

# Predict Customer Churn in Streaming Service Model

Toan Nguyen

01 July 2025

## Contents

Introduction . . . . .	1
Library setup (Assume all packages are pre-installed) . . . . .	2
Data loading . . . . .	2
Building decision tree models . . . . .	4
Parameter Tuning . . . . .	5
Check and verify which values are more suitable for each parameter . . . . .	6
Tree 1: Using default parameters . . . . .	6
Tree 2: Limit the maximum depth of tree . . . . .	7
Tree 3: Complex tree with many splits . . . . .	9
Tree 4: Pruned tree with the best complexity parameter . . . . .	10
Model Performance Summary . . . . .	11
Performance Metric Definitions . . . . .	15
Model comparison . . . . .	15
Recommendation . . . . .	16

## Introduction

In the context of predicting customer churn for our subscription-based digital streaming service, a decision tree model was chosen for its ability to balance accuracy with interpretability. Our dataset includes a diverse mix of variables such as subscription type, preferred device, payment frequency, customer demographics, and engagement metrics like total watch time and number of support tickets. A decision tree is particularly well-suited for handling this combination of numerical and categorical data with minimal preprocessing.

More importantly, the model generates easily understandable rules that allow us to pinpoint the conditions under which customers are most likely to cancel like low engagement combined with basic-tier subscriptions or high support ticket volumes. This level of transparency is essential for business decision-making, as it enables marketing, product, and support teams to take targeted action based on clear churn indicators. While more advanced ensemble models may offer marginal improvements in predictive accuracy, the decision tree provides a solid foundation for identifying actionable patterns and building a churn mitigation strategy grounded in data-driven insights.

## Library setup (Assume all packages are pre-installed)

```
library(readr)
library(caret)
```

```
## Loading required package: ggplot2
```

```
## Loading required package: lattice
```

```
library(rpart)
library(rattle)
```

```
## Loading required package: tibble
```

```
## Loading required package: bitops
```

```
## Rattle: A free graphical interface for data science with R.
## Version 5.5.1 Copyright (c) 2006-2021 Togaware Pty Ltd.
## Type 'rattle()' to shake, rattle, and roll your data.
```

```
library(pROC)
```

```
## Type 'citation("pROC")' for a citation.
```

```
##
```

```
## Attaching package: 'pROC'
```

```
## The following objects are masked from 'package:stats':
```

```
##
```

```
##      cov, smooth, var
```

```
library(dplyr)
```

```
##
```

```
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
```

```
##
```

```
##      filter, lag
```

```
## The following objects are masked from 'package:base':
```

```
##
```

```
##      intersect, setdiff, setequal, union
```

## Data loading

Load the csv file and take a look at first 5 rows

```
streaming_data <- read_csv('streaming_service_cleaned.csv')
```

```
## Rows: 2375 Columns: 19
## -- Column specification -----
## Delimiter: ","
## chr  (10): Name, Gender, SubscriptionType, PaymentMethod, Email, Country, Re...
## dbl  (6): CustomerID, Age, MonthlyFee, TotalWatchTime, NumSupportTickets, L...
## lgl  (1): Cancelled
## dtm   (2): JoinDate, LastLoginDate
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
head(streaming_data)
```

```
## # A tibble: 6 x 19
##   CustomerID Name      Age Gender JoinDate      SubscriptionType MonthlyFee
##   <dbl> <chr>   <dbl> <chr>   <dtm>         <chr>              <dbl>
## 1         1 David~    19 Male  2022-06-29 00:00:00 Standard            16.4
## 2         2 Emily~    53 Non-b~ 2021-12-15 00:00:00 Standard            15.9
## 3         3 Alex ~    75 Male  2023-09-11 00:00:00 Premium             18.7
## 4         4 Emily~    58 Female 2019-06-15 00:00:00 Basic              10.7
## 5         5 Jane ~    27 Male  2022-03-17 00:00:00 Basic               8.98
## 6         6 David~    58 Male  2023-04-01 00:00:00 Basic               9.72
## # i 12 more variables: PaymentMethod <chr>, TotalWatchTime <dbl>,
## #   Cancelled <lgl>, Email <chr>, Country <chr>, Region <chr>,
## #   LastLoginDate <dtm>, NumSupportTickets <dbl>, LoyaltyPoints <dbl>,
## #   ReferralSource <chr>, PreferredDevice <chr>, PaymentFrequency <chr>
```

```
# Check number of rows
```

```
nrow(streaming_data)
```

```
## [1] 2375
```

