DSE 6111 - Final Report - ABC CORP.

Nathan Monges

2024-08-19

```
suppressMessages(library(tidyverse))
suppressMessages(library(tidymodels))
```

```
## Warning: package 'modeldata' was built under R version 4.3.3
```

```
suppressMessages(library(caret))
suppressMessages(library(randomForest))
suppressMessages(library(xgboost))
```

```
## Warning: package 'xgboost' was built under R version 4.3.3
```

```
suppressMessages(library(pROC))
suppressMessages(library(kknn))
suppressMessages(library(discrim))
suppressMessages(library(xgboost))
suppressMessages(library(vip))
```

Data Preparation & Preprocessing

```
customer <- read.csv("customer_data.csv")[, -1] #rid of irrelevant clientnum col
customer <- customer %>% mutate_if(is.character, as.factor)
#sum(is.na(customer)) #0 na in data

# split  data into training (75%) and testing (25%) sets
set.seed(1)
data_split <- initial_split(customer, prop = 0.75, strata = "Attrition_Flag")
train <- training(data_split)
test <- testing(data_split)

# define recipe
customer_recipe <- recipe(Attrition_Flag ~ ., data = train) %>%
  step_dummy(all_nominal_predictors()) %>%
  step_center(all_predictors()) %>%
  step_scale(all_predictors())

# setting up cv folds
cv_folds <- vfold_cv(train, v = 10, strata = "Attrition_Flag")
```

Logistic Regression

```
# define log model
log_model <- logistic_reg() %>%
  set_engine("glm")

# define workflow
```

```r
log_workflow <- workflow() %>%
  add_recipe(customer_recipe) %>%
  add_model(log_model)

# fit resamples with cross-validation
set.seed(2)
cv_results <- fit_resamples(log_workflow, resamples = cv_folds, metrics = metric_set(accuracy, roc_auc))

# print metrics
cv_metrics <- collect_metrics(cv_results)
print(cv_metrics)
```

```
## # A tibble: 2 x 6
##   .metric  .estimator  mean     n std_err .config
##   <chr>    <chr>      <dbl> <int>   <dbl> <chr>
## 1 accuracy binary     0.902    10 0.00202 Preprocessor1_Model1
## 2 roc_auc  binary     0.922    10 0.00320 Preprocessor1_Model1
```

```r
# final model on the entire training set
final_log <- fit(log_workflow, data = train)

# predictions on test set
log_predictions <- predict(final_log, new_data = test, type = "prob")

# combine predictions with actual values
test_results <- test %>%
  bind_cols(log_predictions) %>%
  mutate(
    p_attrition = log_predictions$`.pred_Attrited Customer`,
    predicted_class = ifelse(p_attrition >= 0.5, "Attrited Customer", "Existing Customer")
  )

# convert factors to numeric in test predictions
test_results <- test_results %>%
  mutate(
    flag_numeric = as.numeric(Attrition_Flag == "Attrited Customer"),
    predicted_numeric = as.numeric(predicted_class == "Attrited Customer")
  )

# accuracy and ROC AUC
test_accuracy <- mean(test_results$predicted_numeric == test_results$flag_numeric)
roc_result <- roc(test_results$flag_numeric, test_results$p_attrition)
```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```

```r
test_auc <- auc(roc_result)

# prrint results
print(test_accuracy)
```

