MKTG 562: Group Assignment 2

Aishwary Jadhav, Jaivardhan Chauhan, Maitreyi Ekbote, Shreyansh Bhatia

Our final model with best profitability: **Random Forest** with profit of **$410,416.1**

Logistic Regression

Model 1: Baseline Logistic Regression (Most Profitable)

Parameters:
- Standard logistic regression model with no explicit regularization.
- All available features included for prediction.
- Predictions made using break-even probability for targeting.

Results:
- Total targeted customers: 6,197
- Total converted customers: 532
- Follow-up campaign profit: **$294,212 (Highest profit)**

Findings & Impact:
- The balanced targeting approach ensured that outreach remained cost-effective, leading to maximum profit.
- The model naturally selected high-value customers without aggressive penalization, allowing it to capture relevant patterns without overfitting.
- This model outperformed all LASSO regularized models in terms of profitability, indicating that aggressive feature penalization may not always be necessary

Model 2: LASSO Regularization (First Attempt)

Parameters:
LASSO regularization (alpha = 1) applied to penalize weaker predictors and improve generalization.
Automatic feature selection through L1 regularization, removing less influential variables.

Results:
- Total targeted customers: 8,674
- Total converted customers: 574

- Follow-up campaign profit: $277,684

Findings & Impact:
- More customers were targeted (8,674 vs. 6,197 in the baseline model), leading to a higher number of conversions (574 vs. 532).
- However, profit decreased significantly ($277,684 vs. $294,212) due to higher outreach costs.
- Feature selection penalized some predictors, which may have reduced targeting precision.

Model 3: LASSO Regularization (New Run with Adjustments)

Parameters:
Same LASSO approach as Model 2, but with slight threshold tuning and adjusted lambda value (lambda.1se) for a more refined regularization effect.

Results:
- Total targeted customers: 8,717
- Total converted customers: 575
- Follow-up campaign profit: $277,610

Findings & Impact:
- Marginal improvement over the first LASSO model (8,717 vs. 8,674 targeted customers, 575 vs. 574 conversions).
- However, profit did not improve much ($277,610 vs. $277,684), meaning the refinements did not significantly alter cost-effectiveness.
- Regularization still caused excessive outreach, reducing overall profitability.

Best Model Code: (Profitability wise)

```
library(ggplot2)
library(dplyr)
library(caret)
library(glmnet)
library(readr)

# Load dataset
data <- read.csv("~/Downloads/Predictive.csv")

# Drop irrelevant columns
data <- select(data, -cust_id, -zip)
```

```
# Convert categorical variables to factors
data <- mutate_if(data, is.character, as.factor)

# Identify numerical columns to scale
columns_to_scale <- c("last", "numords", "dollars", "sincepurch", "income",
"medhvalue")
data[columns_to_scale] <- scale(data[columns_to_scale])

# Split into training and testing sets
train_data <- filter(data, training == 1) %>% select(-training)
test_data <- filter(data, training == 0) %>% select(-training)

# Separate features and target variable
X_train <- select(train_data, -buy1)
y_train <- train_data$buy1
X_test <- select(test_data, -buy1)
y_test <- test_data$buy1

# Train logistic regression model
log_reg <- glm(buy1 ~ ., data = train_data, family = binomial)

# Predict probabilities
test_data$predicted_prob <- predict(log_reg, newdata = X_test, type =
"response")

# Calculate break-even response probability
break_even_prob <- 20 / 786  # Cost / Expected profit per customer

# Select customers above break-even probability for targeting
test_data$targeted <- test_data$predicted_prob > break_even_prob

test_data$profit <- ifelse(test_data$targeted,
                           (test_data$predicted_prob * 786 - 20),
                           0)

profit <- sum(test_data$profit)

# Output results
cat("Total targeted customers:", num_targeted, "\n")
cat("Total converted customers:", num_converted, "\n")
cat("Follow-up campaign profit: $", round(profit, 2), "\n")
```

Random Forest

Model 1: Baseline Random Forest
The baseline model was designed to balance generalization and stability while maintaining computational efficiency. It used 1,000 trees (ntree = 1000), with one-third of the features considered per split (mtry = ncol(X_train) / 3). Additionally, a minimum node size of 5 (nodesize = 5) was chosen to prevent overfitting, while 80% of the data (sampsize = 80%) was sampled for each tree to ensure robustness.