Select features for predicting (based on previous data exploratory performance) with target variable “Cancelled”

```
# Filter to use only the specified variables
streaming_data_filtered <- streaming_data %>% select(
  Region,
  Country,
  PreferredDevice,
  Gender,
  SubscriptionType,
  PaymentFrequency,
  PaymentMethod,
  Age,
  NumSupportTickets,
  ReferralSource,
  Cancelled
)
```

Make sure all variables have appropriate types

```
# Convert variables to appropriate types
streaming_data_filtered$Region <- as.factor(streaming_data_filtered$Region)
streaming_data_filtered$Country <- as.factor(streaming_data_filtered$Country)
streaming_data_filtered$PreferredDevice <-
  ↳ as.factor(streaming_data_filtered$PreferredDevice)
streaming_data_filtered$Gender <- as.factor(streaming_data_filtered$Gender)
streaming_data_filtered$SubscriptionType <-
  ↳ as.factor(streaming_data_filtered$SubscriptionType)
streaming_data_filtered$PaymentFrequency <-
  ↳ as.factor(streaming_data_filtered$PaymentFrequency)
streaming_data_filtered$PaymentMethod <- as.factor(streaming_data_filtered$PaymentMethod)
streaming_data_filtered$Age <- as.integer(streaming_data_filtered$Age)
streaming_data_filtered$NumSupportTickets <-
  ↳ as.integer(streaming_data_filtered$NumSupportTickets)
streaming_data_filtered$ReferralSource <-
  ↳ as.factor(streaming_data_filtered$ReferralSource)
streaming_data_filtered$Cancelled <- as.factor(streaming_data_filtered$Cancelled)

# Basic streaming_data_filtered summary
summary(streaming_data_filtered)
```

```
##      Region      Country PreferredDevice      Gender
## North   :260  Australia:385 Desktop: 684   Female   :1103
## Region9 :254   Canada   :380 Mobile :1455   Male     :1033
## Region7 :249   France   :368 Tablet  : 236   Non-binary : 114
## Region10:244   Germany :414                Prefer not to say: 125
## South   :240    UK       :408
## Central :236    USA      :420
## (Other) :892
## SubscriptionType PaymentFrequency PaymentMethod      Age
## Basic   :967   Annual   : 243 Bank Transfer: 246 Min.    :18.00
## Premium :450   Monthly  :1662 Credit Card  :1134 1st Qu.:34.00
## Standard:958   Quarterly: 470 Debit Card   : 513 Median :49.00
##                                     Paypal      : 482 Mean   :49.33
##                                     3rd Qu.:64.00
##                                     Max.    :80.00
##
## NumSupportTickets ReferralSource Cancelled
## Min.    :0.000    Ad           :481 FALSE:1692
## 1st Qu.:1.000    Friend        :522 TRUE : 683
## Median :1.000    OrganicSearch:238
## Mean    :1.689    SocialMedia  :447
## 3rd Qu.:2.000    Website      :687
## Max.    :6.000
##
```

## Building decision tree models

As seen in the summary, the target variable “Cancelled” consist of total 2375 samples, with 29% of TRUE and 71% of FALSE.

This dataset is slightly imbalanced, therefore a 70/30 split with stratify can help with evaluating the model more reliably while keeping the class distribution.

```
set.seed(42) # for reproducibility

# Create stratified split (70% training, 30% testing)
train_index <- createDataPartition(streaming_data_filtered$Cancelled,
                                   p = 0.7,
                                   list = FALSE)

train_data <- streaming_data_filtered[train_index, ]
test_data  <- streaming_data_filtered[-train_index, ]

# Check class distribution
print("Train data distribution:")
```

```
## [1] "Train data distribution:"
```

```
prop.table(table(train_data$Cancelled))
```

```
##
##      FALSE      TRUE
## 0.7121394 0.2878606
```

```
print("Test data distribution:")
```

```
## [1] "Test data distribution:"
```

```
prop.table(table(test_data$Cancelled))
```

```
##
##      FALSE      TRUE
## 0.7130802 0.2869198
```

The next step is to fit the decision tree.