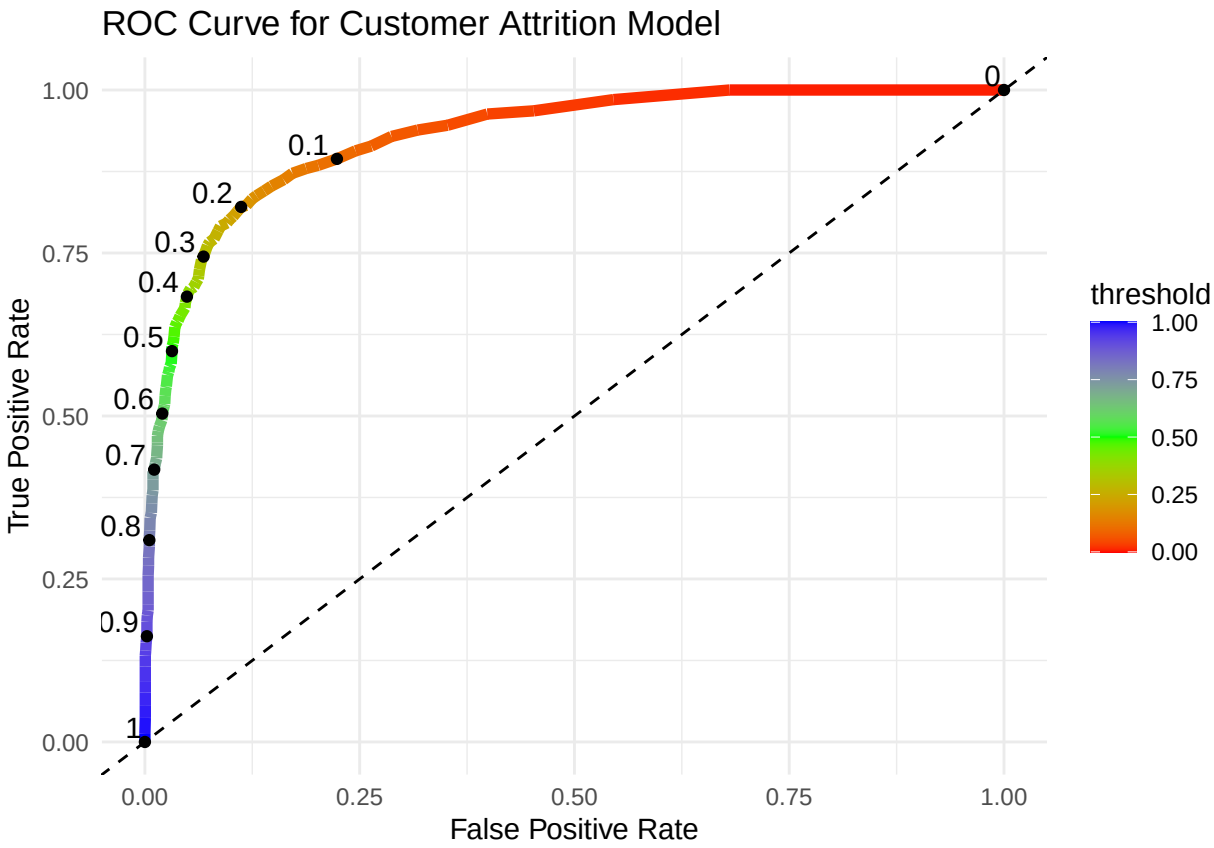
```
## [1] 0.9091627
```

```r
print(test_auc)
```

```
## Area under the curve: 0.9294
```

Logistic Regression ROC Plot

```
roc_data <- data.frame(threshold=seq(1, 0, -0.01), fpr=0, tpr=0)

for (i in roc_data$threshold) {

  # predictions above the threshold
  over_threshold <- test_results[test_results$p_attrition >= i, ]

  # false positive rate
  roc_data[roc_data$threshold==i, "fpr"] <- sum(over_threshold$flag_numeric == 0) / sum(test_results$fla

  # true positive rate
  roc_data[roc_data$threshold==i, "tpr"] <- sum(over_threshold$flag_numeric == 1) / sum(test_results$fla
}

# ROC plot
ggplot() +
  geom_line(data = roc_data, aes(x = fpr, y = tpr, color = threshold), linewidth = 2) +
  scale_color_gradientn(colors = rainbow(3)) +
  geom_abline(intercept = 0, slope = 1, linetype = "dashed") +
  geom_point(data = roc_data[seq(1, 101, 10), ], aes(x = fpr, y = tpr)) +
  geom_text(data = roc_data[seq(1, 101, 10), ],
            aes(x = fpr, y = tpr, label = round(threshold, 2), hjust = 1.2, vjust = -0.2)) +
  labs(title = "ROC Curve for Customer Attrition Model",
       x = "False Positive Rate",
       y = "True Positive Rate") +
  theme_minimal()
```



K - Nearest Neighbors

```
# define kNN model
knn_model <- nearest_neighbor(neighbors = 5) %>%
  set_engine("kknn") %>%
  set_mode("classification")

# define workflow
knn_workflow <- workflow() %>%
  add_recipe(customer_recipe) %>%
  add_model(knn_model)

# fit resamples with cross-validation
set.seed(2)
cv_results_knn <- fit_resamples(knn_workflow, resamples = cv_folds, metrics = metric_set(accuracy, roc_a

# prrint metrics
cv_metrics_knn <- collect_metrics(cv_results_knn)
print(cv_metrics_knn)
```

```
## # A tibble: 2 x 6
##   .metric  .estimator  mean     n std_err .config
##   <chr>    <chr>      <dbl> <int>   <dbl> <chr>
## 1 accuracy binary     0.846    10 0.00412 Preprocessor1_Model1
## 2 roc_auc  binary     0.809    10 0.00685 Preprocessor1_Model1
```

```
# final model on the entire training set
final_knn <- fit(knn_workflow, data = train)
```

```r
# predicttions on  test set
knn_predictions <- predict(final_knn, new_data = test, type = "prob")

# combine predictions with actual values
test_results_knn <- test %>%
  bind_cols(knn_predictions) %>%
  mutate(
    p_attrition = knn_predictions$`.pred_Attrited Customer`,
    predicted_class = ifelse(p_attrition >= 0.5, "Attrited Customer", "Existing Customer")
  )

# converting factors to numeric in test results
test_results_knn <- test_results_knn %>%
  mutate(
    flag_numeric = as.numeric(Attrition_Flag == "Attrited Customer"),
    predicted_numeric = as.numeric(predicted_class == "Attrited Customer")
  )

# accuracy and ROC AUC for kNN
test_accuracy_knn <- mean(test_results_knn$predicted_class == test$Attrition_Flag)
roc_result_knn <- roc(as.numeric(test$Attrition_Flag == "Attrited Customer"), test_results_knn$p_attriti
```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```

```r
test_auc_knn <- auc(roc_result_knn)

# print kNN results
print(test_accuracy_knn)
```

```
## [1] 0.8515008
```

