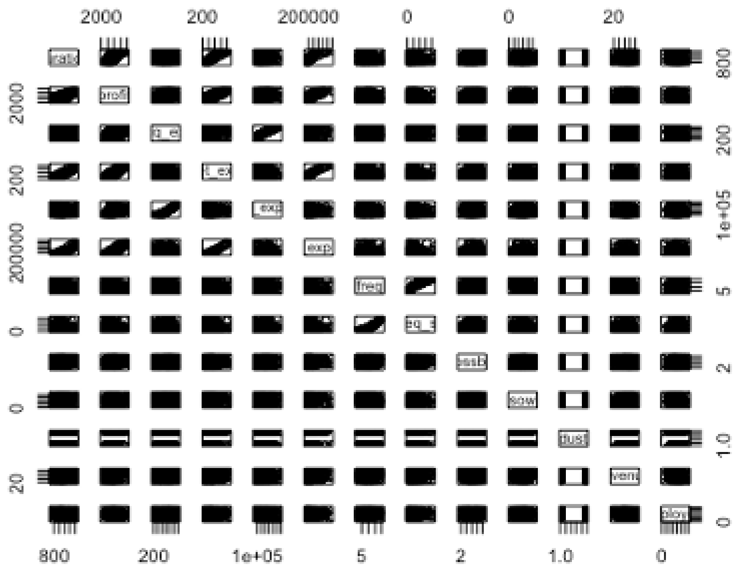


The box plots display that the above variables with high correlation are also perfectly separated. This means that when *acquisition* is zero, the variables are also zero. The exception is *profit,* which displays perfect separation by having negative values whenever *acquisition* is zero, and positive values whenever *acquisition* is one. Due to perfect separation, *duration*, *profit*, *ret_exp*, *ret_exp_sq*, *freq*, *freq_sq*, *crossbuy*, and *sow* must be removed. We also converted two binary variables to factors: *acquisition* and *industry*. Now we are ready to split our data into train and test sets. We perform an 80/20 split and check the balance of our data.

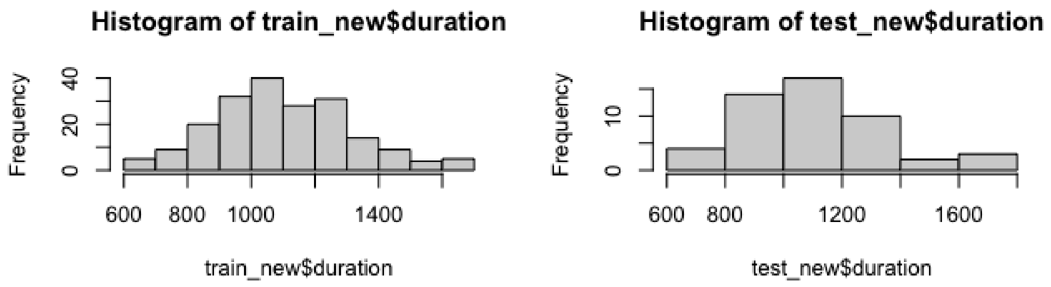
```
# Checking balance of data
summary(train$acquisition)
##   0   1
## 126 274
summary(test$acquisition)
##  0  1
## 36 64
```

We proceed with our classification analysis after finding the data to be decently balanced. After getting predictions for *acquisition*, we prepare the data for the regression task. By creating a temporary dataset that combines the original dataset and our *acquisition* predictions, we are able to create a filtered dataset that only contains those that are predicted as successfully acquired. We then examine the correlation plot for our new filtered dataset.





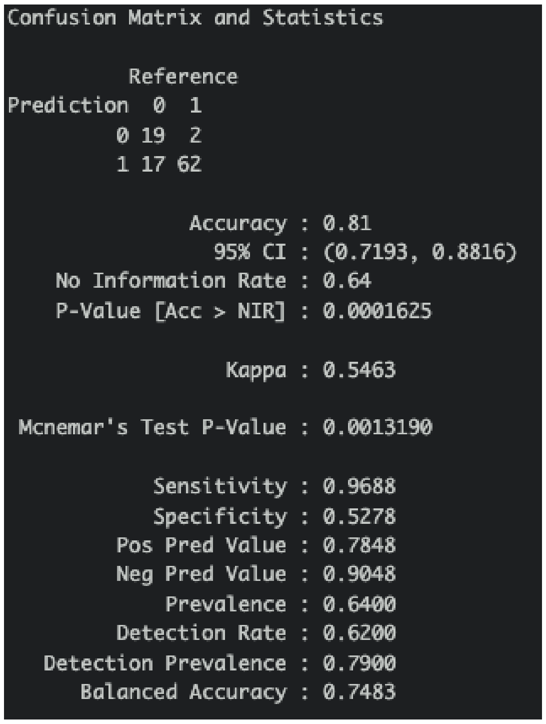
We see that *profit* and *ret_exp* are highly correlated, and remove them from the dataset. Finally, we perform an 80/20 split to separate our data in train and test sets. Though the random forest is generally robust to skewed data, we check the distribution of our train and test sets to ensure there isn't extreme skew.



We find the distribution adequate and continue on to analysis.

