DECISION TREE

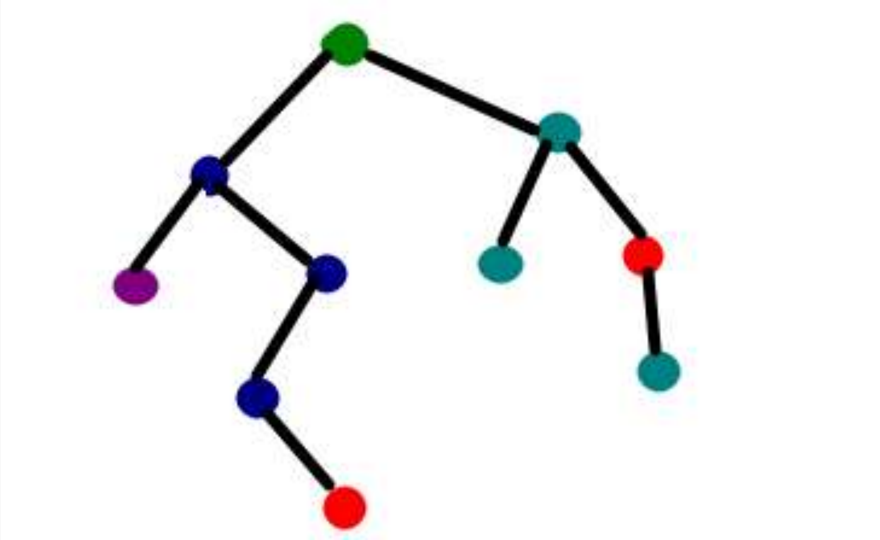
Mr. Ashok B. Bhosale

Assistant Professor

Department of Statistics

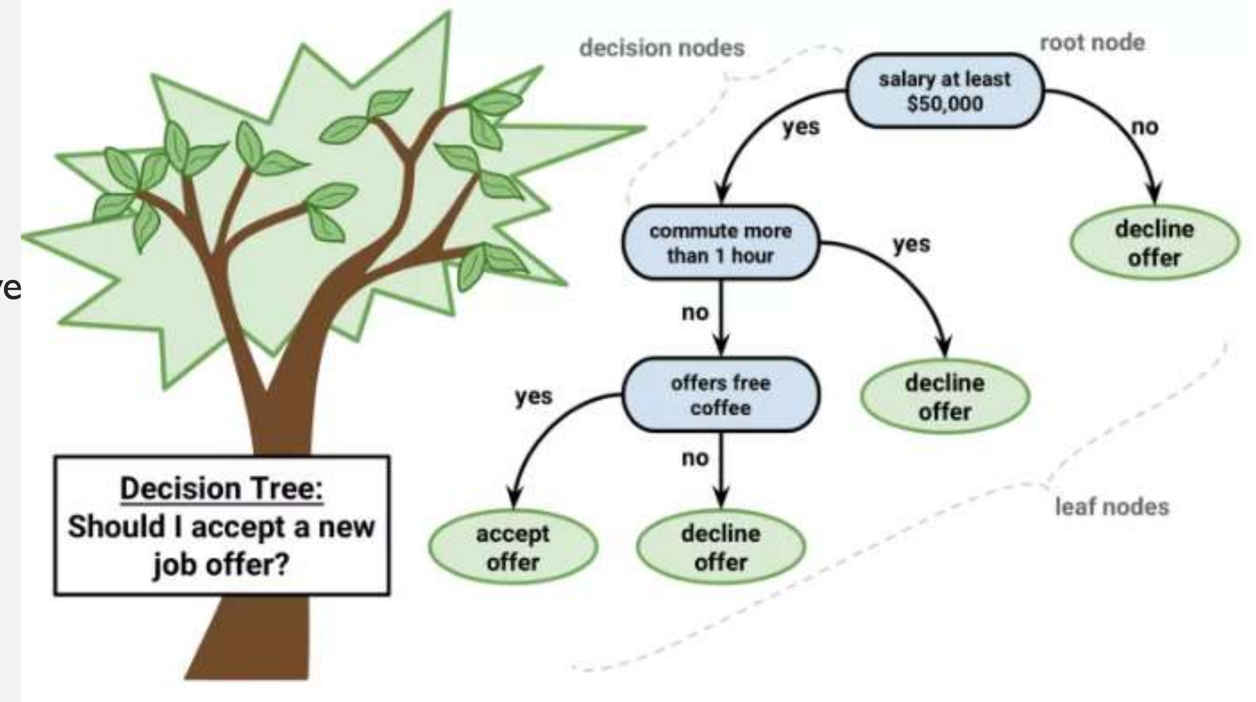
Vivekanand College, Kolhapur

- What it is?
- python implementation
- Use cases



WHAT IT IS

- Decision tree is a natural process of conscious and subconscious interpretation of rules and taking actions.
- Data Science, does the same !!



Is this real

that such simple algorithm can solve

complicated classification problem?

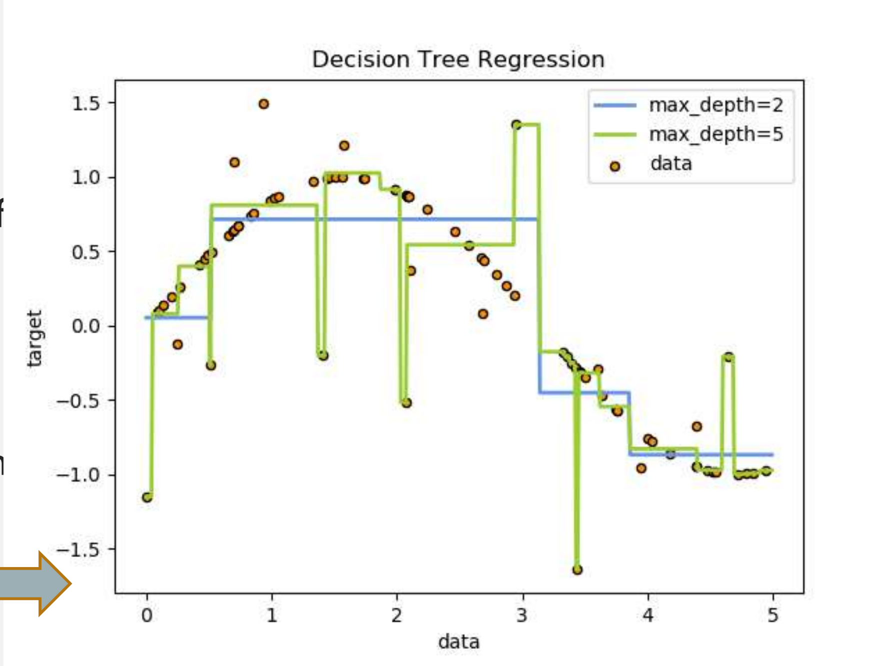
The answer is: Yes!

