Classification and Regression Trees
Part 1: Introduction to Classification Trees

Aram Balagyozyan

Department of Operations and Information Management
Kania School of Management
The
University of Scranton

August 18, 2020

Starter Case: *Deja vu*

- Consider again the case of Universal Bank that is attempting to identify customers who are likely to accept a loan offer in the future.
- The bank has a dataset that includes 5000 customers. The data include the customers' response to the last personal loan campaign (`Personal Loan`), as well as customer demographic information (`Age`, `Education`, `Income`, etc.), and the customer's relationship with the bank (`Mortgage`, `SecuritiesAccount`, etc.).
- The data are stored in file *UniversalBank.csv*.
- Among these 5000 customers, only 480 (= 9.6%) accepted the personal loan offered to them in a previous campaign.

Starter Case: *Deja vu*

- Let's again download the data and run a quick visualization

```
bank.df<-read.csv("UniversalBank.csv")
#Drop ID and Zip Code columns
bank.df<-bank.df[,-c(1,5)]
```

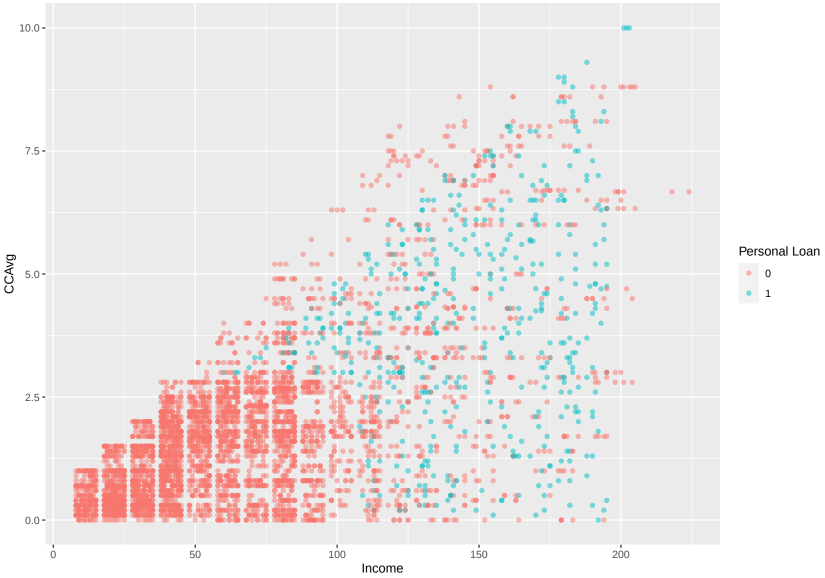
Starter Case: *Deja vu*

- Just for your reference, below again is the data dictionary

Variable	Description
Age	Customer's age in completed years
Experience	Number of years of professional experience
Income	Annual income of the customer ($000s)
Family	Family size of the customer
CCAvg	Average spending on credit cards per month ($000s)
Education	Undergrad; Graduate; Advanced/Professional
Mortgage	Value of house mortgage if any ($000s)
PersonalLoan	1 if customer has accepted a personal loan offer in the past
SecuritiesAccount	1 if customer has securities account with bank
CDAccount	1 if customer has certificate of deposit (CD) account with the Bank
Online	1 if customer uses Internet banking facilities
CreditCard	1 if customer uses credit card issued by the Bank

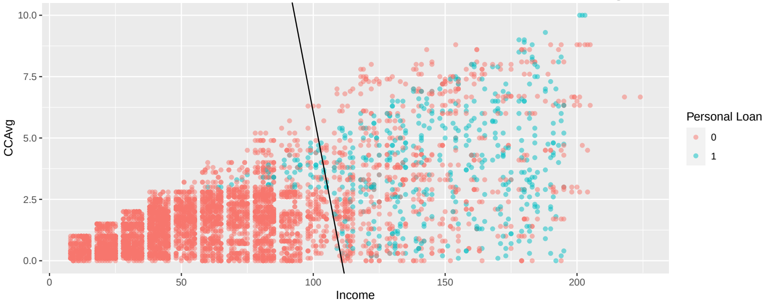
- Suppose again the analytics department at the bank is interested to find out if knowledge about the customer's income and family size can help to classify the customer as someone who will likely accept a personal loan offer. The plot on the following slide may shed light.