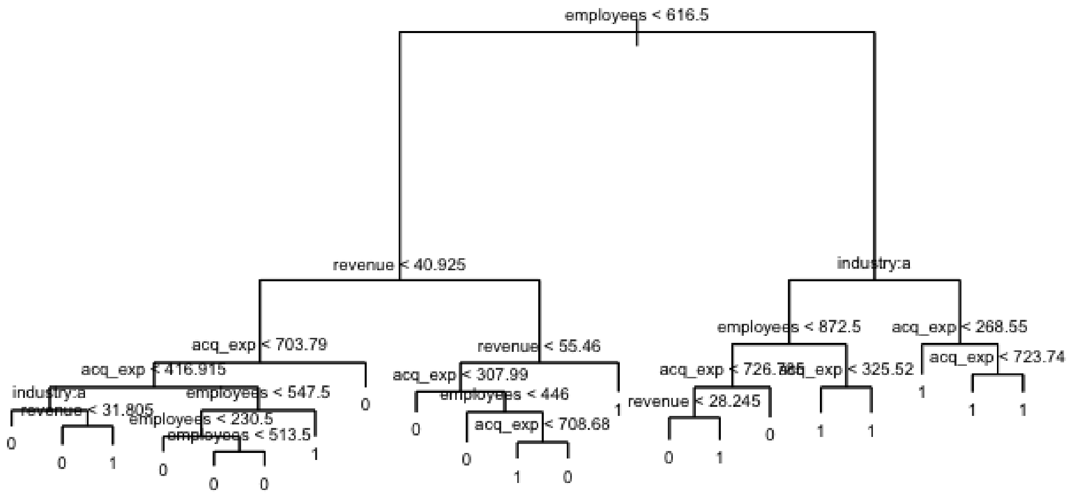
Results

First, we addressed the classification task using *acquisition* as our response variable. The first model we explored was the logistic regression model. Before creating the model, we checked for multicollinearity for each variable and removed the ones that had a value greater than 10. Our final predictors are *acq_exp_sq, industry, revenue, and employees*. Using the confusion matrix, we were able to evaluate the logistic regression model seen below. We can see that the model achieved an accuracy of 0.81, indicating that it correctly predicted the acquisition outcome for 81% of the cases in the test dataset. Additionally, it demonstrated high sensitivity (0.9688), suggesting its ability to correctly identify true positive cases. However, the specificity (0.5278) indicates a relatively lower ability to correctly identify true negative cases.

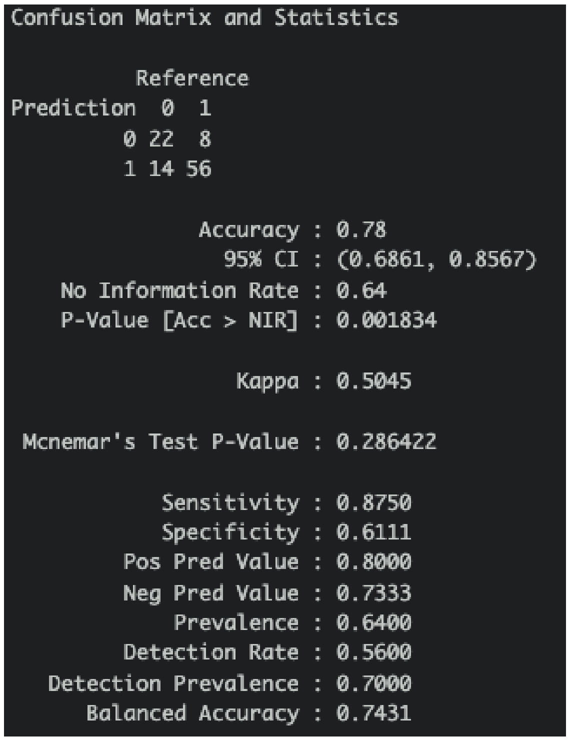


```
Confusion Matrix and Statistics

          Reference
Prediction  0  1
         0 19  2
         1 17 62

               Accuracy : 0.81
                 95% CI : (0.7193, 0.8816)
    No Information Rate : 0.64
    P-Value [Acc > NIR] : 0.0001625

                  Kappa : 0.5463

 Mcnemar's Test P-Value : 0.0013190

            Sensitivity : 0.9688
            Specificity : 0.5278
         Pos Pred Value : 0.7848
         Neg Pred Value : 0.9048
             Prevalence : 0.6400
         Detection Rate : 0.6200
   Detection Prevalence : 0.7900
      Balanced Accuracy : 0.7483
```

Next we created a decision tree model. The decision tree model was constructed using the predictors: *employees, revenue, acq_exp, and industry*, which match the predictors in the logistic regression model.



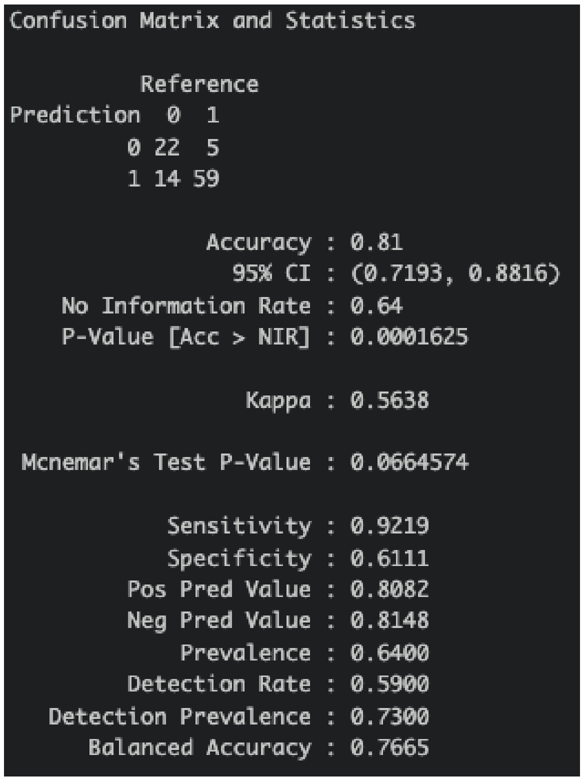
(Figure 1. Decision tree model map)



```
Confusion Matrix and Statistics

          Reference
Prediction  0  1
         0 22  8
         1 14 56

               Accuracy : 0.78
                 95% CI : (0.6861, 0.8567)
    No Information Rate : 0.64
    P-Value [Acc > NIR] : 0.001834

                  Kappa : 0.5045

 Mcnemar's Test P-Value : 0.286422

            Sensitivity : 0.8750
            Specificity : 0.6111
         Pos Pred Value : 0.8000
         Neg Pred Value : 0.7333
             Prevalence : 0.6400
         Detection Rate : 0.5600
   Detection Prevalence : 0.7000
      Balanced Accuracy : 0.7431
```

