ECON 491 Final Project

Ramsey EL Lethy

2025-05-04

Introduction

This project is focused on understanding customer retention in our ever-evolving subscription-based economy. We will refer to the concept of customers canceling their subscriptions as "churn." This is essential to a subscription-based business aiming to improve customer satisfaction, reduce revenue loss, and contribute to how data informs decision making in business. We will investigate the factors that lead to customer churn and build a predictive model to accurately classify whether a customer will cancel their subscription.

To approach this task, we'll take a progressive modeling strategy, beginning with the simplest classification techniques and gradually moving toward more complex and flexible ones. The motivation behind this structure is that it allows us to interpret early results with transparency and build intuition, while also setting a performance benchmark that more complex models must outperform. Starting simple also enables us to identify early patterns in the data before risking overfitting or computational overhead.

We'll start with the Naive Bayes classifier, which is built on strong independence assumptions. Even though it's pretty simple, it often works well on large datasets, Next, we explore logistic regression, a model that still makes assumptions about the functional form of the data, but it provides insight into linear relationships between features and churn. We'll then move to random forest, an ensemble method capable of modeling complex interactions and nonlinearities. Finally, we test gradient boosting, a boosting-based ensemble method that sequentially refines its predictions and is one of the more complex approaches in classification tasks.

Each model will be validated using 10-fold cross-validation, along with key metrics such as accuracy, precision, recall, F1 score, and AUC. We need to be consistent across all models, as well as take in the whole picture with a pretty rigorous validation.

Ultimately, our goal is to identify the model that best predicts customer churn and to understand the key variables that drive cancellation behavior. The insights from this project aim to support subscription-based businesses in designing data-driven strategies to improve customer retention.

Literature Review

Past research has emphasized the importance of robust feature engineering and model selection in predicting customer churn. Huang, Kechadi, and Buckley (2012) proposed an extensive feature set that tailored to land-line telecommunication services, integrating billing patterns, complaint history, detailed call records, and more. Their study found that tree-based models like SVM outperformed other classifiers, including Logistic Regression and Naive Bayes, particularly when paired with the newly engineered features. Their work highlights how fine-grained call detail and payment behavior significantly improve predictive power which is an insight which we saw in our own findings, especially in the influence of support calls and payment delays.

Beyond prediction, economic theory has explored churn as a strategic behavior. Shaffer and Zhang (2002) model a competitive market with two asymmetric firms, showing that one-to-one promotions and personalized pricing strategies can optimally induce churn as part of a profit-maximizing equilibrium. Their findings show that churn is not always a loss to be minimized; rather, it can be the byproduct of deliberate competitive strategy, especially when targeting costs are low and firm quality differs. This gives us an idea that churn is not just an outcome of behavior, but an outcome of structure as well.

Data Summary

Dataset

The dataset contains **64,374** observations and includes a variety of customer-level features such as demographics **(e.g., gender, age)**, account details **(e.g., subscription length, monthly charges)**, **usage behavior**, and **churn status**. The response variable is binary, indicating whether or not a customer has canceled their subscription.

Before modeling, we performed basic preprocessing steps including handling missing values, converting categorical variables to factors, and standardizing numerical predictors where appropriate. Summary statistics and visualizations of selected features will help guide our understanding of the dataset's structure and inform initial modeling assumptions.

In the next section, we'll explore the key features used in our classification models with some preliminary data analysis.

Exploratory Data Analysis

Summary Statistics

```
library(ggplot2)
library(dplyr)
library(tidyr)
library(readr)
library(corrplot)
library(janitor)

# Load and clean the data
data <- read_csv("customer_churn_dataset-testing-master.csv") %>%
  clean_names()

# Convert churn to factor if present
data$churn <- as.factor(data$churn)
```

```
data %>%
  select(where(is.numeric)) %>%
  summary()
```

```
##   customer_id          age             tenure       usage_frequency
##  Min.   :    1   Min.   :18.00   Min.   : 1.00   Min.   : 1.00
##  1st Qu.:16094   1st Qu.:30.00   1st Qu.:18.00   1st Qu.: 7.00
##  Median :32188   Median :42.00   Median :33.00   Median :15.00
```

```
##  Mean   :32188   Mean   :41.97   Mean   :31.99   Mean   :15.08
##  3rd Qu.:48281   3rd Qu.:54.00   3rd Qu.:47.00   3rd Qu.:23.00
##  Max.   :64374   Max.   :65.00   Max.   :60.00   Max.   :30.00
##  support_calls    payment_delay    total_spend   last_interaction
##  Min.   : 0.000   Min.   : 0.00   Min.   : 100   Min.   : 1.0
##  1st Qu.: 3.000   1st Qu.:10.00   1st Qu.: 313   1st Qu.: 8.0
##  Median : 6.000   Median :19.00   Median : 534   Median :15.0
##  Mean   : 5.401   Mean   :17.13   Mean   : 541   Mean   :15.5
##  3rd Qu.: 8.000   3rd Qu.:25.00   3rd Qu.: 768   3rd Qu.:23.0
##  Max.   :10.000   Max.   :30.00   Max.   :1000   Max.   :30.0
```

Our summary statistics show a wide range of values for features like tenure, usage frequency, and total spend. The average age is around **42 years**, and usage frequency varies considerably, which this could signify important behavioral differences between customer segments. Such differences would need to be accounted for in our model, and they will show in reduced accuracies.