## Parameter Tuning

There are lots of parameters, however these are the most relevant:

Parameters:

Parameter	Description
maxdepth	Maximum depth of any node of the final tree (limits complexity).
minsplit	Minimum number of observations required to attempt a split.
cp	Complexity parameter: smaller values allow more splits. Used for pruning and controlling overfitting.
xval	number of cross-validations.

## Check and verify which values are more suitable for each parameter

```
for (cp_val in c(0.01, 0.005, 0.001)) {  
  for (minsplit_val in c(10, 15, 20)) {  
    model <- rpart(Cancelled ~ ., data = train_data, method = "class",  
                  control = rpart.control(cp = cp_val, minsplit = minsplit_val, maxdepth  
                    = 10))  
    pred <- predict(model, newdata = test_data, type = "class")  
    acc <- mean(pred == test_data$Cancelled)  
    cat("cp =", cp_val, "minsplit =", minsplit_val, "Accuracy =", acc, "\n")  
  }  
}
```

```
## cp = 0.01 minsplit = 10 Accuracy = 0.7130802  
## cp = 0.01 minsplit = 15 Accuracy = 0.7130802  
## cp = 0.01 minsplit = 20 Accuracy = 0.7130802  
## cp = 0.005 minsplit = 10 Accuracy = 0.7130802  
## cp = 0.005 minsplit = 15 Accuracy = 0.7130802  
## cp = 0.005 minsplit = 20 Accuracy = 0.7130802  
## cp = 0.001 minsplit = 10 Accuracy = 0.6202532  
## cp = 0.001 minsplit = 15 Accuracy = 0.6469761  
## cp = 0.001 minsplit = 20 Accuracy = 0.6540084
```

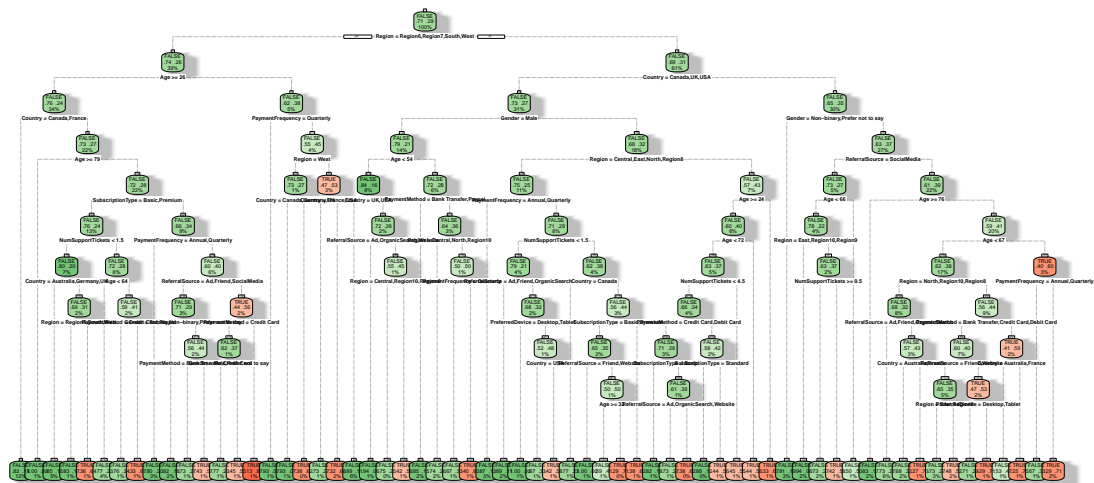
Models with  $cp = 0.01$  and  $cp = 0.005$ , regardless of the  $minsplit$  value, all resulted in an accuracy of 71.3%. However, this matches the proportion of the majority class ( $Cancelled = FALSE$ ), indicating that the trees were over-pruned and likely made no meaningful splits - simply defaulting to predicting the most common outcome. In contrast, lowering the complexity parameter to  $cp = 0.001$  allowed the tree to start forming actual branches. Among these, the model with  $minsplit = 20$  achieved the best accuracy at 65.4%, which, although lower than the previous models, reflects a genuine attempt to classify both TRUE and FALSE cases of cancellation. This suggests that it is the first model to move beyond the baseline and begin learning from the features in the data.

