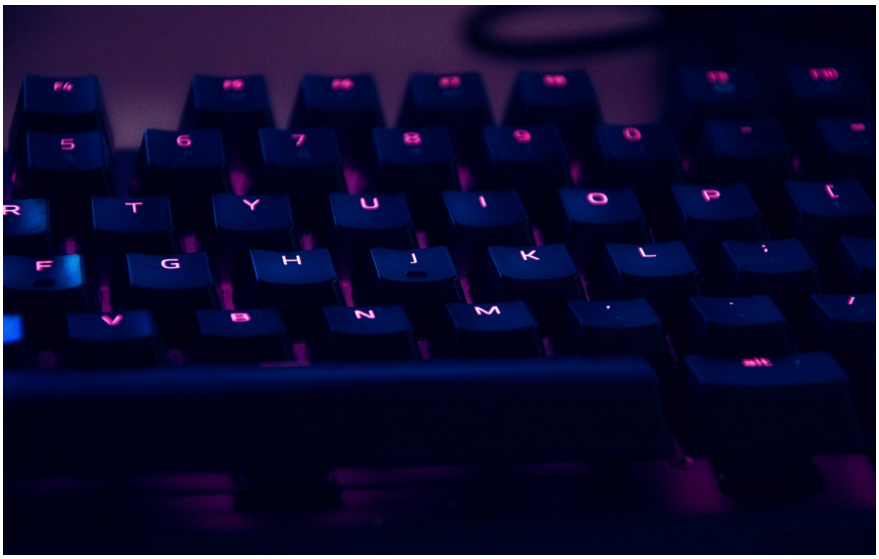
Decision Tree from the Scratch

Rakend Dubba Follow
Jun 10, 2018 · 4 min read



Decision tree from scratch (Photo by Anas Alshanti on Unsplash)

Python algorithm built from the scratch for a simple Decision Tree.

This is a continuation of the post Decision Tree and Math.

We have just looked at Mathematical working for ID3, this post we will see how to build this in Python from the scratch. We will make it simple by using Pandas dataframes.

Python code

Load the prerequisites

```
1  import numpy as np
2  import pandas as pd
3  eps = np.finfo(float).eps
4  from numpy import log2 as log
```

'eps' here is the smallest representable number. At times we get log(0) or 0 in the denominator, to avoid that we are going to use this.

Since the dataset, we looked in the previous post is very small one, we'll just represent that with a dictionary.

```
dataset = {'Taste':['Salty','Spicy','Spicy','Spicy','Spicy',
       'Temperature':['Hot','Hot','Hot','Cold','Hot','Cold',
       'Texture':['Soft','Soft','Hard','Hard','Hard','Soft',
       'Eat':['No','No','Yes','No','Yes','Yes','No','Yes','Y
```

We will read this with Pandas dataframe

```
df = pd.DataFrame(dataset,columns=['Taste','Temperature','Te
```

If we load this dataframe and see

```
df
```

	Taste	Temperature	Texture	Eat
0	Salty	Hot	Soft	No
1	Spicy	Hot	Soft	No
2	Spicy	Hot	Hard	Yes
3	Spicy	Cold	Hard	No
4	Spicy	Hot	Hard	Yes
5	Sweet	Cold	Soft	Yes
6	Salty	Cold	Soft	No
7	Sweet	Hot	Soft	Yes
8	Spicy	Cold	Soft	Yes
9	Salty	Hot	Hard	Yes

Pandas datframe of kid's eating preferences

We have seen from an earlier post we need to find the Entropy and then Information Gain for splitting the data set.

$$S = -\sum_{i=1}^{N} p_i \log_2 p_i,$$

We'll define a function that takes in class (target variable vector) and finds the entropy of that class.

Here the fraction is 'pi', it is the proportion of a number of elements in that split group to the number of elements in the group before splitting(parent group).

```
entropy_node = 0  #Initialize Entropy
values = df.Eat.unique()  #Unique objects - 'Yes', 'No'
for value in values:
    fraction = df.Eat.value_counts()[value]/len(df.Eat)
```

Code for calculating entropy at node

Summation is taken care of by 'entropy +='

We can check this by supplying our data frame to this function

```
entropy_node
0.9709505944546686
```

Entropy at node

This is same as the entropy (Eo) we calculated in the previous post. Now the entropy of the attribute Taste

```
attribute = 'Taste'
target_variables = df.Eat.unique()  #This gives all 'Yes' a
variables = df[attribute].unique()    #This gives different
entropy_attribute = 0
for variable in variables:
    entropy_each_feature = 0
    for target_variable in target_variables:
        num = len(df[attribute][df[attribute]==variable][df
        den = len(df[attribute][df[attribute]==variable])
```

Code for calculating entropy for an attribute

```
abs(entropy_attribute)
```

```
0.7609640474436806
```

Entropy of an attribute—Taste

This is same as the entropy E_Taste we calculated in the previous post.

