Decision Tree

Supervised ML Algorithm

Sahasra Ranjan

April 2020

A decision tree is a map of the possible outcomes of a series of a related choices. It allows to weigh possible actions against one another based of various factors.

It uses a tree-like model of decision. It typically starts with a single node, which branches into possible outcomes. Each of those outcomes leads to additional nodes, which branch off into other possiblities.

Day	*Outlook*	*Temperature*	*Humidity*	*Wind*	*PlayTennis*
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Figure 1: Dataset for possiblity of a tennis match

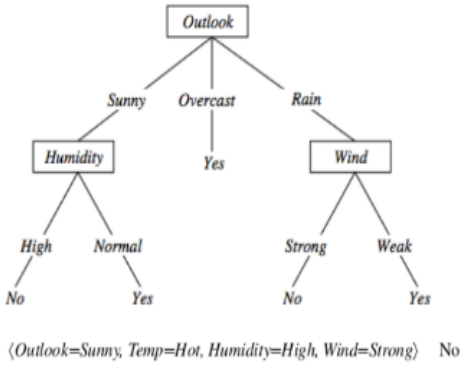

Figure 2: Decision Tree for the same

Advantages and Disadvantages of Decision Trees

Advantages:

- Performs classification without requiring much computation.
- Provides clear indication of important fields for prediction of classification.

Disadvantages:

- Less appropriate for predicting continious attributes.
- Computationally expensive to train

Creating a Decision Tree

For every node, we have to create subtrees with all the possiblities. and then further repeat for other features.

For eg., In the tennis match problem, for the first node let's check outlook, since having three possiblity (viz. Sunny, Overcast, Rainy), we created three subtrees and then further we keep asking for other features like Humidity & wind to get the final tree.

Greedy Approach for creating decision tree

Greedy approach is implemented by making an optimal local choice at each node. By making these local optimal choices, we reach the approximate optimal solution globaly.

The algorithm can be summarized as:

1. At each stage (node), pick out the best feature as the test conditiion.
2. Now split the node into possibel outcomes (internal nodes)
3. Repeat the above steps till all the test conditions have been exhausted into leaf nodes.

Continuous Features

There might be some feature which are not categorical, for these we need to create possibilities on the basis of aprropriate ranges. One such tree is shown below:

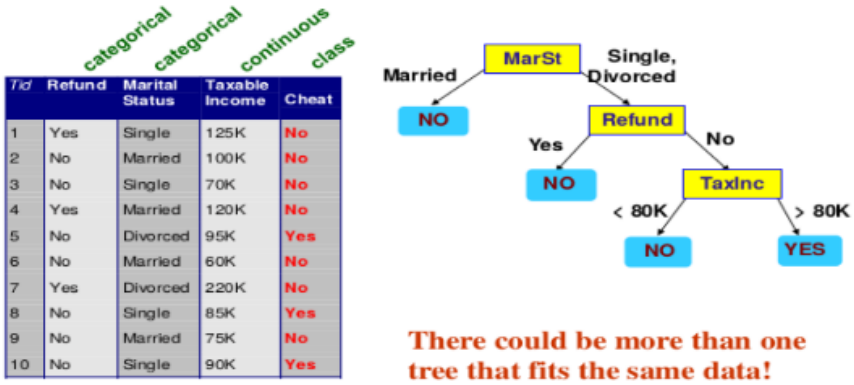

Figure 3: Decision tree with continuous feature

Entropy

In the most layman terms, Entropy is nothing but the **The measure of disorder**. Why is it important to study entropy for machine learning?

Entropy is a measure of disorder or uncertainity and the goal of machine learning models and Data Scientists in general is to reduce uncertainity.

The Mathematical formula for entropy is -

$$E(S) = \sum_{i=1}^{c} -p_i log_2 p_i \tag{1}$$

Where p_i is the frequentist probability of an element/class i in out data.

Let's say we have only two classes, a positive and a negative class. Out of 100 data, suppose that 70 belongs to $-$ve class and 30 to $+$ve. Then, P$+$ will be 0.3 and P$-$ will be 0.7. Entropy E will be given by:

$$E = -\frac{3}{10} \times log_2 \frac{3}{10} - \frac{7}{10} \times log_2 \frac{7}{10} \approx \mathbf{0.88} \tag{2}$$

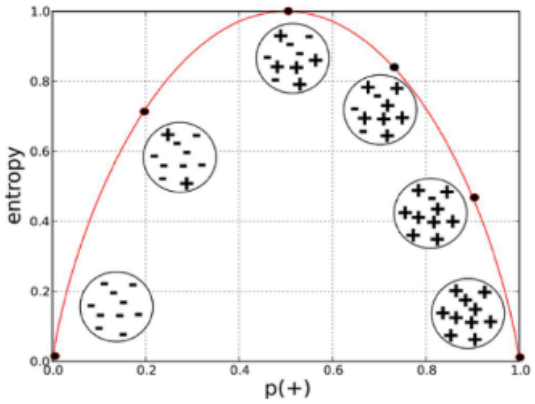

Figure 4: Entropy distribution frequentist probability

Information Gain

Information gain is basically how much Entropy is removed after training a decision tree.
Higher information gain = more entropy removed.
In technical terms, Information Gain from X on Y is defined as:

$$IG(Y, X) = E(Y) - E(Y|X) \tag{3}$$

Basics of inforation gain is well explained here: A Simple Explanation of Information Gain and Entropy