## Tree 1: Using default parameters

Using  $cp = 0.001$ ,  $minsplit = 20$ ,  $maxdepth = 10$

```
tree1 <- rpart(  
  Cancelled ~ .,  
  data = train_data,  
  method = "class",  
  control = rpart.control(  
    maxdepth = 10,  
    minsplit = 20,  
    cp = 0.001  
  )  
)  
fancyRpartPlot(  
  tree1,  
  palettes = c("Greens", "Reds"),  
  sub = "Default Parameters"  
)
```

```
## Warning: labs do not fit even at cex 0.15, there may be some overplotting
```



Default Parameters

Figure 1: Decision Tree 1

Export the tree model as an image so we can view clearer.

```
# Save as large PNG
png("tree_1.png", width = 5000, height = 1800, res = 200)
fancyRpartPlot(tree1, palettes = c("Greens", "Reds"), sub = "Pruned Tree", cex = 0.6)
dev.off()
```

```
## pdf
## 2
```

## Tree 2: Limit the maximum depth of tree

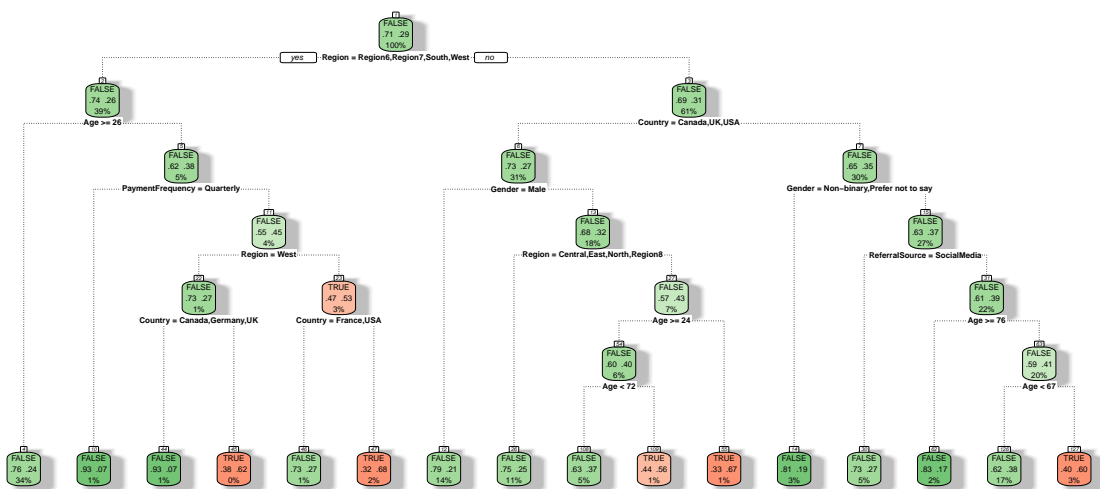
Using  $cp = 0.001$ ,  $minsplit = 20$ ,  $maxdepth = 6$

```
tree2 <- rpart(
  Cancelled ~ .,
  data = train_data,
  method = "class",
```

```

control = rpart.control(
  maxdepth = 6,
  minsplit = 20,
  cp = 0.001
)
fancyRpartPlot(
  tree2,
  palettes = c("Greens", "Reds"),
  sub = "Default Parameters"
)

```



Default Parameters

Figure 2: Decision Tree 2

Export the tree model as an image so we can view clearer.

```

# Save as large PNG
png("tree_2.png", width = 5000, height = 1800, res = 200)
fancyRpartPlot(tree1, palettes = c("Greens", "Reds"), sub = "Pruned Tree", cex = 0.6)
dev.off()