Now the Information Gain is simply

IG_Taste = entropy_node—entropy_attribute = 0.21

We will continue this for the other attributes ‘Temperature’ and ‘Texture’.

We just need to replace attribute= ‘Taste’ with ‘Temperature’ and ‘Texture’

We get,

E_Temperature = 0.9509

E_Texture = 0.9245

Then Information Gain,

IG_Temperature = 0.02

IG_Texture = 0.05

Next process:

We'll find the winner node, the one with the highest Information Gain. We repeat this process to find which is the attribute we need to consider to split the data at the nodes.

We build a decision tree based on this. Below is the complete code.

```
def find_entropy(df):
    Class = df.keys()[-1]   #To make the code generic, chan
    entropy = 0
    values = df[Class].unique()
    for value in values:
        fraction = df[Class].value_counts()[value]/len(df[C
        entropy += -fraction*np.log2(fraction)
    return entropy


def find_entropy_attribute(df,attribute):
  Class = df.keys()[-1]   #To make the code generic, changi
  target_variables = df[Class].unique()  #This gives all 'Y
  variables = df[attribute].unique()    #This gives differe
  entropy2 = 0
  for variable in variables:
      entropy = 0
      for target_variable in target_variables:
          num = len(df[attribute][df[attribute]==variable][
          den = len(df[attribute][df[attribute]==variable])
          fraction = num/(den+eps)
          entropy += -fraction*log(fraction+eps)
      fraction2 = den/len(df)
      entropy2 += -fraction2*entropy
  return abs(entropy2)


def find_winner(df):
    Entropy_att = []
    IG = []
    for key in df.keys()[:-1]:
#         Entropy_att.append(find_entropy_attribute(df,key)
        IG.append(find_entropy(df)-find_entropy_attribute(d
    return df.keys()[:-1][np.argmax(IG)]


def get_subtable(df, node,value):
  return df[df[node] == value].reset_index(drop=True)


def buildTree(df,tree=None):
```

```
42        class = df.keys()[-1]   #To make the code generic, chan
43
44        #Here we build our decision tree
45
```

Code functions for building the tree

Now we call the buildTree function and print the tree we built.

```
tree = buildTree(df)
```

```
import pprint
pprint.pprint(tree)
```

```
{'Taste': {'Salty': {'Texture': {'Hard': 'Yes', 'Soft': 'No'}},
           'Spicy': {'Temperature': {'Cold': {'Texture': {'Hard': 'No',
                                                          'Soft': 'Yes'}},
                                     'Hot': {'Texture': {'Hard': 'Yes',
                                                         'Soft': 'No'}}}},
           'Sweet': 'Yes'}}
```

Printing tree structure

You can see the tree structure is formed based on the priority list to split the data.

We can write an algorithm to predict using this tree structure.

```
1   def predict(inst,tree):
2       #This function is used to predict for any input variabl
3
4       #Recursively we go through the tree that we built earli
5
6       for nodes in tree.keys():
7
8           value = inst[nodes]
9           tree = tree[nodes][value]
10          prediction = 0
11
12          if type(tree) is dict:
```

Function to predict for any input instance

```
inst = df.iloc[6] #This takes row with index 6
```

```
inst
```

```
Taste          Salty
Temperature     Cold
Texture         Soft
Eat               No
Name: 6, dtype: object
```

Sample instance to check the prediction

We can check with our available data.

We took a row with index 6 from our dataframe to use this for our prediction.

```
#Get prediction
prediction = predict(inst, tree)
prediction
```

```
'No'
```

Prediction with the trained data

Our tree has rightly predicted that the kid is not going to eat this food. Of course this is training data, not advisable to use this for testing.

We'll try with new data.

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

```
inst = pd.Series( data)
```

```
#Get prediction
prediction = predict(inst, tree)
prediction
```

```
'Yes'
```

Prediction for new data

Our tree predicted kid will eat this food (which is Salty, Cold and Hard).

This is a simple tree building algorithm without much control parameters.

Some of the contents here, I am inspired from this book ‘Ensemble machine learning-Ankit Dixit’. I encourage you to read it for further explorations.