DECISION TREES

- Decision Trees (DTs) are a non-parametric supervised learning method used for classification and regression.

- The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features.

- For instance, in the example below, decision trees learn from data to approximate a sine curve with a set of if-then-else decision rules.



- The deeper the tree, the more complex the decision rules and the fitter the model.

TYPES OF DECISION TREES

Categorical Variable Decision Tree	Continuous Variable Decision Tree
A decision tree which has a categorical target variable	A decision tree which has continuous target variable
Example:- Let's say we have a problem to predict whether a bike is good or not. This can be judged by using a decision tree classifier.	However, to qualify the bike into the good or bad category, mileage becomes an important factor. Mileage is measured using a contiguous value hence it can be measured using the decision tree regressor.

TERMS

Terms	Description
Root Node	It represents the entire population or sample, and this further gets divided into two or more homogeneous sets.
Splitting	It is a process of dividing a node into two or more sub-nodes.
Decision Node	When a sub-node splits into further sub-nodes, then it is called a decision node.
Leaf/ Terminal Node	Nodes that do not split are called Leaf or Terminal nodes.
Pruning	When we remove sub-nodes of a decision node, this process is called pruning. You can say the opposite process of splitting.
Branch / Sub-Tree:	A sub-section of entire tree is called a branch or sub-tree.
Parent and Child Node:	A node, which is divided into sub-nodes is called the parent node of sub-nodes whereas sub-nodes are the children of a parent node.

DECISION CRITERIA

- So how do we decide on which feature/column/dimension to start with?
  - It is not done randomly !!! It is based on some considerations
  - Each time a subset is created out of parent set, the considerations are repeated
  - Why? Because the decision tree algorithm is a greedy one!

- Algorithms behind the decision tree
  - ID3 - uses Entropy function and Information gain as metrics..
  - C4.5 or C5.0
  - CART - uses Gini Index(Classification) as metric.
  - CHAID: Chi-Square Automatic Interaction Detection
  - MARs

TREE ALGORITHMS: ID3, C4.5, C5.0 AND CART

ID3 (Iterative Dichotomiser 3	C4.5	C5.0	CART (Classification and Regression Trees)
• Developed in 1986 by Ross Quinlan. • The algorithm creates a multiway tree, finding for each node the categorical feature that will yield the largest information gain for categorical targets. • Trees are grown to their maximum size and then a pruning step is usually applied to improve the ability of the tree to generalize the unseen data.	• is the successor to ID3 • removed the restriction that features must be categorical • converts the trained trees (i.e. the output of the ID3 algorithm) into sets of if-then rules	• Quinlan's latest version release under a proprietary license. • It uses less memory and builds smaller rulesets than C4.5 while being more accurate.	• is very similar to C4.5, but it differs in that it supports numerical target variables (regression) • does not compute rule sets. • CART constructs binary trees using the feature and threshold that yield the largest information gain at each node. • *scikit-learn uses an optimized version of the CART algorithm.*

ENTROPY (IMPURITY)

- According to Wikipedia, … Entropy refers to disorder or uncertainty.

- **Definition**: Entropy is the measures of impurity, disorder or uncertainty in a bunch of examples.

$$\text{Entropy} = -\sum p_j \log_2 p_j$$

There are 3 commonly used impurity measures used in binary decision trees:
- Entropy,
- Gini index,
- and Classification Error.

MATHEMATICAL INTUITION OF ENTROPY

- Let us imagine we have a set of N items.

These items fall into two categories,

- n have Label 1 and
- m = (N - n) have Label 2.

In terms of ratio,

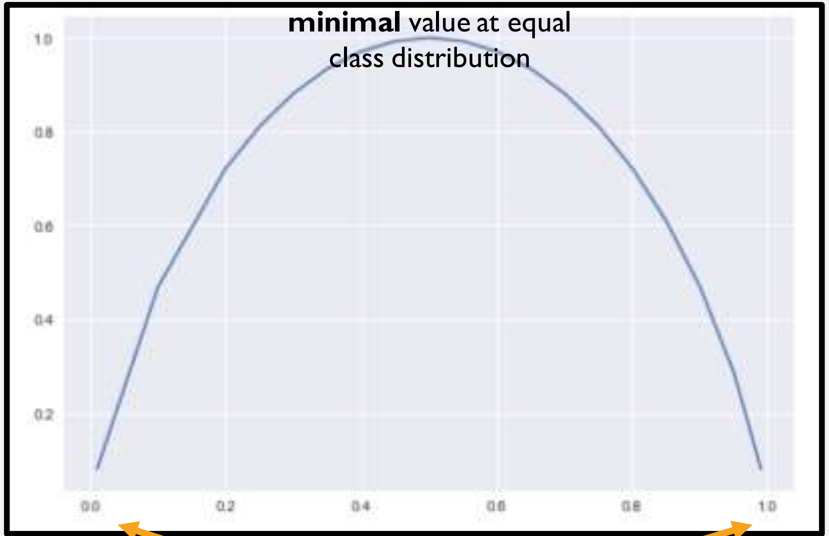
Refer entropy.xlsx

$$p = \frac{n}{N}$$

and

$$q = \frac{m}{N} = \frac{N-n}{N} = 1 - \frac{n}{N} = 1 - p$$

$$Entropy = -plog_2(p) - qlog_2(q)$$



minimal value if only
1 class left in S

- A set is tidy if it contains only items with the same label and messy if it is a mix of items with different labels.
- With no item with label 1 (p=0) or if the set is full of items with Label 1 (p=1), the entropy is zero. LEAST MESSY
- With half in Label 1, half in Label 2 (p=1/2), the entropy is maximal (equals to 1) .. MOST MESSY, symmetric , among the two categories to classify, there not one which is messier than the other.

MEANING

- Entropy = 0, This is not a good set for training.
- Entropy = 1, This is a good set for training.
- The entropy is an absolute measure which provides a number between 0 and 1,

EVOLUTION OF ENTROPY

- In decision trees, at each branching, the input set is split in 2.
- Compare entropy before and after the split.
- E.g. start with a messy set with entropy one (half/half, p=q).
- In the worst case, it could be split into 2 messy sets where half of the items are labeled 1 and the other half have Label 2 in each set. Hence the entropy of each of the two resulting sets is 1. In this scenario, the messiness has not changed
- We can not sum the entropies of the two sets.
- A solution, often used in mathematics, is to compute the mean entropy of the two sets. In this case, the mean is one.
- However, in decision trees, a weighted sum of entropies is computed instead (weighted by the size of the two subsets)

IT MEANS …

Entropy at split = $E_{split} = \frac{N_1}{N}.E_1 + \frac{N_2}{N}.E_2$

- N_1 and N_2 are the number of items of each sets after the split and E_1 and E_2 are their respective entropy.

