Data-Driven Decision Trees for Business

Felipe O. Cerezer

2024-08-25

Step 1: Required Libraries

We will use the following R packages: - `rpart` for building decision trees - `caret` for various modeling techniques, including decision trees - `ggplot2` for data visualizations

```r
## Install required packages if not already installed
#install.packages(c("rpart", "caret", "ggplot2"))

## Load libraries
library(rpart)
library(caret)
```

```
## Loading required package: ggplot2

## Loading required package: lattice
```

```r
library(ggplot2)
```

Step 2: Generate Fake Data

First, I will create a synthetic dataset representing customers information, such as Age, Income, spending habits score, and Segment (the target variable).

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

# Generate fake data
n <- 200  # Number of observations
data <- data.frame(
  Age = sample(18:70, n, replace = TRUE),
  Income = sample(20000:120000, n, replace = TRUE),
  SpendingScore = sample(1:100, n, replace = TRUE),
  Segment = sample(c("Low", "Medium", "High"), n, replace = TRUE)
)

# View the first few rows of the data
head(data)
```

```
##   Age Income SpendingScore Segment
## 1  66 117123            75     Low
## 2  54  95338            13    High
## 3  18  81640            18  Medium
## 4  42  32349            85     Low
## 5  27  54782             5    High
```

```
## 6  53  55376           31    High
```

Explanation: Here, I used the sample() function to randomly generate values for Age, Income, and SpendingScore. The Segment is a categorical variable indicating whether a customer belongs to the "Low", "Medium", or "High" segment.

Step 3: Splitting the Data

To evaluate the decision tree's performance, I split the dataset into training (70%) and testing (30%) sets. The model will be trained on the training set and tested on the testing set to check its accuracy.

```r
# Create a training (70%) and testing (30%) split
set.seed(42)
trainIndex <- createDataPartition(data$Segment, p = 0.7, list = FALSE)
trainData <- data[trainIndex, ]
testData <- data[-trainIndex, ]
```

Explanation: The createDataPartition() function from the caret package creates an index to split the data into training and testing sets. I used a 70/30 split, which is a common practice to ensure the model has enough data to learn from while still being tested on unseen data.

Step 4: Building the Decision Tree

Next, I use the rpart() function to build the decision tree model. This function will automatically determine the best splits in the data based on the Segment variable.

```r
# Build the decision tree model
treeModel <- rpart(Segment ~ Age + Income + SpendingScore,
                   data = trainData,
                   method = "class")

# Print the model summary
summary(treeModel)
```

```
## Call:
## rpart(formula = Segment ~ Age + Income + SpendingScore, data = trainData,
##     method = "class")
##   n= 141
##
##           CP nsplit rel error   xerror       xstd
## 1 0.11956522      0 1.0000000 1.228261 0.05148976
## 2 0.04891304      1 0.8804348 1.076087 0.05902620
## 3 0.02173913      3 0.7826087 1.097826 0.05818262
## 4 0.01630435      7 0.6956522 1.086957 0.05861308
## 5 0.01000000      9 0.6630435 1.097826 0.05818262
##
## Variable importance
## SpendingScore        Income           Age
##            60            26            14
##
## Node number 1: 141 observations,    complexity param=0.1195652
##   predicted class=Medium  expected loss=0.6524823  P(node) =1
##     class counts:    47    45    49
##    probabilities: 0.333 0.319 0.348
##   left son=2 (39 obs) right son=3 (102 obs)
```