Starter Case: *Deja vu*



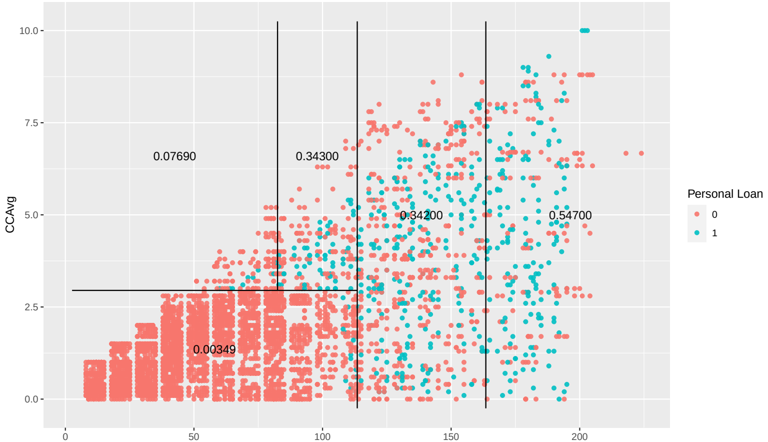
Starter Case: *Deja vu*

- In Module 7, we learned how to estimate a logistic regression and obtain a **linear** separation rule such that given a customer's income and the average credit-card spending, we could classify the customer s someone who will likely accept a loan offer. And example of the separation line is presented on the graph below:



Starter Case: *Deja vu*

- As it must be obvious from the graph above and from the classifier accuracy measures that we obtained in Module 7, the linear separation line does a fairly poor job in separating acceptors from non-acceptors.
- It would be great if our *non-linear* separation scheme looked like the following.

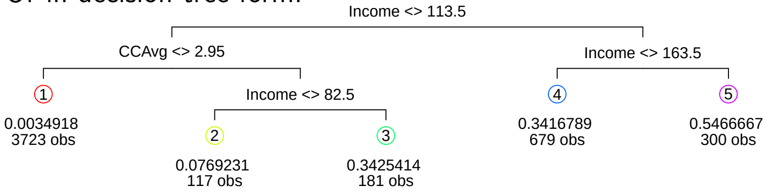


Starter Case: *Deja vu*

- The resulting decision boundaries would result in a rule-based classification as follows:
  1. If Income < 82.5 **and** CCAvg < 2.95, then there is only a 0.349% chance that the customer will accept the offer.
  2. If Income < 82.5 **and** CCAvg > 2.95, then the chance of acceptance is 7.69%.
  3. If $82.5 <$ Income < 113.5 **and** CCAvg > 2.95, there is a 34.25% probability that the customer will accept the offer.
  4. If $113.5 <$ Income < 163.5, there is 34.17% chance that the customer will accept the offer.
  5. If Income > 163.5 then there is 54.67% chance that the customer will accept the loan offer.

Starter Case: *Deja vu*

- Or in decision-tree form:



- Such non-linear decision boundaries and rules can be produced using classification (and regression) trees.
- The equivalence of the tree above and the previous set of rules for classifying must be obvious. For example, by looking at the leftmost *terminal node*, we can state that IF a customer's ACCAvg < 2.95 AND Income $<$ \$113.5K there is only a 0.035% chance that the customer will accept the loan offer.

Classification Trees

- Classification (and regression) trees (developed by Brieman et al. 1984) are one of the most flexible and easily interpretible classification methods.
- This method can be mostly used for classification but, as we will demonstrate later, can also be used as a tool for predicting numerical outcomes.
- Two key ideas underlie classification trees:
  1. *Recursive Partitioning* of the space pf the predictor variables.
  2. *Pruning of a tree* using validation data.
- We will describe these two ideas below.