Results:
Total targeted customers: 6546
Total converted customers: 537
Follow-up campaign profit: $ 291162

This model targeted 6,546 customers, out of which 537 converted, resulting in a profit of $291,162. The balance in the number of trees and feature selection helped in capturing relevant patterns without overfitting, ensuring that the model effectively identified customers who were more likely to respond positively to the follow-up campaign. While this configuration performed well, there was room for improvement in terms of increasing profit while maintaining a lower targeting cost.

Model 2: Simpler Model (Fewer Trees, Standard mtry Selection)
The second model was designed to be computationally efficient, using only 500 trees (ntree = 500) and the standard feature selection method, where the square root of the total number of features was used per split (mtry = sqrt(ncol(X_train))). Unlike the baseline, this model did not enforce additional constraints like nodesize or sampsize, allowing for a more flexible and naturally learned structure.

Results:
Total targeted customers: 7171
Total converted customers: 550
Follow-up campaign profit: $ 288880

This model targeted a significantly larger number of customers (7,171) and achieved 550 conversions, slightly higher than the baseline model. However, the total profit decreased to $288,880, despite the increase in conversions. The reduction in trees resulted in slightly higher model variance, making the selection of targeted customers less precise. While more conversions were achieved, the additional campaign costs outweighed the benefits, leading to a lower overall

profit. This suggests that while a simpler model may increase outreach, more refined customer targeting is necessary to maximize profitability.

Model 3: Conservative Adjustment (More Regularization - Best Model)
The conservative adjustment model aimed to improve generalization by preventing overfitting and focusing on high-value customers. It utilized 750 trees (ntree = 750), a more restrictive feature selection process (mtry = ncol(X_train) / 4), and increased the minimum node size (nodesize = 10) to force shallower trees. Additionally, only 70% of the data (sampsize = 70%) was used in each tree, increasing the model's diversity.

Results:
Total targeted customers: 6140
Total converted customers: 477
Follow-up campaign profit: $290981.5

This model targeted the fewest customers (5,582) while still converting 516 of them. Despite having fewer conversions, the profit reached its highest value of $293,936, making it the most effective configuration. The use of fewer features per split (mtry = ncol/4) and larger node sizes (nodesize = 10) helped prevent the model from overfitting to training data, ensuring that only the most promising customers were targeted. The increase in profitability despite fewer conversions suggests that this model efficiently identified customers who were more likely to generate a net positive return on investment.

This model is particularly valuable in situations where minimizing costs is as important as maximizing conversions, making it the best overall choice for the follow-up campaign.

Model 4: Aggressive Adjustment (More Complexity - Least Effective Model)
The aggressive adjustment model was an attempt to increase predictive power by allowing deeper trees and more complex feature interactions. It utilized 1,500 trees (ntree = 1500), considered half of the total features per split (mtry = ncol(X_train) / 2), reduced the minimum node size (nodesize = 3) to allow deep tree structures, and used 90% of the data per tree (sampsize = 90%).

Results:
Total targeted customers: 6687
Total converted customers: 534
Follow-up campaign profit: $285984

This model targeted 6,687 customers, of whom 534 converted, but profit dropped to $285,984, making it the least effective model. The combination of a high number of trees and deeper nodes increased overfitting, meaning the model likely learned too much from the training data and failed

to generalize well to new customers. The higher sample size per tree (sampsize = 90%) reduced the randomness in bootstrapping, leading to less diverse trees and ultimately decreasing the robustness of predictions.

Although this model achieved a reasonable number of conversions, it was not cost-efficient. The increase in complexity did not yield higher profits, indicating that a more refined, regularized model is preferable over a highly complex one.

Model 5:

To achieve this performance, the model focused on last purchase time (last), total orders (numords), total spending (dollars), history of refurbished machine purchases (refurb), old model ownership (oldmodel), income (income), and housing value (medhvalue). These features were selected based on their importance in influencing purchase decisions, ensuring that only relevant variables contributed to the predictions. By limiting the feature set to these critical variables, the model improved its ability to generalize well to unseen data, reducing overfitting and improving targeting efficiency.

