Predicting Cutomer Churn with Classification Decision Tree

Aleksandra Tatko (648925)

November, 2025

Introduction

Retaining existing customers is substantially more cost-effective than acquiring new ones. Research shows that attracting a new customer can cost, depending one the industry, 5 to 25 times more than keeping an existing one (Gallo 2014). This makes the churn prevention a critical priority for companies such as telecom providers, where recurring revenue and long-term customer relationships are central to business performance. In this assignment, churn behaviour is examined using a dataset of 2,000 telecom customers with information on contract type, service usage, monthly charges, tenure, and related attributes. The purpose is to identify which features are most strongly associated with churn and to model this behaviour using a CART classification decision tree model. The guiding research question is: *Which customer characteristics are most informative for predicting churn, and how accurately can a decision tree classify churn versus non-churn customers?*

Data Description

The dataset contains information on 2,000 telecom customers and 13 features, with Churn (Yes/No) as the binary outcome variable. This variable is imbalanced (Yes = 536, No = 1464) highlighting the need to use evaluation metrics that account for class imbalance. Most predictors are unordered factor variables, which can be grouped into four categories: customer demographics (Partner), phone and line services (e.g., Phone Service, Multiple Lines), internet and add-on services (e.g., Internet Service), and contract and billing information (e.g., Tenure, Contract Type). Three variables—tenure, MonthlyCharges, and TotalCharges—are numeric. One advantage of decision trees is that they do not rely on assumptions about normality, linearity, or correlation structures. Since the dataset also contains no missing values, only minimal preprocessing was required.

Methodology

A classification decision tree (CART) partitions the data into increasingly homogeneous groups using binary splits chosen to maximise node purity. Purity is typically measured using the Gini impurity, which quantifies how mixed the classes are within a node. Gini is the default in CART because it is computationally efficient and tends to create slightly more stable, balanced splits than entropy, although both criteria usually lead to similar trees (James et al. 2023). Decision trees are well suited for churn prediction because they can naturally model non-linear relationships, high-order interactions, and work seamlessly with both categorical and numeric predictors without requiring transformations or scaling. At each split, the algorithm evaluates all possible cut-points or category partitions and chooses the option that most reduces impurity. This process continues until the tree is fully grown, which often results in overfitting. To improve generalisation, CART uses cost–complexity pruning, which evaluates a sequence of simpler subtrees and selects the one that best balances fit and complexity. The pruning objective is:

$$\min_{T} \{R(T) \; + \; \alpha \, |T|\}$$

where R(T) is misclassification error, |T| is the number of terminal nodes, and alpha (the complexity parameter, cp) penalises overly large trees. The optimal cp is typically chosen through k-fold cross-validation, and repeated cross-validation. To assess model stability, further robustness checks includes alternative train/test splits, comparison of the full and pruned trees, and exploratory resampling (ROSE) to probe potential effects of class imbalance. Because churn data are typically skewed, model performance was evaluated using metrics beyond accuracy, including kappa (randomness measure), sensitivity, specificity, F1-score, balanced accuracy, and the ROC/AUC.

Results

A 70/30 train–test split was used due to the limited number of observations, with stratified sampling applied to preserve the churn distribution. After growing the maximal tree (cp = 0), the cost–complexity table showed that the lowest cross-validated error occurred at cp = 0.009 (xerror = 0.79), where xerror represents the estimated misclassification rate obtained through cross-validation and therefore reflects how well the tree is expected to generalise to unseen data. This value corresponds to a tree with nine splits and ten terminal nodes. The CP plot indicated a broad flat region around the minimum, with several neighbouring cp values yielding statistically similar errors. Applying the 1-SE rule suggested that slightly larger cp values (0.012–0.017) would give a simpler and more interpretable tree without a meaningful loss in predictive performance. To refine the pruning level, 10-fold cross-validation was performed across 20 candidate cp values. This tuning procedure identified cp = 0.010 as the best-performing model (accuracy = 0.7930, = 0.43). To obtain a more stable estimate, the tuning was repeated using three times repeated 10-fold CV. This shifted the optimal value to cp = 0.016, which produced the highest repeated-CV accuracy (0.79) and a consistent kappa (0.4). Given its lower variance and preference for simpler but comparably accurate models, cp = 0.016 was selected as the final pruning parameter. Pruning the maximal tree at this value produced the final decision tree used for out-of-sample evaluation (*Appendix A.1*).