Recursive Partitioning

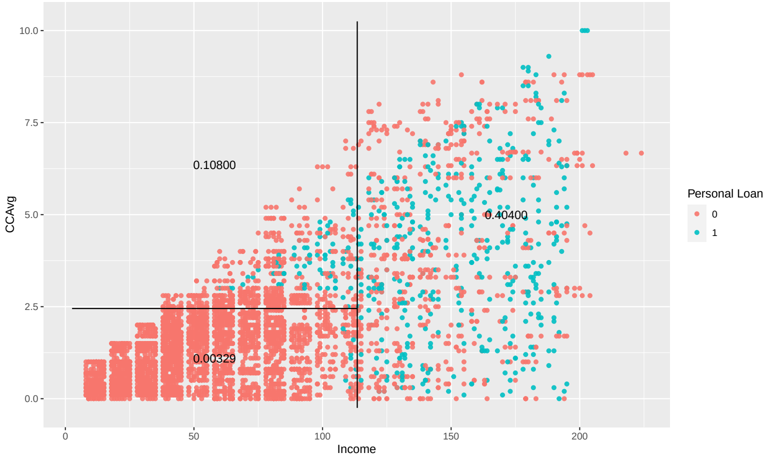
- How is the entire space of predictors (in our case `CCAvg` and `Income`) is partitioned into classification rectangles?
- Consider again our example above.

Recursive Partitioning

- First, one of the predictors (say `Income`) and a value of the predictor (say $113.5K) are optimally chosen to split the 2-dimensional space (of `Income` and `CCAvg`) into two rectangular parts: one part that contains all the points with `Income < 113.5` and the other with all the points with `Income > 113.5`.
- Then, one of these two parts is divided in a similar manner by again optimally choosing a predictor variable (it could be `Income` or `CCAvg` but in our case it is `CCAvg`) and a split value for that variable (e.g. 2.95). This results in three (two-dimensional) rectangular regions.
- This process is continued so that we get smaller and smaller rectangular regions.
- The idea is to divide the entire space of `Income` and `CCAvg` up into rectangles such that each rectangle is as homogeneous or "pure" (in terms of loan acceptance) as possible.
- By pure, we mean containing records that belong to just one class.

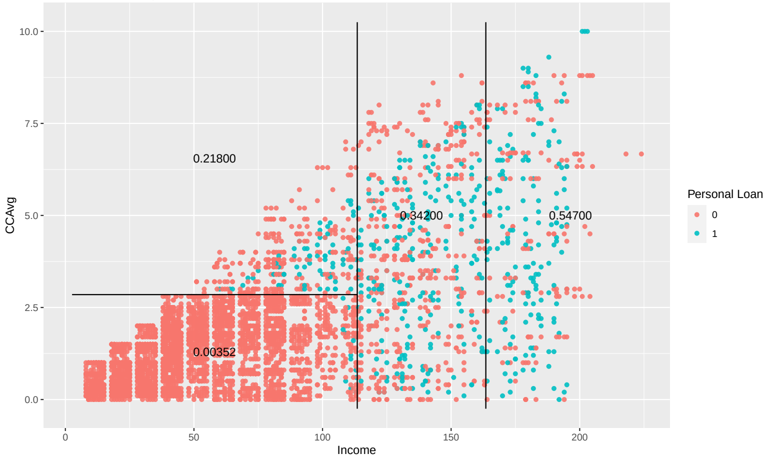
Recursive Partitioning

- As a graphical demonstration, the first two steps of the the tree algorithm above will make the following three partitions, with `Income` partitioned first and `CCAvg` second:



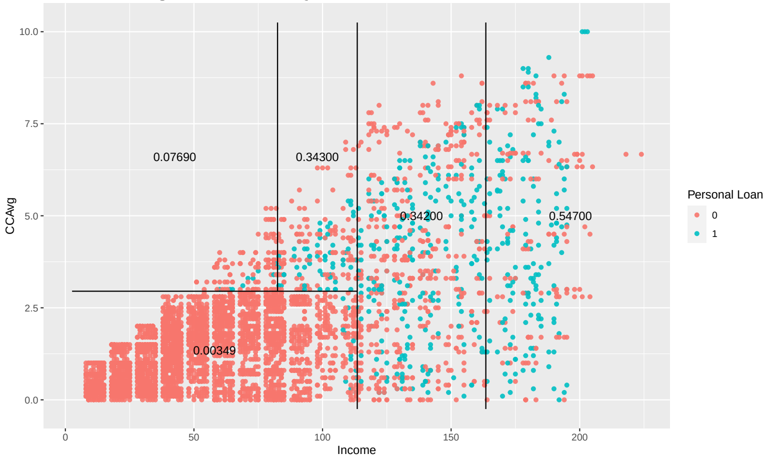
Recursive Partitioning

- At the next step, the algorithm partitions the predictor space as the following:



Recursive Partitioning

- The following step adds yet another partition:



Recursive Partitioning More Generally

- The same idea of recursive partitioning holds for the case when there are more than two predictor variables.
- With *i* predictors variables, a predictor and its value are optimally selected at first to split the entire *i*-dimensional space of predictors into two parts.
- Then one of the parts is split in a similar manner by again optimally choosing a predictor variable (it could be the same or a different variable as in the previous step) and a split value for that variable.
- This process continues so that we get smaller and smaller and increasingly homogeneous rectangular (*i*-dimensional) regions.