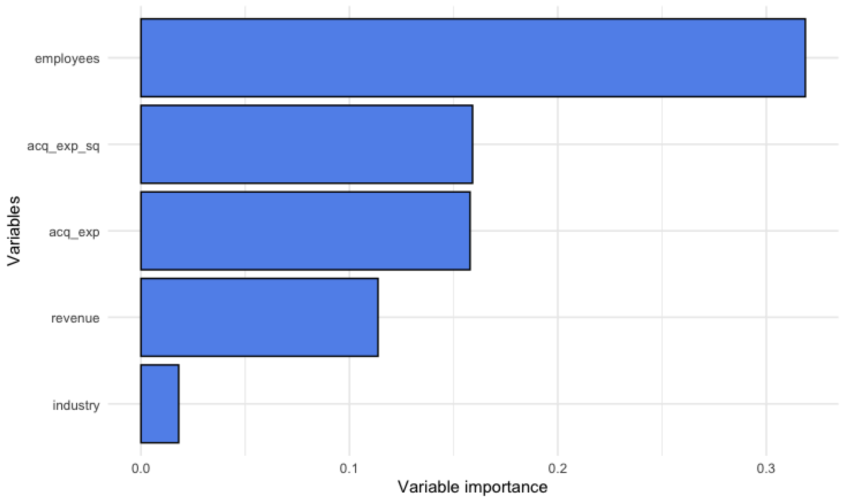
The tree model has 21 terminal nodes, with a misclassification error rate of 0.13, meaning that approximately 13% of cases were misclassified. According to the confusion matrix result above, the decision tree model achieved an accuracy of 78%, with a relatively high sensitivity of 87.5% and a moderate specificity of 61.1%.

Lastly, we explored two random forest models, one with manual tuning and one model with grid search tuning. The parameters for the first model were found by manually evaluating different combinations of settings. The values that resulted in the best performance on the test set were 1000 trees, 9 nodes, and a node depth of 5. As indicated in the output on the following page, this random forest model achieved an accuracy of 81% with a high sensitivity of 92.2% and a moderate specificity of 61.1%.



```
Confusion Matrix and Statistics

          Reference
Prediction  0  1
         0 22  5
         1 14 59

               Accuracy : 0.81
                 95% CI : (0.7193, 0.8816)
    No Information Rate : 0.64
    P-Value [Acc > NIR] : 0.0001625

                  Kappa : 0.5638

 Mcnemar's Test P-Value : 0.0664574

            Sensitivity : 0.9219
            Specificity : 0.6111
         Pos Pred Value : 0.8082
         Neg Pred Value : 0.8148
             Prevalence : 0.6400
         Detection Rate : 0.5900
   Detection Prevalence : 0.7300
      Balanced Accuracy : 0.7665
```

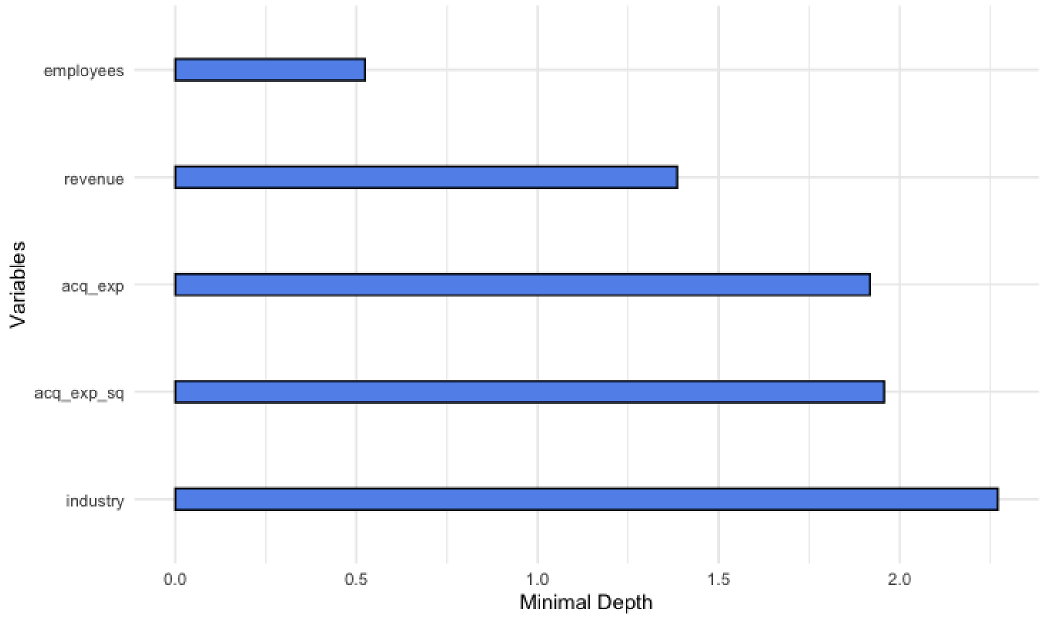
Before tuning the random forest model using grid search, we explored variable importance, minimal depth, and interactions between variables. The variable importance plot visualizes the importance of each predictor variable in the random forest model. It helps identify which variables have the most significant impact on the outcome.