```

```

## pdf
## 2

```

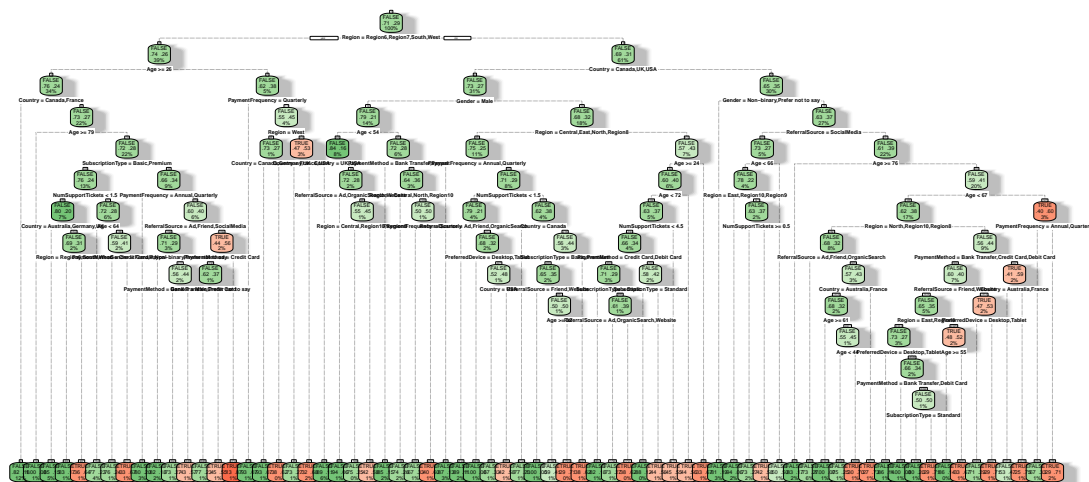


Using `cp = 0.001`, `minsplit = 20`, `maxdepth = 30`

```
tree3 <- rpart(
  Cancelled ~ .,
  data = train_data,
  method = "class",
  control = rpart.control(
    maxdepth = 30,
    minsplit = 20,
    cp = 0.001
  )
)

rfpartPlot(
  tree3,
  palettes = c("Greens", "Reds"),
  sub = "Default Parameters"
)
```

```
## Warning: labs do not fit even at cex 0.15, there may be some overplotting
```



## Default Parameters

Figure 3: Decision Tree 3

Export the tree model as an image so we can view clearer.

```
# Save as large PNG
png("tree_3.png", width = 5000, height = 1800, res = 200)
fancyRpartPlot(tree1, palettes = c("Greens", "Reds"), sub = "Pruned Tree", cex = 0.6)
dev.off()
```

```
## pdf
## 2
```

## Tree 4: Pruned tree with the best complexity parameter

Using  $cp = 0$

Expected impact: Simpler tree with balance performances

```
# Build the full decision tree with the specified complexity parameter
c.tree.full <- rpart(
  Cancelled ~ .,
  data = train_data,
  method = "class",
  control = rpart.control(cp = 0)
)

# Prune the tree using the specified CP value
tree4 <- prune(c.tree.full, cp = 0)

# Visualize the pruned tree
fancyRpartPlot(tree4, palettes = c("Greens", "Reds"), sub = "")
```

```
## Warning: labs do not fit even at cex 0.15, there may be some overplotting
```



```

model <- get(paste0("tree", i))

# Class predictions
pred_class <- predict(model, test_data, type = "class")
pred_class <- factor(pred_class, levels = c("FALSE", "TRUE"))

# Probabilistic predictions for AUC
pred_prob <- predict(model, test_data, type = "prob")[, "TRUE"]

# Confusion matrix with accuracy CIs
cm <- confusionMatrix(
  data      = pred_class,
  reference = true_labels,
  positive  = "TRUE"
)

# Extract key metrics
accuracy    <- cm$overall["Accuracy"]
acc_lower   <- cm$overall["AccuracyLower"]
acc_upper   <- cm$overall["AccuracyUpper"]
precision   <- cm$byClass["Precision"]
recall      <- cm$byClass["Recall"]
f1          <- cm$byClass["F1"]

# ROC and AUC
roc_obj     <- pROC::roc(response = true_labels, predictor = pred_prob)
roc_list[[paste0("Tree ", i)]] <- roc_obj
auc_value   <- pROC::auc(roc_obj)

# Report
cat("\n===== \n")
cat("Tree", i, "Performance: \n")
cat("===== \n")
print(cm$table)
cat(sprintf(
  "Accuracy      : %.3f (95%% CI: %.3f-%.3f) \n",
  accuracy, acc_lower, acc_upper
))
cat(sprintf("Misclass. err : %.3f \n", 1 - accuracy))
cat(sprintf("Precision   : %.3f \n", precision))
cat(sprintf("Recall      : %.3f \n", recall))
cat(sprintf("F1-Score    : %.3f \n", f1))
cat(sprintf("AUC         : %.3f \n", auc_value))
}

