Machine Learning for Predicting Motor Insurance Outcome

Nick Aristidou

10/01/2022

Abstract

In this study, machine learning models have been applied to a motor insurance dataset to predict if a policy has had a claim or not. The target attribute, outcome, is a binary classification problem where the data displays a 70:30 class imbalance in favor of the no claim (0) class. Accuracy, sensitivity and specificity have been used to evaluate the models predictive performance to ensure that the best model for identifying the minor class is selected. The study finds that an optimally tuned xgbTree model yields the best results, with an accuracy of 86%, a sensitivity of 90% and a specificity of 77%.

Introduction

This research capstone project aims to explore the use of machine learning in predicting the Outcome of an insurance policy. The data used herein was obtained from kaggle, "https://www.kaggle.com/sagnik1511/car-insurance-data", and compiles motor insurance policy information, such as gender, age, vehicle type and more. Additionally, the data contains information on if the policy has had a claim or not, this is noted as the target field, Outcome, and is a binary field where 0 indicates a policy is claim free whilst a 1 shows the policy has had a claim. The project will first explore the data and note correlations between the variables and the target and then will explore the use of a number of machine learning algorithms with the aim of predicting if a policy has had a claim or not. The models explored will aim to solve the binary classification problem and will be compared using accuracy, sensitivity and specificity, A. Kumar (2018). The best model will be optimized using grid search cross validation to determine the best combination of hyperparameters.

Accuracy measures the number of correct predicted samples over the total number of samples.

$Accuracy = \frac{\text{no correct predictions}}{\text{no total predictions}}$

Sensitivity is a measure for identifying how well the model identifies all the samples in a given class. It is defined as the number of True Positives over the sum of the True Positive and False Negatives.

$Sensitivity = \frac{TP}{TP+FN}$

Specificity, also known as the true negative rate, is the measure of the proportion of actual negatives that are correctly identified.

$Specificity = \frac{TN}{TN+FP}$

These latter metrics are important for evaluating models, especially under the circumstances that the data used exhibits a class imbalance. Here the data shows an approximately 70:30 split for the outcomes 0:1.

Exploratory Data Analysis

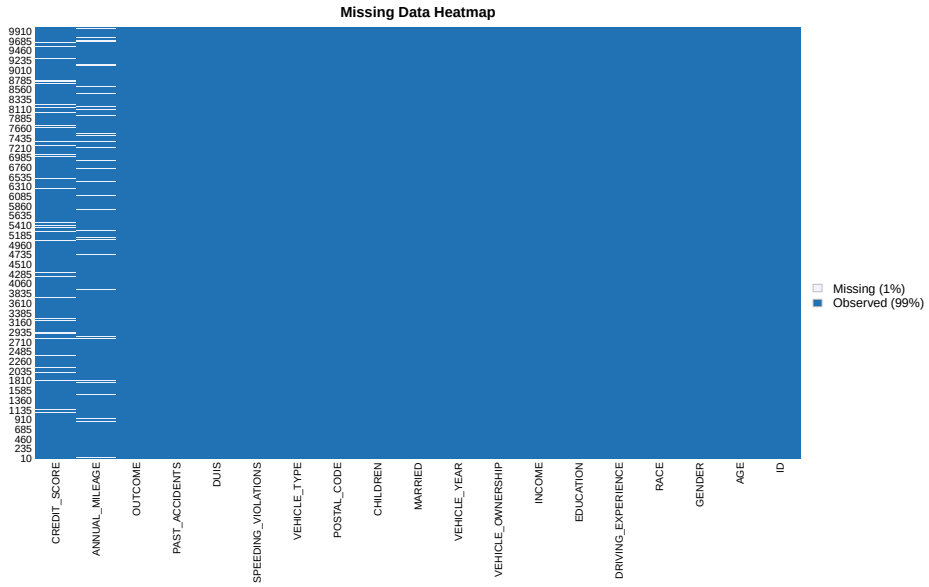
The first step is to load in the data from the csv file exported from kaggle and inspect the dataframe. The str() function has been employed achieve this and shows that there ar 19 variables and 10,000 rows contained in the data and displays the data types:

```
## 'data.frame':    10000 obs. of  19 variables:
##  $ ID                 : int  569520 750365 199901 478866 731664 877557 930134 461006 68366 445911 ..
##  $ AGE                : chr  "65+" "16-25" "16-25" "16-25" ...
##  $ GENDER             : chr  "female" "male" "female" "male" ...
##  $ RACE               : chr  "majority" "majority" "majority" "majority" ...
##  $ DRIVING_EXPERIENCE : chr  "0-9y" "0-9y" "0-9y" "0-9y" ...
##  $ EDUCATION          : chr  "high school" "none" "high school" "university" ...
##  $ INCOME             : chr  "upper class" "poverty" "working class" "working class" ...
##  $ CREDIT_SCORE       : num  0.629 0.358 0.493 0.206 0.388 ...
##  $ VEHICLE_OWNERSHIP  : num  1 0 1 1 1 1 0 0 0 1 ...
##  $ VEHICLE_YEAR       : chr  "after 2015" "before 2015" "before 2015" "before 2015" ...
##  $ MARRIED            : num  0 0 0 0 0 0 1 0 1 0 ...
##  $ CHILDREN           : num  1 0 0 1 0 1 1 1 0 1 ...
##  $ POSTAL_CODE        : int  10238 10238 10238 32765 32765 10238 10238 10238 10238 32765 ...
##  $ ANNUAL_MILEAGE     : num  12000 16000 11000 11000 12000 13000 13000 14000 13000 11000 ...
##  $ VEHICLE_TYPE       : chr  "sedan" "sedan" "sedan" "sedan" ...
##  $ SPEEDING_VIOLATIONS: int  0 0 0 0 2 3 7 0 0 0 ...
##  $ DUIS               : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ PAST_ACCIDENTS     : int  0 0 0 0 1 3 3 0 0 0 ...
##  $ OUTCOME            : num  0 1 0 0 1 0 0 1 0 1 ...
```

Next the data is examined for duplicates, which as shown below is not an issue where the get_dupes() function returns an empty tibble. Otherwise, duplicates would need to be removed as these could influence the models and it may allow them to have seen the data before.

```
##  [1] ID                  AGE                 GENDER
##  [4] RACE                DRIVING_EXPERIENCE  EDUCATION
##  [7] INCOME              CREDIT_SCORE        VEHICLE_OWNERSHIP
## [10] VEHICLE_YEAR        MARRIED             CHILDREN
## [13] POSTAL_CODE         ANNUAL_MILEAGE      VEHICLE_TYPE
## [16] SPEEDING_VIOLATIONS DUIS                PAST_ACCIDENTS
## [19] OUTCOME             dupe_count
## <0 rows> (or 0-length row.names)
```