Example: Decision Tree

Consider an example where we are building a decision tree to predict whether a loan given to a person would result in a write-off or not. Our entire population consists of 30 instances. 16 belong to the write-off class and the other 14 belong to the non-write-off class. We have two features, namely "Balance" that can take on two values: "$<$ 50K" or "$>$ 50K" and "Residence" that can take on three values: "OWN", "RENT" or "OTHER". I'm going to show you how a decision tree algorithm would decide what attribute to split on first and what feature provides more information, or reduces more uncertainty about our target variable out of the two using the concepts of Entropy and Information Gain. The dots are the data points with class right-off and the

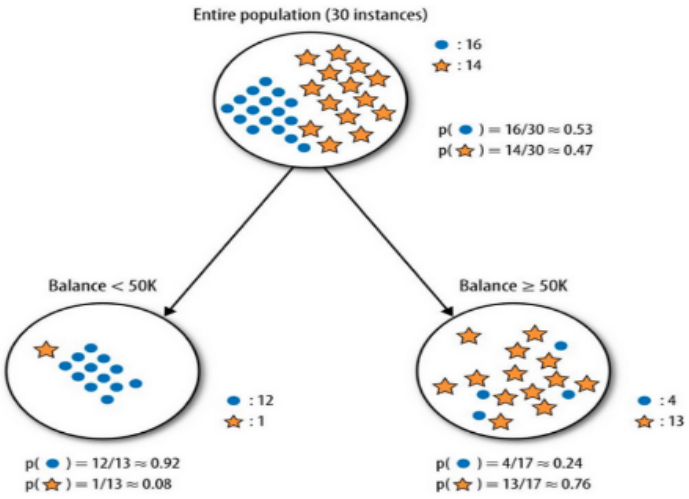

Figure 5: Feature 1: Balance

stars are the non-write-offs. Splitting the parent node on attribute balance gives us 2 child nodes. The left node gets 13 of the total observations with 12/13 (0.92 probability) observations from the write-off class and only 1/13(0.08 probability) observations from the non-write of class. The right node gets 17 of the total observation with 13/17(0.76 probability) observations from the non-write-off class and 4/17 (0.24 probability) from the write-off class.

Let's calculate[1] the entropy for the parent node and see how much uncertainty the tree can reduce by splitting on Balance. Splitting on feature ,"Balance" leads to an information gain of 0.37 on our target variable. Let's do the same thing for feature, "Residence" to see how it compares.

Splitting the tree on Residence gives us 3 child nodes. The left child node gets 8 of the total observations with 7/8 (0.88 probability) observations from the write-off class and only 1/8 (0.12 probability) observations from the non-write-off class. The middle child nodes gets 10 of the total observations with

[1]See this for calculations and further refrences: Entropy: How Decision Trees Make Decisions

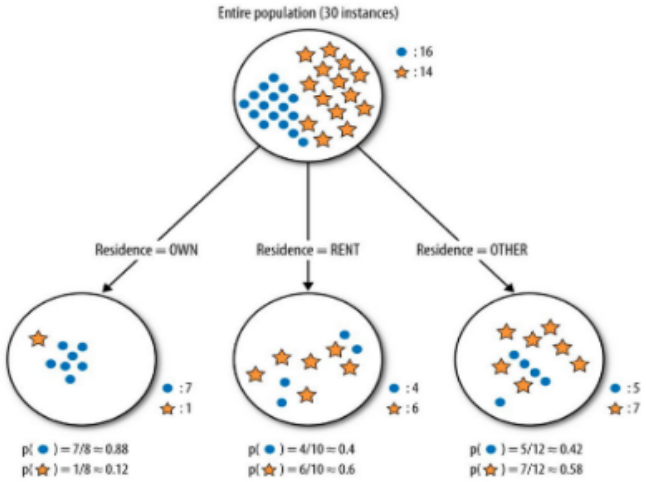

Figure 6: Feature 2: Residence

4/10 (0.4 probability) observations of the write-off class and 6/10(0.6 probability) observations from the non-write-off class. The right child node gets 12 of the total observations with 5/12 (0.42 probability) observations from the write-off class and 7/12 (0.58) observations from the non-write-off class. We already know the entropy for the parent node. We simply need to calculate the entropy after the split to compute the information gain from "Residence"

The information gain from feature, Balance is almost 3 times more than the information gain from Residence! If you go back and take a look at the graphs you can see that the child nodes from splitting on Balance do seem purer than those of Residence. However the left most node for residence is also very pure but this is where the weighted averages come in play. Even though that node is very pure, it has the least amount of the total observations and a result contributes a small portion of it's purity when we calculate the total entropy from splitting on Residence. This is important because we're looking for overall informative power of a feature and we don't want our results to be skewed by a rare value in a feature.