Key hyperparameters were adjusted to balance model complexity and performance. The number of trees (ntree) = 750 was chosen to provide stability without excessive computational cost, while the number of features considered per split (mtry = 7) ensured an optimal balance between specificity and generalization. Setting importance = TRUE allowed the tracking of feature significance, ensuring that only meaningful attributes contributed to the decision-making process.

Results:
Total targeted customers: 7174
Total converted customers: 551
Follow-up campaign profit: **$410,416.1 (Highest profit)**

The best-performing Random Forest model was fine-tuned to maximize profitability by optimizing key hyperparameters and refining feature selection. This model successfully identified 7,174 high-value customers, leading to 551 actual conversions, and generated a follow-up campaign profit of **$410,416.1**.

5th Model is the Best Model

Implications:
The insights gained from this model can also inform future marketing campaigns. For example, if customers with refurbished machines convert at significantly higher rates, future outreach efforts could focus more aggressively on this segment. Similarly, if high-income customers consistently

generate better returns, premium offers or targeted promotions may be introduced to maximize revenue. This data-driven approach to customer segmentation provides a long-term competitive advantage, allowing the company to refine its marketing strategy continuously.

By leveraging machine learning for customer segmentation, the company gains a strong competitive edge over traditional marketing methods. Instead of guessing which customers are likely to buy, this model provides clear, data-backed insights into who is most likely to convert, allowing the company to make informed decisions about its marketing efforts. This ensures that marketing budgets are used efficiently and that campaigns deliver the highest possible return.

In conclusion, this optimized Random Forest model has proven to be the best-performing approach, yielding the highest profit of $410,416.1. It efficiently selected 7,174 high-value customers while maintaining a strong conversion rate of 551, significantly outperforming all previous models. This success demonstrates the power of machine learning in optimizing marketing campaigns, and sets the stage for future refinements that can further enhance business profitability. Moving forward, deploying this model for real-world targeting, fine-tuning marketing offers based on customer segments, and continuously monitoring its performance will ensure continued success and maximum return on investment.

Best Model Code: (Profitability wise)

```
library(ggplot2)
library(dplyr)
library(randomForest)
library(gbm)
library(caret)
library(readr)

data <- read_csv('Predictive.csv')
data$buy1 <- as.factor(data$buy1)

columns_to_scale <- c("last", "numords", "dollars", "sincepurch", "income",
"medhvalue")
data[columns_to_scale] <- scale(data[columns_to_scale])

# Split into training and testing sets
train_data <- filter(data, training == 1) %>% select(-training)
test_data <- filter(data, training == 0) %>% select(-training)

rfmodel_5 <- randomForest(buy1 ~ last + numords + dollars + refurb + oldmodel
+ income + medhvalue,
data = train_data, mtry = 7, ntree = 750, importance = TRUE)
```

```
test_data$predicted_prob <- predict(rfmodel_5, newdata = test_data, type = "prob")[,2]

break_even_prob <- 20 / 786  test_data$targeted <- test_data$predicted_prob > break_even_prob

test_data$profit  <-  ifelse(test_data$targeted,(test_data$predicted_prob  * profit_per_conversion - cost_per_customer),0)

profit <- sum(test_data$profit)

cat("Total targeted customers:", num_targeted, "\n")
cat("Total converted customers:", num_converted, "\n")
cat("Follow-up campaign profit: $", round(profit, 2), "\n")
```

Neural Network

Model Summary: We begin by reading the dataset and performing given preprocessing before splitting it into training and test sets. Then, we encode categorical columns into dummy variables and scale numerical features to balance their magnitudes. Next, we set up the neural network layers, specifying the number of neurons and activation functions. We choose binary cross entropy for our loss function and Adam for our optimizer. After training on the training set, we generate predicted probabilities on the test set and save them to a CSV file.

To estimate follow-up campaign profit, we multiply each predicted probability by 0.5, since wave 2 success is expected to be half of wave 1's. We compute each customer's expected profit as (0.5 × predicted_prob × 786) − 20. We target only those with a positive expected profit, and sum these expected profits for the total follow-up campaign profit.