- It gives more importance to the set which is larger

GENERALIZATION

If you have more than 2 labels, you can generalize the Entropy formula as follows:

$$Entropy = -\sum_{i=1}^{n} p_i log_2(p_i)$$

- where the p_i are the ratios of elements of each label in the set.

INFORMATION GAIN

Definition: Information gain (IG) measures how much "information" a feature gives us about the class.

Why it matters ?

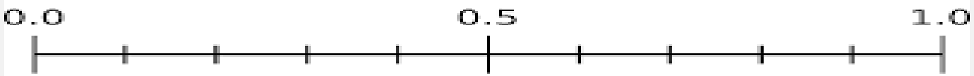
- Decision Trees algorithm will always try to maximize Information gain.
- An attribute with highest Information gain will be tested/split first.

- Information gain = entropy(parent) – [weighted average] * entropy(children)

EXAMPLE – USING IMPURITY (ENTROPY)

- STEP – 1 - Calculate entropy of the target.
- current dataset S.
- compute the Entropy H(S) on S as follows: where K is the number of classes, $p(y_j)$ is the proportion of number of elements of $p(y_j)$ class to the number of entire elements in output of S
- H(S) tell us how uncertain our dataset is. It ranges from 0 to 1, which 0 is the case when the output contains only one class (pure), whereas 1 is the most uncertain case.

$$H(S) = -\sum_{j=1}^{K} p(y_j) \log_2 p(y_j)$$



H(S) 0 – pure set

H(S) 1 – most uncertain

when the proportion of each class is equal to others'

if the number of YES is equal to the number of NO on the considered subset, then it's easy to see that there is a big chance that it can't be fully classified (that's why we call it the most uncertain case).

EXAMPLE – USING IMPURITY (ENTROPY)

- STEP – 2 The dataset is then split on the different attributes. The entropy for each branch is calculated. Then it is added proportionally, to get total entropy for the split. The resulting entropy is subtracted from the entropy before the split. The result is the Information Gain, or decrease in entropy.

- Information Gain is computed separately on each feature of the current dataset S,

- The value indicates how much the uncertainty in S was reduced after splitting S using feature A.

- Lastly, split the current dataset S using the feature which has the highest Information Gain.

$$IG(A,S) = H(S) - \sum_{i=1}^{n} p(t)H(t)$$

EXAMPLE – USING IMPURITY (ENTROPY)

- STEP – 3 Choose attribute with the largest information gain as the decision node, divide the dataset by its branches and repeat the same process on every branch.

DATASET

features

Target/label/prediction

Weather	Temperature	Humidity	Injure	Mood	RUN
clear	<10	<70	slightly	happy	NO
shower	20~30	>80	fit	stressed	YES
storm	10~20	>80	fit	happy	NO
shower	10~20	>80	slightly	stressed	YES
clear	>30	70~80	fit	lazy	YES
storm	20~30	>80	fit	stressed	NO
clear	>30	70~80	severe	happy	NO
clear	10~20	<70	severe	stressed	NO
shower	10~20	70~80	slightly	happy	NO
shower	>30	>80	fit	happy	YES
storm	20~30	70~80	slightly	happy	NO
clear	10~20	<70	slightly	happy	?

Training data

Test data

HOW THE ALGORITHM WORKS

weather

clear

shower

storm

Weathe	Temperature	Humidit	Injure	Mood	RUN
clear	<10	<70	slightly	happy	NO
clear	>30	70~80	fit	lazy	YES
clear	>30	70~80	severe	happy	NO
clear	10~20	<70	severe	stressed	NO

Temperature	Humidit	Injure	Mood	RUN
<10	<70	slightly	happy	NO
>30	70~80	fit	lazy	YES
>30	70~80	severe	happy	NO
10~20	<70	severe	stressed	NO

Humidit	Injure	Mood	RUN
70~80	fit	lazy	YES
70~80	severe	happy	NO

Weathe	Temperature	Humidit	Injure	Mood	RUN
shower	20~30	>80	fit	stressed	YES
shower	10~20	>80	slightly	stressed	YES
shower	10~20	70~80	slightly	happy	NO
shower	>30	>80	fit	happy	YES

Temperature	Humidit	Injure	Mood	RUN
>30	>80	fit	happy	YES
10~20	>80	slightly	stressed	YES
10~20	70~80	slightly	happy	NO
20~30	>80	fit	stressed	YES

Weathe	Temperature	Humidit	Injure	Mood	RUN
storm	10~20	>80	fit	happy	NO
storm	20~30	>80	fit	stressed	NO
storm	20~30	70~80	slightly	happy	NO

Continues …

EXAMPLE – HOW DID IT WORK ON PREVIOUS DATASET

H(S) would be the entire original table

Entropy H(S) = $$H(S) = -\sum_{j=1}^{K} p(y_j) \log_2 p(y_j)$$

$= -p(YES) \log_2 p(YES) - p(NO) \log_2 p(NO)$

$= -(4/11) \log_2(4/11) - (7/11) \log_2(7/11)$

= 0.9457

Weathe	Temperature	Humidit	Injure	Mood	RUN
clear	<10	<70	slightly	happy	NO
clear	>30	70~80	fit	lazy	YES
clear	>30	70~80	severe	happy	NO
clear	10~20	<70	severe	stressed	NO
shower	>30	>80	fit	happy	YES
shower	10~20	>80	slightly	stressed	YES
shower	10~20	70~80	slightly	happy	NO
shower	20~30	>80	fit	stressed	YES
storm	10~20	>80	fit	happy	NO
storm	20~30	>80	fit	stressed	NO
storm	20~30	70~80	slightly	happy	NO

EXAMPLE – HOW DID IT WORK ON PREVIOUS DATASET

Next, we will compute Information Gain on each feature. For weather feature

- 3 possible values: clear, shower and storm.
- clear - p(clear) = 4/11
  - YES 1
  - NO 3
- Entropy H(clear)
  = -p(YES) log2p(YES) – p(NO) log2p(NO)
  = $-(1/4)\log_2(1/4) - (3/4)\log_2(3/4)$
  = 0.8113
- shower - p(shower) = 4/11
  - YES 3
  - NO 1
- Entropy H(Shower)
  = -p(YES) log2p(YES) – p(NO) log2p(NO)
  = $-(3/4)\log_2(3/4) - (1/4)\log_2(1/4)$
  = 0.8113

Weather	Temperature	Humidity	Injure	Mood	RUN
clear	<10	<70	slightly	happy	NO
clear	>30	70~80	fit	lazy	YES
clear	>30	70~80	severe	happy	NO
clear	10~20	<70	severe	stressed	NO
shower	>30	>80	fit	happy	YES
shower	10~20	>80	slightly	stressed	YES
shower	10~20	70~80	slightly	happy	NO
shower	20~30	>80	fit	stressed	YES
storm	10~20	>80	fit	happy	NO
storm	20~30	>80	fit	stressed	NO
storm	20~30	70~80	slightly	happy	NO