Questions Looking Ahead

- Two big questions that we must answer are:
  1. How is each particular split selected?
  2. How many partitions should the algorithm make? It can keep on partitioning the data until each partition contains only one class. Although this will produce a perfect classifier for the training set, it will almost certainly be excessively precise and thus overfitted. The question is then at what point it is optimal for the algorithm to stop partitioning?

How is Each Split Selected?

- The algorithm examines each predictor variable (in this case, Income and CCAvg) and all possible split values for each variable to find the best split.
- The possible split points for each variable are simply the distinct values for each predictor. These split points are ranked according to how much they reduce impurity (heterogeneity) in the resulting rectangle.
- A pure rectangle is one that is composed of a single class (e.g., acceptors).
- The reduction in impurity is defined as overall impurity before the split minus the sum of the impurities for the two rectangles that result from a split.

Measures of Impurity: The Gini Index

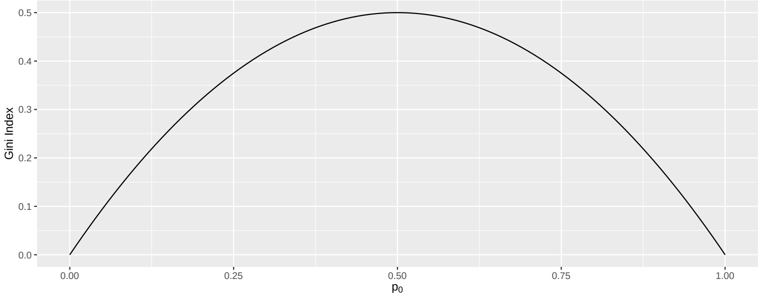
- There are a number of ways to measure impurity. The two most popular measures are the *Gini index* and an *entropy measure*. We describe both next.
- Denote the m classes of the response variable by $k = 1, 2, ..., m$. The Gini impurity index for a rectangle A is defined by:

$$I(A) = 1 - \sum_{k-1}^{m} p_k^2$$

  where p_k is the proportion of records in rectangle A that belong to class k. This measure takes values between 0 (when all the records belong to the same class) and $(m-1)/m$ (when there is an equal number of elements in each of the m classes).
- For example, if a given rectangle A has 20 observations with 2 responses = 1 and 18 responses = 0 (quite pure), then $p_1 = 2/20 = 0.1$ and $p_0 = 18/20 = 0.9$. Thus, $I(A) = 1 - 0.1^2 - 0.9^2 = 0.18$

Measures of Impurity: The Gini Index

- On the other hand, if a rectangle has an equal number of 0s and 1s, ($p_1 = 10/20 = 0.5$ and $p_0 = 10/20 = 0.5$), then $I(A) = 1 - 0.5^2 - 0.5^2 = 0.5$
- The figure below shows the values of the Gini index for a two-class case (0 and 1) as a function of p_0. It can be seen that the impurity measure is at its peak when $p_k = 0.5$ (i.e., when the rectangle contains 50% of each of the two classes)



Measures of Impurity: Entropy

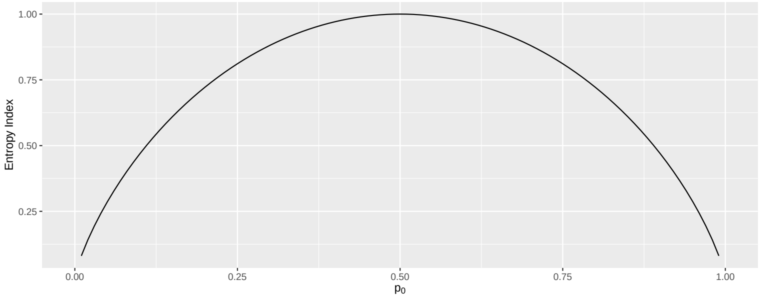
- The entropy for a rectangle A is defined by