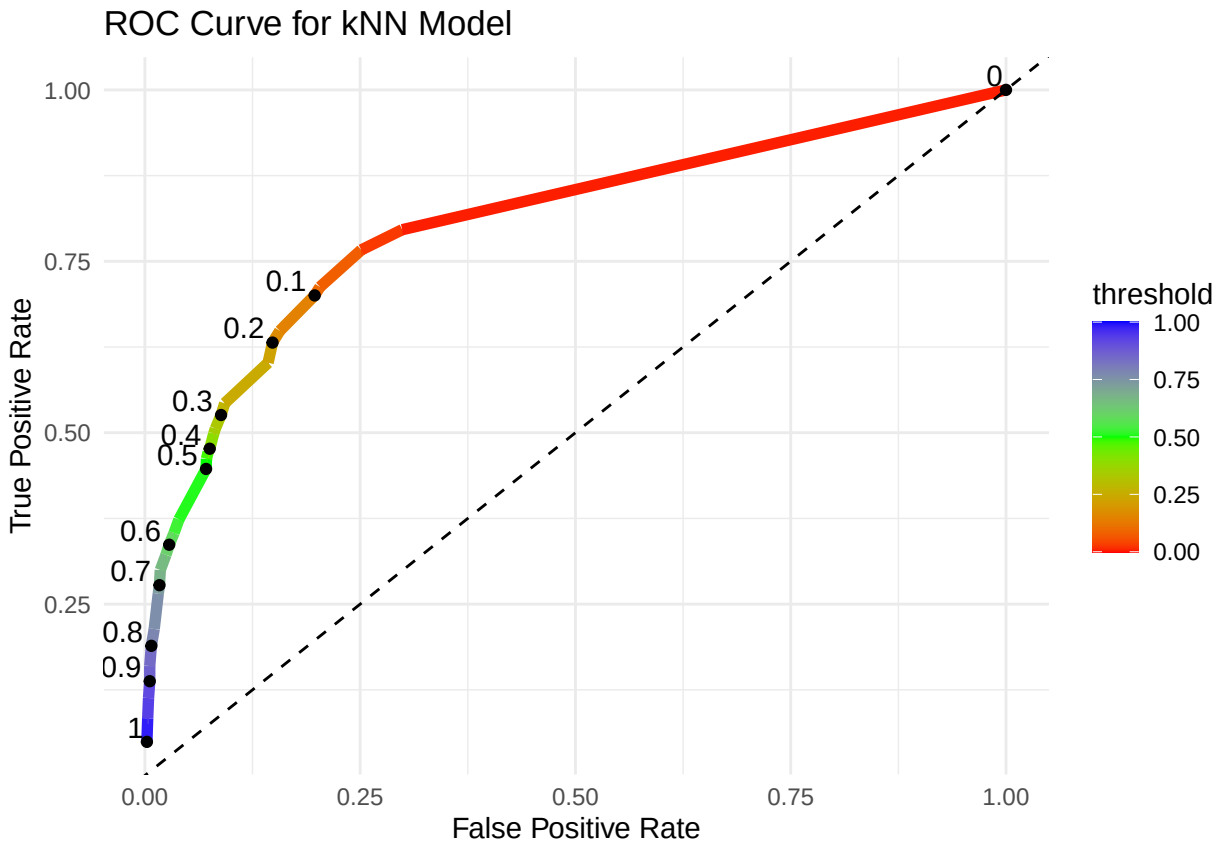
```r
print(test_auc_knn)
```

```
## Area under the curve: 0.8054
```

K - Nearest Neighbors ROC Plot

```r
roc_data_knn <- data.frame(threshold = seq(1, 0, -0.01), fpr = 0, tpr = 0)

for (i in roc_data_knn$threshold) {

  # predictions above the threshold
  over_threshold <- test_results_knn[test_results_knn$p_attrition >= i, ]

  # false positive rate
  roc_data_knn[roc_data_knn$threshold == i, "fpr"] <- sum(over_threshold$flag_numeric == 0) / sum(test_
  # true positive rate
  roc_data_knn[roc_data_knn$threshold == i, "tpr"] <- sum(over_threshold$flag_numeric == 1) / sum(test_
}

# plotting curve
ggplot() +
  geom_line(data = roc_data_knn, aes(x = fpr, y = tpr, color = threshold), linewidth = 2) +
```

```
  scale_color_gradientn(colors = rainbow(3)) +
  geom_abline(intercept = 0, slope = 1, linetype = "dashed") +
  geom_point(data = roc_data_knn[seq(1, 101, 10), ], aes(x = fpr, y = tpr)) +
  geom_text(data = roc_data_knn[seq(1, 101, 10), ],
            aes(x = fpr, y = tpr, label = round(threshold, 2), hjust = 1.2, vjust = -0.2)) +
  labs(title = "ROC Curve for kNN Model",
       x = "False Positive Rate",
       y = "True Positive Rate") +
  theme_minimal()
```