By itself the feature, Balance provides more information about our target variable than Residence. It reduces more disorder in our target variable. A decision tree algorithm would use this result to make the first split on our

data using Balance. From here on, the decision tree algorithm would use this process at every split to decide what feature it is going to split on next. In a real world scenario , with more than two features the first split is made on the most informative feature and then at every split the information gain for each additional feature needs to be recomputed because it would not be the same as the information gain from each feature by itself. The entropy and information gain would have to be calculated after one or more splits have already been made which would change the results. A decision tree would repeat this process as it grows deeper and deeper till either it reaches a predefined depth or no additional split can result in a higher information gain beyond a certain threshold which can also usually be specified as a hyperparameter!

There you have it! You now know what entropy and information gain are and how they are computed. You understand how a decision tree either by itself or in a tree based ensemble decides on the best order of features to split on and decides when to stop when it trains itself on given data. If you every have to explain the intricacies of how decision trees work to someone, hopefully you won't do too bad.

Implementation

```
import pandas as pd
import numpy as np

eps = np.finfo(float).eps

from numpy import log2 as log
dataset = {'Taste':['Salty','Spicy','Spicy','Spicy','Spicy','Sweet','Salty','Sweet','Spicy','Salty'],
        'Temperature':['Hot','Hot','Hot','Cold','Hot','Cold','Cold','Hot','Cold','Hot'],
        'Texture':['Soft','Soft','Hard','Hard','Hard','Soft','Soft','Soft','Soft','Hard'],
        'Eat':['No','No','Yes','No','Yes','Yes','No','Yes','Yes','Yes']}

df = pd.DataFrame(dataset, columns=['Taste','Temperature','Texture','Eat'])
```

```
13
14 def find_entropy(df):
15     '''
16     Function to calculate the entropy of the label i.e. Eat.
17     '''
18     Class = df.keys()[-1]
19     entropy = 0
20     values = df[Class].unique()
21     for value in values:
22         fraction = df[Class].value_counts()[value]/len(df[
   Class])
23         entropy += -fraction*np.log2(fraction)
24     return entropy
25
26
27 def find_entropy_attribute(df,attribute):
28     '''
29     Funtion to calculate the entropy of all the features.
30     '''
31     Class = df.keys()[-1]
32     target_variables = df[Class].unique()
33     variables = df[attribute].unique()
34     entropy2 = 0
35
36     for variable in variables:
37         entropy = 0
38         for target_variable in target_variables:
39             num = len(df[attribute][df[attribute]==variable][
   df[Class] ==target_variable])
40             den = len(df[attribute][df[attribute]==variable])
41             fraction = num/(den+eps)
42             entropy += -fraction*log(fraction+eps)
43         fraction2 = den/len(df)
44         entropy2 += -fraction2*entropy
45     return abs(entropy2)
46
47
48 def find_winner(df):
49     '''
50     Function to find the feature with the highest information
       gain
51     '''
52     Entropy_att = []
53     IG = []
54     for key in df.keys()[:-1]:
```

```
        IG.append(find_entropy(df) - find_entropy_attribute(df,key))

    return df.keys()[:-1][np.argmax(IG)]

def get_subtable(df, node, value):
    '''
    Function to get a subtable of met conditions.

    node: Column name
    value: Unique value of the column
    '''
    return df[df[node] == value].reset_index(drop=True)

def buildTree(df,tree=None):
    '''
    Function to build the ID3 Decision Tree.
    '''
    Class = df.keys()[-1]
    #Here we build our decision tree

    #Get attribute with maximum information gain
    node = find_winner(df)

    #Get distinct value of that attribute e.g Salary is node and Low,Med and High are values
    attValue = np.unique(df[node])

    #Create an empty dictionary to create tree
    if tree is None:
        tree={}
        tree[node] = {}

   #We make loop to construct a tree by calling this function recursively.
    #In this we check if the subset is pure and stops if it is pure.

    for value in attValue:

        subtable = get_subtable(df,node,value)
        clValue,counts = np.unique(subtable['Eat'],return_counts=True)

        if len(counts)==1:#Checking purity of subset
```

```
            tree[node][value] = clValue[0]
        else:
            tree[node][value] = buildTree(subtable) #Calling
    the function recursively

    return tree


tree = buildTree(df)


def predict(inst,tree):
    '''
    Function to predict for any input variable.
    '''
    #Recursively we go through the tree that we built earlier

    for nodes in tree.keys():

        value = inst[nodes]
        tree = tree[nodes][value]
        prediction = 0

        if type(tree) is dict:
            prediction = predict(inst, tree)
        else:
            prediction = tree
            break;

    return prediction

data = {'Taste':'Salty','Temperature':'Cold','Texture':'Hard'
    }

inst = pd.Series(data)

prediction = predict(inst,tree)

print(prediction)

# Prints "Yes"
```

Listing 1: Python Implementation for Decision Tree