$$entropy(A) = -\sum_{k=1}^{m} p_k \log_2(p_k)$$

- To compute $\log_2(x)$ in R, use function `log2()`.
- The measure of entropy ranges between 0 (most pure, all records belong to the same class) and $\log_2(m)$ (when all m classes are represented equally).
- In the two-class case, the entropy measure is maximized (like the Gini index) at $p_k = 0.5$.
- For the same example, if a given rectangle A has 20 observations with 2 responses = 1 and 18 responses = 0 (quite pure), then $p_1 = 2/20 = 0.1$ and $p_0 = 18/20 = 0.9$.
- Thus, in this case, $entropy(A) = -(p_0 \log_2(p_0) + p_1 \log_2(p1)) = -(0.9 \log_2(0.9) + 0.1 \log_2(0.1))$=0.4689956

Measures of Impurity: Entropy

- On the other hand, if a rectangle has an equal number of 0s and 1s, ($p_1 = 10/20 = 0.5$ and $p_0 = 10/20 = 0.5$), then $entropy(A) = -(0.5\log_2(0.5) + 0.5\log_2(0.5))$=1
- The figure below shows the values of entropy for a two-class case (0 and 1) as a function of p_0. You can see again that the impurity measure is at its peak when $p_k = 0.5$ (i.e., when the rectangle contains 50% of each of the two classes)



Measures of Impurity: Entropy

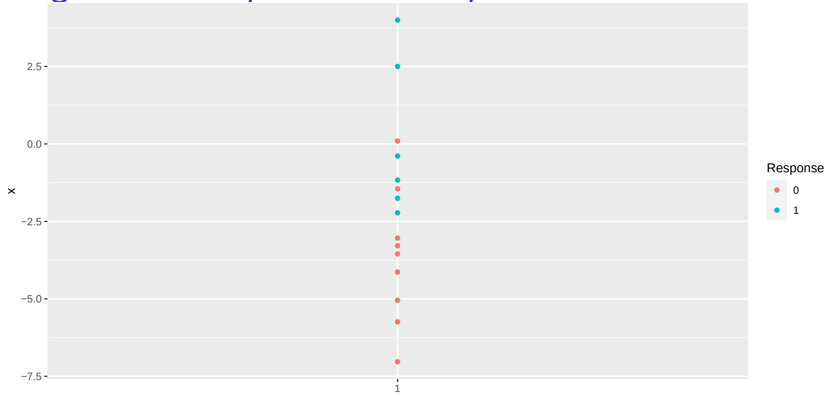
- Note that if a rectangle consists of data of only only 1 class, then at the first glance, the measure of entropy is undefined.
- Say the rectangle consists of 0s only. Then $p_0 = 1$, $p_1 = 0$, and $\mathit{entropy} = -(1\log_2(1) + 0\log_2(0))$.
- The first element $1\log_2(1) = 0$.
- Although $\log(0)$ in the second element is undefined, $\lim_{x\to 0} x\log_2(x) = 0$.
- Thus, when a rectangle is perfectly pure (populated by data of only one class), $\mathit{entropy} = -(1\log_2(1) + 0\log_2(0)) = 0$.

Picking the Best Split: An Example

- Consider the following hypothetical dataset

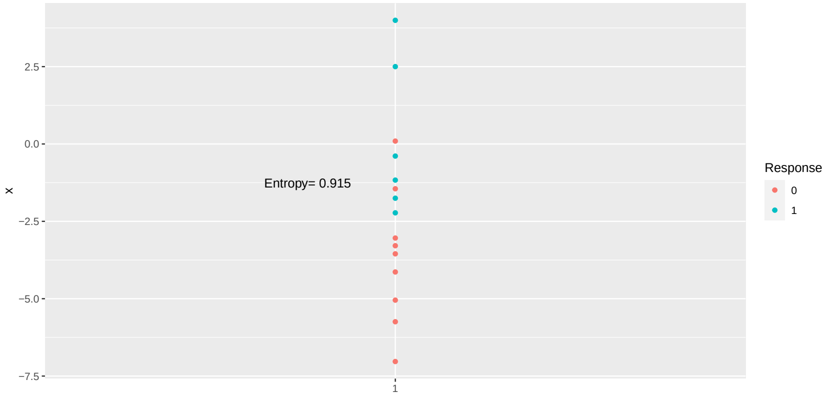
Response	x
1	3.9965056
1	2.5000429
0	0.0933411
1	-0.3906333
1	-1.1674150
0	-1.4470587
1	-1.7527098
1	-2.2248640
0	-3.0394899
0	-3.2877423
0	-3.5492384
0	-4.1362087
0	-5.0459569
0	-5.7442179
0	-7.0301114