```
##   Primary splits:
##       Income        < 45830.5 to the left,  improve=2.6506530, (0 missing)
##       SpendingScore < 67.5    to the left,  improve=1.8436950, (0 missing)
##       Age           < 20.5    to the right, improve=0.7159897, (0 missing)
##   Surrogate splits:
##       SpendingScore < 98.5    to the right, agree=0.738, adj=0.051, (0 split)
##
## Node number 2: 39 observations,    complexity param=0.01630435
##   predicted class=Low     expected loss=0.5128205  P(node) =0.2765957
##     class counts:    12    19     8
##    probabilities: 0.308 0.487 0.205
##   left son=4 (31 obs) right son=5 (8 obs)
##   Primary splits:
##       SpendingScore < 80      to the left,  improve=2.5312240, (0 missing)
##       Income        < 24848   to the right, improve=0.7763278, (0 missing)
##       Age           < 40.5    to the right, improve=0.7065527, (0 missing)
##
## Node number 3: 102 observations,    complexity param=0.04891304
##   predicted class=Medium  expected loss=0.5980392  P(node) =0.7234043
##     class counts:    35    26    41
##    probabilities: 0.343 0.255 0.402
##   left son=6 (72 obs) right son=7 (30 obs)
##   Primary splits:
##       SpendingScore < 67.5    to the left,  improve=1.732353, (0 missing)
##       Income        < 58294   to the left,  improve=1.671242, (0 missing)
##       Age           < 37.5    to the left,  improve=1.368311, (0 missing)
##   Surrogate splits:
##       Age < 68      to the left,  agree=0.725, adj=0.067, (0 split)
##
## Node number 4: 31 observations,    complexity param=0.01630435
##   predicted class=High    expected loss=0.6129032  P(node) =0.2198582
##     class counts:    12    12     7
##    probabilities: 0.387 0.387 0.226
##   left son=8 (17 obs) right son=9 (14 obs)
##   Primary splits:
##       SpendingScore < 41.5    to the left,  improve=1.2466790, (0 missing)
##       Age           < 39.5    to the right, improve=0.9876181, (0 missing)
##       Income        < 35452   to the left,  improve=0.7700579, (0 missing)
##   Surrogate splits:
##       Income < 25799.5 to the right, agree=0.710, adj=0.357, (0 split)
##       Age    < 34.5    to the right, agree=0.645, adj=0.214, (0 split)
##
## Node number 5: 8 observations
##   predicted class=Low     expected loss=0.125  P(node) =0.05673759
##     class counts:     0     7     1
##    probabilities: 0.000 0.875 0.125
##
## Node number 6: 72 observations,    complexity param=0.04891304
##   predicted class=High    expected loss=0.625  P(node) =0.5106383
##     class counts:    27    21    24
##    probabilities: 0.375 0.292 0.333
##   left son=12 (59 obs) right son=13 (13 obs)
##   Primary splits:
##       Age           < 56.5    to the left,  improve=2.214146, (0 missing)
```