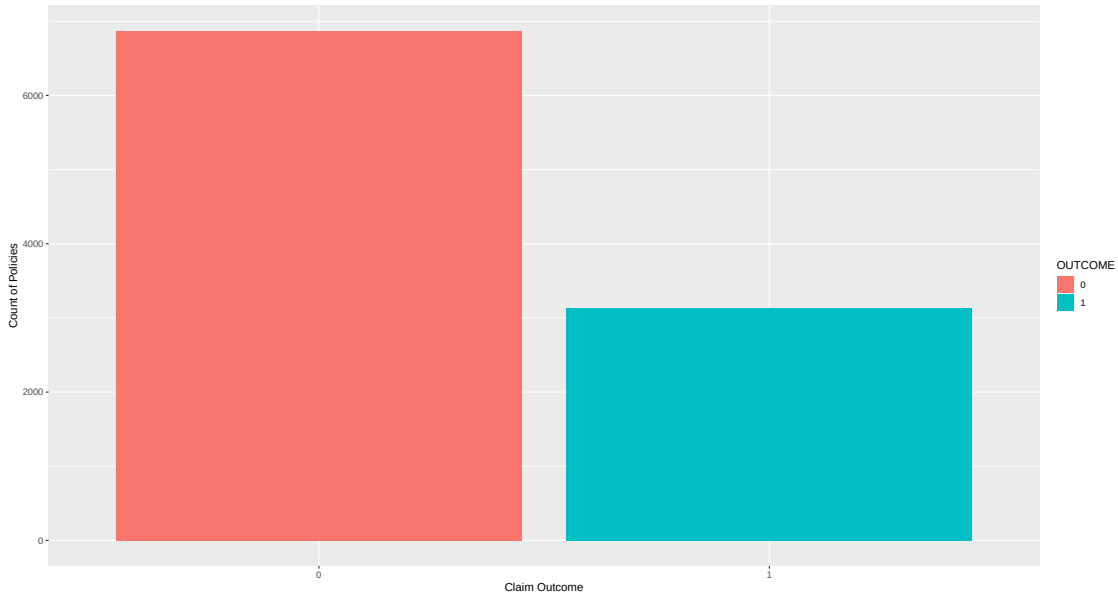
More over the data is explored for missing values and the heatmap displayed below shows that the two variables CREDIT_SCORE and ANNUAL_MILEAGE are sparsely populated. These missing values will be delt with in the following data preparation section.



Investigate Spread of Target Variable

Of the 19 attributes the target class is Outcome and the rest of the features will be used as predictors in the models. Class balance, is examined below and shows the ratio of the binary outcomes 0 and 1. From this it is noted that the data is imbalanced in favor of policies with no claim. The skew is roughly a 70:30 split, and this may cause the models to have a poor predictive performance especially for the minor class, as models were designed to around the assumption of an equal distribution of classes. In these situations a model can have a high accuracy just by default guessing a policy wont have a claim. This also posses a problem as typically the minor class is more important. An insurance company would want to know if a policy is going to have claim so the right premium can be charged for the risk and therefore having good predictive performance for the minor class is important. This means that for evaluating the models, accuracy alone will not be sufficient and consideration of specificity and sensitivity are required.

Var1	Freq
0	0.69
1	0.31



The Outcome attribute also needs to be converted into a factor, before the models are applied.

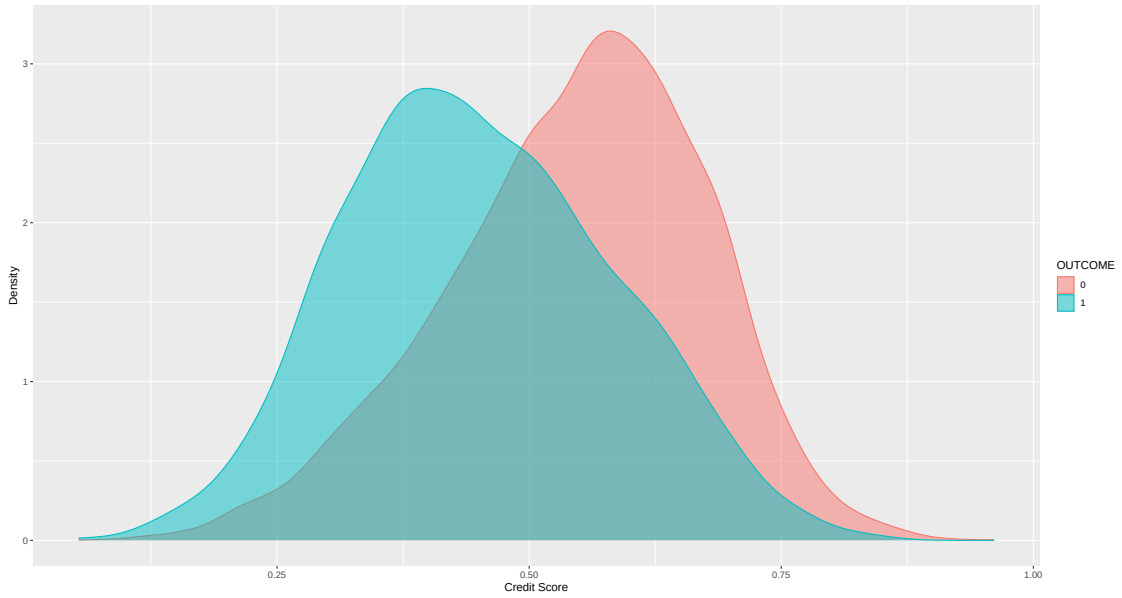
```
claims_df <-
  claims_df %>%
  mutate(OUTCOME=as.factor(OUTCOME))
```

Explore Numerical Attributes

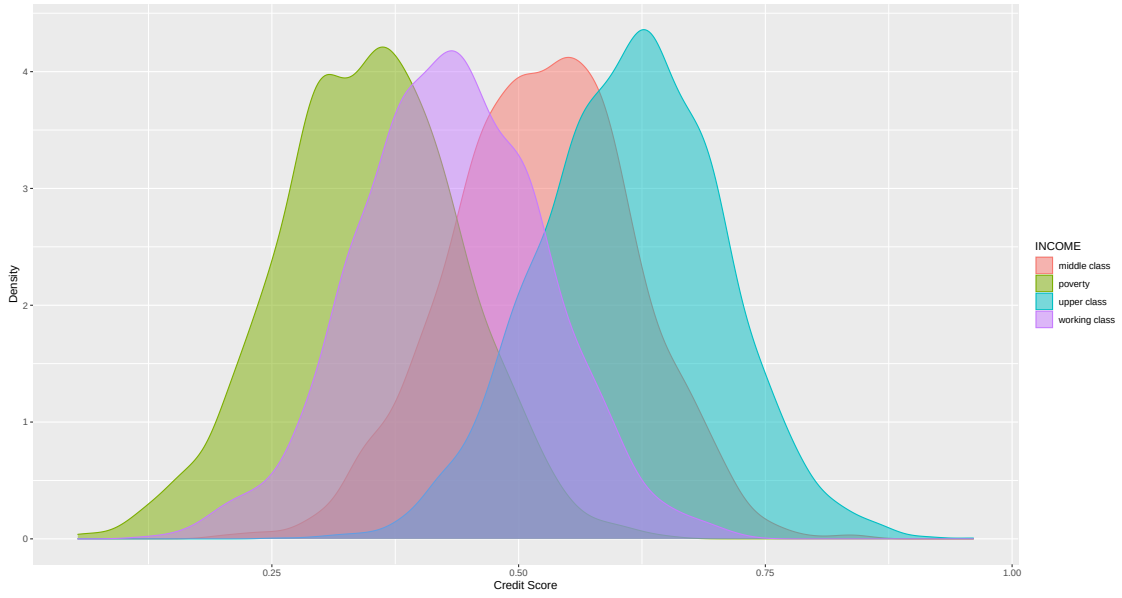
The attributes that have been loaded in as numerical are shown below in the subset of the data.