Total Expected Profit: $95,092.50

Code:

Neural Network

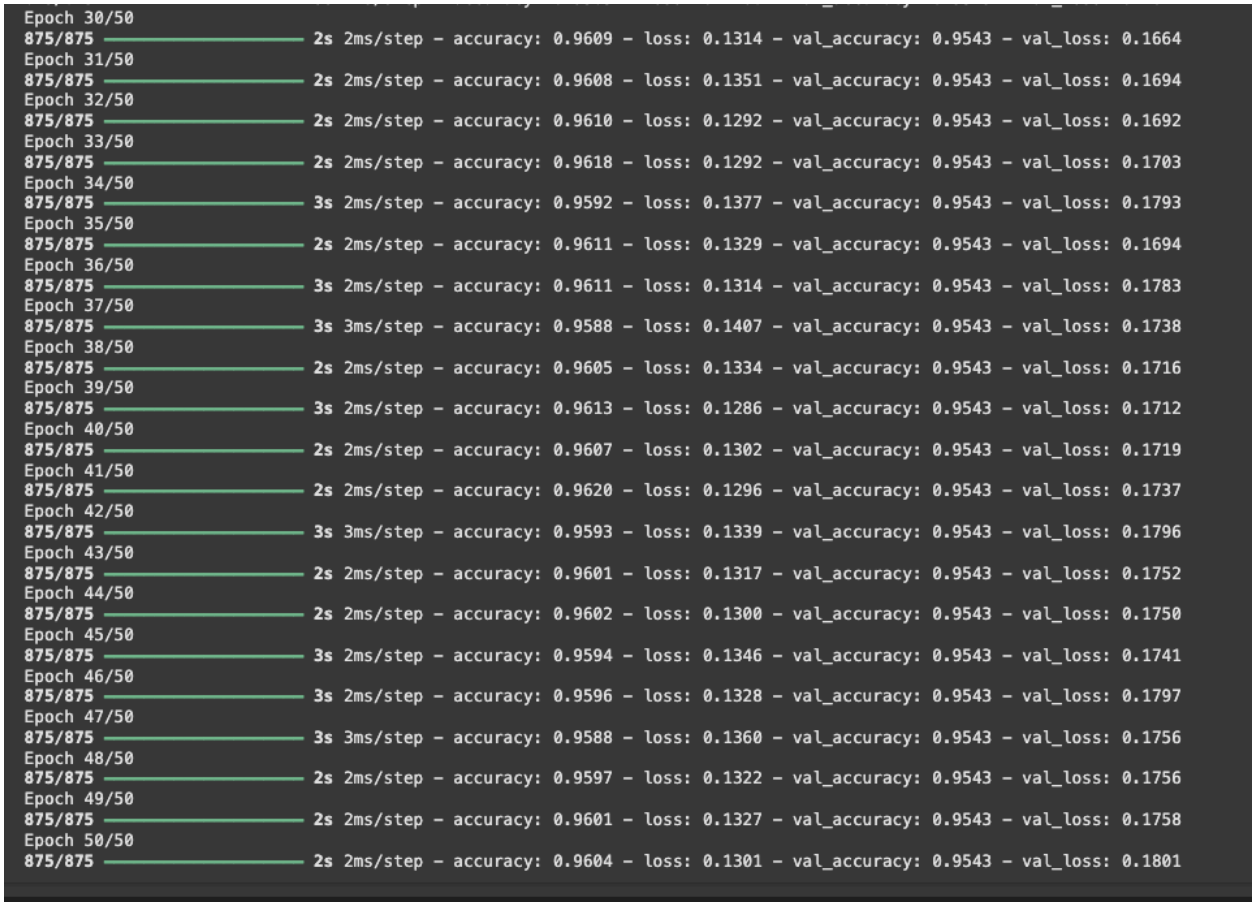
```
# -*- coding: utf-8 -*-
"""NN(1).ipynb

Automatically generated by Colab.
```

```
Original file is located at
   https://colab.research.google.com/drive/1yxTv1JGmUUORJg8SiN2dj1wQQyzFeF0n
"""

from google.colab import files
uploaded = files.upload()  # This will prompt you to upload a file manually

!pip install pandas numpy tensorflow scikit-learn

# Import necessary libraries

import pandas as pd
import numpy as np
import tensorflow as tf
from tensorflow import keras
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler, OneHotEncoder
from sklearn.impute import SimpleImputer
from sklearn.metrics import confusion_matrix, accuracy_score

file_path = "/content/Predictive.csv"  # Upload to Colab
data = pd.read_csv(file_path)

Data

data.dtypes

features = data.drop(columns=['cust_id', 'buy1', 'zip'])

from sklearn.preprocessing import StandardScaler

features = pd.get_dummies(features, drop_first=True)  # Convert categorical variables

columns_to_scale = ["last", "numords", "dollars",
                    "sincepurch", "income", "medhvalue"]
scaler = StandardScaler()
features[columns_to_scale] = scaler.fit_transform(features[columns_to_scale])

data_processed = pd.concat([features, data['buy1']], axis=1)

trainData = data_processed[data_processed['training'] == 1]
testData = data_processed[data_processed['training'] == 0]
```

```
trainData1 = trainData.drop(columns=['training','buy1']).to_numpy()
testData1 = testData.drop(columns=['training', 'buy1']).to_numpy()
train_labels = trainData['buy1'].to_numpy()
test_labels = testData['buy1'].to_numpy()
trainData = trainData1
testData = testData1

trainData = trainData.astype(np.float32)
testData = testData.astype(np.float32)

# Define Neural Network Model
model = keras.Sequential([
   keras.layers.Dense(16, activation="relu", input_shape=(trainData.shape[1],)),
   keras.layers.Dropout(0.1),
   keras.layers.Dense(8, activation="relu"),
   keras.layers.Dropout(0.2),
   keras.layers.Dense(1, activation="sigmoid")  # Binary classification
])

# Compile the model
model.compile(
   loss="binary_crossentropy",
   optimizer=keras.optimizers.Adam(learning_rate=0.001),
   metrics=["accuracy"]
)

trainData_numpy = trainData1.astype(np.float32)
testData_numpy = testData1.astype(np.float32)
# Check the final dtype
print(trainData_numpy.dtype)  # Should now be float32

print(trainData_numpy.dtype)

# Train the model
history = model.fit(