```

```
## Setting levels: control = FALSE, case = TRUE
```

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

```
##
## =====
## Tree 1 Performance:
## =====
```

```
##           Reference
## Prediction FALSE TRUE
##      FALSE   422  161
##      TRUE     85   43
## Accuracy      : 0.654 (95% CI: 0.618-0.689)
## Misclass. err : 0.346
## Precision     : 0.336
## Recall        : 0.211
## F1-Score      : 0.259
## AUC           : 0.546
```

```
## Setting levels: control = FALSE, case = TRUE
## Setting direction: controls < cases
```

```
##
## =====
## Tree 2 Performance:
## =====
##           Reference
## Prediction FALSE TRUE
##      FALSE   478  193
##      TRUE     29   11
## Accuracy      : 0.688 (95% CI: 0.652-0.722)
## Misclass. err : 0.312
## Precision     : 0.275
## Recall        : 0.054
## F1-Score      : 0.090
## AUC           : 0.546
```

```
## Setting levels: control = FALSE, case = TRUE
## Setting direction: controls < cases
```

```
##
## =====
## Tree 3 Performance:
## =====
##           Reference
## Prediction FALSE TRUE
##      FALSE   418  161
##      TRUE     89   43
## Accuracy      : 0.648 (95% CI: 0.612-0.684)
## Misclass. err : 0.352
## Precision     : 0.326
## Recall        : 0.211
## F1-Score      : 0.256
## AUC           : 0.534
```

```
## Setting levels: control = FALSE, case = TRUE
## Setting direction: controls < cases
```

```
##
## =====
```

```
## Tree 4 Performance:
## =====
##           Reference
## Prediction FALSE TRUE
##      FALSE   410  158
##      TRUE     97   46
## Accuracy      : 0.641 (95% CI: 0.605-0.677)
## Misclass. err : 0.359
## Precision     : 0.322
## Recall        : 0.225
## F1-Score      : 0.265
## AUC           : 0.502

# Set up the plotting area with 2x2 grid
par(mfrow = c(2, 2))

# Color palette for the plots
cols <- c("#1f77b4", "#ff7f0e", "#2ca02c", "#d62728")

# Loop through the ROC list and plot each
for (i in 1:4) {
  plot(
    roc_list[[i]],
    col = cols[i],
    main = paste("Tree Model", i),
    lwd = 2,
    xlim = c(1, 0),
    ylim = c(0, 1)
  )

  # Calculate and display the AUC value
  auc_value <- auc(roc_list[[i]])
  text(
    x = 0.4, y = 0.15,
    labels = paste("AUC:", round(auc_value, 3)),
    col = cols[i],
    cex = 1.2,
    font = 2
  )
}
}
```