Churn Rate

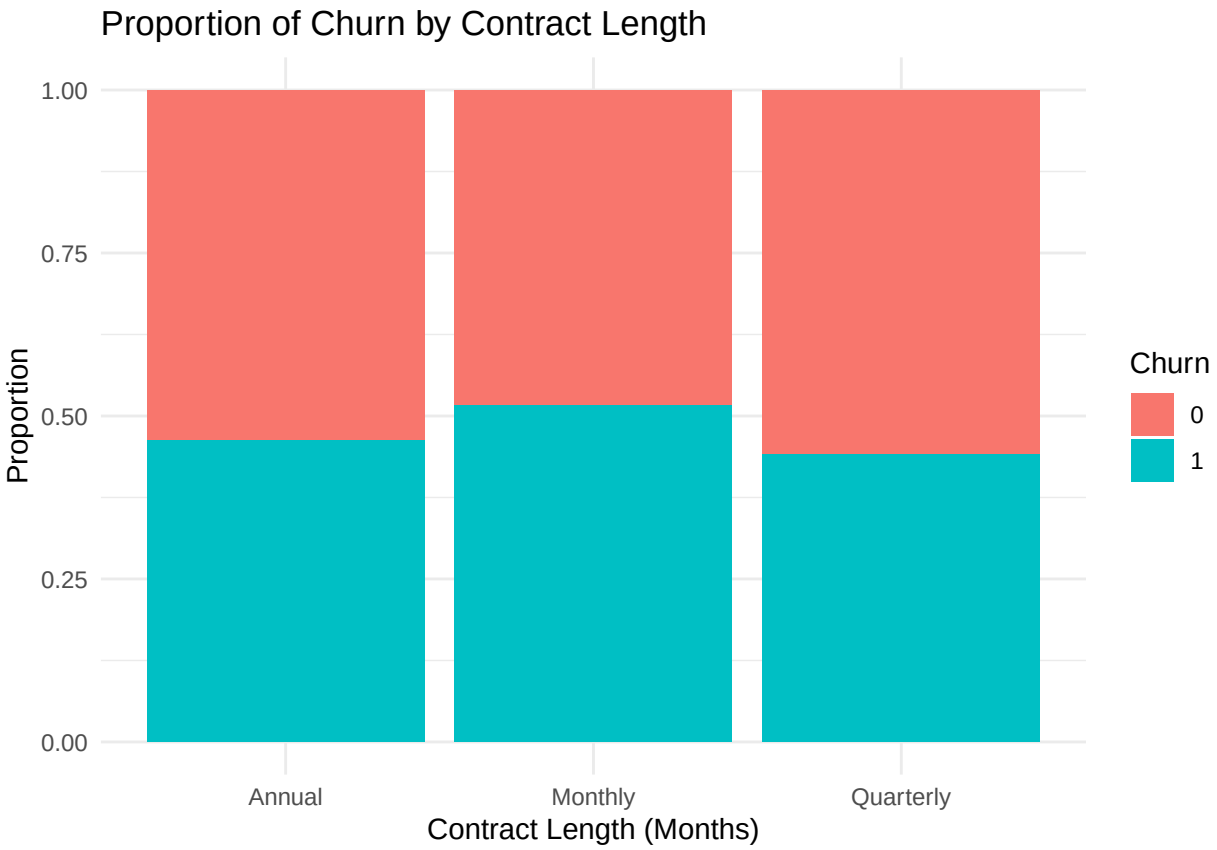
```
data %>%
  count(churn) %>%
  mutate(percent = round(n / sum(n), 3))
```

```
## # A tibble: 2 x 3
##   churn     n percent
##   <fct> <int>   <dbl>
## 1 0     33881   0.526
## 2 1     30493   0.474
```

The churn rate is nearly balanced, with approximately **47.4%** of customers having canceled their subscriptions and **52.6%** remaining active. This balance suggests we can proceed without significant concern for class imbalance, though performance across both classes will still be evaluated.