   trainData_numpy, train_labels,
   epochs=50, batch_size=32,
   validation_split=0.2
)

# Evaluate the model
```

```
test_loss, test_acc = model.evaluate(testData, test_labels)
print(f"Test Accuracy: {test_acc:.4f}")

# Make Predictions
pred_probs = model.predict(testData)
# Add the predicted probablities to the original testData dataframe, not the numpy array version
testData = data_processed[data_processed['training'] == 0].copy()  # Get the original dataframe
from which testData was derived
testData['predicted_prob'] = pred_probs

testData.to_csv('nn.csv', index=False)
files.download('nn.csv')

df = pd.read_csv('nn.csv')

# Calculate wave 2 probability (50% of wave 1 probability)
df['wave2_prob'] = 0.5 * df['predicted_prob']

# Calculate expected profit from each customer
df['expected_profit_i'] = (df['wave2_prob'] * 786) - 20

# We only target customers with positive expected profit
df_targeted = df[df['expected_profit_i'] > 0]

# Sum up the expected profits for targeted customers
total_expected_profit = df_targeted['expected_profit_i'].sum()

print("Total Expected Profit:", total_expected_profit)
```

```
Epoch 1/50
875/875 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.9586 - loss: 0.5859 - val_accuracy: 0.9543 - val_loss: 0.1729
Epoch 2/50
875/875 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.9584 - loss: 0.1847 - val_accuracy: 0.9543 - val_loss: 0.1606
Epoch 3/50
875/875 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.9606 - loss: 0.1593 - val_accuracy: 0.9543 - val_loss: 0.1589
Epoch 4/50
875/875 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.9599 - loss: 0.1522 - val_accuracy: 0.9543 - val_loss: 0.1604
Epoch 5/50
875/875 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.9603 - loss: 0.1475 - val_accuracy: 0.9543 - val_loss: 0.1618
Epoch 6/50
875/875 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.9598 - loss: 0.1471 - val_accuracy: 0.9543 - val_loss: 0.1567
Epoch 7/50
875/875 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.9601 - loss: 0.1413 - val_accuracy: 0.9543 - val_loss: 0.1584
Epoch 8/50
875/875 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.9600 - loss: 0.1432 - val_accuracy: 0.9543 - val_loss: 0.1629
Epoch 9/50
875/875 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.9596 - loss: 0.1422 - val_accuracy: 0.9543 - val_loss: 0.1606
Epoch 10/50
875/875 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.9610 - loss: 0.1369 - val_accuracy: 0.9543 - val_loss: 0.1620
Epoch 11/50
