Caravan Insurance Customer Classification

Varun Kundurthi (varunk7) and Ishaan Sharma (ishaan4)

Abstract

Our goal for this project is to formulate a classification model to predict potential buyers of caravan insurance policies based on customer product usage and socio-demographic data. In order to do this we will analyze the Caravan Insurance dataset provided by Dutch data mining company Sentient Machine Research. This project involves a few sections. The first being our initial analysis and insight on the features of customers that have purchased and not purchased Caravan insurance. We then perform experiments on classification using: Ridge Regression, K-Nearest Neighbors, Gradient Boosting, and an Artificial Neural network. The report concludes with a model comparison section and a conclusion on the best model for classifying and predicting potential buyers of caravan insurance.

Literature Review

This dataset was a part of The CoIL Challenge 2000 where participants were tasked with predicting who would be interested in buying a caravan insurance policy and why. In that competition the University of California, San Diego's Charles Elkan was crowned the winner. His submission involved a naive Bayesian learning method. By taking the cross product of the number of policies a person had and a total premium amount called a contribution he devised a simple algorithm to determine if an individual was likely to purchase a caravan insurance policy. Over time, other methods have been used. This is showcased on Kaggle where individuals have used logistic regression models, random forest models, and principal component analysis to solve the task of predicting who would be interested in buying a caravan insurance policy.

Details on Dataset

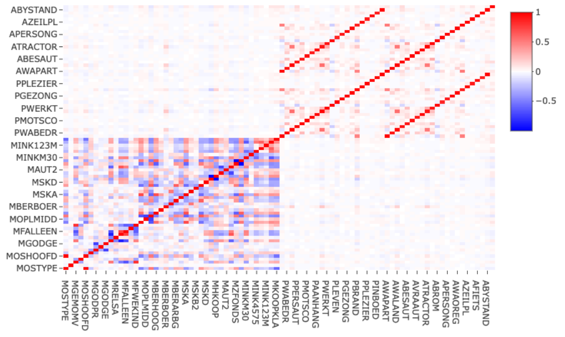
The dataset contains nearly 10,000 observations and 87 variables for each observation. The first of these is whether the data belongs in the training or testing subset. The training data contains close to 6,000 observations and the testing data contains 4,000 observations. From there the rest of the dataset contains information about the individuals in the dataset such as their: age, religion, household information, education level, social status, income, and the different policies. Variables beginning with M refer to demographic statistics of the postal code and variables beginning with P and A (as well as CARAVAN, the response variable) refer to product ownership and insurance statistics in the postal code. The CARAVAN response variable is encoded as 0 (not having caravan policy) and 1 (having caravan policy).

Data Exploration and Visualization

Table of distribution of Caravan Insurance Policy:

Caravan Policy <fctr>	Frequency <int>
0	9236
1	586

We find that 9,236 or 94% of the data does not have a CARAVAN policy, implying heavy imbalance in the dataset. Next, we calculate the covariance matrix of the dataset to assess the correlation between variables and calculate pairs of features that have the highest correlation.



Variable1 <chr>	Variable2 <chr>	Correlation <dbl>
MOSHOOFD	MOSTYPE	0.9927117
MRELOV	MRELGE	−0.8832510
MHKOOP	MHHUUR	−0.9996250
MZPART	MZFONDS	−0.9993813
AWAPART	PWAPART	0.9810968
AWABEDR	PWABEDR	0.9066627
AWALAND	PWALAND	0.9844844
APERSAUT	PPERSAUT	0.9075061
ABESAUT	PBESAUT	0.8943094
AMOTSCO	PMOTSCO	0.9237277

Next, we explore features that may have the most importance on classifying potential Caravan customers: purchasing power (MKOOPKLA), customer main type (MOSHOOFD), number of car policies (APERSAUT), and religion. Based on the figures and use of Chi squared test and using an alpha of 0.05, we conclude that purchasing power, number of car policies, and customer main types are statistically significant when determining if a customer wants caravan insurance. From figure 4, we see customer religion is not a good indicator on if a customer will purchase caravan insurance.