Naive Bayes

```
# define Naive Bayes model
nb_model <- naive_Bayes() %>%
  set_engine("naivebayes") %>%
  set_mode("classification")

# defining workflow
nb_workflow <- workflow() %>%
  add_recipe(customer_recipe) %>%
  add_model(nb_model)

# Fit resamples with cross-validation
set.seed(3)
cv_results_nb <- fit_resamples(nb_workflow, resamples = cv_folds, metrics = metric_set(accuracy, roc_au

# print metrics
```

```r
cv_metrics_nb <- collect_metrics(cv_results_nb)
print(cv_metrics_nb)
```

```
## # A tibble: 2 x 6
##   .metric  .estimator  mean     n std_err .config
##   <chr>    <chr>      <dbl> <int>   <dbl> <chr>
## 1 accuracy binary     0.846    10 0.00125 Preprocessor1_Model1
## 2 roc_auc  binary     0.966    10 0.00116 Preprocessor1_Model1
```

```r
# final model on the entire training set
final_nb <- fit(nb_workflow, data = train)

# prediction on  test set
nb_predictions <- predict(final_nb, new_data = test, type = "prob")

# combine predictions with actual values
test_results_nb <- test %>%
  bind_cols(nb_predictions) %>%
  mutate(
    p_attrition = nb_predictions$`.pred_Attrited Customer`,
    predicted_class = ifelse(p_attrition >= 0.5, "Attrited Customer", "Existing Customer")
  )

#factord to numeric in results
test_results_nb <- test_results_nb %>%
  mutate(
    flag_numeric = as.numeric(Attrition_Flag == "Attrited Customer"),
    predicted_numeric = as.numeric(predicted_class == "Attrited Customer")
  )

# accuracy and ROC AUC for Naive Bayes
test_accuracy_nb <- mean(test_results_nb$predicted_class == test$Attrition_Flag)
roc_result_nb <- roc(as.numeric(test$Attrition_Flag == "Attrited Customer"), test_results_nb$p_attrition
```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```

```r
test_auc_nb <- auc(roc_result_nb)

# Naive Bayes results
print(test_accuracy_nb)
```

```
## [1] 0.8463665
```

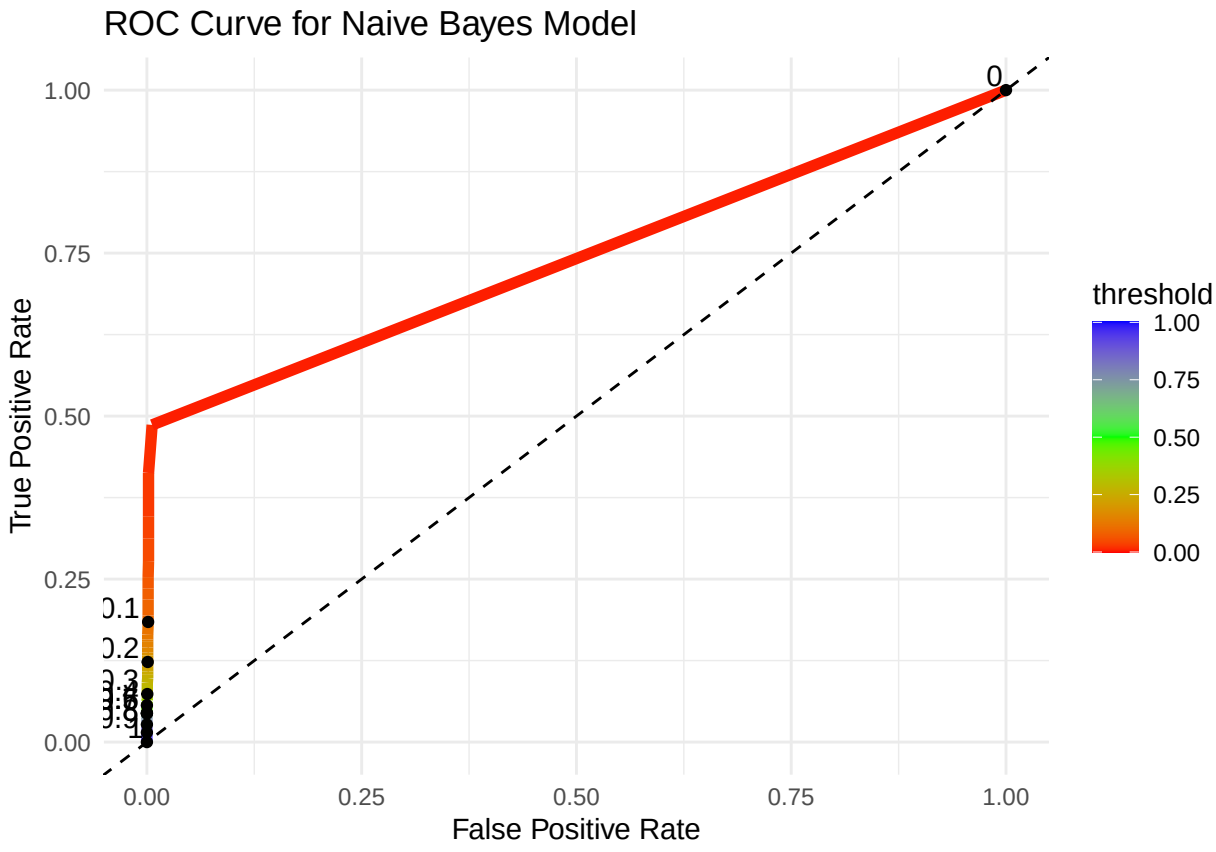
```r
print(test_auc_nb)
```

```
## Area under the curve: 0.9654
```