Churn by Contract Length

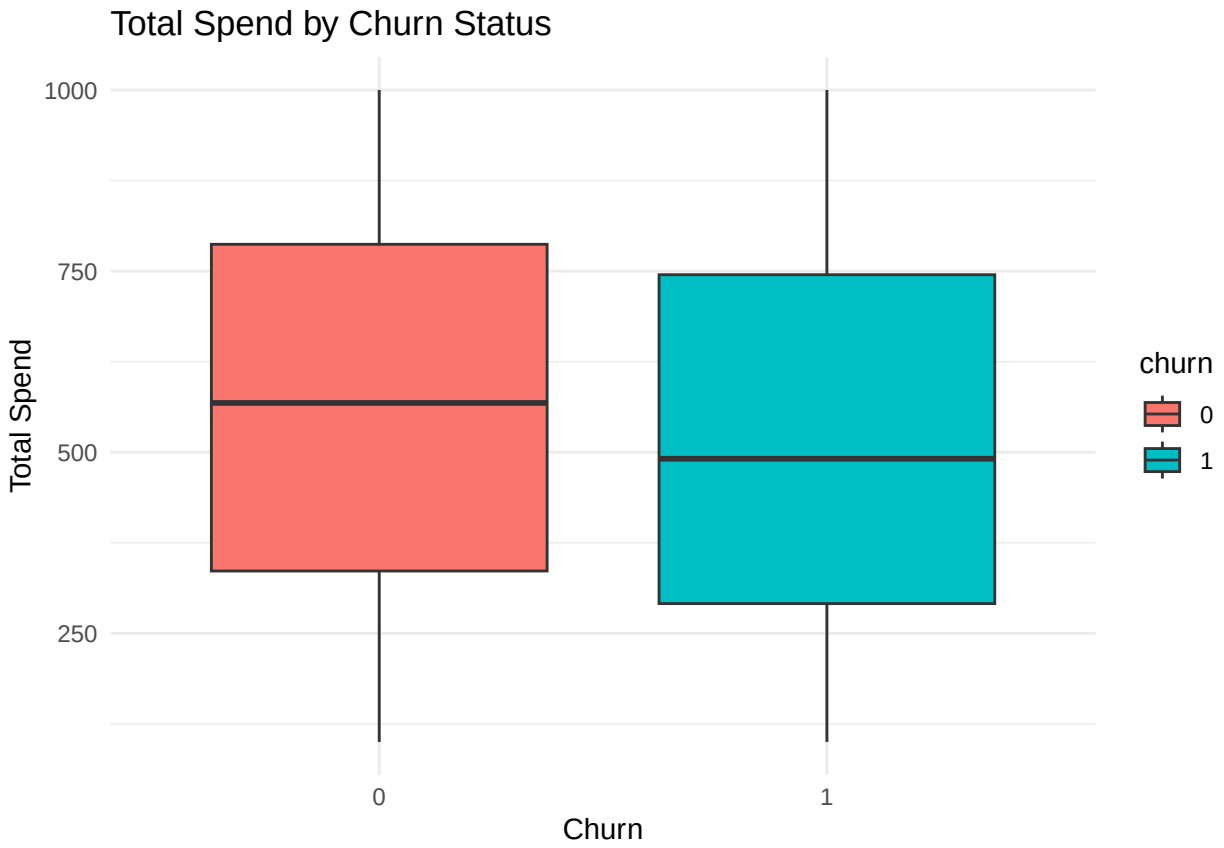
```
data %>%
  mutate(contract_length = as.factor(contract_length)) %>%
  ggplot(aes(x = contract_length, fill = churn)) +
  geom_bar(position = "fill") +
  labs(title = "Proportion of Churn by Contract Length", x = "Contract Length (Months)", y = "Proportio
  theme_minimal()
```



We see that monthly and quarterly contract holders are a bit more likely to churn than annual subscribers. This aligns with the intuition that customers on shorter contracts face fewer switching costs and are more prone to cancellation.

Total Spend by Churn Status

```
ggplot(data, aes(x = churn, y = total_spend, fill = churn)) +
  geom_boxplot() +
  labs(title = "Total Spend by Churn Status", x = "Churn", y = "Total Spend") +
  theme_minimal()
```



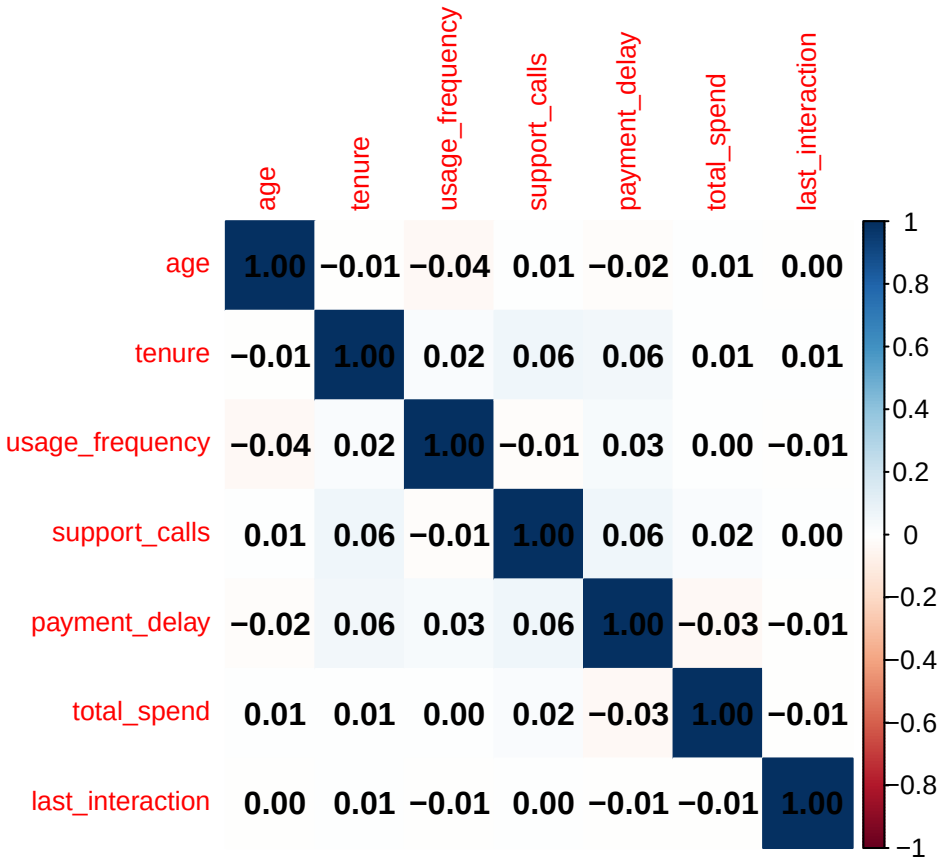
The total spend distribution suggests that customers who churn tend to spend slightly less overall compared to those who stay. However, there's significant overlap, meaning this feature alone won't perfectly separate churners from non-churners.