ID	CREDIT_SCORE	VEHICLE_OWNERSHIP	MARRIED	CHILDREN	POSTAL_CODE	ANNUAL_MILEAGE	SPEEDING_VIOLATIONS	DUIS	PAST_ACCIDENTS
569520	0.63	1	0	1	10238	12000	0	0	0
750365	0.36	0	0	0	10238	16000	0	0	0
199901	0.49	1	0	0	10238	11000	0	0	0
478866	0.21	1	0	1	32765	11000	0	0	0
731664	0.39	1	0	0	32765	12000	2	0	1
877557	0.62	1	0	1	10238	13000	3	0	3

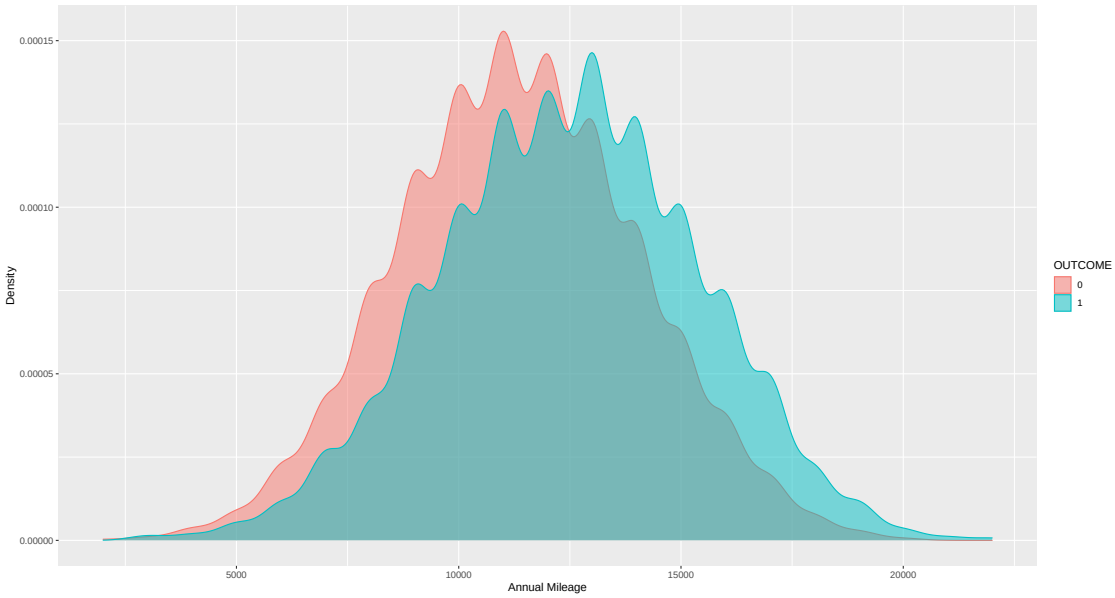
The first numerical field to be examined is the CREDIT_SCORE variable and the density plot below shows that this field has two distinct peaks when split by Outcome. As shown policies with a higher credit score are more likely to be claim free.

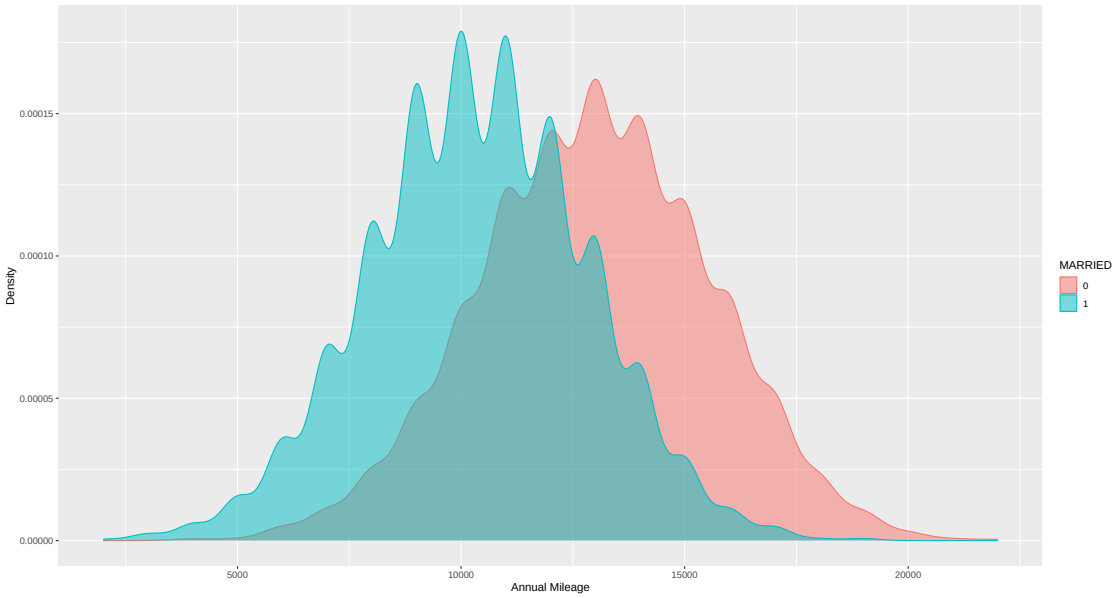


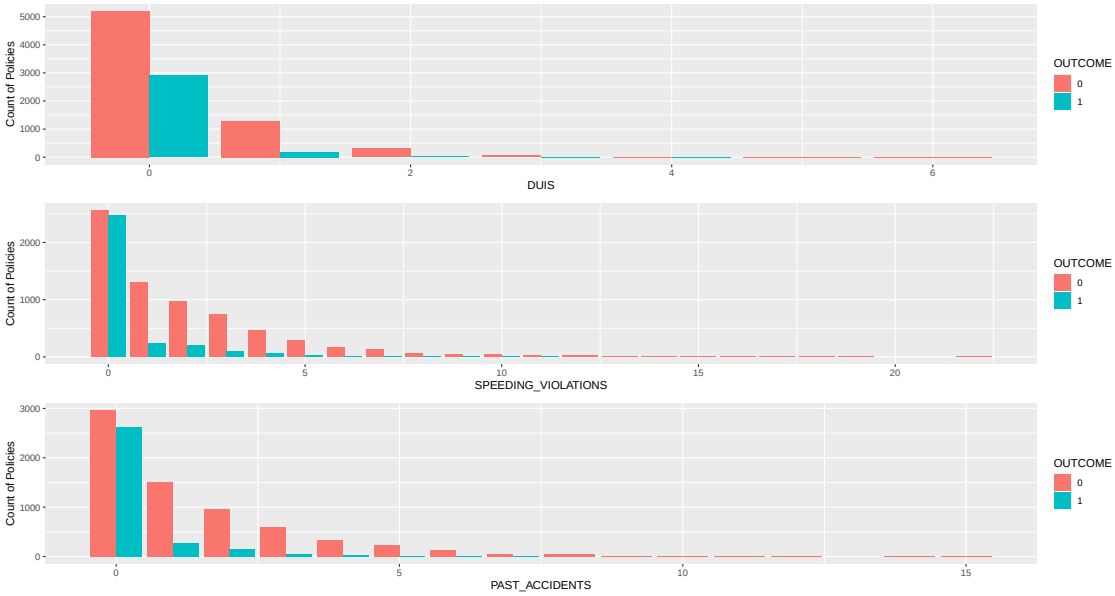
It is also noted that credit score is related to the Income Attribute and this will be used to help fill in missing values in the credit score attribute.



Annual mileage shows the trend that as the distance increases a policy is more likely to have a claim. It is noted that annual mileage correlates to the Mrried attribute, where those policies with a married classification have lower mileage.