- storm - p(storm) = 3/11
  - YES 0
  - NO 3
- Entropy H(storm)
  = -p(YES) log2p(YES) – p(NO) log2p(NO)
  = $-(0/3)\log_2(0/3) - (3/3)\log_2(3/3)$
  = 0

EXAMPLE – HOW DID IT WORK ON PREVIOUS DATASET

So now we can compute the Information Gain on the Weather feature as follow

$$IG(Weather, S) = 0.9457 - \frac{4}{11} * 0.8113 - \frac{4}{11} * 0.8113 - \frac{3}{11} * 0 = 0.3557$$

Continue repeat this process with other features, you will likely end up with results like this:

IG (Weather, S)	= 0.3557
IG (Temperature, S)	= 0.1498
IG (Humidity, S)	= 0.2093
IG (Injure, S)	= 0.2093
IG (Mood, S)	= 0.2275

From the results above, IG on Weather has the highest value, so use Weather as a splitting condition will have the highest chance to reduce the uncertainty of dataset S, and may lead to a **good classification** in the end.

EXAMPLE - ENTROPY OF TARGET

- 8 records with negative class and 8 records with positive class. So, we can directly estimate the entropy of target as 1.

Variable label	
pos	neg
8	8

A	B	C	D
>= 5	>= 3.0	>= 4.2	>= 1.4
< 5	< 3.0	< 4.2	< 1.4

- IG for the entire data set
  - E(8, 8) = -1 * [(p(+ve) * log(p(+ve))) + (p(-ve) * log(p(-ve)))]
  - $= -1 * [(8/16) * \log_2(8/16))) \quad + ((8/16) * \log_2(8/16)))]$
  - = 1

ENTROPY OF TARGET

- For the variable A,
- var A has value >=5 for 12 records out of 16 and 4 records with value <5 value.

  - For Var A >= 5 & class == positive: 5/12
  - For Var A >= 5 & class == negative: 7/12
    - Entropy(5, 7) = -1 * ((5/12)*log2(5/12) + (7/12)*log2(7/12)) = 0.9799

  - For Var A <5 & class == positive: 3/4
  - For Var A <5 & class == negative: 1/4
    - Entropy(3, 1) = -1 * ((3/4)*log2(3/4) + (1/4)*log2(1/4)) = 0.81128

- Entropy(Target,A) = P(>=5) * E(5,7) + P(<5) * E(3,1)
- = (12/16) * 0.9799 + (4/16) * 0.81128 = 0.937745
- IG = E(Target) – E(Target,A)
- = 1 – 0.937745 = 0.062255

ENTROPY OF TARGET

- For the variable B,
- var B has value >=3 for 12 records out of 16 and 4 records with value <3 value.

  - For Var B >= 3 & class == positive: 8/12
  - For Var B >= 3 & class == negative: 4/12
    - Entropy(8, 4) = $-1 * ((8/12)*\log_2(8/12) + (4/12)*\log_2(4/12)) = 0.39054$

  - For Var B <3 & class == positive: 0/4
  - For Var B <3 & class == negative: 4/4
    - Entropy(0, 4) = $-1 * ((0/4)*\log_2(3/4) + (4/4)*\log_2(4/4)) = 0$

- Entropy(Target, B) = P(>=5) * E(5,7) + P(<5) * E(3,1)
- = (12/16) * 0.39054 + (4/16) * 0 = 0.292905
- IG = E(Target) – E(Target, B)
- = 1 – 0.292905 = 0.707095

ENTROPY OF TARGET

- For the variable C,
- var C has value >= 4.2 for 6 records out of 16 and 10 records with value < 4.2 value.
  - For Var C >= 4.2 & class == positive: 0/6
  - For Var C >= 4.2 & class == negative: 6/6
    - Entropy(0, 6) = -1 * ((0/6)*$\log_2$(0/6) + (6/6)*$\log_2$(6/6)) = 0
  - For Var C <4.2 & class == positive: 8/10
  - For Var C <4.2 & class == negative: 2/10
    - Entropy(8, 2) = 0.72193
- Entropy(Target, C) = P(>=4.2) * E(0, 6) + P(<4.2) * E(8, 2)
- = (6/16) * 0 + (10/16) * 0.72193 = 0.4512
- IG = E(Target) – E(Target, C)
- = 1 – 0.4512 = 0.5488

ENTROPY OF TARGET

- For the variable D,
- var D has value >= 1.4 for 5 records out of 16 and 11 records with value < 1.4 value.

  - For Var D >= 1.4 & class == positive: 0/5
  - For Var D >= 1.4 & class == negative: 5/5
    - Entropy(0, 5) = 0

  - For Var D < 1.4 & class == positive: 8/11
  - For Var D < 1.4 & class == negative: 3/11
    - Entropy(8, 3) = -1 * ((8/11)*log2(8/11) + (3/11)*log2(3/11)) = 0.84532

- Entropy(Target, D) = P(>= 1.4) * E(0, 5) + P(< 1.4) * E(8, 3)
- = 5/16 * 0 + (11/16) * 0.84532 = 0.5811575
- IG = E(Target) – E(Target, D)
- = 1 – 0.5811575 = 0.41189

DECISION

- build a decision tree.
- place the attributes on the tree according to their values.
- An Attribute with better value than other should position as root
- A branch with entropy 0 should be converted to a leaf node.
- A branch with entropy more than 0 needs further splitting.