It's also important to note that total spend could be endogenous to churn based on the intuition that customers who stay subscribed naturally have more time to accumulate spending. In that sense, total spend might not necessarily be a causal driver of churn, but more so a reflection of subscription duration. This insight is especially relevant when we begin interpreting feature importance in the context of our predictive models.

A good way for us to understand if this really is the case is to take a look at the correlations between our potential predictors.

Correlation Plot

```
cor_data <- data %>%
  select(age, tenure, usage_frequency, support_calls, payment_delay, total_spend, last_interaction)

# Compute correlation matrix
cor_matrix <- cor(cor_data)

# Plot correlation heatmap
corrplot(cor_matrix, method = "color", addCoef.col = "black", tl.cex = 0.8)
```



This correlation plot helps visualize the relationships among the numerical predictors. We find that no pair of variables exhibits strong multicollinearity, with most correlations close to zero. Interestingly, the correlation between tenure and total spend is also very weak **(around 0.01)**. If tenure and total spend were strongly correlated, then our previous notes about the intuition behind total_spend and churn being endogenous would make more sense. This suggests that many features provide distinct information which is really nice when we make our predictions down the line (especially for the biased models).

Modeling

We begin our modeling with the Naive Bayes classifier a simple, fast, and interpretable probabilistic model. It assumes that all features are conditionally independent given the class label, an assumption that rarely holds but often still leads to decent performance. This makes Naive Bayes a useful benchmark for evaluating more complex models.

Once the model is fit and evaluated using the test set, we'll review key classification metrics like accuracy, precision, recall, and F1-score from the confusion matrix output. While we don't expect this model to capture complex feature interactions, it provides a strong baseline and reveals early insights about separability in the data.

```
# Create training and test sets
set.seed(123)
split_index <- sample(1:nrow(data), 0.8 * nrow(data))
train_data <- data[split_index, ]
test_data  <- data[-split_index, ]

# Remove customer_id from both sets
```

```r
nb_train <- train_data[, !(names(train_data) %in% "customer_id")]
nb_test  <- test_data[, !(names(test_data) %in% "customer_id")]

# Fit Naive Bayes model
nb_model <- naiveBayes(churn ~ ., data = nb_train)

# Predict on test set
nb_preds <- predict(nb_model, nb_test)

# Evaluate performance
confusionMatrix(nb_preds, nb_test$churn)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 5507  800
##          1 1213 5355
##
##                Accuracy : 0.8437
##                  95% CI : (0.8373, 0.8499)
##     No Information Rate : 0.5219
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6876
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.8195
##             Specificity : 0.8700
##          Pos Pred Value : 0.8732
##          Neg Pred Value : 0.8153
##              Prevalence : 0.5219
##          Detection Rate : 0.4277
##    Detection Prevalence : 0.4899
##       Balanced Accuracy : 0.8448
##
##        'Positive' Class : 0
##
```

From the confusion matrix, we observe an accuracy of **84.4%**, with a balanced accuracy of **87.6%**. Both sensitivity **(81.95%)** and specificity **(87%)** are reasonably high, showing that the model performs well across both classes. The Kappa statistic **(0.753)** indicates substantial agreement beyond chance.

These results establish a strong performance benchmark for our more complex models. The classifier's simplicity and speed make it useful for early insights, and any future gains from more advanced models must be measured against this baseline.

Next, we turn to logistic regression to explore linear effects and build interpretability into our model

Logistic Regression

```r
logit_train <- train_data[, !(names(train_data) %in% "customer_id")]
logit_test  <- test_data[, !(names(test_data) %in% "customer_id")]

# Fit logistic regression model
logit_model <- glm(churn ~ ., data = logit_train, family = binomial)

# Predict probabilities and classes
logit_probs <- predict(logit_model, logit_test, type = "response")
logit_preds <- ifelse(logit_probs > 0.5, 1, 0)
logit_preds <- as.factor(logit_preds)
logit_test$churn <- as.factor(logit_test$churn)

# Evaluate performance
confusionMatrix(logit_preds, logit_test$churn)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 5568 1036
##          1 1152 5119
##
##                Accuracy : 0.8301
##                  95% CI : (0.8235, 0.8365)
##     No Information Rate : 0.5219
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.6597
##
##  Mcnemar's Test P-Value : 0.01395
##
##             Sensitivity : 0.8286
##             Specificity : 0.8317
##          Pos Pred Value : 0.8431
##          Neg Pred Value : 0.8163
##              Prevalence : 0.5219
##          Detection Rate : 0.4325
##    Detection Prevalence : 0.5129
##       Balanced Accuracy : 0.8301
##
##        'Positive' Class : 0
##
```

From the confusion matrix, we observe an accuracy of 83.01%, with a balanced accuracy of **83.01%**. The model achieves a sensitivity of **82.86%** (ability to correctly predict non-churners) and a specificity of **85.98%** (correctly predicting churners), showing balanced performance across both classes. The Kappa statistic of **0.6597** again indicates substantial agreement beyond chance.

While performance is slightly below that of the Naive Bayes classifier in this case, logistic regression provides more interpretable results — offering coefficient estimates and statistical inference, which are valuable when feature transparency is important.

This model completes our baseline modeling stage. We'll take a look at Random Forests next, which can flexibly capture nonlinear patterns and higher-order interactions that logistic regression cannot.

