# DECISION TREE CLASSIFICATION

*ESRA NUR ERSAN*

*HANYANG UNIVERSITY*

*Department of Computer Science &*

*Software Engineering*

*9325020182*

## Decision Tree

Consider a set of queries that can partition your data set. You choose the enquiry that provides the best split and again find the best queries for the partitions. You stop once all the points you are considering are of the same class. Then the classification part kicks in. The queries should guide to the appropriate class. To build this tree our first decision should show which feature or attribute is used to split our dataset. To determine this we must try all the attributes and find out the best split that will give us the best results. After, we can split the dataset into subtrees.If the data on the branches is the same class, then it means our classification is right and no need of going further. But if something goes wrong, then we have to repeat the splitting part on the subtree. So, we will keep doing this until we classified the entire dataset. Additionally, I would like to mention how we split our dataset. I could have used Gini index etc. But I wanted to go for information gain. Using information theory we can measure the information before and after the split. The change in information known as information gain. So, if we know how to calculate information gain we can split our data across every feature to find the split gives us the highest information gain. This split that has the highest information gain will be our best split. However, in order to calculate information gain, we need to find the entropy. Entropy controls how a decision tree decides to split data. It actually affects how a decision tree draws its boundaries.

## DETAILED DESCRIPTION OF MY CODE

### (Note: I'd like to mention that I didn't use any python library(e.g sklearn) except for math and sys module.)

**ParentEntropy():** Function for calculating the entropy. First, I get the total number of instances in the whole dataset. After, I created a dictionary whose keys are the values in the last attribute. Then I can track how many time label occurs for each key. So the probability comes from here helps me to calculate the entropy so that I can sum this up for the rest of the labels as well.

**InfoGainForRoot() :** In this function we take 3 input, data to be split on, the feature to be split on and the value of the feature to return. I create a new list, in the beginning, each time because we will be calling this function multiple times on the same dataset and we don't want the original dataset to be modified. Inside the if statement I cut the feature that we split on.

**SubsetOfTrainingData():** Actually, I did much research about splitting the datasets and I can say that it is common to use a subset of available data to train a model. Because if you prefer to use a subset of data for training, you can also upload a subset of historical data for training then you can upload the rest for scoring without retraining. We can measure accuracy by using the predictions that will be obtained from the data that is used for scoring. So we can improve accuracy.

**CandidateAttributes():** This function is to decide the attributes that can be candidates for the best options that will be used during the splitting part. In this function, I am also finding the candidate threshold between changed values of attributes. Basically, this approach will be useful to split attributes into multiple intervals(I thought that it could be nice for the age column since we have intervals)

**InformationGain():** This function is to compute the information gain that based on the decrease in entropy after the dataset is split on an attribute. Since building a tree is all about finding attributes that return the highest information gain I tried to maximize the information gain as much as I can.

**ChooseBestFeature():** So this function helps me to find the best attributes according to the information gain function’s outcomes.

**CreateDecisionTree():** Here actually where the actual building tree part is. I wanted to go with the random forest algorithm that cares about the feature importance. Through looking at the feature importance, you can decide which features you may want to drop because they don’t contribute enough or nothing to the prediction process. This is important because a general rule in machine learning is that the more features you have, the more likely your model will suffer from overfitting and vice versa. It ended up being recursive since each group of data from the split keep calling the same function at the end.

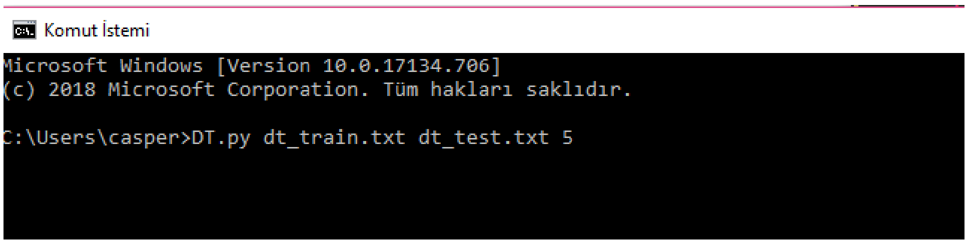
**Classification():** I can say that this function is the prediction phase. Making predictions with a tree involves navigating the tree with the specifically provided row of data so I decide to use the testing set so that at the end I can compare the actual results and the predicted labels.

**ReadFile():** As I mentioned before I didn’t use pandas so I didn’t use read_csv so I created this function to deal with both different datasets and testing sets. Also, I am removing every feature after I used it for splitting and I am done with it.

## How to execute my program

**#** \DT.py training_set testing_set MinSizeOfDataset

(Note:MinSizeOfDataset is a variable that i created for how many instances/rows at least you would to take to build the decision tree on it)


Komut İstemi

```
Microsoft Windows [Version 10.0.17134.706]
(c) 2018 Microsoft Corporation. Tüm hakları saklıdır.

C:\Users\casper>DT.py dt_train.txt dt_test.txt 5
```