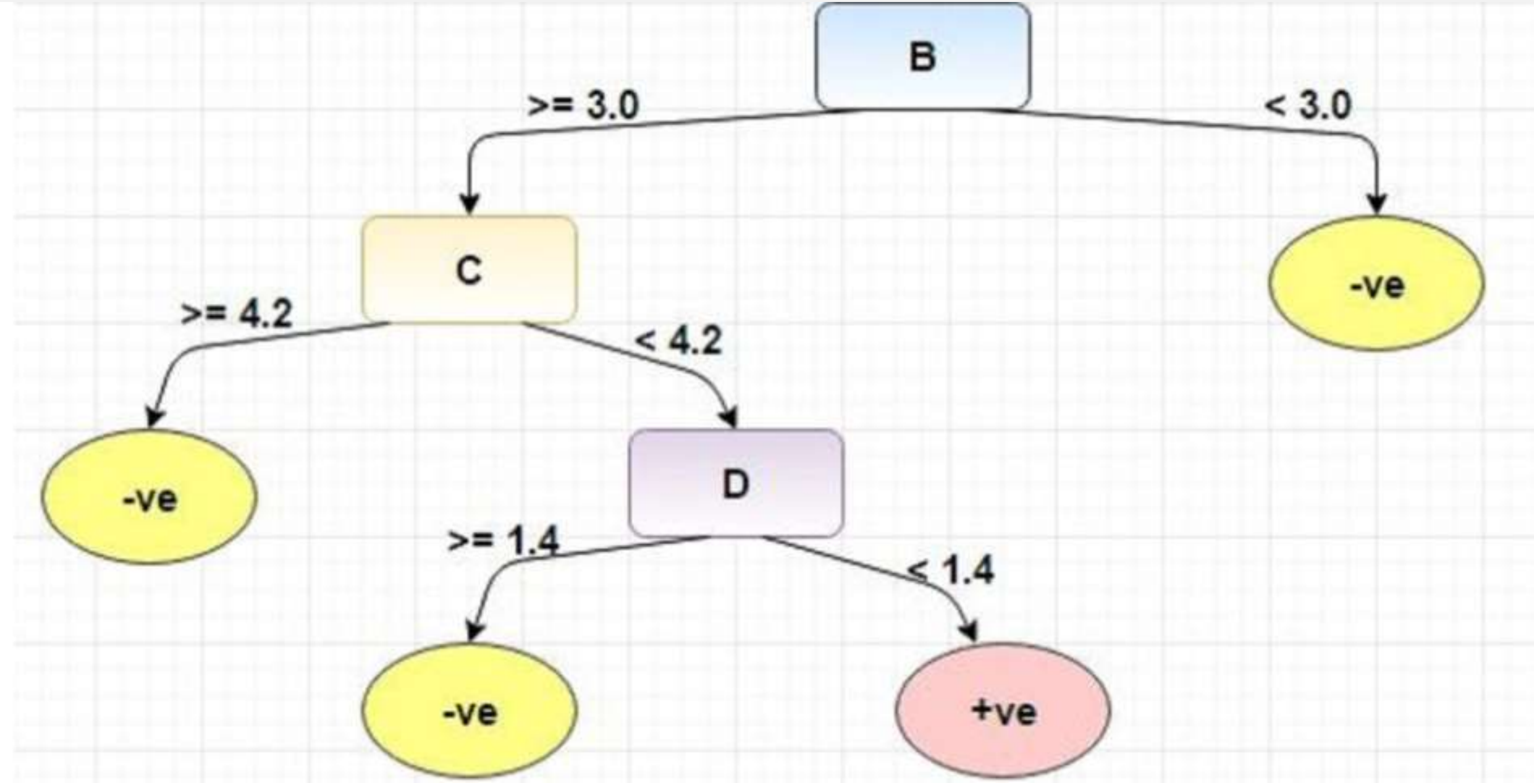
		Target	
		Positive	Negative
A	>= 5.0	5	7
	<5	3	1
Information Gain of A = 0.062255			

		Target	
		Positive	Negative
B	>= 3.0	8	4
	< 3.0	0	4
Information Gain of B= 0.7070795			

		Target	
		Positive	Negative
C	>= 4.2	0	6
	< 4.2	8	2
Information Gain of C= 0.5488			

		Target	
		Positive	Negative
D	>= 1.4	0	5
	< 1.4	8	3
Information Gain of D= 0.41189			

AND THE TREE …



SHORTCOMINGS OF THE ENTROPY MEASURE

- The information gain measure is biased towards the attributes that have more number of unique values

- **Problem**: If an attribute has a large number of values probably the resulting tree will be larger

- The **reason** for that bias resides in the weight given to the values

GINI INDEX

- Gini Index is a metric to measure how often a randomly chosen element would be incorrectly identified.
- It means an attribute with lower gini index should be preferred.

GINI Index

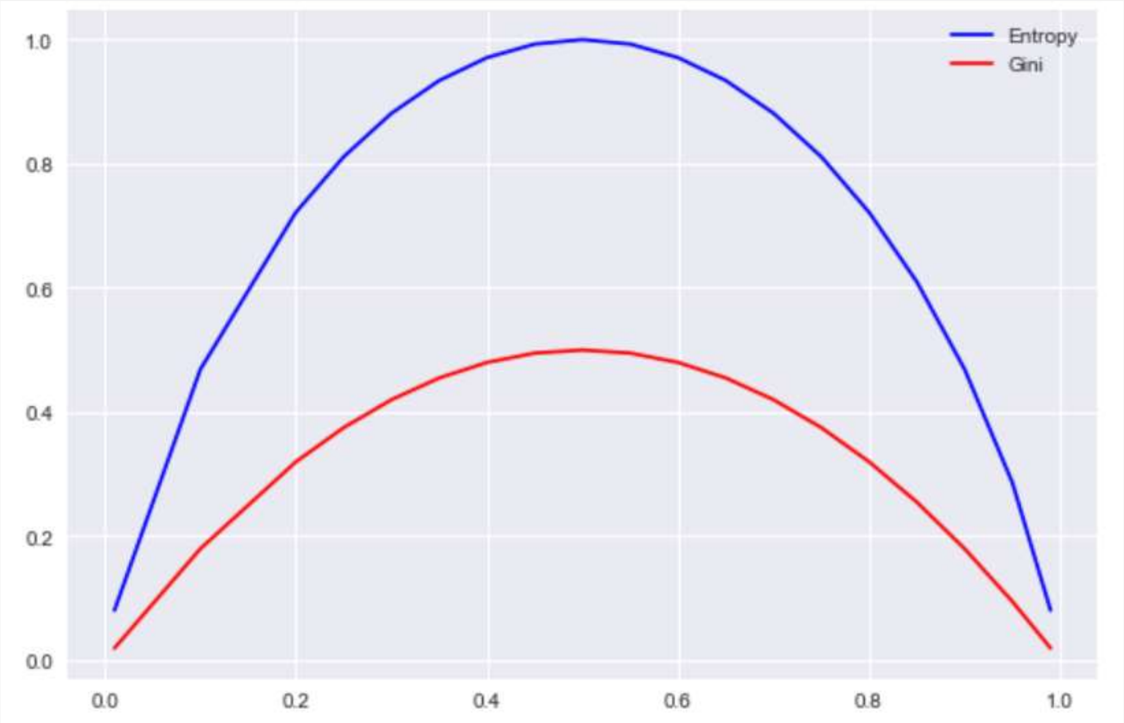
$$Gini = \sum_{i \neq j} p(i)p(j)$$

i and j are levels of the target variable

INTUITION

- According to scikit-learn documentation, gini plays the same role as entropy

- As we can see, there is not much differences.



GINI INDEX

- For the variable A,
- var A has value >=5 for 12 records out of 16 and 4 records with value <5 value.
  - For Var A >= 5 & class == positive: 5/12
  - For Var A >= 5 & class == negative: 7/12
    - gini(5, 7) = 1- ((5/12)^2 + (7/12)^2) = 0.4860
  - For Var A <5 & class == positive: 3/4
  - For Var A <5 & class == negative: 1/4
    - gini(3,1) = 1- ((3/4)^2 + (1/4)^2) = 0.375
- By adding weight and sum each of the gini indices:
- gini(Target, A) = (12/16) * 0.4860 + (4/16) * 0.375 = 0.45825

GINI INDEX

- For the variable B,
- var B has value >=3 for 12 records out of 16 and 4 records with value <3 value.