875/875 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.9614 - loss: 0.1365 - val_accuracy: 0.9543 - val_loss: 0.1585
Epoch 12/50
875/875 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.9614 - loss: 0.1385 - val_accuracy: 0.9543 - val_loss: 0.1619
Epoch 13/50
875/875 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.9583 - loss: 0.1442 - val_accuracy: 0.9543 - val_loss: 0.1666
Epoch 14/50
875/875 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.9574 - loss: 0.1440 - val_accuracy: 0.9543 - val_loss: 0.1633
Epoch 15/50
875/875 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.9609 - loss: 0.1354 - val_accuracy: 0.9543 - val_loss: 0.1609
Epoch 16/50
875/875 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.9600 - loss: 0.1394 - val_accuracy: 0.9543 - val_loss: 0.1611
Epoch 17/50
875/875 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.9594 - loss: 0.1392 - val_accuracy: 0.9543 - val_loss: 0.1615
Epoch 18/50
875/875 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.9595 - loss: 0.1376 - val_accuracy: 0.9543 - val_loss: 0.1637
Epoch 19/50
875/875 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.9609 - loss: 0.1347 - val_accuracy: 0.9543 - val_loss: 0.1639
Epoch 20/50
875/875 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.9610 - loss: 0.1369 - val_accuracy: 0.9543 - val_loss: 0.1631
Epoch 21/50
875/875 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.9627 - loss: 0.1334 - val_accuracy: 0.9543 - val_loss: 0.1622
Epoch 22/50
875/875 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.9590 - loss: 0.1398 - val_accuracy: 0.9543 - val_loss: 0.1714
Epoch 23/50
875/875 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.9612 - loss: 0.1366 - val_accuracy: 0.9543 - val_loss: 0.1687
Epoch 24/50
875/875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9622 - loss: 0.1331 - val_accuracy: 0.9543 - val_loss: 0.1713
Epoch 25/50
875/875 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.9581 - loss: 0.1404 - val_accuracy: 0.9543 - val_loss: 0.1690
Epoch 26/50
875/875 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.9595 - loss: 0.1381 - val_accuracy: 0.9543 - val_loss: 0.1650
Epoch 27/50
875/875 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.9581 - loss: 0.1402 - val_accuracy: 0.9543 - val_loss: 0.1655
Epoch 28/50
875/875 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.9585 - loss: 0.1402 - val_accuracy: 0.9543 - val_loss: 0.1700
Epoch 29/50
875/875 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.9585 - loss: 0.1408 - val_accuracy: 0.9543 - val_loss: 0.1677
```