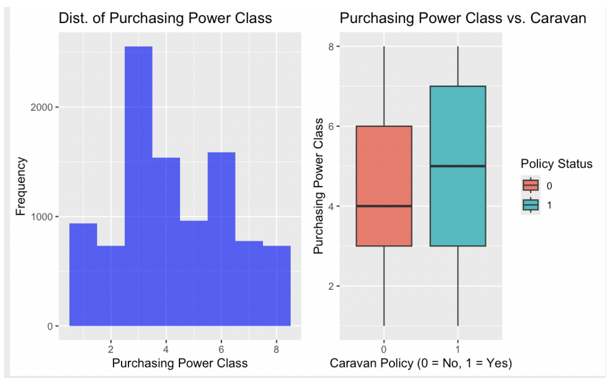


Figure 1: Purchasing Power Plots

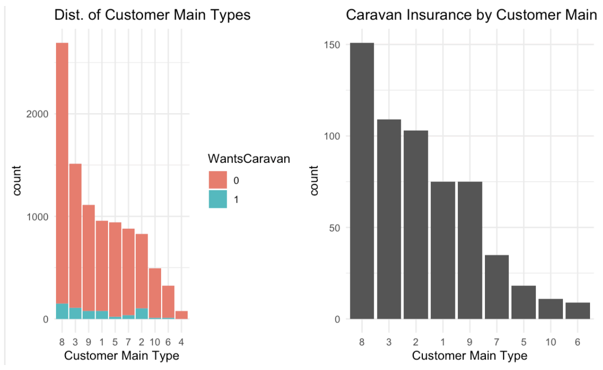


Figure 2: Customer Main Type Plots

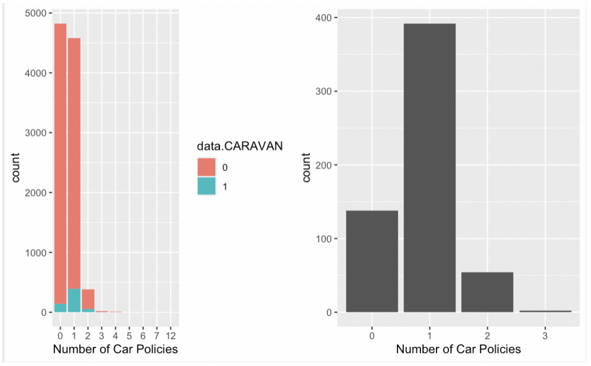


Figure 3: Number of Policies Plot

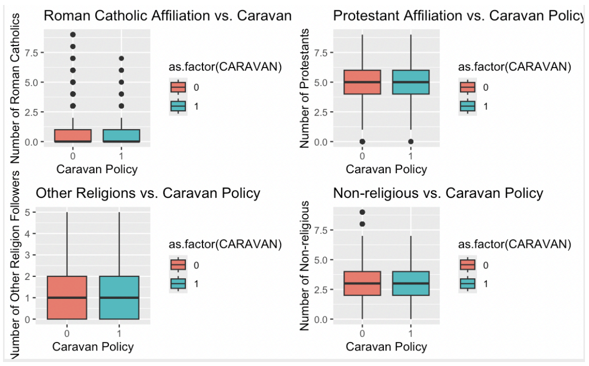


Figure 4: Religion Analysis

Data Preprocessing

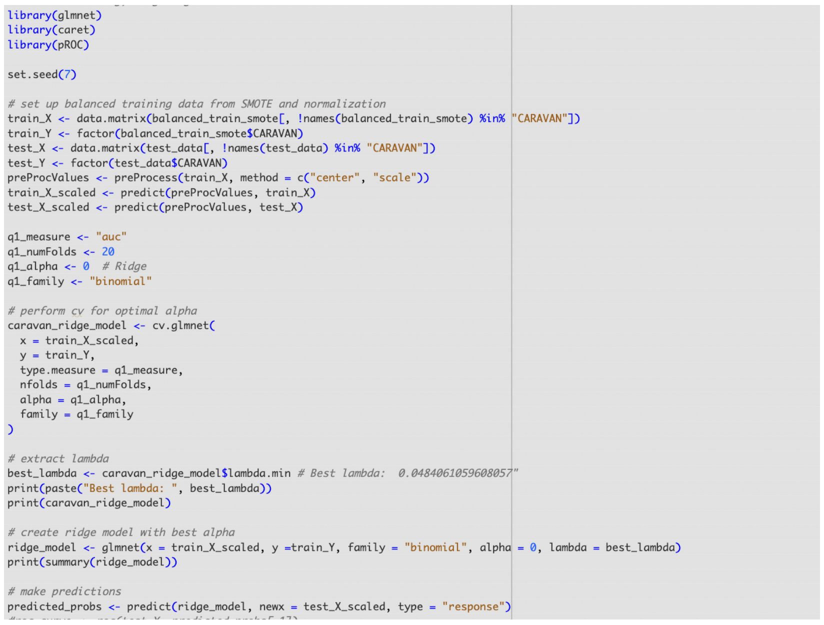
In the initial assessment of our training data, we identified a significant class imbalance in the response variable CARAVAN from the disproportionate distribution between the two classes (0 for "not purchased" and 1 for "purchased"). We employed balancing techniques on only the training set to prevent the classification model bias toward the majority class (0- not purchased). Using SMOTE and a k-value of 5, we corrected the imbalance of the dataset and our training set had 51.19% observations of Caravan 0 and 48.81% of Caravan 1 to ensure models could adequately train on (see Appendix Figure 1). We use normalization techniques to scale and standardize the training and testing sets to ensure features contribute equally in decision making outcomes.

Classification

<u>Model 1: Ridge Regression</u>

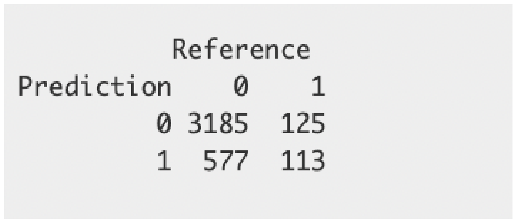
We chose to implement a ridge regression model because of its ability to handle multicollinearity of the 85 input variables. The alpha penalization coefficient shrinks the coefficients of less important features to 0, which provides information on important predictive features and prevents overfitting and is especially helpful for classifying an unbalanced data set. Using cross validation of 20 folds, we obtain and use the best alpha value for the ridge regression model which is 0.01749.

Code:

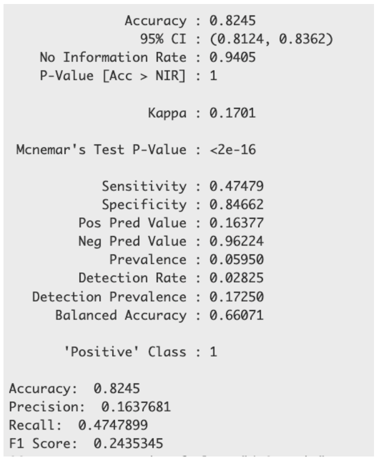


```r
library(glmnet)
library(caret)
library(pROC)

set.seed(7)

# set up balanced training data from SMOTE and normalization
train_X <- data.matrix(balanced_train_smote[, !names(balanced_train_smote) %in% "CARAVAN"])
train_Y <- factor(balanced_train_smote$CARAVAN)
test_X <- data.matrix(test_data[, !names(test_data) %in% "CARAVAN"])
test_Y <- factor(test_data$CARAVAN)
preProcValues <- preProcess(train_X, method = c("center", "scale"))
train_X_scaled <- predict(preProcValues, train_X)
test_X_scaled <- predict(preProcValues, test_X)

q1_measure <- "auc"
q1_numFolds <- 20
q1_alpha <- 0  # Ridge
q1_family <- "binomial"

# perform cv for optimal alpha
caravan_ridge_model <- cv.glmnet(
  x = train_X_scaled,
  y = train_Y,
  type.measure = q1_measure,
  nfolds = q1_numFolds,
  alpha = q1_alpha,
  family = q1_family
)

# extract lambda
best_lambda <- caravan_ridge_model$lambda.min # Best lambda:  0.0484061059608057"
print(paste("Best lambda: ", best_lambda))
print(caravan_ridge_model)

# create ridge model with best alpha
ridge_model <- glmnet(x = train_X_scaled, y =train_Y, family = "binomial", alpha = 0, lambda = best_lambda)
print(summary(ridge_model))

# make predictions
predicted_probs <- predict(ridge_model, newx = test_X_scaled, type = "response")
```