  - For Var B >= 3 & class == positive: 8/12
  - For Var B >= 3 & class == negative: 4/12
    - gini(8,4) = 1- ((8/12)2 + (4/12)2) = 0.446

  - For Var B <3 & class == positive: 0/4
  - For Var B <3 & class == negative: 4/4
    - gin(0,4) = 1- ((0/4)2 + (4/4)2) = 0

- By adding weight and sum each of the gini indices:
- gini(Target, B) = (12/16) * 0.446 + (4/16) * 0 = 0.3345

DECISION

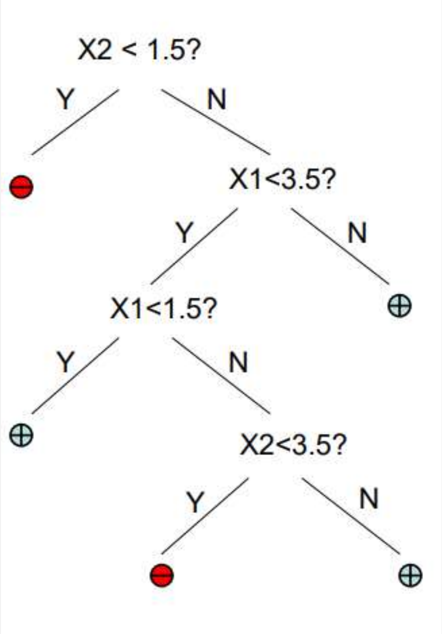
		Target	
		Positive	Negative
A	>= 5.0	5	7
	<5	3	1
Gini Index of X1 = 0.45825			

		Target	
		Positive	Negative
B	>= 3.0	8	4
	< 3.0	0	4
Gini Index of X2= 0.3345			

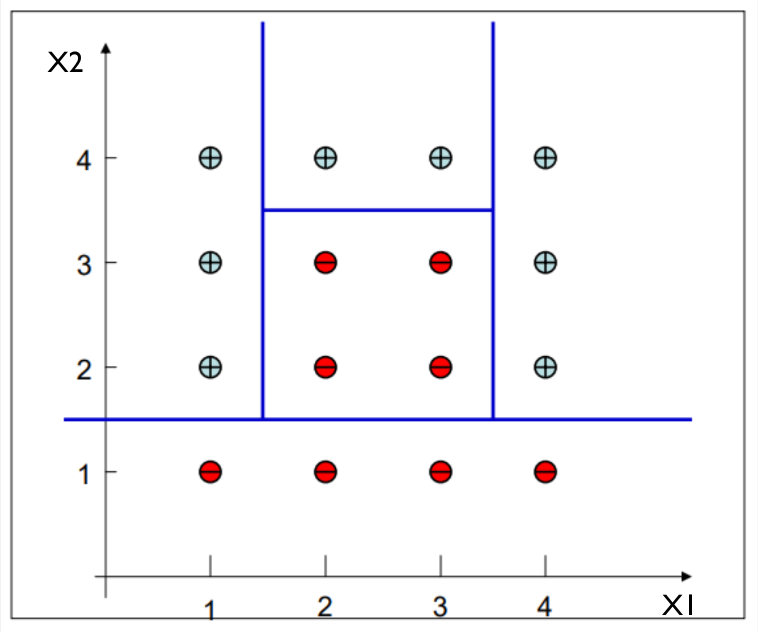
		Target	
		Positive	Negative
C	>= 4.2	0	6
	< 4.2	8	2
Gini Index of X3= 0.2			

		Target	
		Positive	Negative
D	>= 1.4	0	5
	< 1.4	8	3
Gini Index of X4= 0.273			

DECISION BOUNDARIES



Decision Trees divide the input space into axis-parallel rectangles and label each rectangle with one of the K classes



ENTROPY VS GINI

- Gini is intended for continuous attributes,
- Entropy for attributes that occur in classes
- Entropy may be a little slower to compute

DECISION TREE VARIATIONS

Input/ predictor variables	Target/ output variable	ML type	Decision criteria
Discreet	Discreet	Classification	(entropy, gini)
Discreet	Continuous	Regression	(entropy, gini)
Continuous	Continuous	Regression	(threshold split)
Continuous	Discreet	Classification	(threshold split)
Continuous/ Discreet	Discreet	Classification	Mix
Continuous/ Discreet	Continuous	Regression	Mix

NUMERIC VARIABLES - ENTROPY-BASED DISCRETIZATION

data relating the number of hours various students studied in an attempt to determine its effect on their test performance

Sort the feature (hours studied)

HOURS STUDIED	GRADE A ON TEST
4	N
5	Y
8	N
12	Y
15	Y

NUMERIC VARIABLES - ENTROPY-BASED DISCRETIZATION

Step 1: start by calculating entropy of the data set itself

	A ON TEST	LOWER THAN A
Overall	3	2

$E(D) = -\ (3/5 \log_2(3/5) + 2/5 \log_2(2/5)) = .529 + .442 = .971$

NUMERIC VARIABLES - ENTROPY-BASED DISCRETIZATION

Step 2: let's iterate through and see which splits give us the maximum entropy gain. To find a split, we average two neighboring values in the list.

HOURS STUDIED	GRADE A ON TEST	SPLIT POINT #	SPLIT VALUE
4	N	1	(4+ 5)/2 = 4.5
5	Y		
8	N		
12	Y		
15	Y		

Now we get 2 bins, as follows:

	A ON TEST	LOWER THAN A
<=4.5	0	1
>4.5	3	1

calculate entropy for each bin and find the information gain of this split:

$E(D <= 4.5) = - (1/1 \log_2(1/1) + 0/1 \log_2(0/1)) = 0 + 0 = 0$

$E(D > 4.5) = - (1/4 \log_2(1/4) + 3/4 \log_2(3/4)) = .311 + .5 = .811$

NUMERIC VARIABLES - ENTROPY-BASED DISCRETIZATION

Step 2: let's iterate through and see which splits give us the maximum entropy gain. To find a split, we average two neighboring values in the list.

HOURS STUDIED	GRADE A ON TEST	SPLIT POINT #	SPLIT VALUE
4	N	1	(4+ 5)/2 = 4.5
5	Y		
8	N		
12	Y		
15	Y		

Now we get 2 bins, as follows:

	A ON TEST	LOWER THAN A
<=4.5	0	1
>4.5	3	1

calculate entropy for each bin and find the information gain of this split:

$E(D <= 4.5) = - (1/1 \log_2(1/1) + 0/1 \log_2(0/1)) = 0 + 0 = 0$

$E(D > 4.5) = - (1/4 \log_2(1/4) + 3/4 \log_2(3/4)) = .311 + .5 = .811$

$E_{net} = 1/5\ (0) + 4/5\ (.811) = .6488$

Gain = .971 - .6488 = .322

NUMERIC VARIABLES - ENTROPY-BASED DISCRETIZATION

Step 2: let's iterate through and see which splits give us the maximum entropy gain. To find a split, we average two neighboring values in the list.