0.00015
0.00010
0.00005
0.00000
Density
5000
10000
15000
20000
Annual Mileage
OUTCOME
0
1

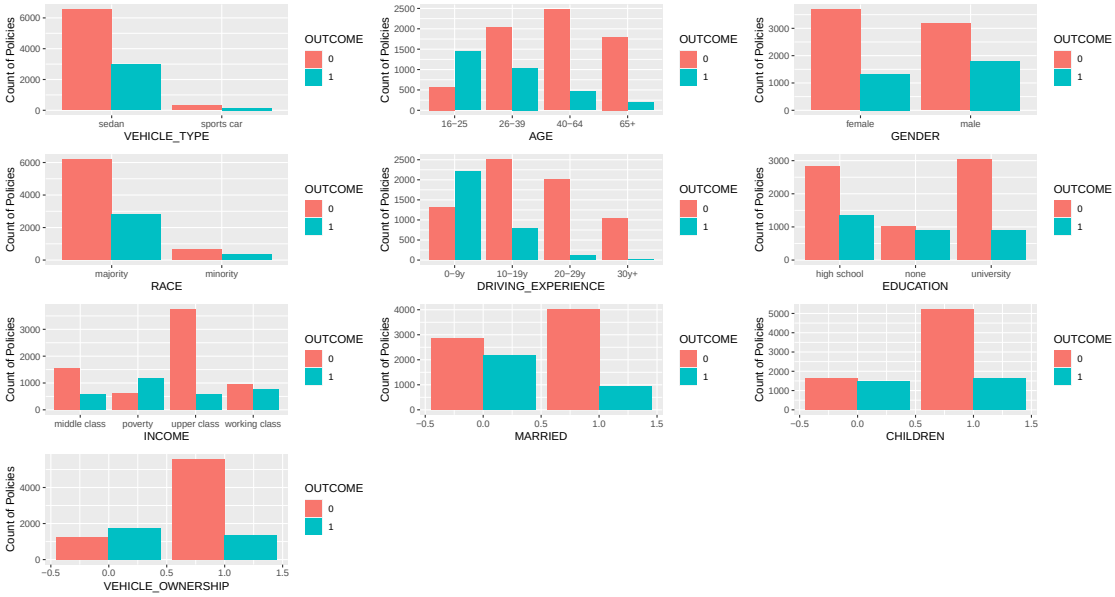

0.00015
0.00010
0.00005
0.00000
Density
5000
10000
15000
20000
Annual Mileage
MARRIED
0
1



All other values that have been set as numeric upon loading the data actually appear to be factors and will be explored in the following Categorical Variables section

Explore Categorical Attributes

All categorical variables are plotted as a function of the number of policies and split by the target attribute Outcome. A few observations to note, the data is heavily skewed to sedans, which are more likely to be claim free. Age correlates to the probability of a policy having a claims, were as expected older and therefore likely more experienced drivers are less likely to have a claim. Males are more likely to have a claim than females. Very few policies are associated with the minor race category and therefore will yield little difference on the outcome as predicting factor. Those with no education are more likely to have a claim than those who have some level of education. Likewise lower income levels are more likely to have an outcome of 1, relative to those that are in the middle class or upper class bracket.



Data Preperation

As noted previously, the two variables CREDIT_SCORE and ANNUAL_MILEAGE have a number of missing entries, ca. 1000 rows each. One method to handle this would be to drop these rows, however doing so causes the class imbalance to very slightly deteriorate further. Consequently, the aim is to retain these rows so that as much information is kept as possible to train the models on.

For credit score, it was observed that the variable was correlated to income and so to populate the missing blanks the average value of the credit score as a function of income is used.

```
for (i in unique(claims_df$INCOME)) {
  mu <- claims_df %>%
    filter(INCOME == i) %>%
    pull(CREDIT_SCORE) %>%
    mean(.,na.rm=T)

  claims_df$CREDIT_SCORE[
    is.na(claims_df$CREDIT_SCORE) & claims_df$INCOME == i
    ] <- mu
}
```

Whilst for annual mileage, the average value of the dataset is used to populate the missing values.

```
claims_df$ANNUAL_MILEAGE[is.na(claims_df$ANNUAL_MILEAGE)] <-
  mean(claims_df$ANNUAL_MILEAGE,na.rm=T)
```

The ID column is removed as this is a unique feature to every policy and is not required as a predictive attribute.

```
claims_df <-
  claims_df %>%
  select(-ID)
```

The data can be randomly split the data into a training set and a test set using the caret createDataPartition function to generate an 80:20 train:test split

```
test_index <- createDataPartition(y = claims_df$OUTCOME, times = 1, p = 0.2, list = FALSE)
train <- claims_df[-test_index,]
test <- claims_df[test_index,]
```

The train data has rows vs the test data set which has showing that the 80:20 split has been achieved. Additionally both the train and test data sets have retained a very similar class imbalance.

Train class balance:

Var1	Freq
0	0.69
1	0.31

Test class balance:

Var1	Freq
0	0.69
1	0.31

Applying Models

Now that the data is prepared, the classification models can be applied to the train data set with the aim of predicting the outcome attribute. It is worth noting the following code to train all the models takes a reasonable amount of time to run.

```
#XGBtree ----
fit_xgb <-
  caret::train(OUTCOME ~ .,
               data=train,
               method='xgbTree')

#GLM ----
fit_glm <-
  caret::train(OUTCOME ~ .,
               data=train,
               method="glm")

#LDA ----
fit_lda <-
  caret::train(OUTCOME ~ .,
               data=train,
               method="lda")

#QDA ----
fit_qda <-
  caret::train(OUTCOME ~ .,
               data=train,
               method="qda")

#kNN ----
fit_knn <-
  caret::train(OUTCOME ~ .,
               data=train,
               method="knn",
               tuneGrid=data.frame(k=seq(1,16,2)),
               trControl = trainControl(method = "cv",
                                        number = 10))

#Classification Tree ----
cp_grid <- data.frame(cp=seq(0, 0.03, 0.002))
fit_ct <- caret::train(OUTCOME ~ .,
                       data = train,
                       method="rpart",
                       tuneGrid = cp_grid)

#Random Forest ----
mtry_grid = data.frame(mtry = seq(1,7,2))
```

```
fit_rf <- caret::train(OUTCOME ~ .,
                       data = train,
                       method='rf',
                       tuneGrid=mtry_grid,
                       ntree=100)

#treebag ----
fit_tb <-
  caret::train(OUTCOME ~ .,
               data=train,
               method="treebag")

#C4.5 ----
fit_c4.5 <- caret::train(OUTCOME ~ .,
                         data = train,
                         method='J48')

#C5 ----
fit_c5 <- caret::train(OUTCOME ~ .,
                       data = train,
                       method='C5.0')
#FDA ----
fit_fda <- caret::train(OUTCOME ~ .,
                       data = train,
                       method='fda')
```

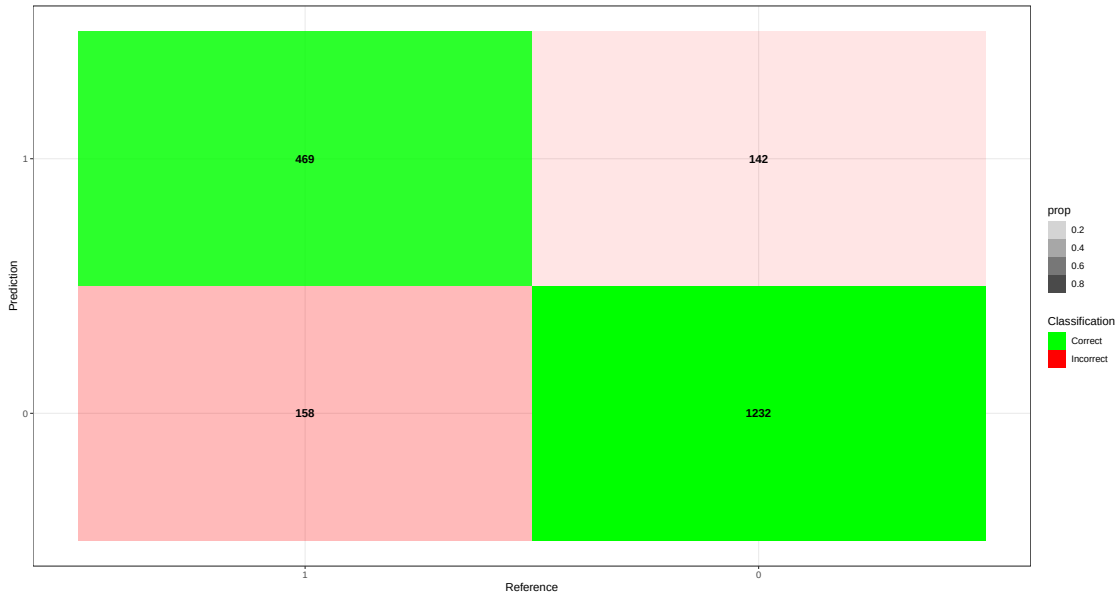
Model Performance Summary

Having fitted the models to the data the models are used to predict the Outcome attribute for the test set. The results of which are then used against the actual value to return the models accuracy, sensitivity and specificity. The results are displayed in the table below and have been ordered by accuracy (highest to lowest).