Table 1: Performance Metrics for Final Pruned Decision Tree

Metric1	Value1	Metric2	Value2
True Negatives (TN)	407	Sensitivity	0.927
False Positives (FP)	84	Specificity	0.475
False Negatives (FN)	32	Balanced Accuracy	0.701
True Positives (TP)	76	F1 Score (No)	0.875
Accuracy	0.806	AUC	0.810
Kappa	0.448	Best CP	0.016

This pruned decision tree achieved with the results visible in Table 1. The model reached accuracy of 0.806, though accuracy alone is less informative given the class imbalance. More meaningful measures show that the model performs reasonably well. With 'No' set as the positive class, sensitivity = 0.927 indicates that most non-churners are correctly identified. On the other hand, the model struggles to detect churners (specificity 0.48), misclassifying 76 churners as non-churners. The F1-score for the majority class (0.88) and balanced accuracy (0.70) confirm this asymmetry, and = 0.45 suggests moderate agreement beyond chance. The ROC curve shows good separation between classes, with an AUC of 0.81, indicating solid discriminatory ability. Variable importance measures highlight Contract, tenure, Total Charges, Monthly Charges, and service-related features (Security, TechSupport) as the most influential predictors, aligning with typical churn dynamics (*Appendix A.2*). Robustness checks supported these findings: re-fitting the model under a different train/test split produced slightly lower performance (= 0.36), and comparing the pruned tree to the full unpruned tree showed that pruning improved generalisation (AUC: 0.81 vs. 0.807). Finally, ROSE resampling was tested to increase churn detection. However, this approach substantially degraded overall performance (accuracy = 0.643, AUC = 0.719), indicating that synthetic balancing introduced too much noise for this model (*Appendix A.3*). Overall, the pruned decision tree provided the best balance between interpretability and predictive performance.