HOURS STUDIED	GRADE A ON TEST	SPLIT POINT #	SPLIT VALUE
4	N		
5	Y	2	(5 + 8)/2 = 6.5
8	N		
12	Y		
15	Y		

Now we get 2 bins, as follows:

	A ON TEST	LOWER THAN A
<=6.5	1	1
>6.5	2	1

calculate entropy for each bin and find the information gain of this split:

$E(D <= 6.5) = - (1/2 \log_2(1/2) + 1/2 \log_2(1/2)) = 1$

$E(D > 6.5) = - (2/2 \log_2(2/2) + 1/3 \log_2(1/3)) = .389 + .528 = .917$

$E_{net} = 2/5\ (1) + 3/5\ (.917) = .950$

Gain = .971 - .950 = .021

NUMERIC VARIABLES - ENTROPY-BASED DISCRETIZATION

Step 2: let's iterate through and see which splits give us the maximum entropy gain. To find a split, we average two neighboring values in the list.

HOURS STUDIED	GRADE A ON TEST	SPLIT POINT #	SPLIT VALUE
4	N		
5	Y		
8	N	3	(8+12)/2 = 10
12	Y		
15	Y		

Now we get 2 bins, as follows:

	A ON TEST	LOWER THAN A
<=10	1	2
>10	2	0

calculate entropy for each bin and find the information gain of this split:

$E(D <= 10) = - (1/3 \log_2(1/3) + 2/3 \log_2(2/3)) = .917$

$E(D > 10) = - (2/2 \log_2(2/2) + 0/0 \log_2(0/0)) = 0$

$E_{net} = 2/5 (0) + 3/5 (.917) = ..55$

Gain = .971 - .55 = .421

NUMERIC VARIABLES - ENTROPY-BASED DISCRETIZATION

Step 2: let's iterate through and see which splits give us the maximum entropy gain. To find a split, we average two neighboring values in the list.

HOURS STUDIED	GRADE A ON TEST	SPLIT POINT #	SPLIT VALUE
4	N		
5	Y		
8	N		
12	Y	4	(12 + 15)/2 = 13.5
15	Y		

Now we get 2 bins, as follows:

	A ON TEST	LOWER THAN A
<=13.5	2	2
>13.5	1	0

calculate entropy for each bin and find the information gain of this split:

$E(D <= 13.5) = -(2/2 \log_2(2/2) + 2/2 \log_2(2/2)) = 1$

$E(D > 13.5) = -(1/1 \log_2(1/1) + 0/1 \log_2(0/1)) = 0$

$E_{net} = 4/5\ (1) = .80$

Gain = .971 - .80 = .117

NUMERIC VARIABLES - ENTROPY-BASED DISCRETIZATION

- Step 3

- After calculating everything, we find that our best split is split 3, which gives us the best information gain of .421. We will partition the data there!

- According to the algorithm, we now can further bin our attributes in the bins we just created. This process will continue until we satisfy a termination criteria.

SCIKIT LEARN PARAMETERS

- class sklearn.tree.DecisionTreeClassifier(criterion='gini', splitter='best', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=None, random_state=None, max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, class_weight=None, presort=False)

- criterion : string, optional (default="gini")
- The function to measure the quality of a split.
- Supported criteria are
  - "gini" for the Gini impurity
  - "entropy" for the information gain.

SCIKIT LEARN PARAMETERS

- class sklearn.tree.DecisionTreeClassifier(criterion='gini', splitter='best', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=None, random_state=None, max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, class_weight=None, presort=False)

- max_depth : int or None, optional (default=None)
- The maximum depth of the tree.
- If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples.

SCIKIT LEARN PARAMETERS

- class sklearn.tree.DecisionTreeClassifier(criterion='gini', splitter='best', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=None, random_state=None, max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, class_weight=None, presort=False)

- min_samples_split : int, float, optional (default=2)
- The minimum number of samples required to split an internal node:
- If int, then consider min_samples_split as the minimum number.
- If float, then min_samples_split is a fraction and ceil (min_samples_split * n_samples) are the minimum number of samples for each split.

SCIKIT LEARN PARAMETERS

- class sklearn.tree.DecisionTreeClassifier(criterion='gini', splitter='best', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=None, random_state=None, max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, class_weight=None, presort=False)

- min_samples_leaf : int, float, optional (default=1)
- The minimum number of samples required to be at a leaf node. A split point at any depth will only be considered if it leaves at least min_samples_leaf training samples in each of the left and right branches. This may have the effect of smoothing the model, especially in regression.
- If int, then consider min_samples_leaf as the minimum number.
- If float, then min_samples_leaf is a fraction and ceil(min_samples_leaf * n_samples) are the minimum number of samples for each node.

SCIKIT LEARN PARAMETERS

- class sklearn.tree.DecisionTreeClassifier(criterion='gini', splitter='best', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=None, random_state=None, max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, class_weight=None, presort=False)

- max_features : int, float, string or None, optional (default=None)
- The number of features to consider when looking for the best split:
- If int, then consider max_features features at each split.
- If float, then max_features is a fraction and int(max_features * n_features) features are considered at each split.
- If "auto", then max_features=sqrt(n_features).
- If "sqrt", then max_features=sqrt(n_features).
- If "log2", then max_features=log2(n_features).
- If None, then max_features=n_features.

SCIKIT LEARN PARAMETERS

- class sklearn.tree.DecisionTreeClassifier(criterion='gini', splitter='best', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=None, random_state=None, max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, class_weight=None, presort=False)

- Weights associated with classes in the form {class_label: weight}.
- If not given, all classes are supposed to have weight one.
- For multi-output problems, a list of dicts can be provided in the same order as the columns of y.
- for multioutput (including multilabel) weights should be defined for each class of every column in its own dict. For example, for four-class multilabel classification weights should be [{0: 1, 1: 1}, {0: 1, 1: 5}, {0: 1, 1: 1}, {0: 1, 1: 1}] instead of [{1:1}, {2:5}, {3:1}, {4:1}].

- The "balanced" mode uses the values of y to automatically adjust weights inversely proportional to class frequencies in the input data as n_samples / (n_classes * np.bincount(y))

ATTRIBUTES

- classes_ : array of shape = [n_classes] or a list of such arrays, The classes labels (single output problem), or a list of arrays of class labels (multi-output problem).
- feature_importances_ : array of shape = [n_features] , Return the feature importances.
- max_features_ : int, The inferred value of max_features.
- n_classes_ : int or list, The number of classes (for single output problems), or a list containing the number of classes for each output (for multi-output problems).
- n_features_ : int, The number of features when fit is performed.
- n_outputs_ : int, The number of outputs when fit is performed.