Picking the Best Split: An Example



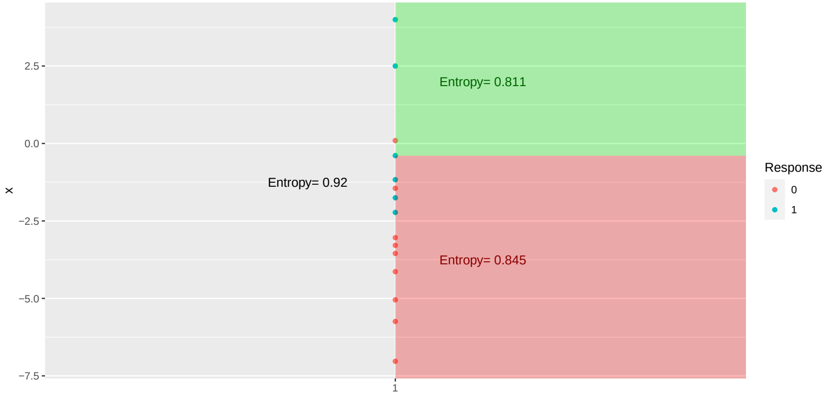
- Let's concentrate on only one of the measures of impurity, say entropy (picking the Gini index would not make much difference).
- If the dataset isn't partitioned, then $p_0 = 10/15 = 0.6667$ and $p_1 = 5/15 = 0.33$, then
- Entropy$= -(0.33\log_2(0.33) + 0.67\log_2(0.67)) =$ 'r round(binaryEntropy(0.33),3)

Picking the Best Split: An Example



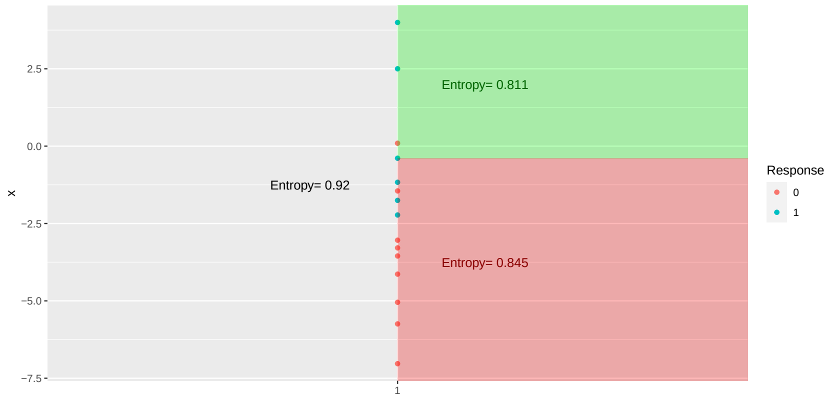
Picking the Best Split: An Example

- Suppose now we partition the dataset arbitrarily at one of the values of x = -0.39063326



- In the top rectangle, $p_0 = 1/4 = 0.25$ and $p_1 = 3/4 = 0.75$.
  - Entropy: $-(0.25 \log_2(0.25) + 0.75 \log_2(0.75)) = 0.811$
- In the bottom rectangle, $p_0 = 8/11 = 0.73$ and $p_1 = 3/11 = 0.27$.
  - Entropy: $-(0.73 \log_2(0.73) + 0.27 \log_2(0.27)) = 0.845$