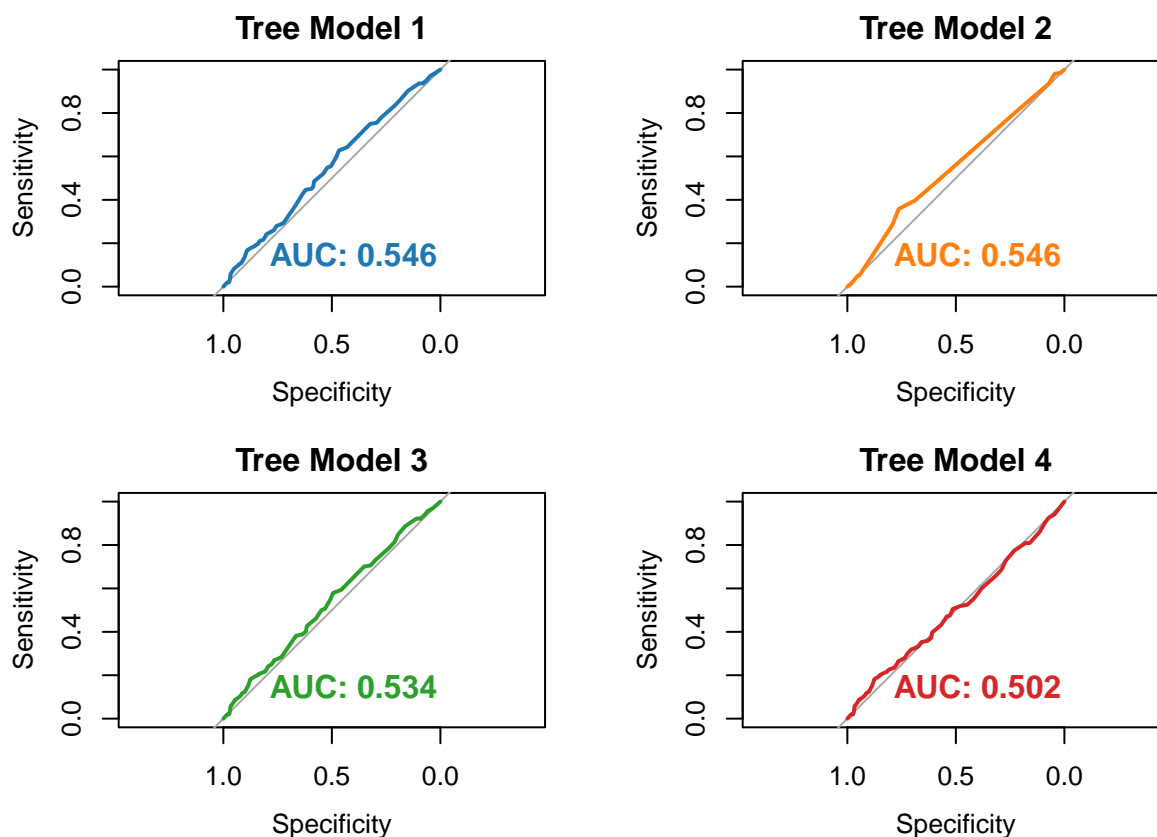


Figure 5: AUC comparison

## Performance Metric Definitions

Performance Metric	Definition
Accuracy	How many predictions were correct overall.
Precision	Out of all predicted churners, how many actually churned.
Recall	Out of all actual churners, how many we correctly predicted.
F1-Score	A balance between precision and recall.
AUC	How well the model separates churners from non-churners.

## Model comparison

In evaluating the decision tree models for predicting customer churn, we compared performance across multiple key metrics, including accuracy, precision, recall, F1-score, and AUC, all of which are crucial in understanding how well each model can help the business proactively retain customers.

Tree 2 achieved the highest accuracy (68.8%), with a 95% Confidence Interval (CI) of [65.2%, 72.2%], suggesting that it correctly predicts churn and non-churn outcomes more often than the other models. However, accuracy alone isn't

enough when dealing with customer churn, because a high accuracy can be misleading if the model fails to identify the customers most likely to churn. Tree 2 has a very low recall (5.4%), which means it misses a significant portion of actual churned customers. In the context of customer churn, recall is critical because it tells us how many of the customers at risk of leaving we are able to identify. Missing churned customers reduces our ability to take action and retain them. Thus, despite Tree 2's high accuracy, its low recall makes it unsuitable for proactively targeting customers who are likely to churn.

Tree 1, with an accuracy of 65.4% and a 95% CI of [61.8%, 68.9%], performs better than Tree 2 in terms of recall (21.1%). This indicates that it captures more of the churned customers compared to Tree 2, though still not enough. Precision (33.6%) in Tree 1 is also higher than Tree 2, meaning it is less likely to incorrectly flag non-churned customers as churned. However, its F1-score (0.259)—which balances precision and recall—shows that the model is still not good enough at identifying churned customers while avoiding false positives. The trade-off between precision and recall in Tree 1 means that it does better at reducing errors but still misses a significant portion of the customers we want to retain.