(Figure 2. Variable Importance Plot of the Random Forest Model)

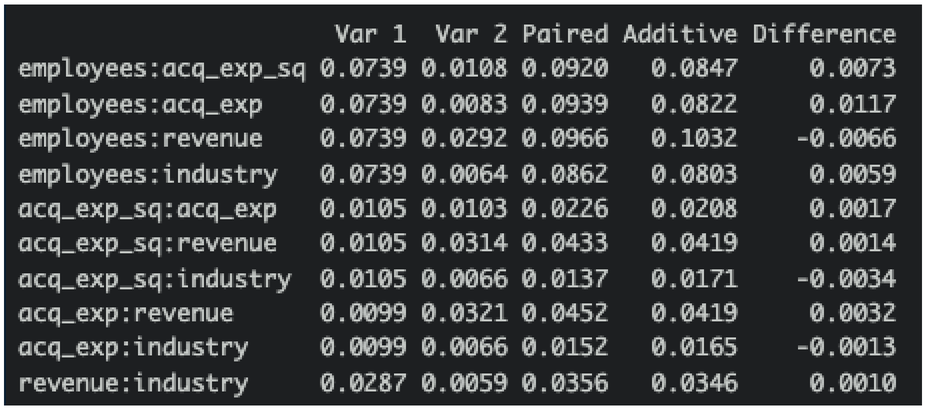
In the variable importance plot above, we can see that the variable *employees* is the most significant variable in predicting acquisition.

Minimal depth indicates how close variables are to the root of the trees in the Random Forest model. Lower minimal depth values suggest greater importance or relevance of the variables in predicting the outcome.



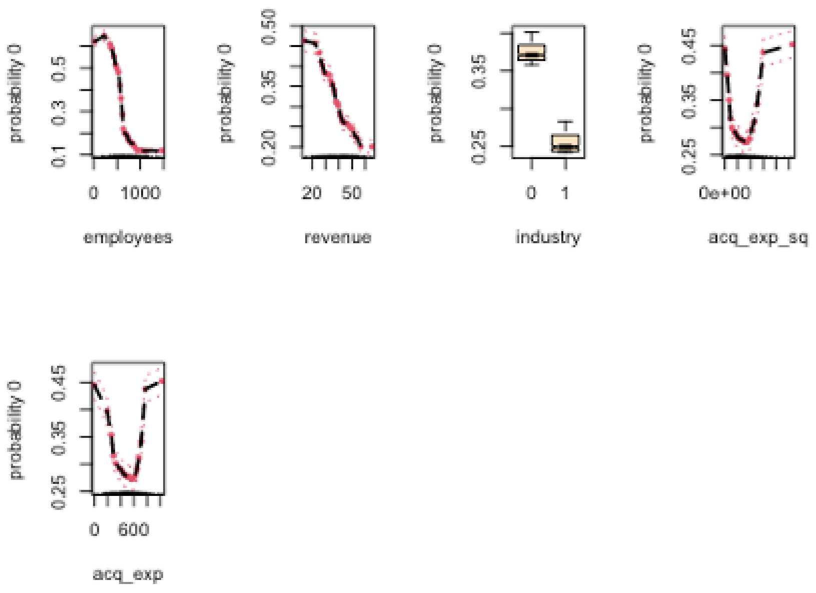
(Figure 3. Minimal Depth of Variables in the Random Forest Model)

As shown in Figure 3, the variable *employees* has the lowest minimal depth, which matches the result from Figure 2.



```
                     Var 1  Var 2 Paired Additive Difference
employees:acq_exp_sq 0.0739 0.0108 0.0920   0.0847     0.0073
employees:acq_exp    0.0739 0.0083 0.0939   0.0822     0.0117
employees:revenue    0.0739 0.0292 0.0966   0.1032    -0.0066
employees:industry   0.0739 0.0064 0.0862   0.0803     0.0059
acq_exp_sq:acq_exp   0.0105 0.0103 0.0226   0.0208     0.0017
acq_exp_sq:revenue   0.0105 0.0314 0.0433   0.0419     0.0014
acq_exp_sq:industry  0.0105 0.0066 0.0137   0.0171    -0.0034
acq_exp:revenue      0.0099 0.0321 0.0452   0.0419     0.0032
acq_exp:industry     0.0099 0.0066 0.0152   0.0165    -0.0013
revenue:industry     0.0287 0.0059 0.0356   0.0346     0.0010
```

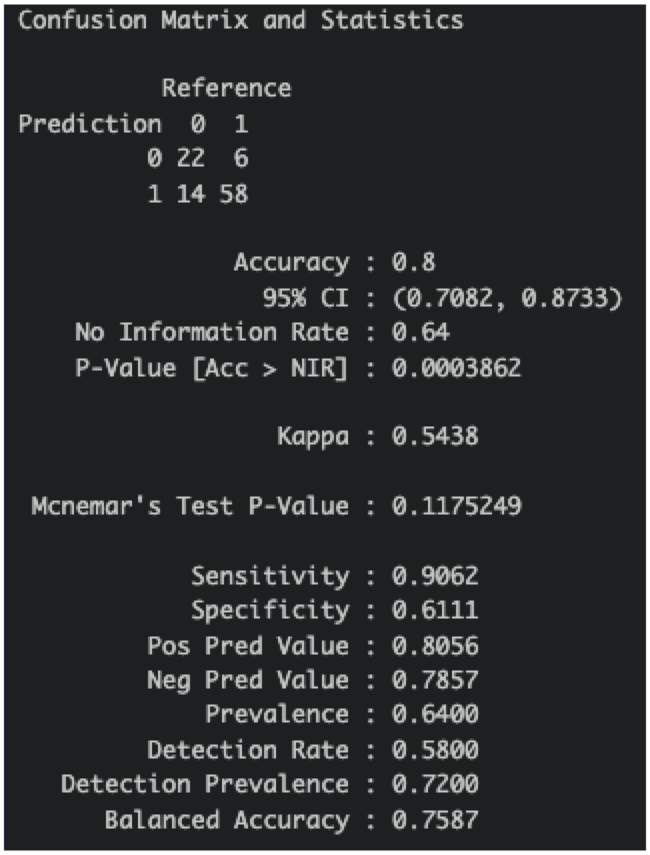
Interactions between variables can also provide insights into how predictors influence each other in predicting the outcome. In the analysis result above, the most significant interaction terms were found to be between *employees* and *industry, employees* and *revenue, employees* and *acq_exp_sq,* and *employees* and *acq_exp.* Seeing that variable *employees* has the highest importance, lowest minimal depth, and is the common variable in all interaction pairs, it is safe to say that *employees* is the most relevant variable in predicting the outcome, *acquisition*.