Model	Accuracy	Specificity	Sensitivity
xgbTree	0.85	0.75	0.90
Classification Tree	0.85	0.74	0.90
C5	0.85	0.71	0.92
FDA	0.85	0.73	0.90
GLM	0.85	0.72	0.90
Treebag	0.84	0.73	0.89
Random forest	0.84	0.70	0.90
C4.5	0.84	0.70	0.91
LDA	0.84	0.69	0.90
kNN	0.82	0.71	0.86
QDA	0.77	0.87	0.72

From this initial result, it can be seen that the best performing model taking into account all three metrics is the xgbTree model. This not only has the one of the highest accuracies of 85%, but has the highest combination of sensitivity, 90%, and specificity, 75%. Consequently, this model is the best at correctly predicting the outcome variable and on the whole handles both the minor class (1) and major class (0) better than any other model. This is demonstrated in the figure below, which plots out the heatmap for the

confusion matrix of the xgbTree model and shows how it identified the policies outcome (Prediction) relative to the actual (Reference) classification.



Model Optimisation

Having identified the xgbTree model as the best classifier for predicting the Outcome, the model hyperparameters will be optimised to further improve the predictive performance of the model. To achieve this a tuning grid to search across the hyperparameter space is set up and three fold cross validation is employed to determine the best possible combination of the values explored. To start the tuning protocol suggestions were taken from O. Zhang (2015).

Number of iterations and learning rate

In the first instance, the dominant feature of the maximum number of trees, max_depth, is examined in combination with the learning rate, eta. The tuning grid is set up as per the below code and the model is trained and the resulting best parameters are displayed in the following table.

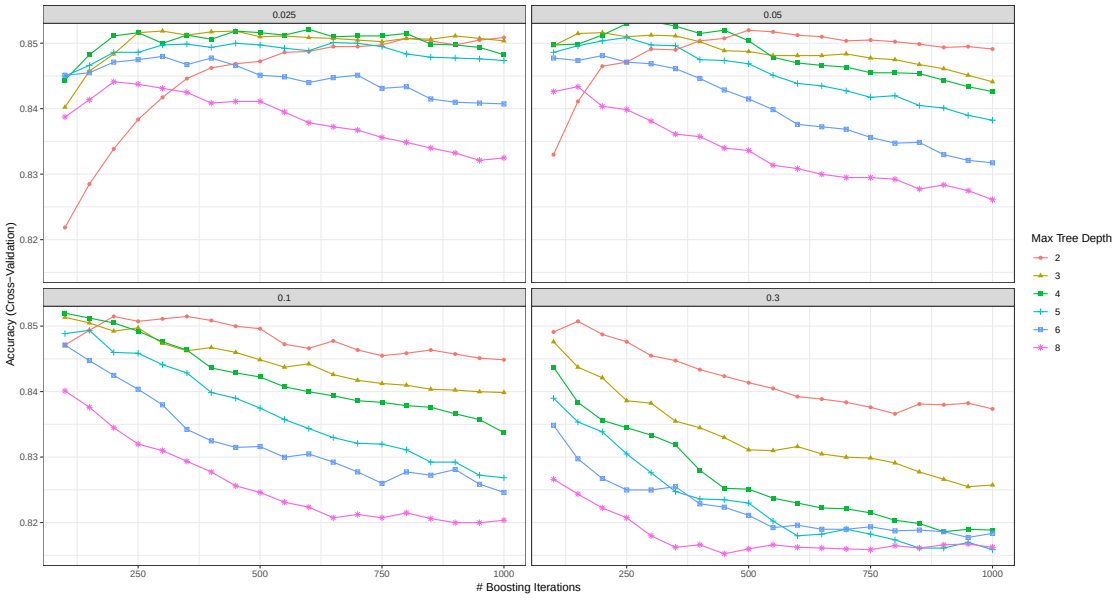
```
tune_grid <- expand.grid(
  nrounds = seq(from = 100, to = 1000, by = 50),
  eta = c(0.025, 0.05, 0.1, 0.3),
  max_depth = c(2, 3, 4, 5, 6, 8),
  gamma = 0,
  colsample_bytree = 1,
  min_child_weight = 1,
  subsample = 1
)

tune_control <- caret::trainControl(
  method = "cv", # cross-validation
  number = 3, # with n folds
  verboseIter = FALSE, # no training log
  allowParallel = TRUE
```

```
)

xgb_tune <- caret::train(
  OUTCOME ~ .,
  data=train,
  trControl = tune_control,
  tuneGrid = tune_grid,
  method = "xgbTree",
  verbose = TRUE
)
```

	nrounds	max_depth	eta	gamma	colsample_bytree	min_child_weight	subsample
157	300	4	0.05	0	1	1	1

The function below has been created to help visualize the tuning process as a function of the learning rate and will be recycled through the optimization process. The first tuning process is visualized below from this function:

```
tuneplot <- function(x, probs = .95) {
  ggplot(x) +
    coord_cartesian(ylim = c(min(x$results$Accuracy),
                             quantile(x$results$Accuracy,
                                      probs = probs))
                    ) +
    theme_bw()
}

tuneplot(xgb_tune)
```

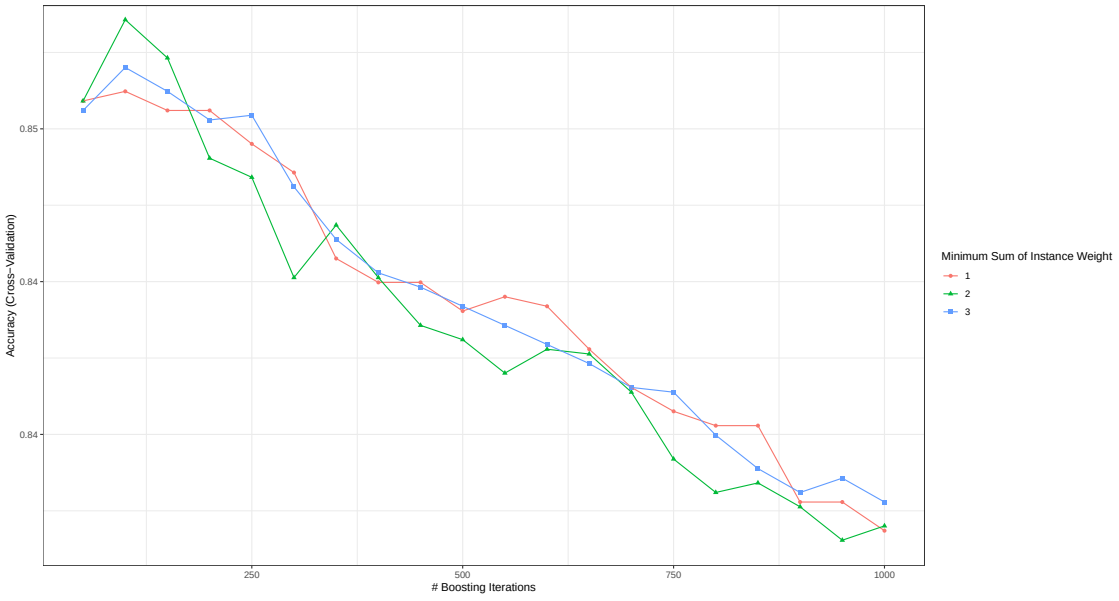


From the plots it can be confirmed that the best set of parameters, as per the accuracy measurement, are achieved from a max tree depth of 4 and a learning rate of 0.05