REGRESSION WITH DECISION TREES

- Replacing INFORMATION GAIN with Standard Deviation Reduction

- A decision tree is built top-down from a root node and involves partitioning the data into subsets that contain instances with similar values (homogeneous)

- We use standard deviation to calculate the homogeneity of a numeric sample

- If the numeric sample is completely homogeneous, it’s S.D = 0

STANDARD DEVIATION REDUCTION

- The SD reduction is based on the decrease in the SD after a dataset is split on an attribute
- Constructing a decision tree is all about finding attribute that returns the highest SD reduction
- The split is done on the feature which returns max SD reduction
- Dataset is divided based on the values of the selected feature
- A branch set with SD > 0 needs further splitting, the process is repeated on the non-leaf branches, until all data is processed
- When the number of instances is more than 1 at a leaf node, we calculate the average as the final value for the prediction

SCIKIT LEARN PARAMETERS

- class sklearn.tree.DecisionTreeRegressor(criterion='mse', splitter='best', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=None, random_state=None, max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, presort=False)

- criterion : string, optional (default="mse")
- The function to measure the quality of a split.
- Supported criteria are
  - "mse" for the mean squared error, which is equal to variance reduction as feature selection criterion and minimizes the L2 loss using the mean of each terminal node,
  - "friedman_mse", which uses mean squared error with Friedman's improvement score for potential splits, and
  - "mae" for the mean absolute error, which minimizes the L1 loss using the median of each terminal node.

SCIKIT LEARN PARAMETERS

- class sklearn.tree.DecisionTreeRegressor(criterion=’mse’, splitter=’best’, max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=None, random_state=None, max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, presort=False)
- max_depth : int or None, optional (default=None)
- min_samples_split : int, float, optional (default=2)
- min_samples_leaf
- min_weight_fraction_leaf : float, optional (default=0.)
- max_features : int, float, string or None, optional (default=None)

ATTRIBUTES

- feature_importances_ : array of shape = [n_features] - Return the feature importances.
- max_features_ : int, - The inferred value of max_features.
- n_features_ : int - The number of features when fit is performed.
- n_outputs_ : int - The number of outputs when fit is performed.

USE CASES

- Building knowledge management platforms for customer service that improve first call resolution, average handling time, and customer satisfaction rates
- In finance, forecasting future outcomes and assigning probabilities to those outcomes
- Binomial option pricing predictions and real option analysis
- Customer's willingness to purchase a given product in a given setting, i.e. offline and online both
- Product planning; for example, Gerber Products, Inc. used decision trees to decide whether to continue planning PVC for manufacturing toys or not
- General business decision-making
- Loan approval

ADVANTAGES

- Simple to understand and to interpret. Trees can be visualized.
- Requires little data preparation.
  - Other techniques often require
    - data normalization,
    - dummy variables need to be created
    - blank values to be removed.
- Able to handle both numerical and categorical data.
- Able to handle multi-output problems.
- Resistant to outliers, hence require little data preprocessing
- Highly flexible hypothesis space, as the # of nodes (or depth) of tree increases, decision tree can represent increasingly complex decision boundaries

DISADVANTAGES

- Prone to overfitting (overly-complex)

- Can create biased learned trees if some classes dominate.
  - It is therefore recommended to balance the dataset prior to fitting with the decision tree.

- Decision trees can be unstable because small variations in the data might result in a completely different tree being generated.
  - This problem is mitigated by using decision trees within an ensemble.

DECISION TREE - OVERFITTING

- Overfitting is a significant practical difficulty for decision tree models and many other predictive models.
- Overfitting happens when the learning algorithm continues to develop hypotheses that reduce training set error at the cost of an increased test set error.
- There are several approaches to avoiding overfitting in building decision trees.
  - Pre-pruning that stop growing the tree earlier, before it perfectly classifies the training set.
  - Post-pruning that allows the tree to perfectly classify the training set, and then post prune the tree.
- Practically, the second approach of post-pruning overfit trees is more successful because it is *not easy to precisely estimate when to stop growing the tree*.

TIPS ON PRACTICAL USE

- Decision trees tend to overfit on data with a large number of features. Getting the right ratio of samples to number of features is important, since a tree with few samples in high dimensional space is very likely to overfit.
- Consider performing dimensionality reduction (PCA, ICA, or Feature selection) beforehand to give your tree a better chance of finding features that are discriminative.
- Visualize your tree as you are training by using the export function.
- Use max_depth=3 as an initial tree depth to get a feel for how the tree is fitting to your data and then increase the depth.
- Remember that the number of samples required to populate the tree doubles for each additional level the tree grows too. Use max_depth to control the size of the tree to prevent overfitting.

STEPS

The important step of tree pruning is to define a criterion be used to determine the correct final tree size using one of the following methods:

1. Use a distinct dataset from the training set (called validation set), to evaluate the effect of post-pruning nodes from the tree.

2. Build the tree by using the training set, then apply a statistical test to estimate whether pruning or expanding a particular node is likely to produce an improvement beyond the training set.
   - Error estimation
   - Significance testing (e.g., Chi-square test)

3. Minimum Description Length principle : Use an explicit measure of the complexity for encoding the training set and the decision tree, stopping growth of the tree when this encoding size (size(tree) + size(misclassifications(tree)) is minimized.

TIPS ON PRACTICAL USE

- Use min_samples_split or min_samples_leaf to ensure that multiple samples inform every decision in the tree, by controlling which splits will be considered.

- A very small number will usually mean the tree will overfit, whereas a large number will prevent the tree from learning the data.

- Try min_samples_leaf=5 as an initial value. If the sample size varies greatly, a float number can be used as percentage in these two parameters.

- While min_samples_split can create arbitrarily small leaves, min_samples_leaf guarantees that each leaf has a minimum size, avoiding low-variance, over-fit leaf nodes in regression problems.

- For classification with few classes, min_samples_leaf=1 is often the best choice.

TIPS ON PRACTICAL USE

- Balance your dataset before training to prevent the tree from being biased toward the classes that are dominant.
- Class balancing can be done by sampling an equal number of samples from each class, or preferably by normalizing the sum of the sample weights (sample_weight) for each class to the same value.
- Also note that weight-based pre-pruning criteria, such as min_weight_fraction_leaf, will then be less biased toward dominant classes than criteria that are not aware of the sample weights, like min_samples_leaf.
- If the samples are weighted, it will be easier to optimize the tree structure using weight-based pre-pruning criterion such as min_weight_fraction_leaf, which ensure that leaf nodes contain at least a fraction of the overall sum of the sample weights.
- If the input matrix X is very sparse, it is recommended to convert to sparse csc_matrix before calling fit and sparse csr_matrix before calling predict. Training time can be orders of magnitude faster for a sparse matrix input compared to a dense matrix when features have zero values in most of the samples.