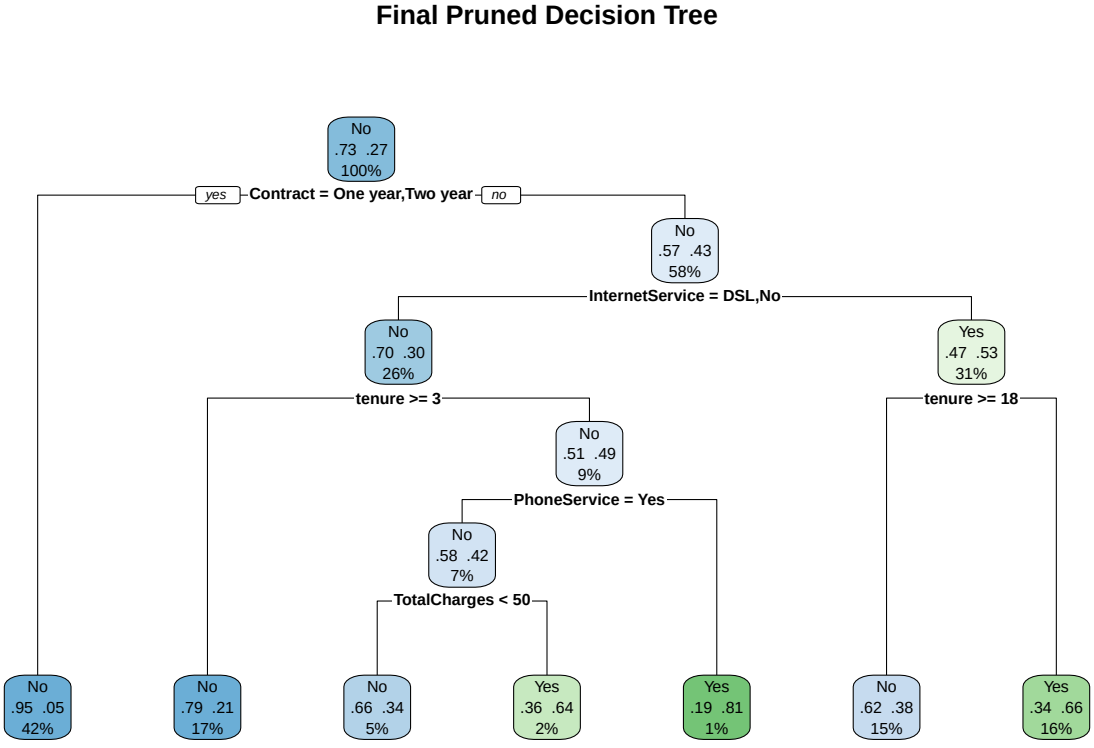
Conclusion and Discussion

The final pruned decision tree (9 terminal nodes) achieved moderate but solid performance, with a kappa of 0.45 and strong majority-class F1 (0.88), though it remains less effective at identifying churners due to class imbalance. The most influential predictors—Contract type, tenure and Total Charges, highlight that short-tenure, month-to-month customers with higher charges are the highest-risk segment, suggesting clear opportunities for targeted retention incentives. A main limitation is the tree's instability and reduced specificity, which indicates high- risk of missing actual churners. More advanced models such as Random Forests or Gradient Boosting could improve churn detection and provide more robust predictive performance in future work.

