

A walk through Decision Trees and Random Forests

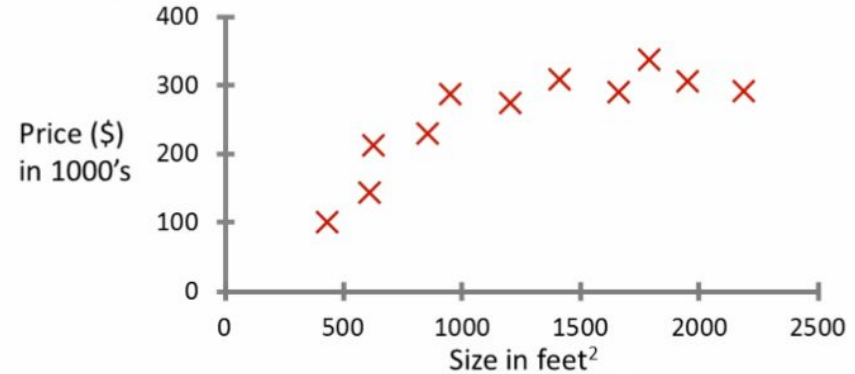
-MANSI BREJA

Kinds of ML Algorithms

Supervised Learning

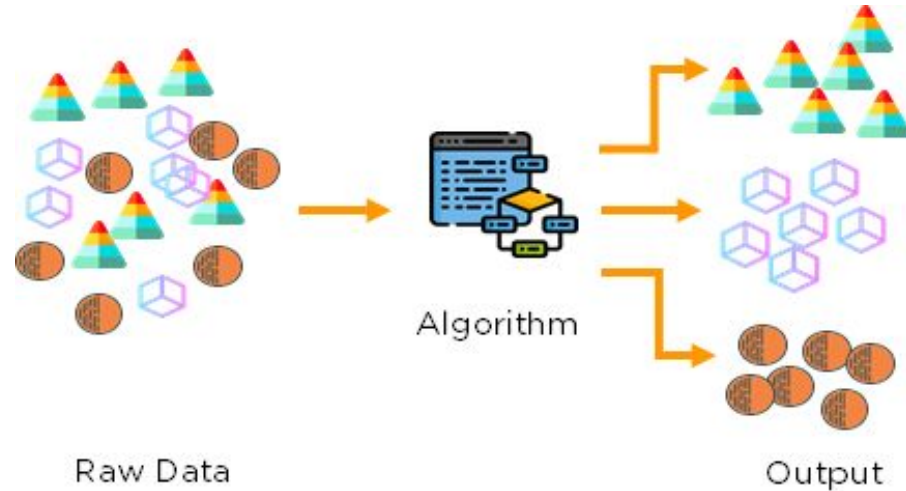
- Supervised learning as the name indicates, is the presence of a supervisor as teacher.
- Basically supervised learning is a learning in which we teach or train the machine using data which is well labeled that means some data is already tagged with correct answer.
- After that, machine is provided with new set of examples(data) so that supervised learning algorithm analyses the training data(set of training examples) and produces a correct outcome from labeled data.

Housing price prediction.



Unsupervised Learning

- Unsupervised learning is the training of machine using information that is neither classified nor labeled and allowing the algorithm to act on that information without guidance.
- Here the task of machine is to group unsorted information according to similarities, patterns and differences without any prior training of data.
- Unlike supervised learning, no teacher is provided that means no training will be given to the machine.
- Therefore machine is restricted to find the hidden structure in unlabeled data by itself.



Supervised learning classified into two categories of algorithms:

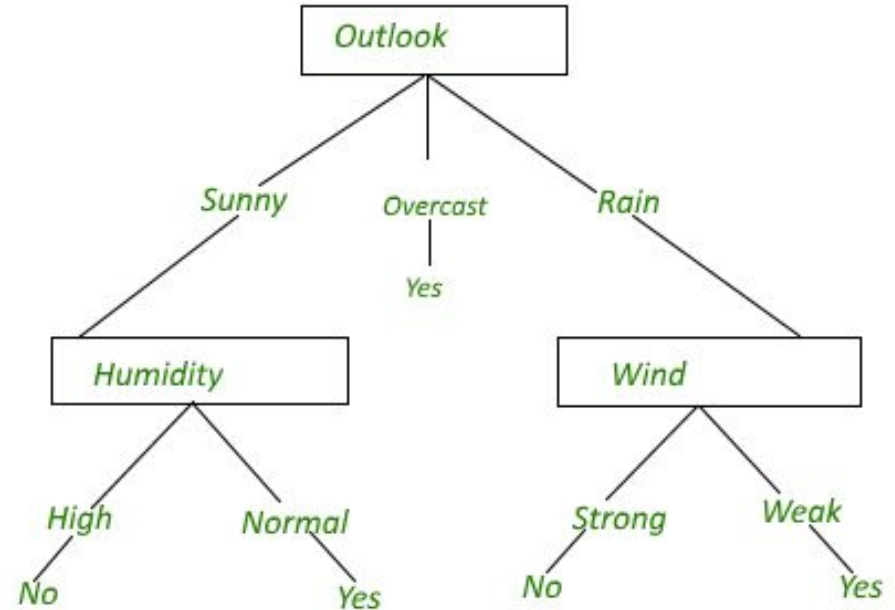
- Classification: A classification problem is when the output variable is a category, such as “Red” and “blue” or “disease” and “no disease”.
- Regression: A regression problem is when the output variable is a real value, such as “dollars” or “weight”.

Decision Trees

Decision tree is the most powerful and popular tool for classification and prediction.