Naive Bayes ROC Plot

```r
roc_data_nb <- data.frame(threshold = seq(1, 0, -0.01), fpr = 0, tpr = 0)

for (i in roc_data_nb$threshold) {

  # predictions above the threshold
  over_threshold <- test_results_nb[test_results_nb$p_attrition >= i, ]
```

```
  # false positive rate
  roc_data_nb[roc_data_nb$threshold == i, "fpr"] <- sum(over_threshold$flag_numeric == 0) / sum(test_res

  # true positive rate
  roc_data_nb[roc_data_nb$threshold == i, "tpr"] <- sum(over_threshold$flag_numeric == 1) / sum(test_res
}

# Plot ROC Curve
ggplot() +
  geom_line(data = roc_data_nb, aes(x = fpr, y = tpr, color = threshold), linewidth = 2) +
  scale_color_gradientn(colors = rainbow(3)) +
  geom_abline(intercept = 0, slope = 1, linetype = "dashed") +
  geom_point(data = roc_data_nb[seq(1, 101, 10), ], aes(x = fpr, y = tpr)) +
  geom_text(data = roc_data_nb[seq(1, 101, 10), ],
            aes(x = fpr, y = tpr, label = round(threshold, 2), hjust = 1.2, vjust = -0.2)) +
  labs(title = "ROC Curve for Naive Bayes Model",
       x = "False Positive Rate",
       y = "True Positive Rate") +
  theme_minimal()
```



Random Forest

```
mtry_val <- round(sqrt(ncol(train))) # mtry value - using sqrt(p)

# define rf model
rf_model <- rand_forest(mtry = mtry_val, trees = 500) %>%
  set_engine("randomForest") %>%
```

```
  set_mode("classification")

# define workflow
rf_workflow <- workflow() %>%
  add_recipe(customer_recipe) %>%
  add_model(rf_model)

# fit resamples with cross-validation
set.seed(4)
cv_results_rf <- fit_resamples(rf_workflow, resamples = cv_folds, metrics = metric_set(accuracy, roc_au

# print metrics
cv_metrics_rf <- collect_metrics(cv_results_rf)
print(cv_metrics_rf)
```

```
## # A tibble: 2 x 6
##   .metric  .estimator  mean     n std_err .config
##   <chr>    <chr>      <dbl> <int>   <dbl> <chr>
## 1 accuracy binary     0.950    10 0.00181 Preprocessor1_Model1
## 2 roc_auc  binary     0.985    10 0.00159 Preprocessor1_Model1
```

```
# final model on training set
final_rf <- fit(rf_workflow, data = train)

# predictions on the test set
rf_predictions <- predict(final_rf, new_data = test, type = "prob")

# combine predictions with actual values
test_results_rf <- test %>%
  bind_cols(rf_predictions) %>%
  mutate(
    p_attrition = rf_predictions$`.pred_Attrited Customer`,
    predicted_class = ifelse(p_attrition >= 0.5, "Attrited Customer", "Existing Customer")
  )

# convert factors to numeric in results
test_results_rf <- test_results_rf %>%
  mutate(
    flag_numeric = as.numeric(Attrition_Flag == "Attrited Customer"),
    predicted_numeric = as.numeric(predicted_class == "Attrited Customer")
  )

# accuracy and ROC AUC for rf
test_accuracy_rf <- mean(test_results_rf$predicted_class == test$Attrition_Flag)
roc_result_rf <- roc(as.numeric(test$Attrition_Flag == "Attrited Customer"), test_results_rf$p_attritio
```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```

```
test_auc_rf <- auc(roc_result_rf)

# rf results
print(test_accuracy_rf)
```