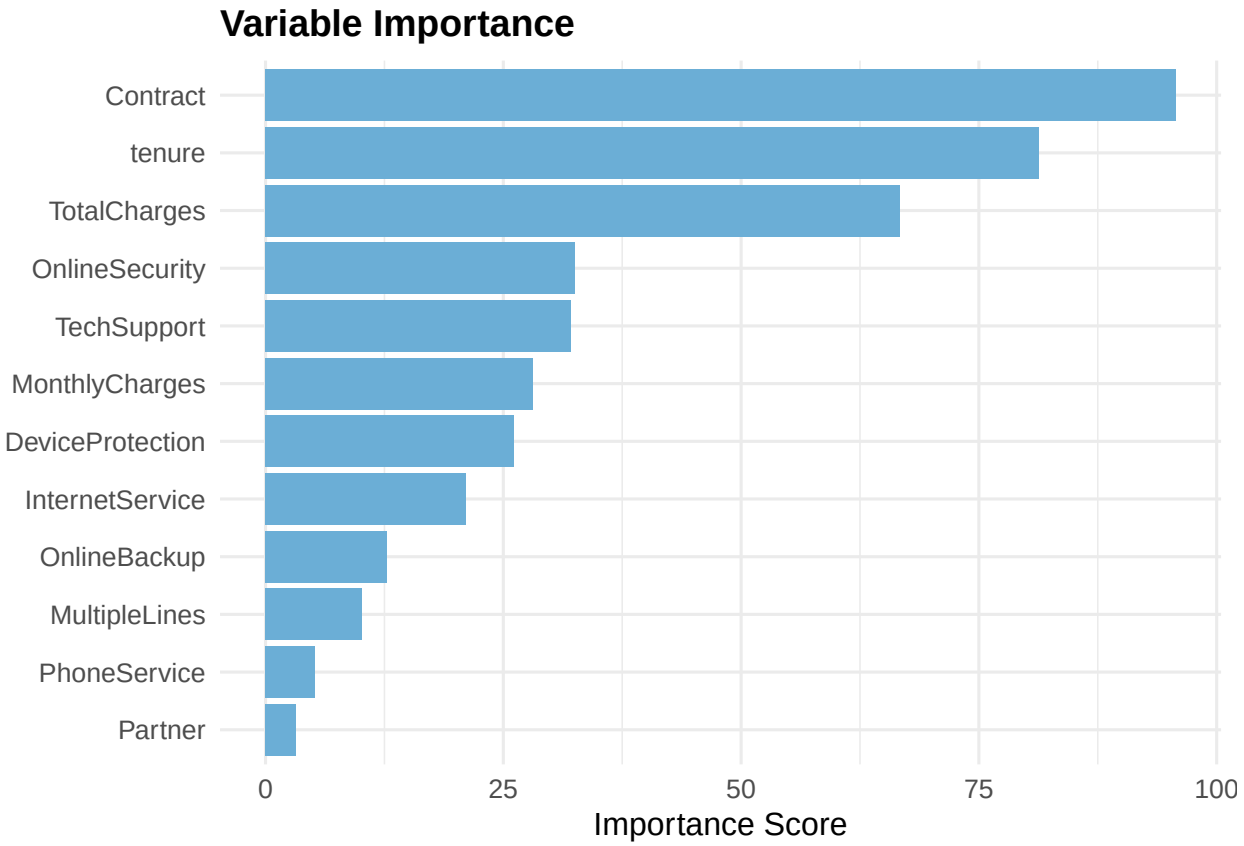
References

Gallo, Amy. 2014. "The Value of Keeping the Right Customers." *Harvard Business Review*. https://hbr.org/2014/10/the-value-of-keeping-the-right-customers.

James, Gareth, Daniela Witten, Trevor Hastie, Robert Tibshirani, and Jonathan Taylor. 2023. *An Introduction to Statistical Learning: With Applications in r*. 3rd ed. New York: Springer. https://www.statlearning.com/.



Appendix A.1 Final Tree for cp = 0.0157, plotted on a train data.



Appendix A.2 Variable Importance for the Final Tree

Table 2: Comparison of Decision Tree Models

Model	TP	TN	FP	FN	Accuracy	Precision	Recall	Specificity	F1	Kappa	BalancedA
Final	76	407	84	32	0.806	0.475	0.704	0.475	0.567	0.448	0.70
Full	80	378	80	61	0.765	0.500	0.567	0.500	0.532	0.375	0.68
80/20 Split	43	268	64	24	0.779	0.402	0.642	0.402	0.494	0.363	0.66
ROSE	141	244	19	195	0.643	0.881	0.420	0.881	0.569	0.324	0.71