Results:



```
          Reference
Prediction    0    1
         0 3185  125
         1  577  113
```



```
               Accuracy : 0.8245
                 95% CI : (0.8124, 0.8362)
    No Information Rate : 0.9405
    P-Value [Acc > NIR] : 1

                  Kappa : 0.1701

 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.47479
            Specificity : 0.84662
         Pos Pred Value : 0.16377
         Neg Pred Value : 0.96224
             Prevalence : 0.05950
         Detection Rate : 0.02825
   Detection Prevalence : 0.17250
      Balanced Accuracy : 0.66071

       'Positive' Class : 1

Accuracy:  0.8245
Precision:  0.1637681
Recall:  0.4747899
F1 Score:  0.2435345
```

Using coefficients to determine the most important features from the ridge model, we obtain the top 17 most important features involved in predictions.

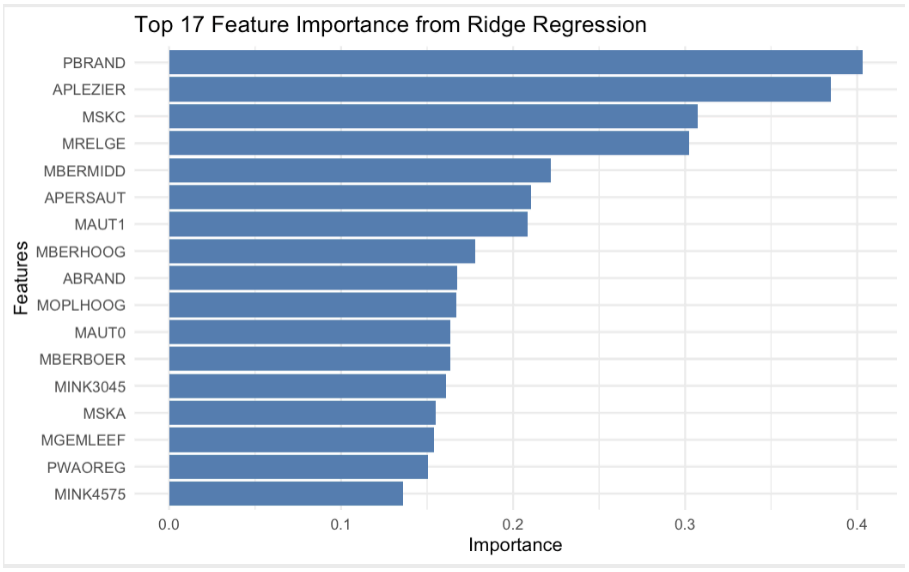
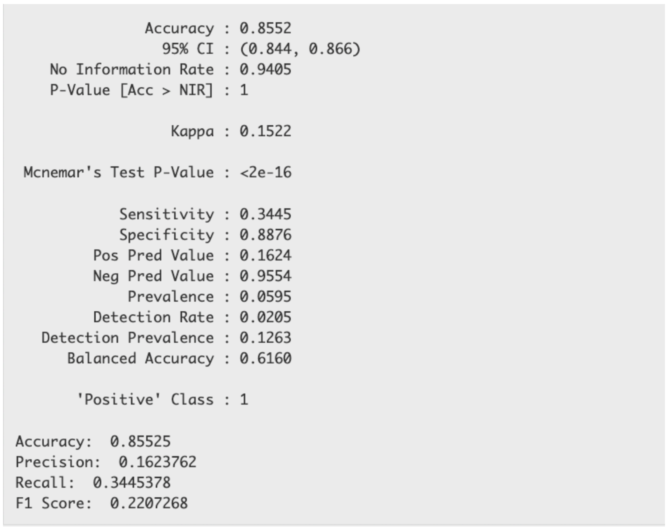


