# Title: Decision Trees

Course: MBA business analytics Module: 07 Author: Reuben \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# Run this code before you start.

# load needed packages  
library(tidyverse)

## ── Attaching packages ─────────────────────────────────────── tidyverse 1.3.2 ──  
## ✔ ggplot2 3.3.6 ✔ purrr 0.3.5   
## ✔ tibble 3.1.8 ✔ dplyr 1.0.10  
## ✔ tidyr 1.2.1 ✔ stringr 1.4.1   
## ✔ readr 2.1.3 ✔ forcats 0.5.2   
## ── Conflicts ────────────────────────────────────────── tidyverse\_conflicts() ──  
## ✖ dplyr::filter() masks stats::filter()  
## ✖ dplyr::lag() masks stats::lag()

# Confusion Matrix function  
my\_confusion\_matrix <- function(cf\_table) {  
 true\_positive <- cf\_table[4]  
 true\_negative <- cf\_table[1]  
 false\_positive <- cf\_table[2]  
 false\_negative <- cf\_table[3]  
 accuracy <- (true\_positive + true\_negative) / (true\_positive + true\_negative + false\_positive + false\_negative)  
 sensitivity\_recall <- true\_positive / (true\_positive + false\_negative)   
 specificity\_selectivity <- true\_negative / (true\_negative + false\_positive)  
 precision <- true\_positive / (true\_positive + false\_positive)   
 neg\_pred\_value <- true\_negative/(true\_negative + false\_negative)  
 print(cf\_table)  
 my\_list <- list(sprintf("%1.0f = True Positive (TP), Hit", true\_positive),  
 sprintf("%1.0f = True Negative (TN), Rejection", true\_negative),  
 sprintf("%1.0f = False Positive (FP), Type 1 Error", false\_positive),  
 sprintf("%1.0f = False Negative (FN), Type 2 Error", false\_negative),  
 sprintf("%1.4f = Accuracy (TP+TN/(TP+TN+FP+FN))", accuracy),   
 sprintf("%1.4f = Sensitivity, Recall, Hit Rate, True Positive Rate (How many positives did the model get right? TP/(TP+FN))", sensitivity\_recall),  
 sprintf("%1.4f = Specificity, Selectivity, True Negative Rate (How many negatives did the model get right? TN/(TN+FP))", specificity\_selectivity),  
 sprintf("%1.4f = Precision, Positive Predictive Value (How good are the model's positive predictions? TP/(TP+FP))", precision),  
 sprintf("%1.4f = Negative Predictive Value (How good are the model's negative predictions? TN/(TN+FN)", neg\_pred\_value)  
 )  
 return(my\_list)  
}

We will now us the decision tree algorithm to address our business problem.

TECA is planning for the future and would like to set up their business so they are not so reliant on selling gasoline. One way to do that is to increase the sales of profitable products. Thus, TECA would like to predict when a transaction is a going to be a sale of a high gross profit margin product. Profit margin is the percentage of gross profit to revenue, or revenue minus costs divided by revenue. It is a percentage of how much profit each product makes or what percentage of profit is earned for each dollar of revenue.

The first thing we need to do is bring in the dataset we are going to work with. This is TECA data that has been transformed to be used in our decision tree analysis. Let’s load it and examine it a bit.

We first notice that the dataset has 200,000 rows and 13 features. This data is aggregated at the level of a purchase. Thus, each example or row is the purchase of one item.

* The first feature is the target feature or the dependent variable. This is the variable we are trying to predict. It is called high\_gpm. It is a factor that is either low or high.
* The second variable is a categorical variable loyalty2. This feature indicates whether the purchase was made by a loyalty customer (a customer who scanned their loyalty card) or not. Note, two things here. First, note that the data was adjusted so that about half of the purchases in this dataset were made by loyalty customers, since TECA is particularly interested in these customers. Second, note that the decision tree method does not require us to make all of our variables numerical. Thus, we do not have to convert our character/categorical features to multiple columns of dummy variables.
* Next we see revenue. This variable is the amount of revenue TECA makes for each purchase. Revenue is a continuous variable.
* Next we have the quarter feature. This variable indicates the quarter of the year in which the purchase took place.
* Next, income, bachelors\_degree, and population list these averages for the location of the store where the purchase took place.