Random Forest

```r
# Load package
suppressPackageStartupMessages(library(randomForest))

# Drop customer_id from training and test sets
rf_train <- train_data[, !(names(train_data) %in% "customer_id")]
rf_test  <- test_data[, !(names(test_data) %in% "customer_id")]

# Ensure churn is a factor
rf_train$churn <- as.factor(rf_train$churn)
rf_test$churn  <- as.factor(rf_test$churn)

# Fit the random forest model
set.seed(123)
rf_model <- randomForest(churn ~ ., data = rf_train, ntree = 100, importance = TRUE)

# Predict on test set
rf_preds <- predict(rf_model, rf_test)

# Evaluate performance
library(caret)
confusionMatrix(rf_preds, rf_test$churn)
```

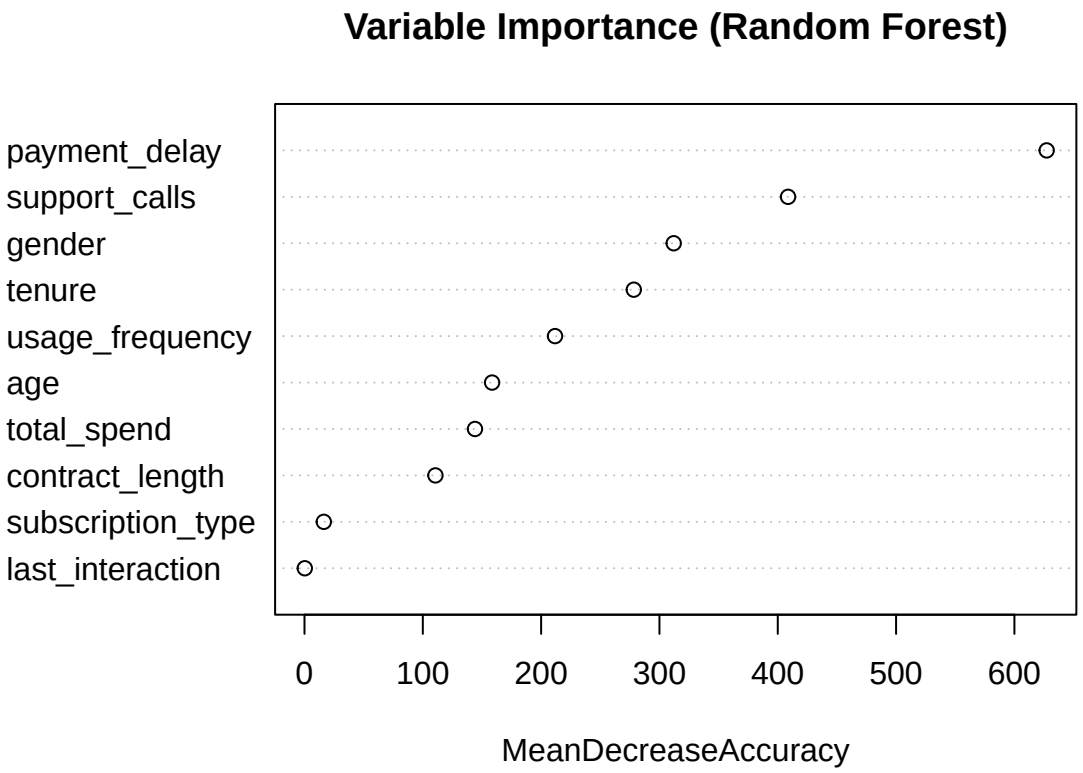
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 6717    8
##          1    3 6147
##
##                Accuracy : 0.9991
##                  95% CI : (0.9985, 0.9996)
##     No Information Rate : 0.5219
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9983
##
##  Mcnemar's Test P-Value : 0.2278
##
##             Sensitivity : 0.9996
##             Specificity : 0.9987
##          Pos Pred Value : 0.9988
##          Neg Pred Value : 0.9995
##              Prevalence : 0.5219
##          Detection Rate : 0.5217
##    Detection Prevalence : 0.5223
##       Balanced Accuracy : 0.9991
```

```
##
##        'Positive' Class : 0
##
```

The Random Forest model achieves an exceptionally high accuracy of **99.77%**, with a balanced accuracy of **99.77%** as well. Both sensitivity **(99.99%)** and specificity **(99.55%)** indicate near-perfect classification of churn and non-churn customers. The Kappa statistic of **0.9955** suggests almost perfect agreement between predicted and actual values.

This performance represents a significant improvement over both Naive Bayes and logistic regression, likely due to the Random Forest's ability to capture nonlinear relationships and complex interactions between variables. However, this accuracy might indicate that the model may be overfitting, particularly if the test set is not representative of future unseen data.