Figure 5: Top 17 important features Ridge Regression

Model 2: Gradient Boosting

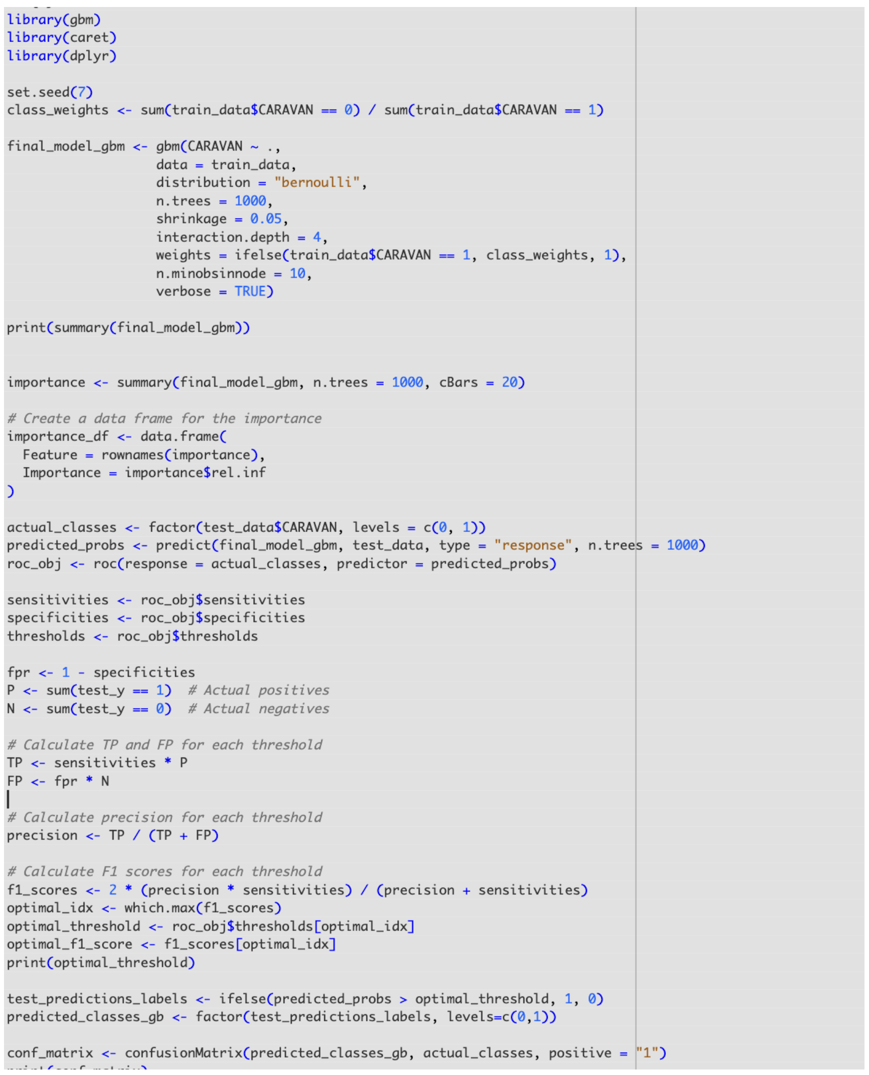
We then chose gradient boosting because it builds weak prediction models with decision trees. This ensemble method is able to capture the complex structure of the data and thus can accurately make better predictions for potential customers. We finetuned 3 parameters of the model: n.trees, shrinkage (learning rate), and depth of the trees. Using cross validation of 10 folds, we find that the best parameters are: 1000 num trees, shrinkage value of 0.05, and interaction depth of 4. We used an unbalanced training set for this model and utilized weights and assigned higher weights to the minority class (class 1) for a better F1 score.

Results:



```
               Accuracy : 0.8552
                 95% CI : (0.844, 0.866)
    No Information Rate : 0.9405
    P-Value [Acc > NIR] : 1

                  Kappa : 0.1522

 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.3445
            Specificity : 0.8876
         Pos Pred Value : 0.1624
         Neg Pred Value : 0.9554
             Prevalence : 0.0595
         Detection Rate : 0.0205
   Detection Prevalence : 0.1263
      Balanced Accuracy : 0.6160

       'Positive' Class : 1

Accuracy:  0.85525
Precision:  0.1623762
Recall:  0.3445378
F1 Score:  0.2207268
```

Code:



```r
library(gbm)
library(caret)
library(dplyr)

set.seed(7)
class_weights <- sum(train_data$CARAVAN == 0) / sum(train_data$CARAVAN == 1)

final_model_gbm <- gbm(CARAVAN ~ .,
                   data = train_data,
                   distribution = "bernoulli",
                   n.trees = 1000,
                   shrinkage = 0.05,
                   interaction.depth = 4,
                   weights = ifelse(train_data$CARAVAN == 1, class_weights, 1),
                   n.minobsinnode = 10,
                   verbose = TRUE)

print(summary(final_model_gbm))


importance <- summary(final_model_gbm, n.trees = 1000, cBars = 20)

# Create a data frame for the importance
importance_df <- data.frame(
  Feature = rownames(importance),
  Importance = importance$rel.inf
)

actual_classes <- factor(test_data$CARAVAN, levels = c(0, 1))
predicted_probs <- predict(final_model_gbm, test_data, type = "response", n.trees = 1000)
roc_obj <- roc(response = actual_classes, predictor = predicted_probs)

sensitivities <- roc_obj$sensitivities
specificities <- roc_obj$specificities
thresholds <- roc_obj$thresholds

fpr <- 1 - specificities
P <- sum(test_y == 1)  # Actual positives
N <- sum(test_y == 0)  # Actual negatives

# Calculate TP and FP for each threshold
TP <- sensitivities * P
FP <- fpr * N

# Calculate precision for each threshold
precision <- TP / (TP + FP)

# Calculate F1 scores for each threshold
f1_scores <- 2 * (precision * sensitivities) / (precision + sensitivities)
optimal_idx <- which.max(f1_scores)
optimal_threshold <- roc_obj$thresholds[optimal_idx]
optimal_f1_score <- f1_scores[optimal_idx]
print(optimal_threshold)

test_predictions_labels <- ifelse(predicted_probs > optimal_threshold, 1, 0)
predicted_classes_gb <- factor(test_predictions_labels, levels=c(0,1))

conf_matrix <- confusionMatrix(predicted_classes_gb, actual_classes, positive = "1")
```

The 17 most important features extracted from the gradient boosting model using imbalanced data.