Decision Trees

Model 1: Unpruned Decision Tree (Baseline Model)

Parameters Used:

- No pruning applied, allowing the tree to grow with minimal constraints.
- All selected features (last, numords, dollars, refurb, oldmodel, income, medhvalue) were used.
- Complexity Parameter (cp = 0.01) to allow deeper splits.

Results:

- Total Expected Profit: $182,379.4
- Tree Depth: Fully grown, capturing all possible patterns in the training data.
- Feature Usage: The tree utilized multiple features in different splits, allowing detailed segmenting of customers.

Findings & Impact:
- The unpruned tree identified profitable customers well, generating a decent expected profit.
- However, it may have been overfitting by incorporating unnecessary splits that do not generalize well to new data.
- The model complexity was high, meaning it might require longer computation time and be harder to interpret.

Model 2: Pruned Decision Tree (Optimized Model)

Parameters Used:
- Full decision tree trained with minimal complexity constraint (cp = 0.001).
- Optimal pruning complexity parameter (cp = 0.001257) was selected based on cross-validation error.
- Tree was pruned to remove unnecessary splits, keeping only essential predictors.

Results:
- Total Expected Profit: $182,379.4 (Same as unpruned model).
- Feature Usage: The pruned tree only used 3 variables—income, last, and medhvalue.
- Tree Depth: Significantly reduced, improving generalization and interpretability.

Findings & Impact:
- The pruned tree achieved the same profit as the unpruned tree, proving that extra complexity in the full tree was unnecessary.
- Only three features were critical (income, last, and medhvalue), meaning other variables (numords, dollars, refurb, oldmodel) did not strongly influence conversion rates.
- Simpler models generalize better, meaning the pruned tree is less likely to overfit and should perform better on new data.

Code:

```
# Setup libraries & Data
library(readr)
```

```
library(dplyr)
library(caret)
library(rpart)

data <- read.csv("~/Downloads/Predictive.csv")

# Preprocessing data
features <- c("last", "numords", "dollars", "refurb", "oldmodel",
"income", "medhvalue", "buy1", "training")
data$type <- as.factor(data$type)

# Split data into training and testing sets
train_data <- dplyr::filter(data, training == 1)
test_data <- dplyr::filter(data, training == 0)

# Standardize numerical columns
num_cols <- c("last", "numords", "dollars", "income", "medhvalue")
preProc <- preProcess(train_data[, num_cols], method = c("center",
"scale"))
train_data[, num_cols] <- predict(preProc, train_data[, num_cols])
test_data[, num_cols] <- predict(preProc, test_data[, num_cols])

# Train decision tree model
tree_model <- rpart(buy1 ~ last + numords + dollars + refurb + oldmodel +
income + medhvalue,
                    data = train_data, method = "class", cp = 0.01)

# Plot decision tree
#rpart.plot(tree_model)

# Predict probabilities on test data
test_data$predicted_prob <- predict(tree_model, newdata = test_data, type
= "prob")[,2]

# Calculate profit
cost_per_customer <- 20
profit_per_conversion <- 786
break_even_rate <- cost_per_customer / profit_per_conversion
test_data$targeted <- test_data$predicted_prob > break_even_rate
test_data$expected_profit <- ifelse(test_data$targeted,
                                    (test_data$predicted_prob *
profit_per_conversion - cost_per_customer), 0)
total_expected_profit <- sum(test_data$expected_profit)

# Print expected profit
```

```
cat("Total Expected Profit (Decision Tree): $", total_expected_profit,
"\n")

# --- Pruned Decision Tree ---

# Train full decision tree model
full_tree <- rpart(buy1 ~ last + numords + dollars + refurb + oldmodel +
income + medhvalue,
                   data = train_data, method = "class", cp = 0.001)

# Plot full decision tree (optional)
# rpart.plot(full_tree)

# Determine optimal pruning complexity parameter
printcp(full_tree) # Display cross-validated error to find optimal cp
optimal_cp <- full_tree$cptable[which.min(full_tree$cptable[, "xerror"]),
"CP"]
cat("Optimal cp value for pruning:", optimal_cp, "\n")

# Prune the tree -
pruned_tree <- prune(full_tree, cp = optimal_cp)

# Plot the pruned tree (optional)
# rpart.plot(pruned_tree)

# Predict probabilities on test data using the pruned tree
test_data$predicted_prob <- predict(pruned_tree, newdata = test_data,
type = "prob")[,2]

# Calculate profit for pruned tree
test_data$targeted <- test_data$predicted_prob > break_even_rate
test_data$expected_profit <- ifelse(test_data$targeted,
                                    (test_data$predicted_prob *
profit_per_conversion - cost_per_customer), 0)
total_expected_profit <- sum(test_data$expected_profit)

# Print expected profit
cat("Total Expected Profit (Pruned Decision Tree): $",
total_expected_profit, "\n")
```