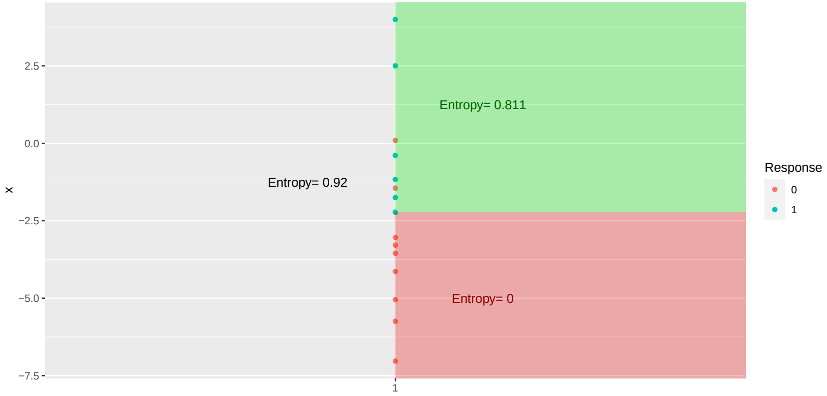
Picking the Best Split: An Example



- Thus, by splitting the dataset at x = -0.39063326, we reduced the overall impurity in the data (as measured by entropy) from 0.918 to (0.811+0.845)/2=0.828
- An overall reduction of 0.09

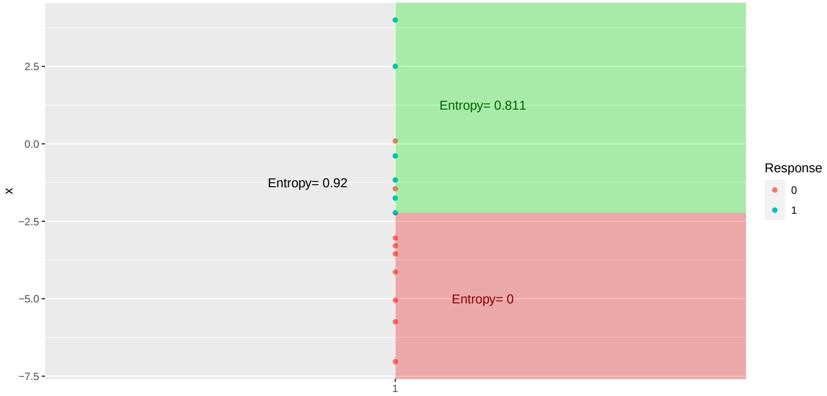
Picking the Best Split: An Example

- Suppose we now pick another partition, say at x = -2.22486401



- In the top rectangle, $p_0 = 2/8 = 0.25$ and $p_1 = 6/8 = 0.75$.
  - Entropy: $-(0.25 \log_2(0.25) + 0.75 \log_2(0.75)) = 0.811$
- The bottom rectangle consists of 0s only, $p_0 = 7/7 = 1$ and $p_1 = 0/7 = 0$.
  - Entropy of a pure rectangle $= 0$

Picking the Best Split: An Example



- Thus, by splitting the dataset at at x = -2.22486401, we reduced the overall impurity in the data (as measured by entropy) from 0.91 to (0.811+0)/2=0.423
- An overall reduction of 0.496

Picking the Best Split: An Example

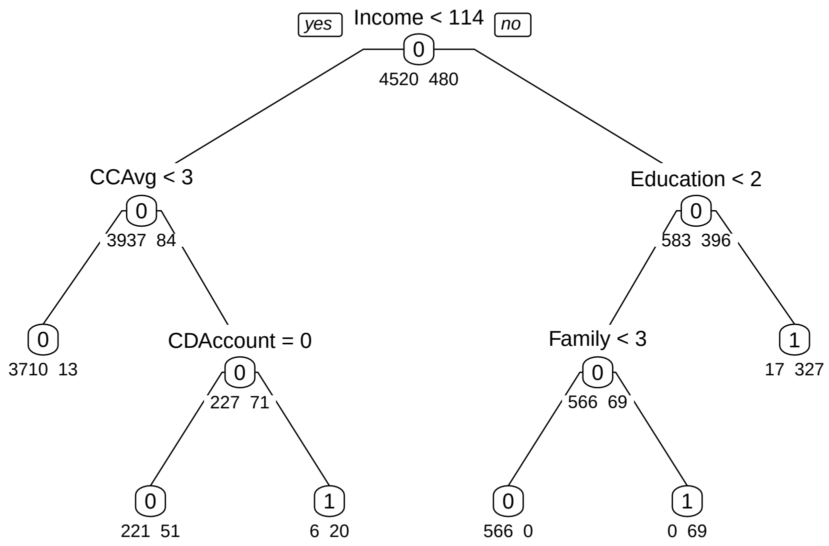
- The latter split at x = -2.22486401 is a better split than the former split at x = -0.39063326 because it reduces the overall impurity (entropy) in the data by much more (by 0.496 as opposed to 0.09).
- By comparing the reduction in impurity across all possible splits we can decide the best possible data partition.
- In reality, data usually have more than one predictor. To make a single partition, the tree algorithm combs through not only the possible partitions of one predictor but also through all possible partitions across all different predictor variables and picks the one that reduces impurity in rectangles the most.
- By continually splitting the data across different predictors, the tree procedure chooses splits that increase the purity of the resulting rectangles.