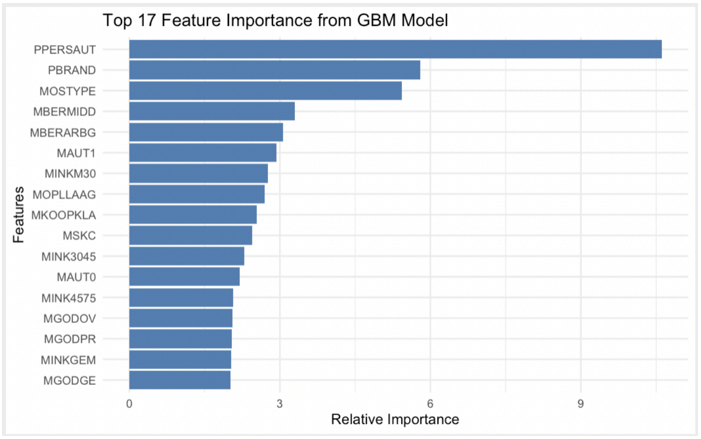


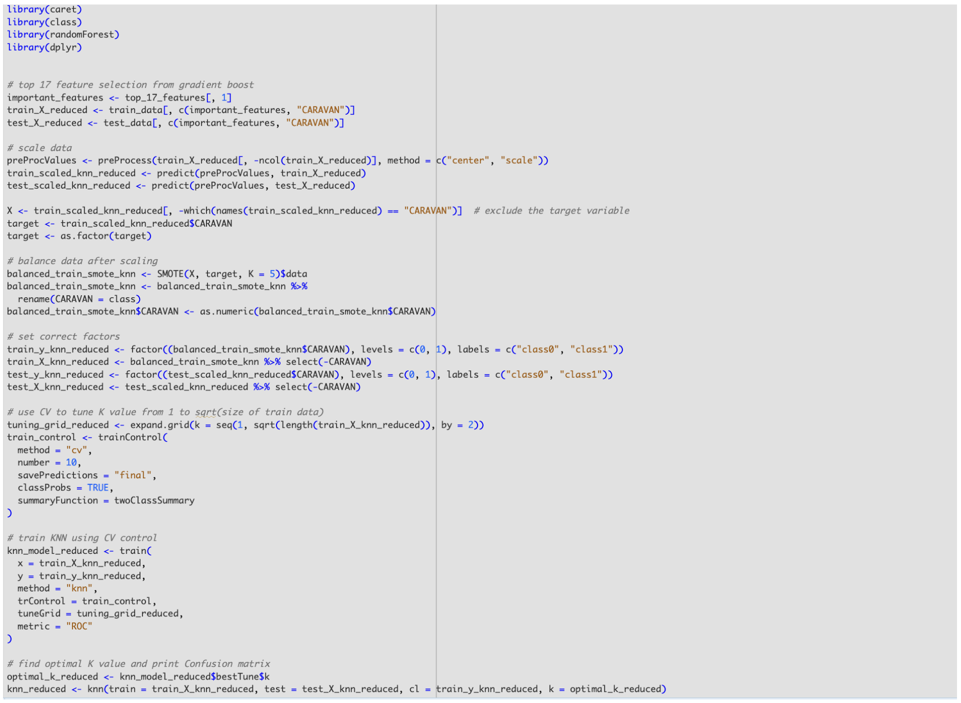
Figure 6: Top 17 important features Gradient Boost

The GBM tells us that these features are the most important variables for predicting whether a customer has or has not purchased Caravan insurance.

Model 3: k-Nearest-Neighbor

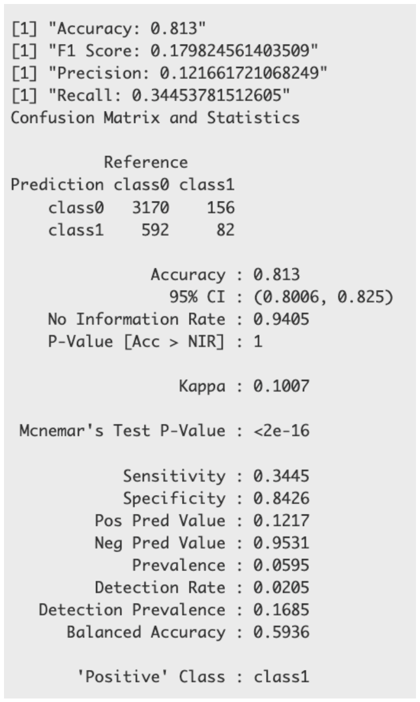
We chose the kNN model because it is non-parametric and does not make assumptions of the underlying distribution of the data. We use a subset of features for this model using top 17 features from the gradient boost model as a feature subset, training the kNN model on the balanced training dataset after scaling and centering the data. We used cross validation of 10 for training control to find the optimal K-value and normalize the data before training and testing the model with k values from 1 to the square root of the size and found the optimal k value was 3.

Code:



```
library(caret)
library(class)
library(randomForest)
library(dplyr)


# top 17 feature selection from gradient boost
important_features <- top_17_features[, 1]
train_X_reduced <- train_data[, c(important_features, "CARAVAN")]
test_X_reduced <- test_data[, c(important_features, "CARAVAN")]

# scale data
preProcValues <- preProcess(train_X_reduced[, -ncol(train_X_reduced)], method = c("center", "scale"))
train_scaled_knn_reduced <- predict(preProcValues, train_X_reduced)
test_scaled_knn_reduced <- predict(preProcValues, test_X_reduced)

X <- train_scaled_knn_reduced[, -which(names(train_scaled_knn_reduced) == "CARAVAN")]  # exclude the target variable
target <- train_scaled_knn_reduced$CARAVAN
target <- as.factor(target)

# balance data after scaling
balanced_train_smote_knn <- SMOTE(X, target, K = 5)$data
balanced_train_smote_knn <- balanced_train_smote_knn %>%
  rename(CARAVAN = class)
balanced_train_smote_knn$CARAVAN <- as.numeric(balanced_train_smote_knn$CARAVAN)

# set correct factors
train_y_knn_reduced <- factor((balanced_train_smote_knn$CARAVAN), levels = c(0, 1), labels = c("class0", "class1"))
train_X_knn_reduced <- balanced_train_smote_knn %>% select(-CARAVAN)
test_y_knn_reduced <- factor((test_scaled_knn_reduced$CARAVAN), levels = c(0, 1), labels = c("class0", "class1"))
test_X_knn_reduced <- test_scaled_knn_reduced %>% select(-CARAVAN)

# use CV to tune K value from 1 to sqrt(size of train data)
tuning_grid_reduced <- expand.grid(k = seq(1, sqrt(length(train_X_knn_reduced)), by = 2))
train_control <- trainControl(
  method = "cv",
  number = 10,
  savePredictions = "final",
  classProbs = TRUE,
  summaryFunction = twoClassSummary
)

# train KNN using CV control
knn_model_reduced <- train(
  x = train_X_knn_reduced,
  y = train_y_knn_reduced,
  method = "knn",
  trControl = train_control,
  tuneGrid = tuning_grid_reduced,
  metric = "ROC"
)

# find optimal K value and print Confusion matrix
optimal_k_reduced <- knn_model_reduced$bestTune$k
knn_reduced <- knn(train = train_X_knn_reduced, test = test_X_knn_reduced, cl = train_y_knn_reduced, k = optimal_k_reduced)
```