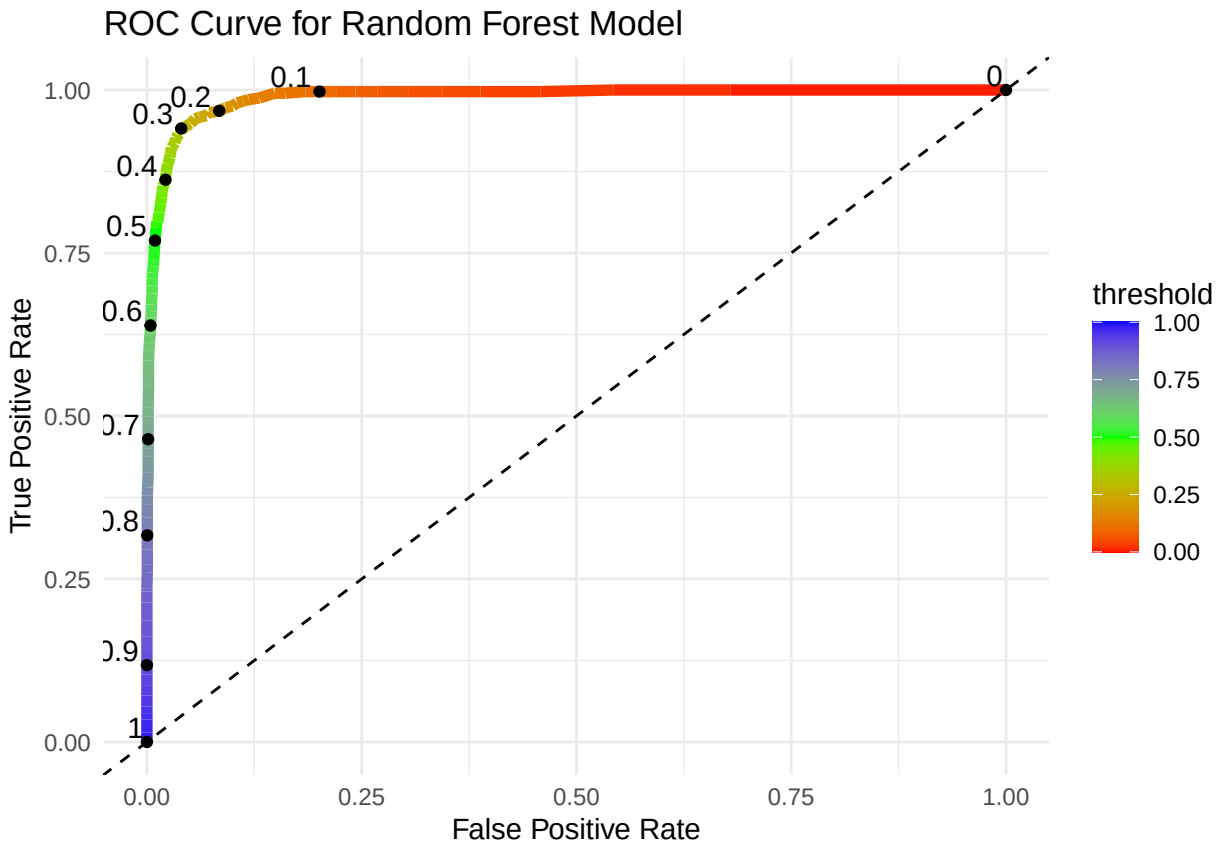
```
## [1] 0.9549763
```

```r
print(test_auc_rf)
```

```
## Area under the curve: 0.9886
```

Random Forest ROC Plot

```r
roc_data_rf <- data.frame(threshold = seq(1, 0, -0.01), fpr = 0, tpr = 0)

for (i in roc_data_rf$threshold) {

  # predictions above the threshold
  over_threshold <- test_results_rf[test_results_rf$p_attrition >= i, ]

  # false positive rate
  roc_data_rf[roc_data_rf$threshold == i, "fpr"] <- sum(over_threshold$flag_numeric == 0) / sum(test_res

  # true positive rate
  roc_data_rf[roc_data_rf$threshold == i, "tpr"] <- sum(over_threshold$flag_numeric == 1) / sum(test_res
}

# Plot ROC Curve for Random Forest
ggplot() +
  geom_line(data = roc_data_rf, aes(x = fpr, y = tpr, color = threshold), linewidth = 2) +
  scale_color_gradientn(colors = rainbow(3)) +
  geom_abline(intercept = 0, slope = 1, linetype = "dashed") +
  geom_point(data = roc_data_rf[seq(1, 101, 10), ], aes(x = fpr, y = tpr)) +
  geom_text(data = roc_data_rf[seq(1, 101, 10), ],
            aes(x = fpr, y = tpr, label = round(threshold, 2), hjust = 1.2, vjust = -0.2)) +
  labs(title = "ROC Curve for Random Forest Model",
       x = "False Positive Rate",
       y = "True Positive Rate") +
  theme_minimal()
```