(Figure 4. PDP plot of predictors for acquisition)

We also generated partial dependence plots for all predictors of *acquisition*. As Figure 4 shows above, both *employees* and *revenue* have a steep and negative slope, indicating a stronger negative influence in predicting *acquisition*. It also revealed a non-linear relationship between *acq_exp_sq* and *acq_exp* with *acquisition*.

Next, we employed a grid search approach to tune hyper-parameters for the random forest model. We explored various combinations of *mtry*, *nodesize*, and *ntree* values to optimize model performance. The grid search revealed that setting *mtry* = 3, *nodesize* = 8, and *ntree* = 7000 yielded the lowest out-of-bag error, indicating the optimal set of hyper-parameters.



```
Confusion Matrix and Statistics

          Reference
Prediction  0  1
         0 22  6
         1 14 58

               Accuracy : 0.8
                 95% CI : (0.7082, 0.8733)
    No Information Rate : 0.64
    P-Value [Acc > NIR] : 0.0003862

                  Kappa : 0.5438

 Mcnemar's Test P-Value : 0.1175249

            Sensitivity : 0.9062
            Specificity : 0.6111
         Pos Pred Value : 0.8056
         Neg Pred Value : 0.7857
             Prevalence : 0.6400
         Detection Rate : 0.5800
   Detection Prevalence : 0.7200
      Balanced Accuracy : 0.7587
```

The model tuned using grid search achieved an accuracy of 80% on the test dataset and demonstrated a sensitivity of 90.62% and a specificity of 61.11%. In other words, the tuned model did not outperform the initial random forest that we tuned by manually evaluating different combinations of settings. This scenario reveals that grid search does not always guarantee optimal performance on the test set. Nonetheless, using grid search to tune parameters should generally improve results, and it could be worth investigating this further, for instance by testing with different seeds and other ratios for the train/test split.

Finally, we prepared the data for a regression task aimed at predicting the number of days the customer was a customer of the firm (*duration*). As mentioned earlier in the data section, we isolated instances where our random forest model *correctly* predicted successful acquisitions and filtered out variables highly correlated with each other. We then utilized the Random Forest algorithm to predict *duration* using the modified dataset. Initially, we trained a Random Forest model (`rf_model2`) with default hyper-parameter values, resulting in a Mean Squared Error (MSE) of 5390.555 on the test set.

To optimize model performance, we conducted a grid search over a range of hyper-parameter values including *mtry*, *nodesize*, and *ntree*. The grid search identified the optimal set of hyper-parameters as `mtry = 9`, `nodesize = 4`, and `ntree = 1000`. We applied these tuned parameters to train a new Random Forest model (`rf2.tuned`).

The tuned Random Forest model achieved a significant performance improvement, as evidenced by a substantially reduced MSE of 2804.261 on the test set. This demonstrates the effectiveness of grid search in enhancing model accuracy, leading to a considerable reduction in prediction errors compared to the initial model.

Conclusion & Recommendations

In conclusion, this case study explored the realm of customer acquisition and retention, using machine learning techniques to forecast customer behavior. By addressing both classification and regression tasks, we aimed to predict customer acquisition and the duration of customer relationships. Through the exploration of logistic regression, decision trees, and random forest models, coupled with variable importance analysis and hyperparameter optimization, we developed strong predictive models to enhance customer relationship management strategies.

While our strategies gave valuable insights into the factors influencing customer acquisition and retention, our analysis also encountered challenges. Despite extensive testing and optimization efforts, we only observed significant improvements in model accuracy beyond certain thresholds. The grid search approach proved effective in optimizing the Random Forest model for *duration* but not for *acquisition*. It's important to acknowledge that the performance of these models may not always generalize well to unseen data, emphasizing the need for ongoing refinement and validation.

Moving forward, further exploration into alternative methodologies, parameter settings, and data preprocessing techniques could enhance the predictive capabilities of our models. Additionally, conducting sensitivity analyses and exploring different data partitioning strategies may provide deeper insights into model robustness and generalizability. Overall, this study highlights the complexity of customer acquisition and retention dynamics and emphasizes the ongoing evolution of machine learning methodologies in addressing these challenges.