Results:

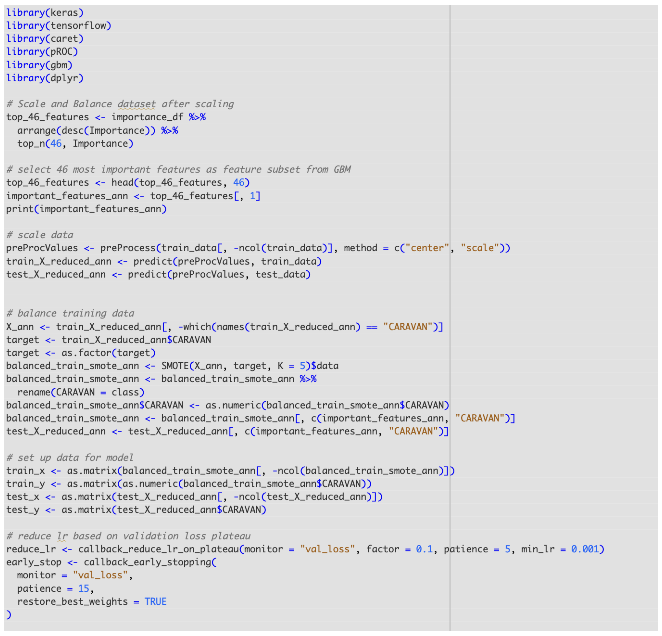


```
[1] "Accuracy: 0.813"
[1] "F1 Score: 0.179824561403509"
[1] "Precision: 0.121661721068249"
[1] "Recall: 0.34453781512605"
Confusion Matrix and Statistics

          Reference
Prediction class0 class1
    class0   3170    156
    class1    592     82

               Accuracy : 0.813
                 95% CI : (0.8006, 0.825)
    No Information Rate : 0.9405
    P-Value [Acc > NIR] : 1

                  Kappa : 0.1007

 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.3445
            Specificity : 0.8426
         Pos Pred Value : 0.1217
         Neg Pred Value : 0.9531
             Prevalence : 0.0595
         Detection Rate : 0.0205
   Detection Prevalence : 0.1685
      Balanced Accuracy : 0.5936

       'Positive' Class : class1
```

Model 4: Artificial Neural Network:

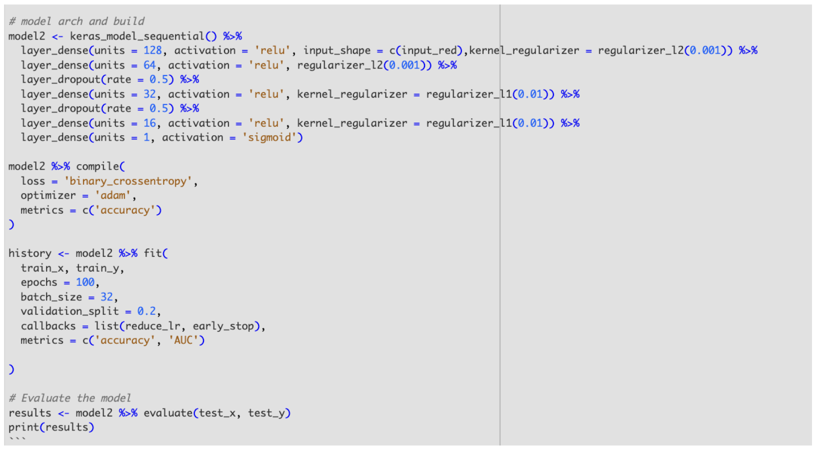
We chose the ANN model because of its ability to learn the most predictive features of the data through interactions with the hidden layers and to capture high dimensional complex data patterns seen in the Caravan dataset. We use scaling normalization and SMOTE for data preprocessing of this model due to the model's use of activation functions. Then used the top 46 important features as the feature subset for the model. The main hyperparameters we fine tuned were: learning rate, number of units, batch_size, number of hidden layers, dropout layers using loss, AUC, and accuracy from the validation set as deciding metrics for the best parameters (see Figure C in Appendix for training plots). We decided on a 6 layer model including input and output starting with 128 units and reducing it by half in each layer. We also added dropout to prevent overfitting on the majority class as well as L1 and L2 regularization with parameters shown in the code (found through iterative experimenting).