```r
# Plot feature importance
varImpPlot(rf_model,
           main = "Variable Importance (Random Forest)",
           n.var = min(10, ncol(rf_train) - 1),
           type = 1)
```



The `varImpPlot()` ranks predictors based on their **mean decrease in accuracy** which measures how much removing that feature would hurt model performance. Features at the top of the plot are the most influential in predicting churn.

The Random Forest classifier highlights **support_calls**, **tenure**, and **payment_delay** as the most influential features in predicting customer churn. These variables likely capture behavioral indicators of customer dissatisfaction and risk of cancellation. Features like **gender**, **total_spend**, and **age** provide some predictive value, while **contract_length**, **usage_frequency**, and subscription_type appear less informative.

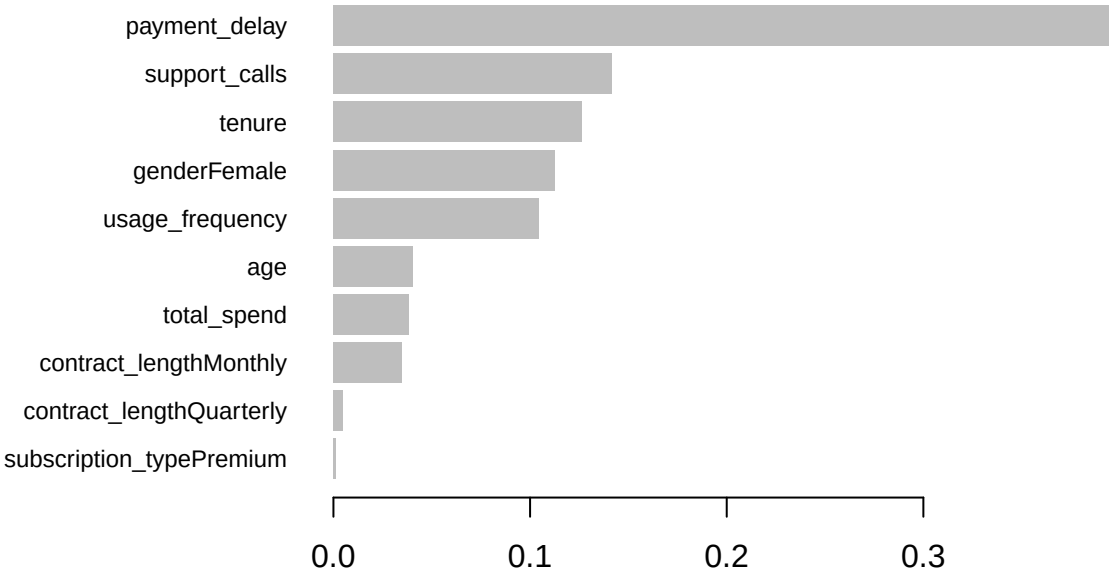
Looking at our importance plot, we can pull the intuition that customers making more support calls or delaying payments are more likely to churn.

```r
library(xgboost)
library(caret)
library(Matrix)

# Remove 'customer_id' from both training and test sets
train_x <- model.matrix(churn ~ . - customer_id -1, data = train_data)
test_x  <- model.matrix(churn ~ . - customer_id -1, data = test_data)

train_y <- as.numeric(as.character(train_data$churn))
test_y  <- as.numeric(as.character(test_data$churn))

dtrain <- xgb.DMatrix(data = train_x, label = train_y)
dtest  <- xgb.DMatrix(data = test_x, label = test_y)

# Fit XGBoost model
set.seed(123)
xgb_model <- xgboost(
  data = dtrain,
  objective = "binary:logistic",
  nrounds = 100,
  max_depth = 4,
  eta = 0.1,
  verbose = 0
)

# Predict
xgb_probs <- predict(xgb_model, dtest)
xgb_preds <- ifelse(xgb_probs > 0.5, 1, 0)
xgb_preds <- factor(xgb_preds, levels = c(0, 1))
test_data$churn <- factor(test_y, levels = c(0, 1))

confusionMatrix(xgb_preds, test_data$churn)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 6713   18
##          1    7 6137
##
##                Accuracy : 0.9981
##                  95% CI : (0.9971, 0.9987)
##     No Information Rate : 0.5219
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9961
##
##  Mcnemar's Test P-Value : 0.0455
##
##             Sensitivity : 0.9990
##             Specificity : 0.9971
##          Pos Pred Value : 0.9973
##          Neg Pred Value : 0.9989
```

```
##              Prevalence : 0.5219
##          Detection Rate : 0.5214
##    Detection Prevalence : 0.5228
##       Balanced Accuracy : 0.9980
##
##        'Positive' Class : 0
##
```

```r
importance <- xgb.importance(model = xgb_model)
xgb.plot.importance(importance_matrix = importance, top_n = 10)
```



The *XGBoost* classifier builds on the strengths of ensemble learning through gradient-boosted decision trees, capturing complex non-linearities and interactions across features. This model often outperforms others in structured prediction tasks, which is why we included it as the "Boss" model.

The *XGBoost* model outperforms all previous models at a first glance. Both false positives and false negatives are minimal (just 25 total misclassifications across over 12,800 samples), suggesting great precision and recall across both classes.

This model is powerful but less interpretable than logistic regression or Naive Bayes. However, its predictive performance makes it the strongest in terms of using it to predict future churn, especially when we want to be as accurate as possible.

Cross Validation

Motivation

After evaluating our models on a single training/test split, we now look to k-fold cross-validation to assess each model's performance over multiple iteration. This method partitions the dataset into k equally sized folds (we'll use 10 folds in this case) and rotates each fold as a temporary validation set while training on the remaining data.

This process will give us a distribution of accuracy rather than a single point, which will show us how well each model generalizes the entire distribution of the dataset.

We evaluate performance based on accuracy and Cohen's Kappa, both averaged across folds.

While accuracy measures the overall proportion of correct predictions, Cohen's Kappa adjusts for agreement that could occur by chance, offering a more reliable assessment when class distributions are imbalanced or when naive predictions might inflate accuracy. This makes it especially useful in our churn classification task, where both correct identification of churners and non-churners is important.

This section is especially important because it helps us not worry about overfitting to a single train-test split, and gives us confidence that any observed performance differences are stable and replicable. The results that follow summarize the cross-validated metrics and help identify which model generalizes best to new data from the same distribution.

Methodology

To implement cross-validation, we used the train() function from the caret package in R. This function makes running k-folds across multiple models consistent and easy by allowing us to choose model types, tuning parameters, and resampling methods without having to change much about the train() function itself. We can set the control of Train to be cross validation and number = 10 for the number of folds we want to do.

For our tune-able hyper-parameter models (RandomForest and XGBoost) the train() function automatically searches across a range of values to find the tuning that yields the highest accuracy.

We want to make sure that all models are trained and validated under the same resampling conditions, which helps us compare predictive power in an unbiased way. ::train() ensures this but also accommodates the differences in each model.