Classification Tree Using **rpart**

- There are several R packages that can help you grow a classification tree in R. One of the most polished ones is **rpart**
- To produce a classification tree for the loan acceptance data using the **rpart** library follow the code below.

```r
library(rpart)
library(rpart.plot)
# use rpart() to run a classification tree.
rpart.tree <- rpart(PersonalLoan~ .,
                data = bank.df,
                maxdepth = 3,
                method = "class")
## plot tree
# use prp() to plot the tree.
prp(rpart.tree, type = 1, extra = 1, under = TRUE,
split.font = 1, varlen = -10)
```

Classification Tree Using **rpart**



Interpreting Numbers in a **rpart** Tree

- We have two types of nodes in a tree: *decision nodes* and *terminal nodes*. Nodes that have successors are called decision nodes. Terminal nodes are also sometimes called *leaves*. The top decision node is often referred to as the *root* node
- It is useful to note that the type of trees grown by R's `rpart()` function have the property that the number of terminal nodes is exactly one more than the number of decision nodes. For example, the tree above has 6 terminal and 5 decision nodes.
- How does one read the tree?
- Looking at the top node and considering the numbers below the the node, we can say that there are 5000 responders (4520 + 480) in the data set, among whom, 4520 have reported "No" and 480 have responded "Yes" to the loan offer (I know, it's a little confusing, the "no" and "yes" rectangles above the node split the class into Income $>=$ 114 and Income $<$ 114).
- The number inside the node reflects the majority class. In the root node, the majority (4520/5000) responded "No" (0).

Interpreting Numbers in a **rpart** Tree

- The name of the variable chosen for splitting and its splitting value are above each decision node. The numbers below a node are the number of records in that node that had values lesser than (left side) or larger or equal to (right side) the splitting value. ("yes" and "no" tags at the top node clarify on which side we find the lesser values)
- Among the 5000 responders, three are $583 + 396 = 979$ responders with Income $>= 114$ and $3973 + 84 = 4021$ responders with Income < 114.

Interpreting Numbers in a **rpart** Tree

- Moving to the second-level nodes on the right. Among the 979 responders whose income is $>=$ 114, 396 responded "Yes" and 583 responded "No" to the loan offer. Since the majority responded "No" (583 out of 979), the node is marked with zero.
- Among those 979 high-income responders, 17 + 327 =344 have Education $>=$ 2 and 566 + 69 =535 have Education $<$ 2.
- Among those 344 high-income, high-education responders, 327 responded "Yes" and 17 responded "No" to the loan offer. Since the majority (327 out of 344) responded "Yes", the rightmost leaf on the above tree indicates 1.
- The other two leaves with a majority responders accepting the loan offer are:
  1. Responders with Income $>=$ 114, Education $<$ 2, and Family $>=$ 3 (69 out of 69 accepted the loan offer)
  2. Responders with Income $<$ 114, CCAvg $>=$ 3, CDAccount $=$ 1 (20 out of 26 accepted the loan offer)

Classifying a New Record

- To classify a new record, it is "dropped" down the tree. When it has dropped all the way down to a terminal node, we can assign its class simply by taking a "vote" of all the training data that belonged to the terminal node when the tree was grown.
- The class with the highest vote is assigned to the new record. For instance, a new record reaching the leftmost terminal node in the tree above, which has a majority of records that belong to the non-responder class, would be classified as "non-responder."
- Alternatively, if a single class is of interest, we could counts the proportion (propensity) of "votes" for this class then compares it to a user-specified cutoff value.
- In a binary classification situation, we can also establish a lower cutoff to better capture those rare successes (at the cost of lumping in more failures as successes).