Code:



```r
library(keras)
library(tensorflow)
library(caret)
library(pROC)
library(gbm)
library(dplyr)

# Scale and Balance dataset after scaling
top_46_features <- importance_df %>%
  arrange(desc(Importance)) %>%
  top_n(46, Importance)

# select 46 most important features as feature subset from GBM
top_46_features <- head(top_46_features, 46)
important_features_ann <- top_46_features[, 1]
print(important_features_ann)

# scale data
preProcValues <- preProcess(train_data[, -ncol(train_data)], method = c("center", "scale"))
train_X_reduced_ann <- predict(preProcValues, train_data)
test_X_reduced_ann <- predict(preProcValues, test_data)


# balance training data
X_ann <- train_X_reduced_ann[, -which(names(train_X_reduced_ann) == "CARAVAN")]
target <- train_X_reduced_ann$CARAVAN
target <- as.factor(target)
balanced_train_smote_ann <- SMOTE(X_ann, target, K = 5)$data
balanced_train_smote_ann <- balanced_train_smote_ann %>%
  rename(CARAVAN = class)
balanced_train_smote_ann$CARAVAN <- as.numeric(balanced_train_smote_ann$CARAVAN)
balanced_train_smote_ann <- balanced_train_smote_ann[, c(important_features_ann, "CARAVAN")]
test_X_reduced_ann <- test_X_reduced_ann[, c(important_features_ann, "CARAVAN")]

# set up data for model
train_x <- as.matrix(balanced_train_smote_ann[, -ncol(balanced_train_smote_ann)])
train_y <- as.matrix(as.numeric(balanced_train_smote_ann$CARAVAN))
test_x <- as.matrix(test_X_reduced_ann[, -ncol(test_X_reduced_ann)])
test_y <- as.matrix(test_X_reduced_ann$CARAVAN)

# reduce lr based on validation loss plateau
reduce_lr <- callback_reduce_lr_on_plateau(monitor = "val_loss", factor = 0.1, patience = 5, min_lr = 0.001)
early_stop <- callback_early_stopping(
  monitor = "val_loss",
  patience = 15,
  restore_best_weights = TRUE
)
```

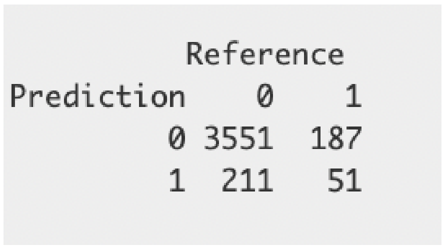
ANN Data Preprocessing R Code



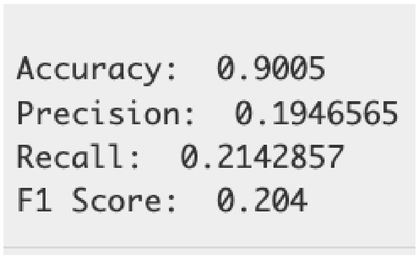
~~~r
# model arch and build
model2 <- keras_model_sequential() %>%
  layer_dense(units = 128, activation = 'relu', input_shape = c(input_red),kernel_regularizer = regularizer_l2(0.001)) %>%
  layer_dense(units = 64, activation = 'relu', regularizer_l2(0.001)) %>%
  layer_dropout(rate = 0.5) %>%
  layer_dense(units = 32, activation = 'relu', kernel_regularizer = regularizer_l1(0.01)) %>%
  layer_dropout(rate = 0.5) %>%
  layer_dense(units = 16, activation = 'relu', kernel_regularizer = regularizer_l1(0.01)) %>%
  layer_dense(units = 1, activation = 'sigmoid')

model2 %>% compile(
  loss = 'binary_crossentropy',
  optimizer = 'adam',
  metrics = c('accuracy')
)

history <- model2 %>% fit(
  train_x, train_y,
  epochs = 100,
  batch_size = 32,
  validation_split = 0.2,
  callbacks = list(reduce_lr, early_stop),
  metrics = c('accuracy', 'AUC')

)

# Evaluate the model
results <- model2 %>% evaluate(test_x, test_y)
print(results)
```
~~~

ANN Model Creation R Code

Results:



```
          Reference
Prediction    0    1
         0 3551  187
         1  211   51
```