Appendix

```
###### Modeling for Acquisition (Classification Task)

### Logistic regression
glm.all = glm(acquisition ~ ., data = train, family = binomial)
summary(glm.all)

##
## Call:
## glm(formula = acquisition ~ ., family = binomial, data = train)
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) -1.436e+01  1.679e+00  -8.553  < 2e-16 ***
## acq_exp      3.032e-02  4.834e-03   6.273 3.53e-10 ***
## acq_exp_sq  -3.025e-05  4.854e-06  -6.232 4.62e-10 ***
## industry1    1.809e+00  3.278e-01   5.520 3.39e-08 ***
## revenue      8.382e-02  1.587e-02   5.283 1.27e-07 ***
## employees    6.971e-03  8.717e-04   7.997 1.27e-15 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 498.43  on 399  degrees of freedom
## Residual deviance: 279.37  on 394  degrees of freedom
## AIC: 291.37
##
## Number of Fisher Scoring iterations: 6

vif(glm.all)

##     acq_exp acq_exp_sq   industry    revenue  employees
##  27.961634  27.750115   1.139053   1.083612   1.234651

# Removing acq_exp due to multicollinearity
glm.fit = glm(acquisition ~ .-acq_exp, data = train, family = binomial)
summary(glm.fit)

##
## Call:
## glm(formula = acquisition ~ . - acq_exp, family = binomial, data = train)
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) -6.610e+00  8.673e-01  -7.622 2.51e-14 ***
## acq_exp_sq  -1.029e-06  7.711e-07  -1.334        0.182
## industry1    1.445e+00  2.853e-01   5.063 4.12e-07 ***
## revenue      7.919e-02  1.448e-02   5.470 4.51e-08 ***
## employees 6.265e-03  7.769e-04   8.064 7.37e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##       Null deviance: 498.43  on 399  degrees of freedom
## Residual deviance: 335.93  on 395  degrees of freedom
## AIC: 345.93
##
## Number of Fisher Scoring iterations: 5
```

```
vif(glm.fit)
```

```
## acq_exp_sq   industry       revenue  employees
##   1.009201   1.087037   1.066564   1.112978
```

```
# Computing probabilities and getting predictions
glm.probs <- predict.glm(glm.fit, test, type = "response")
glm.preds = ifelse(glm.probs >= 0.5, yes = 1, 0)

# Confusion matrix
caret::confusionMatrix(as.factor(glm.preds), test$acquisition, positive = "1")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 19  2
##          1 17 62
##
##                Accuracy : 0.81
##                  95% CI : (0.7193, 0.8816)
##     No Information Rate : 0.64
##     P-Value [Acc > NIR] : 0.0001625
##
##                   Kappa : 0.5463
##
##  Mcnemar's Test P-Value : 0.0013190
##
##             Sensitivity : 0.9688
##                Specificity : 0.5278
##        Pos Pred Value : 0.7848
##        Neg Pred Value : 0.9048
##                Prevalence : 0.6400
##        Detection Rate : 0.6200
##        Detection Prevalence : 0.7900
##        Balanced Accuracy : 0.7483
##
##        'Positive' Class : 1
##
```

```
# Accuracy      : 0.81
# Sensitivity : 0.9688
# Specificity : 0.5278


### Decision Tree
tree.fit <- tree(acquisition ~ ., data = train)
summary(tree.fit)
```

```
##
## Classification tree:
## tree(formula = acquisition ~ ., data = train)
## Variables actually used in tree construction:
## [1] "employees" "revenue"   "acq_exp"   "industry"
## Number of terminal nodes:  21
## Residual mean deviance:  0.5483 = 207.8 / 379
## Misclassification error rate: 0.13 = 52 / 400
```

```
# Plotting the tree
par(mfrow = c(1, 1))
plot(tree.fit)
text(tree.fit, cex = 0.6)

# Getting predictions
tree.preds = predict(tree.fit, test, type = "class")
caret::confusionMatrix(as.factor(tree.preds), test$acquisition, positive = "1")
```

```
## Confusion Matrix and Statistics
##
##        Reference
## Prediction  0  1
##        0 22  8
##        1 14 56
##
##                Accuracy : 0.78
##                95% CI : (0.6861, 0.8567)
##       No Information Rate : 0.64
##       P-Value [Acc > NIR] : 0.001834
##
##                Kappa : 0.5045
##
##  Mcnemar's Test P-Value : 0.286422
##
##          Sensitivity : 0.8750
##                Specificity : 0.6111
##       Pos Pred Value : 0.8000
##       Neg Pred Value : 0.7333
##                Prevalence : 0.6400
##       Detection Rate : 0.5600
##       Detection Prevalence : 0.7000
##       Balanced Accuracy : 0.7431
##
##       'Positive' Class : 1
##

# Accuracy      : 0.78
# Sensitivity : 0.8750
# Specificity : 0.6111



### Random Forest (with manual tuning of parameters)

set.seed(123) ; rf_model1 = rfsrc(acquisition ~ ., data = train,
                            importance = TRUE, ntree = 1000,
                            nodesize = 9, nodedepth = 5)

rf_model1

##                         Sample size: 400
##                Frequency of class labels: 126, 274
##                Number of trees: 1000
##                Forest terminal node size: 9
##       Average no. of terminal nodes: 14.922
## No. of variables tried at each split: 3
##                Total no. of variables: 5
##       Resampling used to grow trees: swor
##       Resample size used to grow trees: 253
##                         Analysis: RF-C
##                         Family: class
##                     Splitting rule: gini *random*
##        Number of random split points: 10
##                  Imbalanced ratio: 2.1746
##                  (OOB) Brier score: 0.13859597
##        (OOB) Normalized Brier score: 0.5543839
##                           (OOB) AUC: 0.86371799
##                           (OOB) PR-AUC: 0.725858
##                           (OOB) G-mean: 0.74551142
##        (OOB) Requested performance error: 0.19, 0.38095238, 0.10218978
##
## Confusion matrix:
##
##        predicted
##   observed  0   1 class.error
##        0 78  48         0.3810
##        1 28 246         0.1022
##
##        (OOB) Misclassification rate: 0.19
```