```
##       SpendingScore < 61.5    to the right, improve=1.569780, (0 missing)
##       Income        < 58643   to the left,  improve=1.337302, (0 missing)
##
## Node number 7: 30 observations
##   predicted class=Medium  expected loss=0.4333333  P(node) =0.212766
##     class counts:     8     5    17
##    probabilities: 0.267 0.167 0.567
##
## Node number 8: 17 observations
##   predicted class=High    expected loss=0.4705882  P(node) =0.1205674
##     class counts:     9     6     2
##    probabilities: 0.529 0.353 0.118
##
## Node number 9: 14 observations
##   predicted class=Low     expected loss=0.5714286  P(node) =0.09929078
##     class counts:     3     6     5
##    probabilities: 0.214 0.429 0.357
##
## Node number 12: 59 observations,    complexity param=0.02173913
##   predicted class=High    expected loss=0.559322  P(node) =0.4184397
##     class counts:    26    16    17
##    probabilities: 0.441 0.271 0.288
##   left son=24 (8 obs) right son=25 (51 obs)
##   Primary splits:
##       SpendingScore < 58.5    to the right, improve=1.579595, (0 missing)
##       Income        < 108279  to the left,  improve=1.414933, (0 missing)
##       Age           < 20.5    to the right, improve=1.057832, (0 missing)
##
## Node number 13: 13 observations
##   predicted class=Medium  expected loss=0.4615385  P(node) =0.09219858
##     class counts:     1     5     7
##    probabilities: 0.077 0.385 0.538
##
## Node number 24: 8 observations
##   predicted class=High    expected loss=0.25  P(node) =0.05673759
##     class counts:     6     0     2
##    probabilities: 0.750 0.000 0.250
##
## Node number 25: 51 observations,    complexity param=0.02173913
##   predicted class=High    expected loss=0.6078431  P(node) =0.3617021
##     class counts:    20    16    15
##    probabilities: 0.392 0.314 0.294
##   left son=50 (9 obs) right son=51 (42 obs)
##   Primary splits:
##       SpendingScore < 49.5    to the right, improve=2.0429510, (0 missing)
##       Income        < 58643   to the right, improve=1.4464200, (0 missing)
##       Age           < 20.5    to the right, improve=0.7579577, (0 missing)
##
## Node number 50: 9 observations
##   predicted class=Low     expected loss=0.3333333  P(node) =0.06382979
##     class counts:     2     6     1
##    probabilities: 0.222 0.667 0.111
##
## Node number 51: 42 observations,    complexity param=0.02173913
```

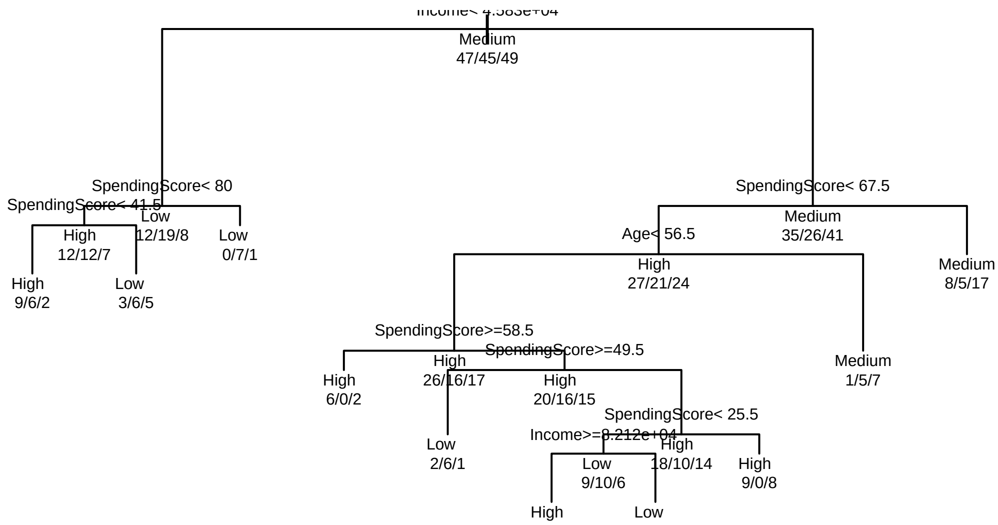
```
##   predicted class=High    expected loss=0.5714286  P(node) =0.2978723
##     class counts:    18    10    14
##    probabilities: 0.429 0.238 0.333 
##   left son=102 (25 obs) right son=103 (17 obs)
##   Primary splits:
##       SpendingScore < 25.5    to the left,  improve=2.4475070, (0 missing)
##       Income        < 56573   to the right, improve=0.9523810, (0 missing)
##       Age           < 44.5    to the right, improve=0.3150183, (0 missing)
##   Surrogate splits:
##       Income < 109366  to the left,  agree=0.690, adj=0.235, (0 split)
##       Age    < 35.5    to the left,  agree=0.643, adj=0.118, (0 split)
## 
## Node number 102: 25 observations,    complexity param=0.02173913
##   predicted class=Low     expected loss=0.6  P(node) =0.177305
##     class counts:     9    10     6
##    probabilities: 0.360 0.400 0.240 
##   left son=204 (10 obs) right son=205 (15 obs)
##   Primary splits:
##       Income        < 82119.5 to the right, improve=1.4533330, (0 missing)
##       Age           < 34      to the right, improve=0.8712821, (0 missing)
##       SpendingScore < 15.5    to the right, improve=0.5200000, (0 missing)
##   Surrogate splits:
##       SpendingScore < 3.5     to the left,  agree=0.68, adj=0.2, (0 split)
## 
## Node number 103: 17 observations
##   predicted class=High    expected loss=0.4705882  P(node) =0.1205674
##     class counts:     9     0     8
##    probabilities: 0.529 0.000 0.471 
## 
## Node number 204: 10 observations
##   predicted class=High    expected loss=0.4  P(node) =0.07092199
##     class counts:     6     3     1
##    probabilities: 0.600 0.300 0.100 
## 
## Node number 205: 15 observations
##   predicted class=Low     expected loss=0.5333333  P(node) =0.106383
##     class counts:     3     7     5
##    probabilities: 0.200 0.467 0.333
```

Explanation: The formula Segment ~ Age + Income + SpendingScore indicates that I predicted the Segment variable based on Age, Income, and SpendingScore. The method = "class" argument specifies that this is a classification problem.

Step 5: Visualizing the Decision Tree

Now, I will visualize the decision tree, which helps us understand the decision rules the model has learned.