Next, we have another multi-level factor variable, state\_province that lists the state the purchase happened in.

* Next, we have num\_trans, which indicates how many purchases were made by this same person as part of this transaction.
* Next, basket indicates whether the purchase was part of a multi-part transaction.
* Next, refill indicate whether or not the purchase was a refill of fountain soda.
* Area is a multi-level categorical variable that indicates the area of the store the purchase was made, as follows:
  + alcohol: products in this area include, for example, wine and beer;
  + cooler: products in this area include, for example, energy drinks, canned soda, and juice;
  + dispensed: products in this area include, for example, cold and hot dispensed (fountain) drinks;
  + fresh: products in this area include, for example, pizza, hot sandwiches, salads, and roller grill items;
  + fuel: products in this area include, for example, gas;
  + grocery: products in this area include, for example, milk, eggs, and cheese;
  + lottery: products in this area include, for example, lottery tickets;
  + miscellaneous: products in this area include, for example, store services and coupons;
  + nongrocery: products in this area include, for example, clothing, magazines, medicine, and newspapers;
  + snacks: products in this area include, for example, candy, gum, chips, and salty snacks; and
  + tobacco: products in this area include, for example, cigarettes and chewing tobacco.
* Next, items\_sold indicate how many of these items were purchased in this transaction.

# Explore the data

tree\_input <- readRDS("C:/Users/User/OneDrive/Documents/projects/decision.rds")  
str(tree\_input)

## 'data.frame': 200000 obs. of 13 variables:  
## $ high\_gpm : Factor w/ 2 levels "low","high": 1 1 1 1 1 2 2 1 2 1 ...  
## $ loyalty2 : Factor w/ 2 levels "not loyal","loyal": 1 1 1 1 1 1 1 1 1 1 ...  
## $ revenue : num 6.35 1.33 5 5.74 6.01 ...  
## $ quarter : int 3 1 3 2 2 2 4 3 3 1 ...  
## $ income : int 43438 58594 59468 38516 52379 61115 69084 44112 54614 65861 ...  
## $ bachelors\_degree: int 51 1167 3808 665 4088 264 8337 1649 1739 164 ...  
## $ population : int 279 8624 22756 9515 41584 2048 51804 19064 23523 1730 ...  
## $ state\_province : Factor w/ 10 levels "Wyoming","Iowa",..: 5 8 5 3 6 3 4 4 4 2 ...  
## $ num\_trans : int 1 1 1 1 1 1 1 1 1 1 ...  
## $ basket : Factor w/ 2 levels "no","yes": 1 1 1 1 1 1 1 1 1 1 ...  
## $ refill : Factor w/ 2 levels "no","yes": 1 1 1 1 1 1 1 1 1 1 ...  
## $ area : Factor w/ 11 levels "fuel","alcohol",..: 5 2 10 5 5 8 8 2 7 1 ...  
## $ items\_sold : num 1 1 1 1 1 1 1 1 1 1 ...

slice\_sample(tree\_input, n=10)

## high\_gpm loyalty2 revenue quarter income bachelors\_degree population  
## 193561 high loyal 0.99 2 58520 533 7692  
## 1670659 high not loyal 1.00 3 61500 129 1302  
## 262097 low loyal 5.13 4 57600 3483 24180  
## 19547 low loyal 2.00 2 61333 15 242  
## 138569 high not loyal 1.99 3 62353 88 757  
## 194792 low loyal 3.33 2 52379 4088 41584  
## 206009 low loyal 1.99 4 44635 1644 21360  
## 355657 high loyal 3.49 3 58386 7602 39333  
## 117521 high not loyal 2.26 4 59000 49 418  
## 261249 low loyal 2.39 2 57600 3483 24180  
## state\_province num\_trans basket refill area items\_sold  
## 193561 Arkansas 1 no no cooler 1  
## 1670659 Iowa 1 no no dispensed 1  
## 262097 Missouri 1 no no tobacco 1  
## 19547 Iowa 1 no no snacks 1  
## 138569 Wyoming 1 no no snacks 1  
## 194792 Arkansas 1 no no cooler 2  
## 206009 Alabama 1 no no fresh 1  
## 355657 Colorado 1 no no snacks 1  
## 117521 Nebraska 1 no no fresh 1  
## 261249 Missouri 1 no no cooler 1