Minimum child weight

Now the hyperparameter, minimum child weight, is explored whilst fixing the learning rate and allowing a tolerance around the number of trees of +/- 1 (or +/- 2 if max_depth =2) for experimentation:

```
tune_grid2 <- expand.grid(
  nrounds = seq(from = 50, to = 1000, by = 50),
  eta = xgb_tune$bestTune$eta,
  max_depth = ifelse(xgb_tune$bestTune$max_depth == 2,
    c(xgb_tune$bestTune$max_depth:4),
    xgb_tune$bestTune$max_depth - 1:xgb_tune$bestTune$max_depth + 1),
  gamma = 0,
  colsample_bytree = 1,
  min_child_weight = c(1, 2, 3),
  subsample = 1
)

xgb_tune2 <- caret::train(
  OUTCOME ~ .,
  data=train,
  trControl = tune_control,
  tuneGrid = tune_grid2,
  method = "xgbTree",
  verbose = TRUE
)

tuneplot(xgb_tune2,0.995)
```



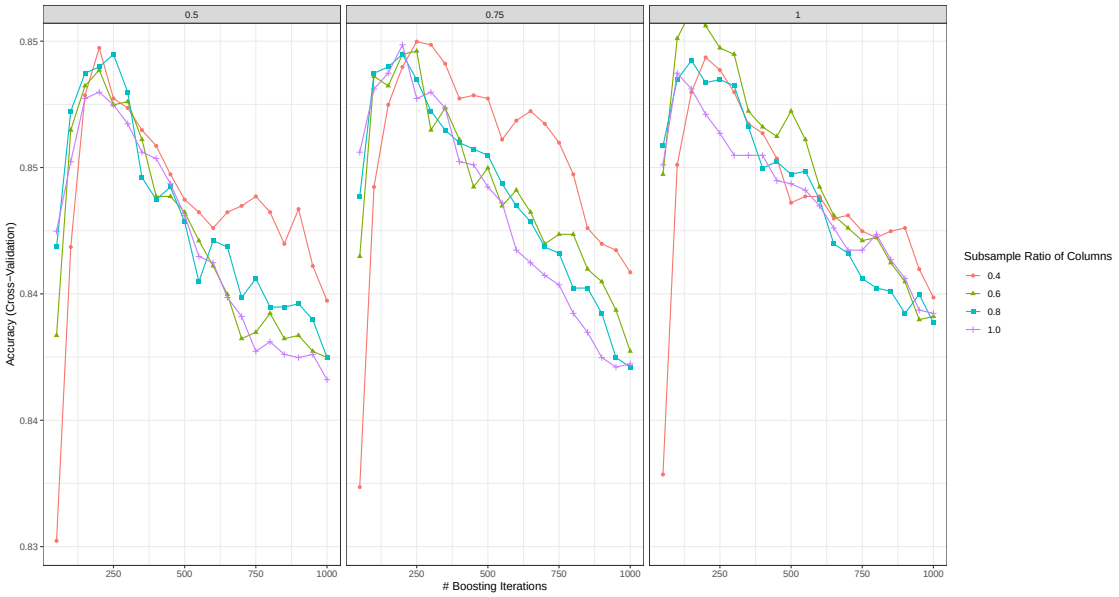
	nrounds	max_depth	eta	gamma	colsample_bytree	min_child_weight	subsample
22	100	4	0.05	0	1	2	1

The second tuning run, shows very little difference between the minimum child weights but the highest accuracy was achieved using 2.

Coulmn and row sampling

From the previous two steps the max depth is now fixed along with the eta and min child weight, whilst the different values for row and column sampling are explored.

```
tune_grid3 <- expand.grid(
  nrounds = seq(from = 50, to = 1000, by = 50),
  eta = xgb_tune$bestTune$eta,
  max_depth = xgb_tune2$bestTune$max_depth,
  gamma = 0,
  colsample_bytree = c(0.4, 0.6, 0.8, 1.0),
  min_child_weight = xgb_tune2$bestTune$min_child_weight,
  subsample = c(0.5, 0.75, 1.0)
)

xgb_tune3 <- caret::train(
  OUTCOME ~ .,
  data=train,
  trControl = tune_control,
  tuneGrid = tune_grid3,
  method = "xgbTree",
  verbose = TRUE
)

tuneplot(xgb_tune3, probs = .975)
```



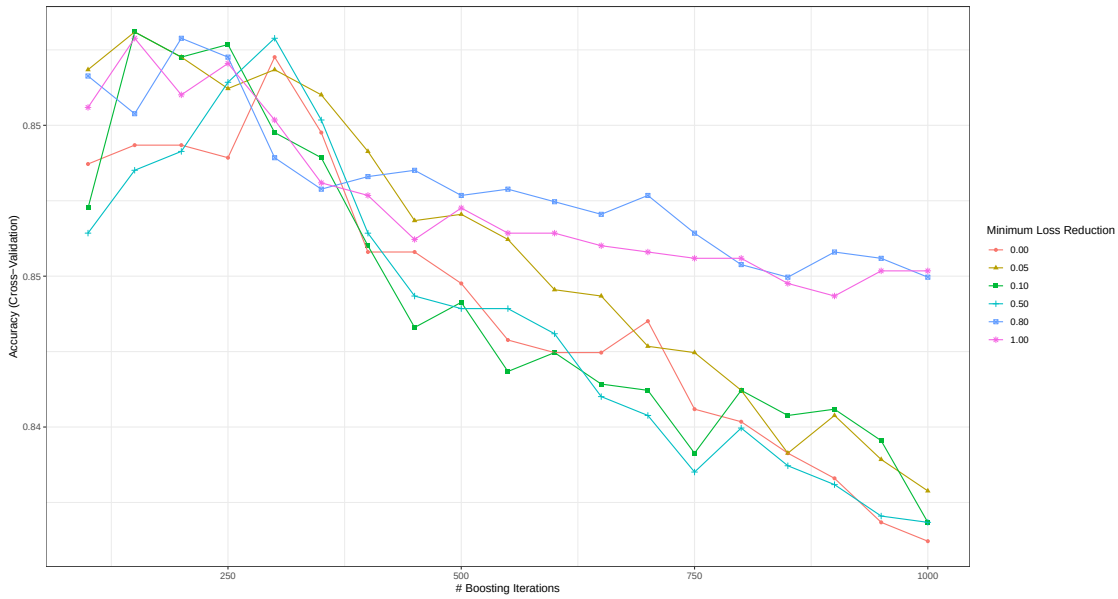
	nrounds	max_depth	eta	gamma	colsample_bytree	min_child_weight	subsample
103	150	4	0.05	0	0.6	2	1

Again very little improvement is seen through tuning these values, as noted by the similarity in the profiles of all the data points. However, the optimal values for these will be st from the tuning process as 0.6 and 1, for column and row sampling respectively.