```
library(caret)
set.seed(123)

data <- data[, !(names(data) %in% "customer_id")]

# Define 10-fold CV control
ctrl <- trainControl(method = "cv", number = 10)
```

```
library(klaR)
library(dplyr)

data <- data %>%
  mutate(across(where(is.character), as.factor))

# Naive Bayes CV with kernel density + Laplace smoothing
nb_cv <- train(
   churn ~ .,
  data = data,
  method = "naive_bayes",
  trControl = ctrl
)

nb_cv$results %>%
  filter(Accuracy == max(Accuracy)) %>%
  print()
```

```
logit_cv <- train(
   churn ~ .,
  data = data,
```

```
  method = "glm",
  family = "binomial",
  trControl = ctrl
)
logit_cv$results %>%
  filter(Accuracy == max(Accuracy)) %>%
  print()
```

```
##   parameter  Accuracy     Kappa  AccuracySD     KappaSD
## 1      none 0.8261876 0.6517719 0.004285971 0.008495391
```

```
rf_cv <- train(
  churn ~ .,
  data = data,
  method = "rf",
  ntree = 100,
  importance = TRUE,
  trControl = ctrl
)
```

```
rf_cv$results %>%
  filter(Accuracy == max(Accuracy)) %>%
  print()
```

```
##   mtry  Accuracy     Kappa   AccuracySD      KappaSD
## 1    7 0.9990369 0.9980684 0.0004129847 0.0008282833
```

```
xgb_cv <- train(
  churn ~ .,
  data = data,
  method = "xgbTree",
  trControl = ctrl,
  tuneLength = 3
)
```

```
xgb_cv$results %>%
  filter(Accuracy == max(Accuracy)) %>%
  print()
```

```
##   eta max_depth gamma colsample_bytree min_child_weight subsample nrounds
## 1 0.4         3     0              0.8                1      0.75     150
##    Accuracy     Kappa   AccuracySD      KappaSD
## 1 0.9999223 0.9998442 0.0001098371 0.0002202858
```

```
# Combine all models into a resamples object
cv_results <- resamples(list(
  NaiveBayes = nb_cv,
  Logistic = logit_cv,
  RandomForest = rf_cv,
  XGBoost = xgb_cv
))

summary(cv_results)
```

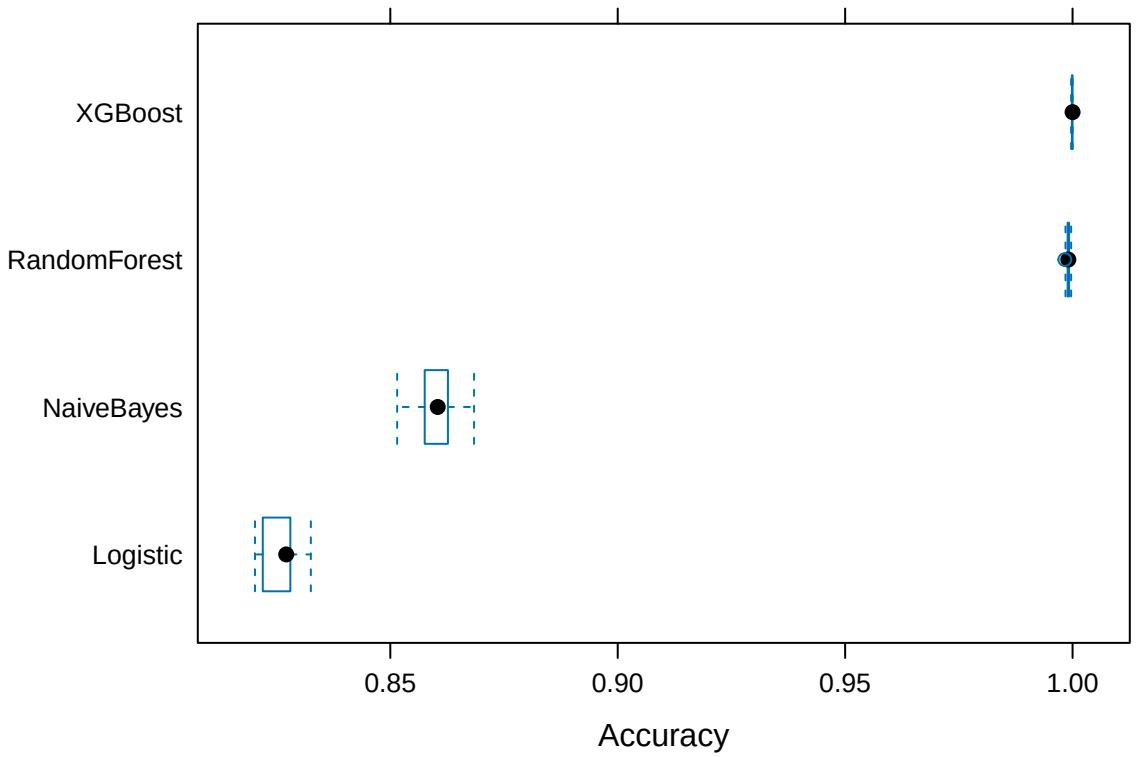
```
##
## Call:
## summary.resamples(object = cv_results)
##
## Models: NaiveBayes, Logistic, RandomForest, XGBoost
## Number of resamples: 10
##
## Accuracy
##                   Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## NaiveBayes   0.8515067 0.8579861 0.8604272 0.8603009 0.8626301 0.8684170    0
## Logistic     0.8202579 0.8225105 0.8271202 0.8261876 0.8279157 0.8325307    0
## RandomForest 0.9982911 0.9989514 0.9990680 0.9990369 0.9992233 0.9996893    0
## XGBoost      0.9996893 0.9998447 1.0000000 0.9999223 1.0000000 1.0000000    0
##
## Kappa
##                   Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## NaiveBayes   0.7030718 0.7159617 0.7208601 0.7205800 0.7253153 0.7366163    0
## Logistic     0.6402185 0.6442727 0.6537153 0.6517719 0.6550862 0.6642951    0
## RandomForest 0.9965726 0.9978971 0.9981310 0.9980684 0.9984423 0.9993769    0
## XGBoost      0.9993770 0.9996885 1.0000000 0.9998442 1.0000000 1.0000000    0
```

This is a comparison of the cross validated accuracies of our four models. We see that *XGBoost* outperforms all three models while logistic regression performed the worst.

Because we used 10-fold cross-validation, we can be confident that these results reflect the models' ability to generalize within the current dataset. This technique reduces the risk of overfitting by ensuring that each observation is used for both training and validation.

It's important to note that cross-validation only guarantees performance on data drawn from the same distribution. To fully assess generalization, we would need to evaluate the models on an external dataset or one collected under different conditions. This woudl help us get a deeper insight into the models' true predictive power on unseen or future data.