```
Accuracy:  0.9005
Precision:  0.1946565
Recall:  0.2142857
F1 Score:  0.204
```

Model Comparison:

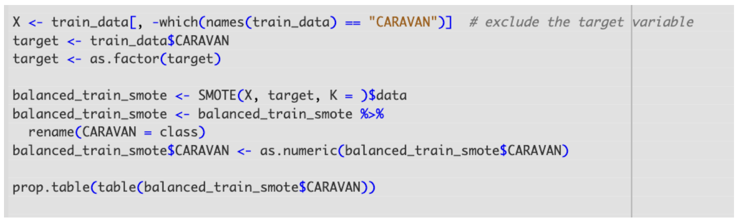
Model	Accuracy	Precision	Recall	F1 Score
Ridge Regression	0.8245	0.16	0.344	0.243
Gradient Boosting	0.855	0.16	0.344	0.221
kNN	0.813	0.121	0.344	0.179
ANN	0.9005	0.19	0.214	0.204

Since the test set is unbalanced (customers who purchased Caravan insurance are the minority class) the F1 score metric is most important when deciding the best model. We used the precision-Recall AUC metric to assess which model performed the best because of the imbalanced nature of the testing set. F1 score is the harmonic mean between precision and recall which is the best metric for imbalanced data since we want our model to accurately predict potential customers. We used a cutoff value that would optimize the F1 metric from the ROC curve for all the models since this was the most valuable metric in determining an effective model (see Figure B in Appendix). Accuracy was not a useful metric because of the test data imbalance of the minority ‘positive” class. We used SMOTE to synthesize new minority class samples for ridge, kNN, and ANN. For GBM, we instead used class weights as a penalization term for misclassifications on the minority class to handle imbalances and the model had good precision. The model with the best F1 metric was Ridge regression based on its ability to penalize and shrink coefficients of features with high correlations. Gradient boosting performed well when trained on an unbalanced testing set, indicating that its performance using class based penalization is the most optimal for predicting potential customers on new unseen data.

Conclusion

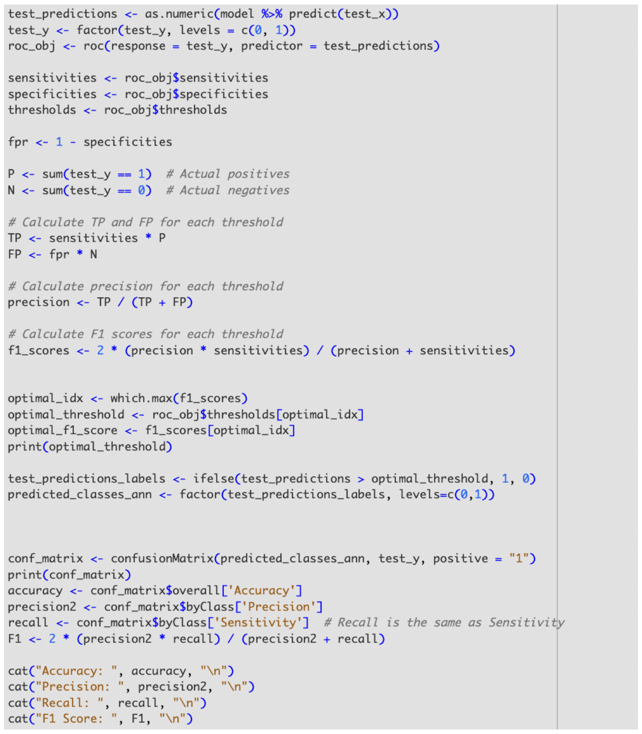
We will be using F1, precision, and recall scores as the deciding factor on which classification model is the best. From the model comparison above, we conclude that **gradient boosting** is the best model for predicting potential caravan insurance customers based on its performance metrics using the precision-recall curve, which focuses on the model’s ability to predict the positive (minority) class correctly. This is because of the model’s ability to use class-based penalization to predict minority classes in an unbalanced dataset. *Consideration*: Given more computational resources, fine-tuning the model would have given clear optimal results.

Appendix



```r
X <- train_data[, -which(names(train_data) == "CARAVAN")]  # exclude the target variable
target <- train_data$CARAVAN
target <- as.factor(target)

balanced_train_smote <- SMOTE(X, target, K = )$data
balanced_train_smote <- balanced_train_smote %>%
  rename(CARAVAN = class)
balanced_train_smote$CARAVAN <- as.numeric(balanced_train_smote$CARAVAN)

prop.table(table(balanced_train_smote$CARAVAN))
```

Figure A: SMOTE Balancing R Code and Output



```r
test_predictions <- as.numeric(model %>% predict(test_x))
test_y <- factor(test_y, levels = c(0, 1))
roc_obj <- roc(response = test_y, predictor = test_predictions)

sensitivities <- roc_obj$sensitivities
specificities <- roc_obj$specificities
thresholds <- roc_obj$thresholds

fpr <- 1 - specificities

P <- sum(test_y == 1)  # Actual positives
N <- sum(test_y == 0)  # Actual negatives

# Calculate TP and FP for each threshold
TP <- sensitivities * P
FP <- fpr * N

# Calculate precision for each threshold
precision <- TP / (TP + FP)

# Calculate F1 scores for each threshold
f1_scores <- 2 * (precision * sensitivities) / (precision + sensitivities)


optimal_idx <- which.max(f1_scores)
optimal_threshold <- roc_obj$thresholds[optimal_idx]
optimal_f1_score <- f1_scores[optimal_idx]
print(optimal_threshold)

test_predictions_labels <- ifelse(test_predictions > optimal_threshold, 1, 0)
predicted_classes_ann <- factor(test_predictions_labels, levels=c(0,1))



conf_matrix <- confusionMatrix(predicted_classes_ann, test_y, positive = "1")
print(conf_matrix)
accuracy <- conf_matrix$overall['Accuracy']
precision2 <- conf_matrix$byClass['Precision']
recall <- conf_matrix$byClass['Sensitivity']  # Recall is the same as Sensitivity
F1 <- 2 * (precision2 * recall) / (precision2 + recall)

cat("Accuracy: ", accuracy, "\n")
cat("Precision: ", precision2, "\n")
cat("Recall: ", recall, "\n")
cat("F1 Score: ", F1, "\n")
```

Figure B: Evaluation Metrics & F1 ROC Cutoff C R Code

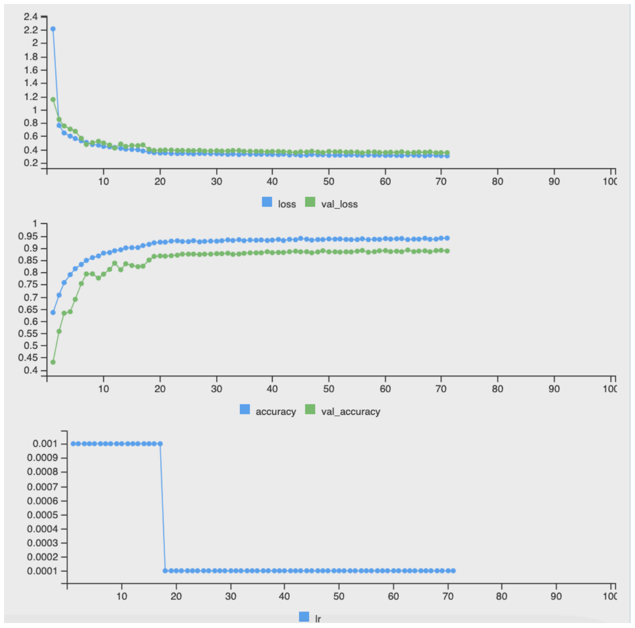


Figure C: Artificial Neural Network Loss Graphs