```
rf1.preds = predict(rf_model1, test)$class
caret::confusionMatrix(as.factor(rf1.preds), test$acquisition, positive = "1")
```

```
## Confusion Matrix and Statistics
##
##        Reference
## Prediction  0  1
##        0 22  5
##        1 14 59
##
##                  Accuracy : 0.81
##                  95% CI : (0.7193, 0.8816)
##        No Information Rate : 0.64
##        P-Value [Acc > NIR] : 0.0001625
##
##                  Kappa : 0.5638
##
##  Mcnemar's Test P-Value : 0.0664574
##
##                  Sensitivity : 0.9219
##                  Specificity : 0.6111
##        Pos Pred Value : 0.8082
##        Neg Pred Value : 0.8148
##                  Prevalence : 0.6400
##        Detection Rate : 0.5900
##        Detection Prevalence : 0.7300
##       Balanced Accuracy : 0.7665
##
##       'Positive' Class : 1
##

# Accuracy      : 0.81
# Sensitivity : 0.9219
# Specificity : 0.6111



###### Random Forest Plots: Variable Importance, Minimal Depth & Interactions

### Variable Importance
importance_df <- as.data.frame(rf_model1$importance) %>%
  tibble::rownames_to_column(var = "variable")

importance_df %>%
  ggplot(aes(x = reorder(variable, all), y = all)) +
  geom_bar(stat = "identity", fill = "cornflowerblue", color = "black") +
  coord_flip() +
  labs(x = "Variables", y = "Variable importance") +
  theme_minimal()

### Minimal Depth
# Helps identify the most important variables in the random forest model by
# indicating how close they are to the root of the trees. Lower minimal depth
# values suggest greater importance or relevance of the variables in predicting
# the outcome.

mindepth <- max.subtree(rf_model1, sub.order = TRUE)

# Print first order depths
print(round(mindepth$order, 3)[, 1])

##       acq_exp acq_exp_sq   industry        revenue  employees
##       1.918   1.957   2.271   1.386   0.524

# Plot minimal depth
md_df <- data.frame(md = round(mindepth$order, 3)[, 1]) %>%
  tibble::rownames_to_column(var = "variable")

ggplot(md_df, aes(x = reorder(variable, desc(md)), y = md)) +
  geom_bar(stat = "identity", fill = "cornflowerblue", color = "black", width = 0.2) +
  coord_flip() +
  labs(x = "Variables", y = "Minimal Depth") +
  theme_minimal()

### Interactions
# Cross-checking with variable importance
find.interaction(rf_model1, method = "vimp", importance = "permute")

## Pairing employees with acq_exp_sq
## Pairing employees with acq_exp
## Pairing employees with revenue
## Pairing employees with industry
## Pairing acq_exp_sq with acq_exp
## Pairing acq_exp_sq with revenue
## Pairing acq_exp_sq with industry
## Pairing acq_exp with revenue
## Pairing acq_exp with industry
## Pairing revenue with industry
##
##                              Method: vimp
##                    No. of variables: 5
##                    Variables sorted by VIMP?: TRUE
##        No. of variables used for pairing: 5
##        Total no. of paired interactions: 10
##                    Monte Carlo replications: 1
##        Type of noising up used for VIMP: permute
##
##                    Var 1  Var 2 Paired Additive Difference
## employees:acq_exp_sq 0.0739 0.0108 0.0920   0.0847  0.0073
## employees:acq_exp    0.0739 0.0083 0.0939   0.0822  0.0117
## employees:revenue    0.0739 0.0292 0.0966   0.1032 -0.0066
## employees:industry   0.0739 0.0064 0.0862   0.0803  0.0059
## acq_exp_sq:acq_exp   0.0105 0.0103 0.0226   0.0208  0.0017
## acq_exp_sq:revenue   0.0105 0.0314 0.0433   0.0419  0.0014
## acq_exp_sq:industry  0.0105 0.0066 0.0137   0.0171 -0.0034
## acq_exp:revenue      0.0099 0.0321 0.0452   0.0419  0.0032
## acq_exp:industry     0.0099 0.0066 0.0152   0.0165 -0.0013
## revenue:industry     0.0287 0.0059 0.0356   0.0346  0.0010

# We found that the most significant interaction terms are:
  # employees:industry
  # employees:revenue
  # employees:acq_exp_sq
  # employees:acq_exp
# We will test how including these affects the model's accuracy below.
```

```
### Tuned Random Forest for `Acquisition` (Using grid search & interaction terms)

# Establishing list of possible values for hyper-parameters
mtry.values <- seq(2,5,1)
nodesize.values <- seq(4,10,1)
#nodedepth.values <- seq(1,9,1)
ntree.values <- seq(1000,9000,2000)

# Creating data frame containing all combinations
hyper_grid = expand.grid(mtry = mtry.values,
                nodesize = nodesize.values,
                #nodedepth = nodedepth.values,
                ntree = ntree.values)

# Creating empty vector to store out-of-bag error values
oob_err = c()

# Looping over the rows of hyper_grid to train the grid of models
for (i in 1:nrow(hyper_grid)) {
  # Training Random Forest
  set.seed(123) ; model = rfsrc(acquisition ~ .+employees*industry,
                         data = train, importance = TRUE,
                         mtry = hyper_grid$mtry[i],
                         nodesize = hyper_grid$nodesize[i],
                         #nodedepth = hyper_grid$nodedepth[i],
                         ntree = hyper_grid$ntree[i])
  # Storing OOB error for the model
  oob_err[i] <- model$err.rate[length(model$err.rate)]
}

# Identifying optimal set of hyper-parameters based on OOB error
opt_i <- which.min(oob_err)
print(hyper_grid[opt_i,])

##     mtry nodesize ntree
## 102 3        8  7000

set.seed(123) ; rf1.tuned = rfsrc(acquisition ~ .+employees*industry,
                         data = train, importance = TRUE,
                         mtry = 3, nodesize = 8, ntree = 7000)
rf1.tuned