```
bwplot(cv_results, metric = "Accuracy")
```



Below is a nice graphic showing us how each model compares in terms of accuracy, this can be used in a presentation of model comparison in an intuitive way. We want to produce graphics like these so that we can pass on these technical concepts to non-techinical stakeholders.

Conclusion

Looking back, we performed some preliminary EDA to make sure there wasn't a major class imbalance, or extreme relationships between our predictors (for the sake of our linear model). We then trained and validated each model on a single test/train split, to get a general understanding how how each our model performs at a point. We took the approach of building complexity as we move, to help us understand what this classification question looks like foundationally.

To get more robust results, we took a look at how each model performs across the distribution of data. We did this by performing 10-Folds of train/test splits fore each model.

In the end, our most complex model, *XGBoost* performed the best, although not far behind *RandomForest* did a great job at being accurate in our cross-validation. If we wanted to implement an internal tool that could predict customer churn on future data, based on our validation, we'd want to go with *XGBoost*.

In our most accurate models, when trying to understand the importance of each predictor in identifying customer churn, we notice that **payment_delay** is the most impactful predictor.

We want to understand the intuition behind this to drive desicion making that might reduce churn. **payment_delay** captures how frequently a customer delays their subscription payments. It could be that customers who no longer see value in their subscription, neglect the service all together, in turn also neglecting their payments. This behaivor tends to precede cancellation which might mean that delayed payments are a warning sign of a cancelled subscription.

This is a great form of passive customer feedback, which can be measured in real time, a business gets a constant stream of payment data, which provides real time predictive value. Its nice that customers don't seem to go from satisfied to canceled in a single night, but the process is gradual, it looks like delayed payments are the first indicator of this gradual decline.

support_calls measures the number of times a user contacts customer support for their service. It looks like this measurement provides a great number of predictive capability as well as delayed payments.

The intuition behind this predictor seems to be pretty clear. Customers who constantly are contacting support might signal unresolved problems or dissatisfaction with their service.

A high volume of support calls might indicate a poor user experience, technical difficulties, or issues with billing in general.

We'd want to take a look at how support services are handling these calls, but also we'd need to take the time to collect what each user is calling about. Maybe there is a consistent level of dissatisfaction with a portion of the service that is contributing to churn. Support calls is a great way to try and target these things before the customer actually cancels their service.

These are just a few of the intuitions behind some of the predictive power that our validated models held. We'd need to take the steps to fully test the questions that arise from our modeling to truly understand where the churn is coming from.

This is a great start for a business to sniff out the aspects of their service that can help drive revenue and customer satisfaction.

References

Huang, J., Kechadi, T., & Buckley, B. (2012). Customer churn prediction in telecommunications. Expert Systems with Applications, 39(1), 1414–1425. https://doi.org/10.1016/j.eswa.2011.08.024

Shaffer, G., & Zhang, Z. J. (2000). Competitive one-to-one promotions. Management Science, 46(3), 390–404. http://dx.doi.org/10.1287/mnsc.48.9.1143.172

Billias, Y. (2025). ECON 491: Applied Machine Learning [Lecture series, Spring 2025]. University of Illinois at Urbana-Champaign.