Gamma

As before the previous tuning parameters are set to the optimal values and the gamma term is now explored to observe the impact on model performance.

```
tune_grid4 <- expand.grid(
  nrounds = seq(from = 100, to = 1000, by = 50),
  eta = xgb_tune$bestTune$eta,
  max_depth = xgb_tune2$bestTune$max_depth,
  gamma = c(0, 0.05, 0.1, 0.5, 0.8, 1.0),
  colsample_bytree = xgb_tune3$bestTune$colsample_bytree,
  min_child_weight = xgb_tune2$bestTune$min_child_weight,
  subsample = xgb_tune3$bestTune$subsample
)

xgb_tune4 <- caret::train(
  OUTCOME ~ .,
  data=train,
  trControl = tune_control,
  tuneGrid = tune_grid4,
  method = "xgbTree",
  verbose = TRUE
)

tuneplot(xgb_tune4,0.995)
```



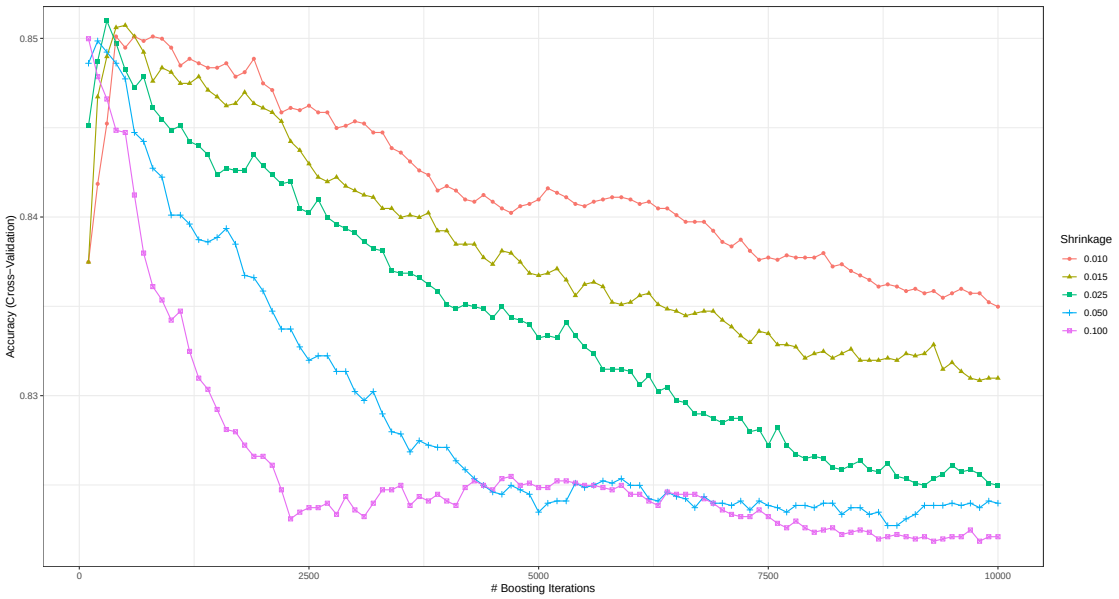
	nrounds	max_depth	eta	gamma	colsample_bytree	min_child_weight	subsample
40	150	4	0.05	0.1	0.6	2	1

From inspection of the results the optimal gamma is found to be, 0.1. Although it is noted that again very little difference in performance is noted between the values examined.

Reducing learning rate

Now that all the hyperparameters have been tuned, the learning rate is reduced.

```
tune_grid5 <- expand.grid(
  nrounds = seq(from = 100, to = 10000, by = 100),
  eta = c(0.01, 0.015, 0.025, 0.05, 0.1),
  max_depth = xgb_tune2$bestTune$max_depth,
  gamma = xgb_tune4$bestTune$gamma,
  colsample_bytree = xgb_tune3$bestTune$colsample_bytree,
  min_child_weight = xgb_tune2$bestTune$min_child_weight,
  subsample = xgb_tune3$bestTune$subsample
)


xgb_tune5 <- caret::train(
  OUTCOME ~ .,
  data=train,
  trControl = tune_control,
  tuneGrid = tune_grid5,
  method = "xgbTree",
  verbose = TRUE
)

tuneplot(xgb_tune5,0.995)
```



	nrounds	max_depth	eta	gamma	colsample_bytree	min_child_weight	subsample
203	300	4	0.03	0.1	0.6	2	1

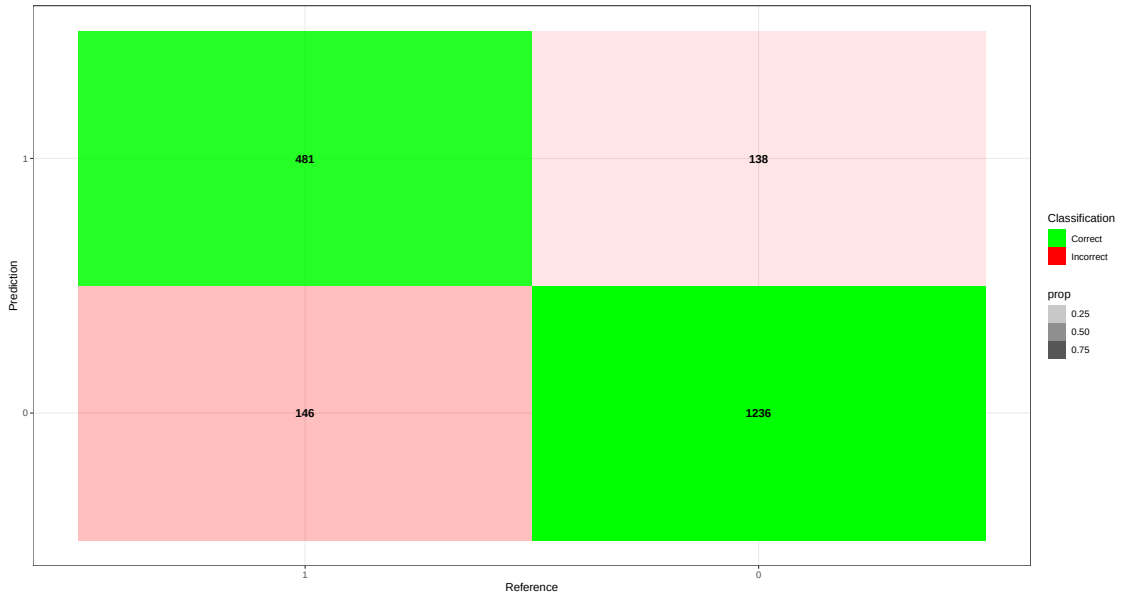
Fitting and evaluating final model

All parameters have been optimized and are now set in a final parameter grid and the new model is trained and tested on the data.

```
final_grid <- expand.grid(
  nrounds = xgb_tune5$bestTune$nrounds,
  eta = xgb_tune5$bestTune$eta,
  max_depth = xgb_tune5$bestTune$max_depth,
  gamma = xgb_tune5$bestTune$gamma,
  colsample_bytree = xgb_tune5$bestTune$colsample_bytree,
  min_child_weight = xgb_tune5$bestTune$min_child_weight,
  subsample = xgb_tune5$bestTune$subsample
)

train_control <- caret::trainControl(
  method = "none",
  verboseIter = FALSE,
  allowParallel = TRUE
)

xgb_model <- caret::train(
  OUTCOME ~ .,
  data=train,
  trControl = train_control,
  tuneGrid = final_grid,
  method = "xgbTree",
  verbose = TRUE
)
```

Model	Accuracy	Specificity	Sensitivity
Tuned xgbTree	0.86	0.77	0.90
xgbTree	0.85	0.75	0.90
Classification Tree	0.85	0.74	0.90
C5	0.85	0.71	0.92
FDA	0.85	0.73	0.90
GLM	0.85	0.72	0.90
Treebag	0.84	0.73	0.89
Random forest	0.84	0.70	0.90
C4.5	0.84	0.70	0.91
LDA	0.84	0.69	0.90
kNN	0.82	0.71	0.86
QDA	0.77	0.87	0.72

The new tuned model performs slightly better in terms of specificity and therefore correctly identifies more of the policies that have an outcome of one. This is shown in the confusion matrix heatmap shown below that shows out of the 2001 policies the model correctly predicts 481 policies with an outcome of one relative to the un-tuned model that predicted 469 correctly (2.5% improvement). Although this is a small enhancement, for the problem at hand this makes the model better as an insurance company will want to more accurately identify policies with an outcome of 1, as these are the policies that are going to cause the company to experience losses and need to be priced accordingly.