Let’s look more in depth at the target feature. For tree analysis, the target feature is a categorical variable. In this implementation we can leave it as a factor. About 44% of these purchases are for high gross profit items.

# Explore the target feature

freq <- table(tree\_input$high\_gpm)  
freq[2]/(freq[1]+freq[2])

## high   
## 0.4341

contrasts(tree\_input$high\_gpm)

## high  
## low 0  
## high 1

Before using the algorithm, we need to prepare the data. The first line below loads the caret package. The next line sets the seed for the randomization that will be used for the data split. The caret::createDataPartition() function uses the caret package to split the data. Basically, it creates a number for p amount, in this case 0.75, of the target feature and lists these numbers in a matrix (since list is FALSE). Next, the training data and testing data are created. These use the numbers from the partition matrix that we just created. data\_train retains each of the rows with numbers in partition while data\_test takes the numbers not in partition, i.e., -partition. We use the set.seed() function with the same number every time to get a reproducible result every time we run this code, since the function relies on R’s random number generator. Of course, your results might not exactly replicate mine if your R version is different, etc.

# Partition the data

library(caret)

## Loading required package: lattice

##   
## Attaching package: 'caret'

## The following object is masked from 'package:purrr':  
##   
## lift

set.seed(77)  
partition <- caret::createDataPartition(y=tree\_input$high\_gpm, p=.75, list=FALSE)  
data\_train <- tree\_input[partition, ]  
data\_test <- tree\_input[-partition, ]

For decision trees, which don’t rely on distance to calculate splits, we do not need to standardize our data. Thus, we are ready to run our model. The tree() function creates our tree object. The first argument in the function is our formula. We are predicting the feature variable, high\_gpm on the left-hand side of the ~, and using all of our other features to predict it, as indicated by .. We do this, of course, with our training data. As usual, we hold back our test data to evaluate our model later.

# Run the model

#install.packages('tree')  
library(tree)

## Warning: package 'tree' was built under R version 4.2.2

model\_tree <- tree(high\_gpm ~ ., data\_train)

First, before we explore the tree and find out what it tells us, let’s see how accurate our model is. If this model is going to help TECA makes decisions, it needs to be accurate at predicting whether a purchase will be for a high gross profit margin product or not.

The first step in this process is using the model we trained on our training data to see how well it predicts on our holdout testing data. The predict function for the tree package does this for us. Notice that the testing data is passed as the second argument. The type='class' argument tells R to give the output as the actual class label–low and high in our case.

# Predict the model on the holdout testing data

predict\_tree <- predict(model\_tree, data\_test, type='class')

Next, let’s use the confusion matrix function created at the beginning of the notebook. Overall, we find that our model is very accurate. The model makes the correct prediction about 85% of the time!

Let’s explore some details of the accuracy of this model. The table shows the following output: \* When a transaction is actually/truthfully a high gross profit margin transaction, the model correctly classifies it as such-by saying “high”, 16186 times. This is called a True Positive (TP), Hit.

* When a transaction is truthfully a low gross profit margin transaction, the model correctly classifies it as such–by saying “low”, 26281 times. This is a True Negative (TN), Rejection.
* On the other hand, the model makes two kinds of errors. When the model transaction is not a high gross profit margin transaction, but the model incorrectly says it is, this is called a False Positive (FP), Type 1 Error. It happens 2014 times.
* Finally, when a transaction is a high gross profit margin transaction and the model says it is not, which here happens 5519 times, it is called a False Negative (FN), Type 2 Error.