Appendix A.3 Different Models Metrics Comparison

Appedix B Code

```
# -------- Preprocessing-------------
str(data) # 3 numeric var, rest factor
colSums(is.na(data)) # no NAs
sapply(data[sapply(data, is.factor)], is.ordered) #no orders
# Correlation for numeric var
corr_matrix <- round(cor(data %>% select(where(is.numeric))),2)
# Convert to long format for ggplot
melted_corr <- melt(corr_matrix)
# Plot
ggplot(melted_corr, aes(x = Var1, y = Var2, fill = value)) +
  geom_tile(color = "white") + geom_text(aes(label = value), color = "black", size = 5) +
  scale_fill_gradient2(low = "#56B1F7", high = "#CA0020", mid = "white",
                       midpoint = 0, limit = c(-1, 1), name = "Correlation") +
  theme_minimal() + theme(
    axis.text.x = element_text(angle = 45, vjust = 1, hjust = 1, size = 12),
    axis.text.y = element_text(size = 12),
    plot.title = element_text(face = "bold", size = 14)
  ) + labs(
    title = "Correlation Heatmap of Numerical Variables",
    subtitle = "Strong correlation (r = 0.83) between tenure and TotalCharges",
    x = "", y = "" )
# Churn distribution
ggplot(data, aes(x = Churn, fill = Churn)) +
  geom_bar() +
  theme_classic() +
  labs(title = "Churn Distribution")
# ------------ Decision Tree -----------
set.seed(123)# We used stratified sampling to preserve the churn rate in training and test sets.
train_index <- createDataPartition(data$Churn, p = 0.7, list = FALSE) # small amount of obs (2000)->  7
train <- data[train_index, ]
test <- data[-train_index, ]
#Fit an Full Decision Tree
tree_full <- rpart(
  Churn ~ .,
  data = train,
  method = "class",
  control = rpart.control(cp = 0)) # grow maximal tree
printcp(tree_full)     # see CP table
plotcp(tree_full)      # visual cp plot
# Automatic cp tuning
fit <- train(
  Churn ~ .,
  data = train,
  method = "rpart",
  trControl = trainControl(method = "cv", number = 10),
  tuneLength = 20)   # test 20 cp values
fit
plot(fit) # Best Kappa at cp 0.01049832 =   0.4273946
# Robustness Check: Repeated 10-fold CV
tr_ctrl <- trainControl(method = "repeatedcv", number = 10, repeats = 3)
fit_rep <- train(
```

```
  Churn ~ .,
  data = train,
  method = "rpart",
  trControl = tr_ctrl,
  tuneLength = 20)
fit_rep
plot(fit_rep)
# Check if cp is stable
fit$bestTune$cp
fit_rep$bestTune$cp
best_cp <- fit_rep$bestTune$cp
# Prune using the chosen cp
final_tree <- prune(tree_full, cp = best_cp)
rpart.plot( # Visualize the final pruned tree
  final_tree,
  type = 2,
  extra = 104,
  fallen.leaves = TRUE,
  main = "Final Pruned Decision Tree")
# Evaluate performance on the test set
pred_class <- predict(final_tree, test, type = "class")
pred_prob <- predict(final_tree, test, type = "prob")[,2]
confusionMatrix(pred_class, test$Churn)
f1_no <- F1_Score(y_pred = pred_class, y_true = test$Churn, positive = "No")
# Plot ROC and compute AUC
pred_obj <- prediction(pred_prob, test$Churn)
perf <- performance(pred_obj, "tpr", "fpr")
plot(perf, col="blue", main="ROC Curve")
abline(0,1, lty=2)
auc <- performance(pred_obj, "auc")@y.values[[1]]
auc
#Extract variable importance
importance <- final_tree$variable.importance
importance_df <- data.frame(
  Variable = names(importance),
  Importance = as.numeric(importance))
importance_df <- importance_df %>% arrange(Importance)
ggplot(importance_df, aes(x = Importance, y = reorder(Variable, Importance))) +
  geom_col(fill = "#6BAED6") +
  theme_minimal(base_size = 12) +
  labs(
    title = "Variable Importance",
    x = "Importance Score",
    y = NULL) +
  theme(plot.title = element_text(face = "bold"),
    axis.text.y = element_text(size = 10))
# ------- Robustness Checks -----
set.seed(456)
train_index2 <- createDataPartition(data$Churn, p = 0.7, list = FALSE)
train2 <- data[train_index2, ]
test2  <- data[-train_index2, ]
tree2 <- rpart(Churn ~ ., train2, method="class", control=rpart.control(cp=0))
fit2 <- train(Churn~., train2, method="rpart", trControl=trainControl(method="cv", number=10), tuneLeng
```

```r
cp2 <- fit2$bestTune$cp
final_tree2 <- prune(tree2, cp=cp2)
pred2 <- predict(final_tree2, test2, type="class")
confusionMatrix(pred2, test2$Churn)
# Compare full vs pruned on test
pred_full <- predict(tree_full, test, type = "class")
conf_full <- confusionMatrix(pred_full, test$Churn)
conf_full
prob_full <- predict(tree_full, test, type="prob")[,2]
auc_full <- performance(prediction(prob_full, test$Churn), "auc")@y.values[[1]]
auc_full
# ROSE resampling to increase detection of churners
library(ROSE)
set.seed(123)
rose_train <- ROSE(Churn ~ ., data = train, seed = 123)$data
table(rose_train$Churn)
tree_rose <- rpart(
  Churn ~ .,
  data = rose_train,
  method = "class",
  control = rpart.control(cp = 0))
fit_rose <- train(
  Churn ~ .,
  data = rose_train,
  method = "rpart",
  trControl = trainControl(method = "cv", number = 10),
  tuneLength = 20)
fit_rose
plot(fit_rose)
best_cp_rose <- fit_rose$bestTune$cp
best_cp_rose
# Prune the ROSE tree
final_tree_rose <- prune(tree_rose, cp = best_cp_rose)
# Evaluate ROSE model on the original (unseen) test set
pred_class_rose <- predict(final_tree_rose, test, type = "class")
pred_prob_rose <- predict(final_tree_rose, test, type = "prob")[,2]
confusionMatrix(pred_class_rose, test$Churn)
pred_obj_rose <- prediction(pred_prob_rose, test$Churn)
auc_rose <- performance(pred_obj_rose, "auc")@y.values[[1]]
```