##                          Sample size: 400
##                 Frequency of class labels: 126, 274
##                      Number of trees: 7000
##            Forest terminal node size: 8
##        Average no. of terminal nodes: 23.5024
## No. of variables tried at each split: 3
##               Total no. of variables: 5
##        Resampling used to grow trees: swor
##     Resample size used to grow trees: 253
##                             Analysis: RF-C
##                               Family: class
##                       Splitting rule: gini *random*
##        Number of random split points: 10
##                     Imbalanced ratio: 2.1746
##                   (OOB) Brier score: 0.13839368
##        (OOB) Normalized Brier score: 0.55357473
##                           (OOB) AUC: 0.862878
##                        (OOB) PR-AUC: 0.72887537
##                        (OOB) G-mean: 0.73589152
##    (OOB) Requested performance error: 0.195, 0.3968254, 0.10218978
##
## Confusion matrix:
##
##           predicted
##   observed  0   1 class.error
##          0 76  50      0.3968
##          1 28 246      0.1022
##
##       (OOB) Misclassification rate: 0.195

# Note: The settings in the model above yielded the best accuracy (0.80) among
# all the combinations of interaction terms and hyper-parameters we tested.

# Getting predictions
rf1.tuned.preds = predict(rf1.tuned, test)$class
caret::confusionMatrix(as.factor(rf1.tuned.preds), test$acquisition, positive = "1")

## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 22  6
##          1 14 58
##
##                Accuracy : 0.8
##                  95% CI : (0.7082, 0.8733)
##     No Information Rate : 0.64
##       P-Value [Acc > NIR] : 0.0003862
##
##                   Kappa : 0.5438
##
##  Mcnemar's Test P-Value : 0.1175249
##
##             Sensitivity : 0.9062
##             Specificity : 0.6111
##          Pos Pred Value : 0.8056
##          Neg Pred Value : 0.7857
##              Prevalence : 0.6400
##          Detection Rate : 0.5800
##    Detection Prevalence : 0.7200
##       Balanced Accuracy : 0.7587
##
##        'Positive' Class : 1
##
```

```
##### Preparing Data for Regression Task

# Creating temporary df combining predictions and original df
temp_df = cbind(df, rf1.preds)

# Filtering predictions for a value of 1 (i.e. successful customer acquisition)
df2 = temp_df %>%
  filter(rf1.preds == 1 & acquisition == 1) %>%
  subset(select = c(-acquisition, -rf1.preds))

# Correlation plot
par(mfrow = c(1, 1))
df2 %>%
  select_if(is.numeric) %>%
  cor() %>%
  corrplot(method = "number", type = "lower", number.cex = 0.65)

pairs(df2)

# Removing highly correlated variables
df2 = df2 %>%
  subset(select = c(-profit, -ret_exp))

# Splitting into train and test sets
set.seed(123)
index = sample(nrow(df2), 0.8*nrow(df2), replace = F) # 80/20 split
train_new = df2[index,]
```

```r
test_new = df2[-index,]



###### Modeling for Duration (Regression Task)

### Random Forest
set.seed(123) ; rf_model2 <- rfsrc(duration ~ ., train_new, ntree = 1000)

# Predicting on the test set and computing MSE
rf2.preds = predict(rf_model2, test_new)
mean((test_new$duration - rf2.preds$predicted)^2)
```

```
## [1] 5390.555
```

```r
# MSE: 5390.555


##### Tuned Random Forest for `Duration` (Using grid search)

# Establishing list of possible values for hyper-parameters
mtry.values <- seq(2,10,1)
nodesize.values <- seq(4,10,2)
#nodedepth.values <-seq(2,8,2)
ntree.values <- seq(1000,9000,2000)

# Creating data frame containing all combinations
hyper_grid = expand.grid(mtry = mtry.values,
                         nodesize = nodesize.values,
                         #nodedepth = nodedepth.values,
                         ntree = ntree.values)

# Creating empty vector to store out-of-bag error values
oob_err = c()

# Looping over the rows of hyper_grid to train the grid of models
for (i in 1:nrow(hyper_grid)) {
  # Training Random Forest
  set.seed(123) ; model = rfsrc(duration ~ .,
                                data = train_new,
                                mtry = hyper_grid$mtry[i],
                                nodesize = hyper_grid$nodesize[i],
                                #nodedepth = hyper_grid$nodedepth[i],
                                ntree = hyper_grid$ntree[i])
  # Storing OOB error for the model
  oob_err[i] <- model$err.rate[length(model$err.rate)]
}

# Identifying optimal set of hyper-parameters based on OOB error
opt_i <- which.min(oob_err)
print(hyper_grid[opt_i,])

##   mtry nodesize ntree
## 8    9        4  1000

# Applying tuned params to the model
set.seed(123) ; rf2.tuned = rfsrc(duration ~ ., data = train_new,
                                  importance = TRUE, ntree = 5000,
                                  mtry = 8, nodesize = 4)
rf2.tuned

##                          Sample size: 197
##                      Number of trees: 5000
##                Forest terminal node size: 4
##        Average no. of terminal nodes: 29.524
## No. of variables tried at each split: 8
##               Total no. of variables: 10
##        Resampling used to grow trees: swor
##     Resample size used to grow trees: 125
##                             Analysis: RF-R
##                               Family: regr
##                       Splitting rule: mse *random*
##        Number of random split points: 10
##                      (OOB) R squared: 0.97011165
##    (OOB) Requested performance error: 1378.13407575

rf2.tuned.preds = predict(rf2.tuned, test_new)
mean((test_new$duration - rf2.tuned.preds$predicted)^2)

## [1] 2804.261

# MSE: 856.1437
# For this task, we find that using grid search significantly improves the test
# accuracy, decreasing the MSE from 2385.8 to 856.1.


###### Extra credit: Generate PDP plots for all variables

plot.variable(rf_model1, partial = TRUE)
```