```r
# Plot the decision tree
plot(treeModel)
text(treeModel, use.n = TRUE, all = TRUE, cex = 0.5)
```



Explanation: The plot() function generates a visual representation of the decision tree, while text() adds labels to the tree nodes. The use.n = TRUE argument includes the number of observations in each node, and cex = 0.8 controls the size of the text. The plot typically shows the decision tree with nodes and branches. Each node represents a decision rule (e.g., "Is Age < 40?"), and branches represent the outcomes of these decisions.

Step 6: Predicting and Evaluating the Model

After training the model, I use it to make predictions on the testing set and evaluate its accuracy using a confusion matrix.

```r
# Predict on the test data
predictions <- predict(treeModel, newdata = testData, type = "class")

#Ensure the reference variable is a factor
testData$Segment <- factor(testData$Segment)

# Predict on test data
predictions <- factor(predictions, levels = levels(testData$Segment))

# Generate the confusion matrix
confusionMatrix(predictions, testData$Segment)
```

```
## Confusion Matrix and Statistics
##
##             Reference
## Prediction High Low Medium
##     High      9   5      7
##     Low       2   5      5
##     Medium    8   9      9
##
## Overall Statistics
##
##                Accuracy : 0.3898
##                  95% CI : (0.2655, 0.5256)
##     No Information Rate : 0.3559
##     P-Value [Acc > NIR] : 0.3379
```

```
##
##                   Kappa : 0.0797
##
##  Mcnemar's Test P-Value : 0.4762
##
## Statistics by Class:
##
##                      Class: High Class: Low Class: Medium
## Sensitivity               0.4737    0.26316        0.4286
## Specificity               0.7000    0.82500        0.5526
## Pos Pred Value            0.4286    0.41667        0.3462
## Neg Pred Value            0.7368    0.70213        0.6364
## Prevalence                0.3220    0.32203        0.3559
## Detection Rate            0.1525    0.08475        0.1525
## Detection Prevalence      0.3559    0.20339        0.4407
## Balanced Accuracy         0.5868    0.54408        0.4906
```

Explanation: The predict() function applies the trained model to the test data to generate predictions. The confusionMatrix() function from the caret package compares the predictions to the actual segments, allowing us to evaluate the model's accuracy.

Step 7: Fine-Tuning the Model

To improve the model's performance, I can use the caret package to fine-tune the decision tree by cross-validating different complexity parameters.

```r
# Define the training control
train_control <- trainControl(method = "cv", number = 10)

# Train the model with caret, applying cross-validation
tunedModel <- train(Segment ~ Age + Income + SpendingScore,
                    data = trainData,
                    method = "rpart",
                    trControl = train_control)