xgBoost

```
# xgboost model, tunable hyperparameters
xgboost_model <- boost_tree(
  trees = tune(),
  tree_depth = tune(),
  learn_rate = tune()
) %>%
  set_engine("xgboost") %>%
  set_mode("classification")

# define  workflow
xgboost_workflow <- workflow() %>%
  add_recipe(customer_recipe) %>%
  add_model(xgboost_model)

xgboost_grid <- grid_latin_hypercube(
  trees(),
  tree_depth(),
  learn_rate(),
  size = 10  # Number of hyperparameter combinations to try
)

# hyperparameter tuning using cross-validation
set.seed(5)
xgboost_results <- xgboost_workflow %>%
  tune_grid(
    resamples = cv_folds,
```

```
    grid = xgboost_grid,
    control = control_grid(save_pred = TRUE),
    metrics = metric_set(roc_auc)
  )

# best hyperparameter values
best_auc <- xgboost_results %>% select_best(metric = "roc_auc")

# final workflow with the best hyperparameters
final_xgboost_workflow <- finalize_workflow(xgboost_workflow, best_auc)

# train final model on training set
final_xgboost_fit <- fit(final_xgboost_workflow, data = train)

# predictions on test set
xgboost_predictions <- predict(final_xgboost_fit, new_data = test, type = "prob")

# combine predictions with actual values
test_results_xgboost <- test %>%
  bind_cols(xgboost_predictions) %>%
  mutate(
    p_attrition = xgboost_predictions$`.pred_Attrited Customer`,
    predicted_class = ifelse(p_attrition >= 0.5, "Attrited Customer", "Existing Customer")
  )

test_results_xgboost <- test_results_xgboost %>%
  mutate(
    flag_numeric = as.numeric(Attrition_Flag == "Attrited Customer"),
    predicted_numeric = as.numeric(predicted_class == "Attrited Customer")
  )

# accuracy and ROC AUC on  test set
test_accuracy_xgboost <- mean(test_results_xgboost$predicted_class == test$Attrition_Flag)
roc_result_xgboost <- roc(as.numeric(test$Attrition_Flag == "Attrited Customer"), test_results_xgboost$
```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```

```
test_auc_xgboost <- auc(roc_result_xgboost)

# xgboost results
print(test_accuracy_xgboost)
```

```
## [1] 0.9735387
```

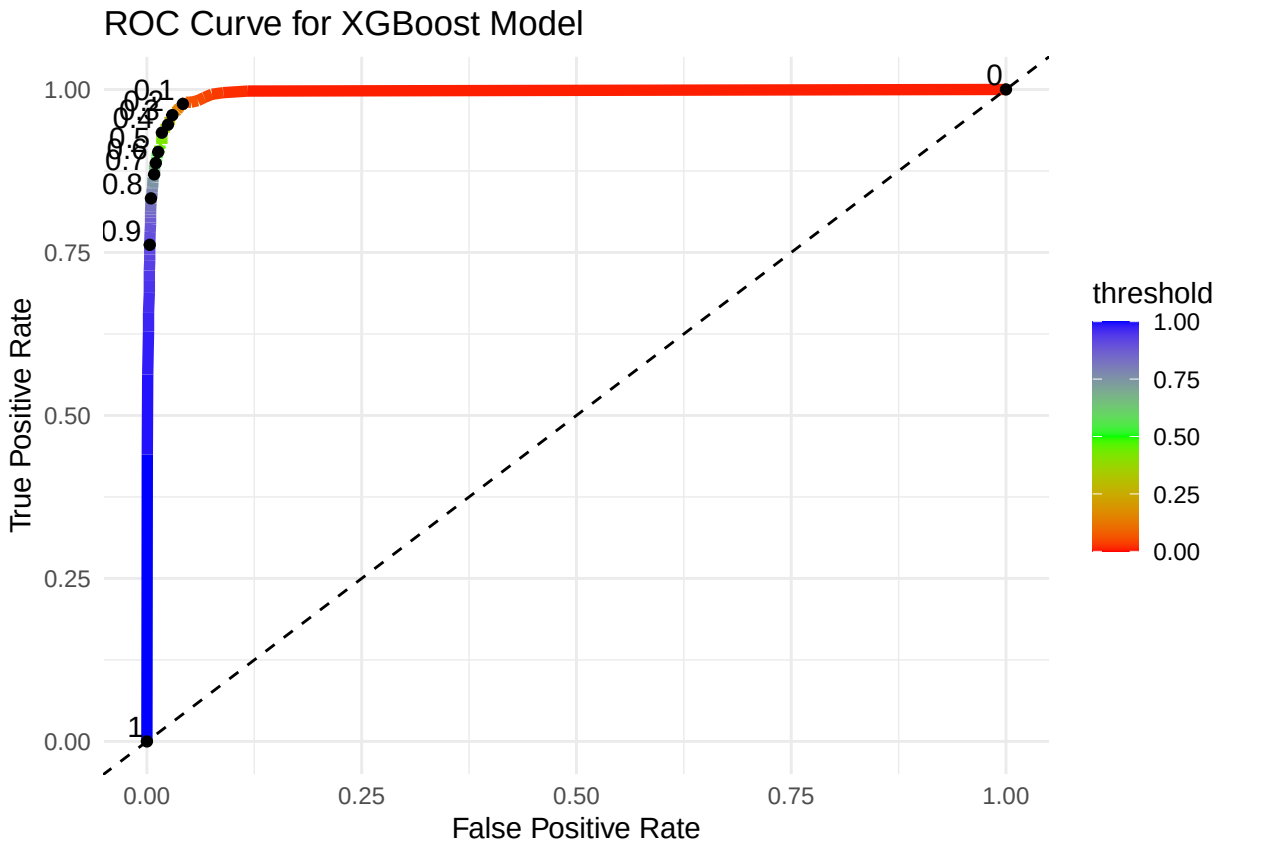
```
print(test_auc_xgboost)
```

```
## Area under the curve: 0.9949
```