These numbers can then be manipulated to create different measures of accuracy, as follows: \* Overall accuracy (TP+TN/(TP+TN+FP+FN)) is 0.8493.

* Sensitivity, Recall, Hit Rate, True Positive Rate (How many positives did the model get right? TP/(TP+FN)), is 0.7457.
* Specificity, Selectivity, True Negative Rate (How many negatives did the model get right? TN/(TN+FP)) is 0.9288.
* Precision, Positive Predictive Value (How good are the model’s positive predictions? TP/(TP+FP)) is 0.8893.
* Negative Predictive Value (How good are the model’s negative predictions? TN/(TN+FN) is 0.8264.

Overall, our model does a great job at predicting when a purchase will be high profit margin versus low profit margin and gets it right 85% of the time. The aspect of the model that is the least effective is its sensitivity and the aspect that is the most effective is the specificity. This means that the model is very good at classifying the low margin purchases successfully (specificity), but slightly worse at classifying the high margin purchases correctly (sensitivity). Thus, while the model is quite good overall, it particularly excels at identifying bad transactions. As we examine the tree below, it should become clear why low margin products are a bit easier to classify.

# Confusion matrix - checking accuracy

table1 <- table(predict\_tree, data\_test$high\_gpm)  
my\_confusion\_matrix(table1)

##   
## predict\_tree low high  
## low 26281 5519  
## high 2014 16186

## [[1]]  
## [1] "16186 = True Positive (TP), Hit"  
##   
## [[2]]  
## [1] "26281 = True Negative (TN), Rejection"  
##   
## [[3]]  
## [1] "2014 = False Positive (FP), Type 1 Error"  
##   
## [[4]]  
## [1] "5519 = False Negative (FN), Type 2 Error"  
##   
## [[5]]  
## [1] "0.8493 = Accuracy (TP+TN/(TP+TN+FP+FN))"  
##   
## [[6]]  
## [1] "0.7457 = Sensitivity, Recall, Hit Rate, True Positive Rate (How many positives did the model get right? TP/(TP+FN))"  
##   
## [[7]]  
## [1] "0.9288 = Specificity, Selectivity, True Negative Rate (How many negatives did the model get right? TN/(TN+FP))"  
##   
## [[8]]  
## [1] "0.8893 = Precision, Positive Predictive Value (How good are the model's positive predictions? TP/(TP+FP))"  
##   
## [[9]]  
## [1] "0.8264 = Negative Predictive Value (How good are the model's negative predictions? TN/(TN+FN)"

Next, let’s examine the details of the tree we have created and what it might tell us that might help TECA increase their sales of profitable products. We can start this process very simply by using the summary() function on our tree object, model\_tree. We learn a couple very useful things here. First, we learn that only two variables were used in our tree–area and revenue. Next, we are told how many terminal nodes exist. Finally, we are given some error measurements. Residual mean deviance is a measure of variance in the model and misclassification error rate is measure of how many examples were misclassified. It shows that 22,723, or about 15%, of our total observations were incorrectly classified (split into the wrong partition).

# Summarize the results from our model

summary(model\_tree)

##   
## Classification tree:  
## tree(formula = high\_gpm ~ ., data = data\_train)  
## Variables actually used in tree construction:  
## [1] "area" "revenue"  
## Number of terminal nodes: 6   
## Residual mean deviance: 0.5741 = 86110 / 150000   
## Misclassification error rate: 0.1515 = 22723 / 150000

Next, let’s look at the tree in text form. To do this, we simply type the name of the tree object that we just created. This is our tree, albeit in text form. It gives us quite a lot of information. the first line is the key to reading the table, node), split, n, deviance, yval, (yprob). It says that each line starts with a numbered node, lists the equation used to split the data, the number of observations following the left side of the branch, the deviance associate with the branch, the predicted value at the node, and the proportion of the values at the branch that are absent and present. This is very informative, but let’s also plot the tree so that we can get a fuller picture (literally) and discuss what the tree is telling us.