Tree 3, with an accuracy of 64.8% and a 95% CI of [61.2%, 68.4%], performs similarly to Tree 1 in terms of recall (21.1%) but slightly outperforms Tree 1 in precision (32.6%). This means that Tree 3 is more accurate in predicting non-churned customers, but still fails to predict a sufficient number of churned customers. While it improves slightly over Tree 1, its F1-score (0.256) is still low, indicating that it's not the best at balancing precision and recall for churn prediction.

In contrast, Tree 4 has a lower accuracy (64.1%), with a 95% CI of [60.5%, 67.7%], but its strength lies in its recall (22.5%), which is the highest among all the models. This is extremely important for predicting customer churn because recall tells us how many actual churned customers the model successfully identifies. A high recall means that the business can act quickly to prevent customers from leaving, which is the ultimate goal when trying to reduce churn. Although Tree 4 has a slightly lower accuracy and the lowest AUC (0.502), its recall is the best, allowing the company to focus on the customers who are most likely to leave and offer interventions to retain them. Tree 4 also achieves a solid F1-score (0.265), striking a balance between reducing false positives and capturing churned customers.

Considering the business objective of reducing customer churn, Tree 4 is the most effective model despite having a slightly lower accuracy compared to Tree 2. Its higher recall is critical in identifying customers who are at risk of leaving, enabling the business to act early and improve retention efforts. Tree 4 captures more churned customers than the other models, making it the best choice for proactively targeting and retaining at-risk customers. Therefore, Tree 4 should be selected as the primary model for churn prediction in customer retention strategies.

## Recommendation

To achieve the highest possible performance in predicting customer churn, it's essential to ensure that we select the model that best balances accuracy, recall, precision, and F1-score while considering business requirements such as interpretability and computational efficiency.

While Tree 4 performed well in terms of recall (22.5%), it is crucial to compare this performance with that of alternative predictive models. Recall is particularly important for churn prediction because it helps identify the maximum number of customers who are likely to churn. However, a model with high recall but lower precision or accuracy may result in too many false positives meaning customers flagged as churn risks when they are actually not, it leads to wasted retention efforts. Therefore, achieving the highest good possibility for a churn prediction model requires testing multiple models to find one that provides the best trade-off between recall (capturing as many churned customers as possible) and precision (ensuring the churned customers flagged by the model are accurate).

To identify the most effective model, we must evaluate a variety of predictive models, each with its strengths and weaknesses. For example, Logistic Regression is simple and easy to interpret but may not capture complex patterns in the data. Support Vector Machines (SVMs) are effective in high-dimensional spaces but can be computationally expensive and hard to interpret. Random Forests and Gradient Boosting Machines (GBMs), which are both ensemble methods, typically offer high accuracy and AUC but may sacrifice some interpretability. Meanwhile, models like k-Nearest Neighbors (k-NN) and Naive Bayes can be useful for certain data types but may struggle to handle large feature sets effectively. More complex models, such as Neural Networks and XGBoost, can provide superior predictive power



but require more training data and computational resources, as well as a more advanced understanding of their internal workings.

By comparing these models, we can select the one that provides the best balance of accuracy (how often it's right), recall (how well it captures churn), precision (how accurate it is when flagging churned customers), and AUC (how well it discriminates between churned and non-churned customers). This rigorous comparison will help ensure that the final model has the highest predictive value, enabling the business to target at-risk customers effectively and take proactive actions to reduce churn.

In summary, the best way to achieve the highest possibility of success with churn prediction is by testing multiple models, carefully comparing them across key performance metrics, and selecting the one that offers the most reliable and actionable insights for customer retention. Tree 4 may be a strong candidate, but a thorough evaluation of other models like Random Forests, Gradient Boosting, and XGBoost, combined with feature engineering and hyperparameter tuning, will maximize the model's effectiveness in a real-world business context.