xgBoost ROC Plot and Feature Importance

```
roc_data_xgboost <- data.frame(threshold = seq(1, 0, -0.01), fpr = 0, tpr = 0)

for (i in roc_data_xgboost$threshold) {

  # predictions above the threshold
```

```
  over_threshold <- test_results_xgboost[test_results_xgboost$p_attrition >= i, ]

  # false positive rate
  roc_data_xgboost[roc_data_xgboost$threshold == i, "fpr"] <- sum(over_threshold$flag_numeric == 0) / su

  # true positive rate
  roc_data_xgboost[roc_data_xgboost$threshold == i, "tpr"] <- sum(over_threshold$flag_numeric == 1) / su
}

# Plot ROC Curve for XGBoost
ggplot() +
  geom_line(data = roc_data_xgboost, aes(x = fpr, y = tpr, color = threshold), linewidth = 2) +
  scale_color_gradientn(colors = rainbow(3)) +
  geom_abline(intercept = 0, slope = 1, linetype = "dashed") +
  geom_point(data = roc_data_xgboost[seq(1, 101, 10), ], aes(x = fpr, y = tpr)) +
  geom_text(data = roc_data_xgboost[seq(1, 101, 10), ],
            aes(x = fpr, y = tpr, label = round(threshold, 2), hjust = 1.2, vjust = -0.2)) +
  labs(title = "ROC Curve for XGBoost Model",
       x = "False Positive Rate",
       y = "True Positive Rate") +
  theme_minimal()
```




```
xgb_model <- extract_fit_engine(final_xgboost_fit)
importance_matrix <- xgb.importance(model = xgb_model)
```

Executive Summary

ABC Corporation, a leading provider of financial services, has identified customer attrition as a significant challenge impacting its bottom line. Customer attrition represents the loss of clients over time, and it is a critical factor in a solid customer base. ABC Corporation is looking to produce a predictive model capable of identifying customers at high risk of attrition. This model would allow the company to target these customers with retention strategies, such as personalized offers and advertisements, to reduce the likelihood of them leaving.

Objective and Approach

The objective of this project was to build a predictive model that accurately estimates the probability of customer attrition. To do this, several machine learning models were developed and compared to identify the most effective one for this task. The models used include: logistic regression, k-Nearest Neighbors (kNN), Naive Bayes, random forest, and xgBoost. These models were observed based on their accuracy and ability to discriminate between customers likely to stay and those likely to leave, measured by the area under the ROC curve (AUC).

The dataset used for this analysis contained a variety of customer attributes, key features such as customer age, income category, credit limit, total transaction amount, and months on book were identified as potential predictors of attrition. The data was ensuring all features were on a scale for comparison of each other.

Key Findings

The performance of the models varied, with some models showing better results than others. The key findings from the evaluation of these models are: 1. Logistic Regression: This model served performed reasonably well with an accuracy of 90.9% and an AUC of 0.9294. However, it was outperformed by more complex models in terms of both accuracy and discriminative power.

2. k-Nearest Neighbors (kNN): The kNN model, while simple, struggled with the complexity of the data, achieving an accuracy of 85.2% and an AUC of 0.8054. This model was less effective in distinguishing between high-risk and low-risk customers, likely due to its sensitivity to the scaling of features.

3. Naive Bayes: This model provided a good balance between simplicity and performance, with an accuracy of 84.6% and an AUC of 0.9654. Its strength in handling categorical data allowed it to perform better than kNN, but it was still outshone by the ensemble methods.

4. Random Forest: Random Forest, which is an ensemble method, showed considerable promise, achieving an accuracy of 95.5% and an AUC of 0.9886. Its ability to capture the complex interactions in the data between variables makes it one of the top performers in this analysis.

5. xgBoost: The xgBoost model proved to be the best-performing model, with an accuracy of 97.4% and an AUC of 0.9949. Its performance can be attributed to its gradient boosting technique, which helps improve the model by focusing on the hardest-to-predict cases. The model's flexibility make it highly suitable for predicting customer attrition.

Recommendations

Based on the model evaluations, the xgBoost model is recommended as the final model for predicting customer attrition at ABC Corporation. Its high accuracy and AUC indicate that it is = effective at identifying customers likely to leave and also provides reliable probability estimates, which can be directly used to prioritize retention efforts.

To implement this model, ABC Corporation should begin to integrate the xgBoost model in there analysis of attrition risk. This will enable monitoring of customer attrition risk, allowing the company to respond with targeted interventions.

Moreover, a feature importance analysis of the xgBoost model identified key drivers of customer attrition, such as Total_Trans_Amt, Credit_Limit, and Months_on_book. ABC Corporation should consider these factors when designing retention strategies, focusing on customers with high transaction amounts but short tenure, as these customers may be more prone to leaving.

Conclusion

The development of a robust predictive model for customer attrition has provided ABC Corporation with a tool to enhance customer retention. By using the xgBoost model, the company can better understand the factors influencing attrition and take steps to reduce it. This approach will not only help retain valuable customers but also strengthen ABC Corporation's market position and long-term profitability.