# Tree in text form

model\_tree

## node), split, n, deviance, yval, (yprob)  
## \* denotes terminal node  
##   
## 1) root 150000 205300.00 low ( 5.659e-01 4.341e-01 )   
## 2) area: fuel,alcohol,tobacco,lottery,miscellaneous 52207 64.59 low ( 9.999e-01 5.746e-05 ) \*  
## 3) area: snacks,fresh,grocery,cooler,dispensed,nongrocery 97793 124600.00 high ( 3.342e-01 6.658e-01 )   
## 6) area: snacks,grocery,cooler 51950 71380.00 low ( 5.553e-01 4.447e-01 )   
## 12) revenue < 1.495 8775 10050.00 high ( 2.597e-01 7.403e-01 ) \*  
## 13) revenue > 1.495 43175 57530.00 low ( 6.154e-01 3.846e-01 ) \*  
## 7) area: fresh,dispensed,nongrocery 45843 26360.00 high ( 8.363e-02 9.164e-01 )   
## 14) area: fresh,nongrocery 22767 20640.00 high ( 1.684e-01 8.316e-01 )   
## 28) revenue < 2.21 11985 14060.00 high ( 2.733e-01 7.267e-01 ) \*  
## 29) revenue > 2.21 10782 4397.00 high ( 5.185e-02 9.482e-01 ) \*  
## 15) area: dispensed 23076 0.00 high ( 0.000e+00 1.000e+00 ) \*

These two functions plot the tree. The first one gives the outline and the second one inserts the text. The heights of the lines are proportional to the decrease in impurity. Thus, the longer the line, the more helpful the split at reducing messiness in the model. The all=TRUE option adds extra labels, cex=.75 provides the size of things, font=2 changes the font, digits=2 changes the number of digits following the decimal point, and pretty=0 keeps the labels in the plot.

So, what do we learn from the text above and the plot? As we move through the tree, moving to the left is “yes” while moving to the right is “no”. “Low” and “high” at the leaf nodes indicates that is the prediction of the profitability of the purchase.

We actually learn a lot that can help TECA. The first, and most significant split that determines whether a purchase will be for a high or low gross profit is whether the purchases is for fuel, alcohol, tobacco, lottery, or miscellaneous. If the answer to this is yes, we move to the left of the tree and the product is very likely to be low profitability. While the management team at TECA surely knows which products are high and low profit. This stark result should make it clear that if individual profitability is the primary concern of management it is important to 1) get customers into the store, and 2) get customers to move beyond just the traditional “vice” products. To the extent management can convert its stores to locations where people do more than just grab cigarettes, beer and lottery tickets after grabbing some gas, the more profitable they will be. Clearly there are other considerations for management, and these products drive traffic into the store and increase revenue, but management can use this knowledge to increase the purchase of higher margin products, perhaps even in concert with these vice products.

Let’s explore the rest of the plot. If participants instead buy snacks, fresh, grocery, cooler, dispensed, or nongrocery items the model gets a bit more complicated. If customers buy snacks, grocery, or cooler items the profitability of the purchase depends on revenue, such that lower profit actually is more likely to lead to higher profitability. This is also likely useful information for TECA. This helps them realize that even the smaller priced items, from the correct areas of the store, are profitable. This might lead to promotion of the products by placing them in the front of coolers or on the counter at checkout. If the customer instead purchases fresh, dispensed, or nongrocery items the data is split between dispensed and fresh and nongrocery. Dispensed items are predicted to be high profit. Thus, TECA should continue promoting and facilitating fountain drinks and coffee. Finally, if fresh food and nongrocery items are purchased, profitability is also predicted to be high, whether revenue is above or below $2.21.

# Plot the tree

plot(model\_tree)  
text(model\_tree, all=TRUE, cex=.75, font=2, digits=2, pretty=0)