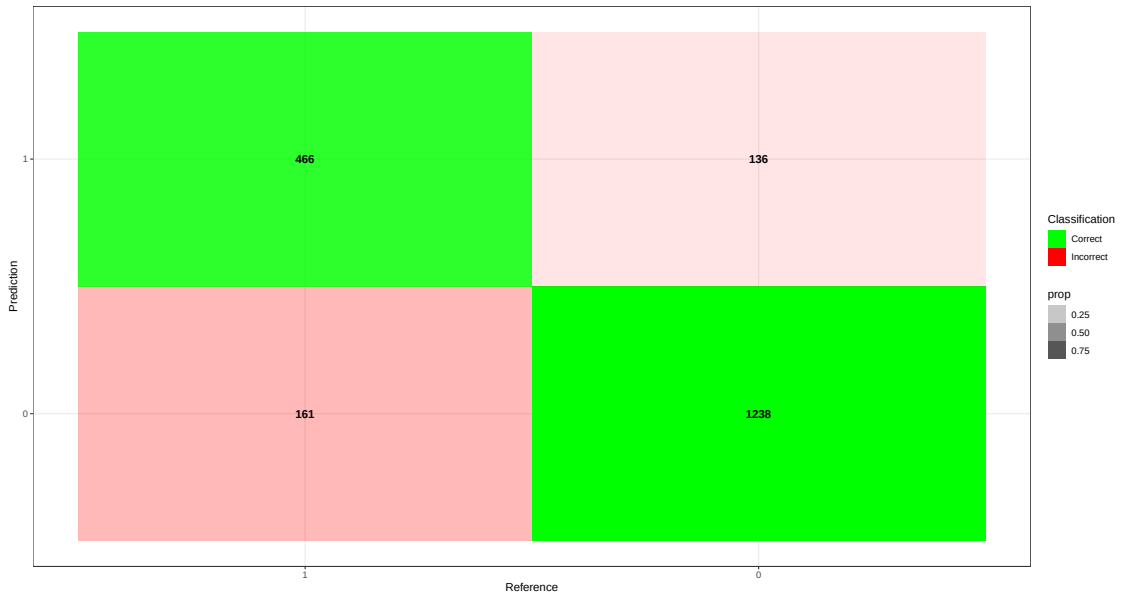
Ensemble

Another technique to enhance model prediction performance is to ensemble models together and combine the predictive power of a collection of models. This technique is explored below using a combination of the models. It is noted that an exhaustive search of all possible permutations was not conducted and there may be a combination that would lead to better predictive performance. However, the model proposed below yielded impressive accuracy, 85%, along with a high specificity 74% and sensitivity of 90%.

```
y_models <-
  bind_cols(
    y_glm = predict(fit_glm,test),
    y_lda = predict(fit_lda,test),
    y_fda = predict(fit_fda,test),
    y_qda = predict(fit_qda,test),
    y_c5 = predict(fit_c5,test),
    y_ct = predict(fit_ct,test),
    y_xgb = predict(xgb_model,test)
  )

ensemble_pred <-
  y_models %>%
  mutate_all(as.numeric) %>%
  rowMeans()

ensemble_pred <-
  ifelse(ensemble_pred > 1.5, 1,0)
```

Once more the confusion matrix heatmap is generated for the ensemble model and shows that model does well compared to all other models in terms of its ability to correctly identify the outcome attribute of 1.

Model	Accuracy	Specificity	Sensitivity
Tuned xgbTree	0.86	0.77	0.90
Ensemble Model	0.85	0.74	0.90
xgbTree	0.85	0.75	0.90
Classification Tree	0.85	0.74	0.90
C5	0.85	0.71	0.92
FDA	0.85	0.73	0.90
GLM	0.85	0.72	0.90
Treebag	0.84	0.73	0.89
Random forest	0.84	0.70	0.90
C4.5	0.84	0.70	0.91
LDA	0.84	0.69	0.90
kNN	0.82	0.71	0.86
QDA	0.77	0.87	0.72



Conclusion

In conclusion, the xgbTree model with tuned hyperparamters delivered the best predictive performance of the outcome attribute from the motor insurance data set. The model achieved a binary classification accuracy of 86%, a sensitivity of 90% and a specificity of 77%. Initially the data was explored to show how the features of the data related to the target variable outcome, which indicates if a policy has had a claim (1) or not (0). The analysis revealed a number of trends in the data for example as age and driving experience increased the likelihood of a policy having a claim decreased, agreeing with natural intuition. Furthermore, the EDA showed that there were no outliers but there were however two variables with approximately 10% of the data missing as blanks. The credit score attribute was populated using the mean of the column values as a function of a co-variate column providing more detail than just the simple average. Whilst the annual mileage attribute was populated with the average value. It is worth noting this technique may mislead the machine learning models and other approaches could be considered, for example dropping all data with blanks. It could also be worth exploring annual mileage as a function of the married status as this was shown to have an impact on this attribute. In this research the blanks were replaced to maximize the number of data points used and ensure the maximum amount of data showing the minor target class were

kept. Importantly the EDA, also highlighted that the data was imbalanced and favored the classes 0:1 in a 70:30 split. Due to this it was important to not only consider accuracy but also specificity when evaluating the predictive performance of the models. In this data knowing the outcome is assigned correctly as class 1 is very important as it is these policies that the modeling needs to find so that adequate insurance pricing measures can be put in place and an equitable and fair premium can be charged to policy holders. In light of this a number of models were searched and the most promising model was taken and tuned for optimal performance. Furthermore, an ensemble model was also created and showed very promising results that were almost equal to the tuned model in isolation.

Future Work

The research reported herein has focused on an initial search of a number of machine learning models with the best performing model being optimized and an ensemble model being generated. There are more avenues to explore that may lead to further enhancements in predictive performance, firstly the challenge of the class imbalance could be addressed through the use over random under sampling. Here a random selection of the major class data points are dropped such that class balance is achieved. Alternatively, a new approach to class imbalance is to generate synthetic versions of the minor class using techniques like SMOTE, J. Brownlee (2020). However, more practically in a business case it may be worth collecting more data points so that a balanced set can be achieved. Striving for class balance is likely to help ensure the true model performance is determined and to ensure that policies are correctly identified as having had a claim.

Another step may be to investigate the use of group bandings as this has been shown to yield better results for a number of models, a specific example of this technique can be seen in the titanic survivour prediction research, as presented by Ekinci et. al (2018). For example the attribute DUIS could be banded into three groups, 0, 1-3, 3+. Further enhancements to the dataset could also include scaling either through normalization or standardization, A. Bhandari (2020).

As noted in the ensemble section an exahustive search of all possible combinations was not explored. Hence for further investigation a function to scan all permutations could be devisedsearching for the combination that yields the optimal balance between accuracy, sensitivity and specificty.

Finally further investigation into the impact of populating the missing values in the credit score and annual mileage attributes should be conducted. In the first instance a direct comparison to dropping these data points that contain blanks and populating the just the average value for the column could be explored.

References

1. https://www.kaggle.com/sagnik1511/car-insurance-data
2. A. Kumar, ML Metrics: Sensitivity vs. Specificity. Available at: https://dzone.com/articles/ml-metrics-sensitivity-vs-specificity-difference, (Accessed on 10 January 2022)
3. O. Zhang, Tips for data science competitions. Available at: https://www.slideshare.net/OwenZhang2/tips-for-data-science-competitions/14, (Accessed on 12 January 2022)
4. J. brownlee, SMOTE for Imbalanced Classification with Python. Available at: https://machinelearningmastery.com/smote-oversampling-for-imbalanced-classification/, (Accessed on 14 January 2022)
5. Ekinci, Ekin & Omurca, Sevinc & Acun, Neytullah. (2018). A Comparative Study on Machine Learning Techniques Using Titanic Dataset.
6. A. Bhandari, Feature Scaling for Machine Learning: Understanding the Difference Between Normalization vs. Standardization. Available at: https://www.analyticsvidhya.com/blog/2020/04/feature-scaling-machine-learning-normalization-standardization/, (Accessed on 14 January 2022)