# Print the best model
print(tunedModel$finalModel)
```

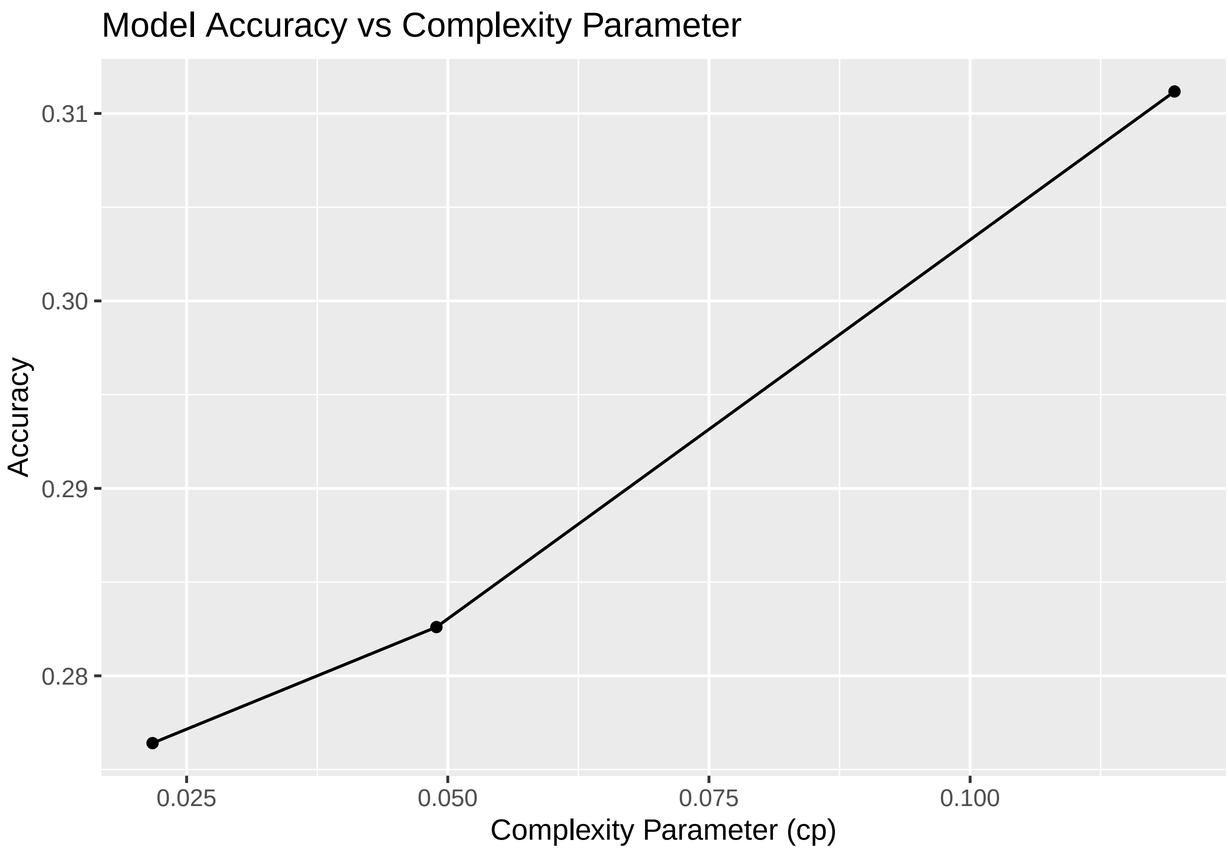
```
## n= 141
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
## 1) root 141 92 Medium (0.3333333 0.3191489 0.3475177) *
```

Explanation: The trainControl() function specifies that we want to use 10-fold cross-validation (method = "cv", number = 10). The train() function from the caret package then trains the model, tuning it by testing different values of the complexity parameter (cp).

Step 8: Visualization

Finally, I can visualize the performance of our tuned model using ggplot2, which allows us to plot the relationship between the complexity parameter and accuracy.

```r
# Plot the complexity parameter (cp) vs accuracy
ggplot(tunedModel) +
  geom_line(aes(x = tunedModel$results$cp, y = tunedModel$results$Accuracy)) +
  labs(x = "Complexity Parameter (cp)", y = "Accuracy") +
  ggtitle("Model Accuracy vs Complexity Parameter")
```



Explanation: Here, ggplot() initializes the plot, and geom_line() creates a line graph showing how the accuracy of the model varies with the complexity parameter (cp).