A Decision tree is a flowchart like tree structure, where each internal node denotes a test on an attribute, each branch represents an outcome of the test, and each leaf node (terminal node) holds a class label.

Decision Tree for *PlayTennis*



Construction of Decision Tree :

- A tree can be “*learned*” by splitting the source set into subsets based on an attribute value test. This process is repeated on each derived subset in a recursive manner called *recursive partitioning*.
- The recursion is completed when in the subset at a node, all have the same value of the target variable, or when splitting no longer adds value to the predictions.
- The construction of decision tree classifier does not require any domain knowledge or parameter setting, and therefore is appropriate for exploratory knowledge discovery. Decision trees can handle high dimensional data.
- In general decision tree classifier has good accuracy.

Decision Tree Representation :

- Decision trees classify instances by sorting them down the tree from the root to some leaf node, which provides the classification of the instance.
- An instance is classified by starting at the root node of the tree, testing the attribute specified by this node, then moving down the tree branch corresponding to the value of the attribute as shown in the above figure. This process is then repeated for the subtree rooted at the new node.
- The decision tree in above figure classifies a particular morning according to whether it is suitable for playing tennis and returning the classification associated with the particular leaf. (in this case Yes or No).

For example, the instance

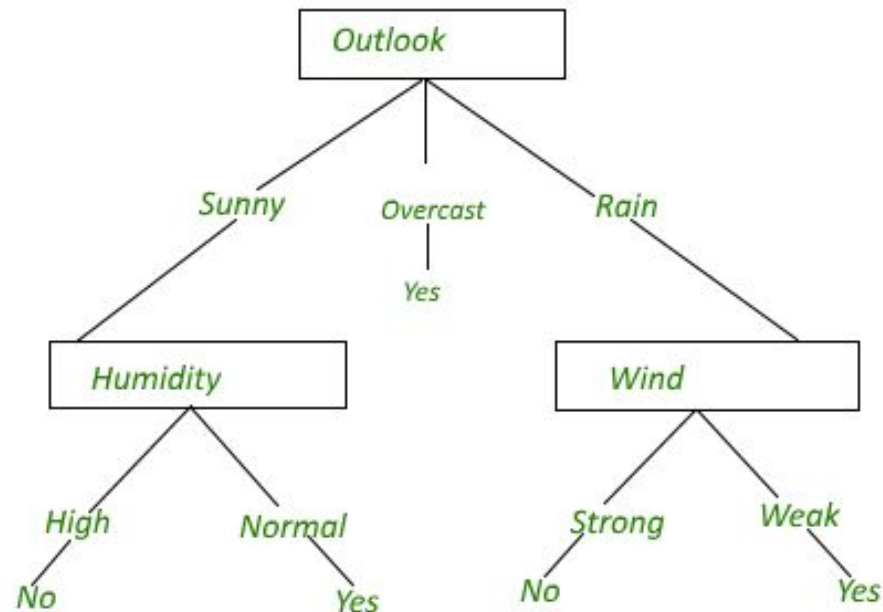
(Outlook = Rain, Temperature = Hot, Humidity = High, Wind = Strong)

would be sorted down the branches of this decision tree and would therefore be classified as a negative instance.

In other words we can say that decision tree represent a disjunction of conjunctions of constraints on the attribute values of instances.

*(Outlook = Sunny \wedge Humidity = Normal) \vee
(Outlook = Overcast) \vee (Outlook = Rain \wedge Wind = Weak)*

Decision Tree for *PlayTennis*



Pruning

Pruning is a technique in machine learning that reduces the size of decision trees by removing sections of the tree that provide little power to classify instances. Pruning reduces the complexity of the final classifier, and hence improves predictive accuracy by the reduction of overfitting.

There are several approaches to avoiding overfitting in building decision trees.

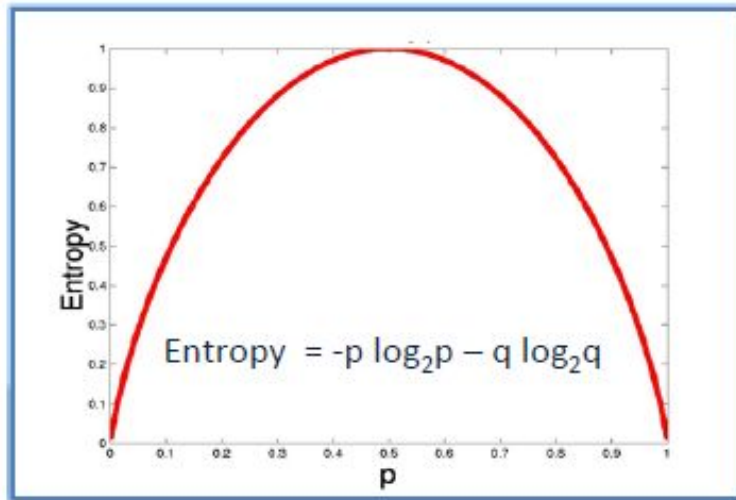
- **Pre-pruning** that stop growing the tree earlier, before it perfectly classifies the training set.
- **Post-pruning** that allows the tree to perfectly classify the training set, and then post prune the tree.

ID3 Algorithm

- Decision tree algorithms transform raw data to rule based decision making trees.
- Herein, ID3 is one of the most common decision tree algorithm.
- Firstly, It was introduced in 1986 and it is an acronym of Iterative Dichotomiser.
- First of all, dichotomisation means dividing into two completely opposite things.
- That's why, the algorithm iteratively divides attributes into two groups which are the most dominant attribute and others to construct a tree.
- Then, it calculates the entropy and information gains of each attribute. In this way, the most dominant attribute can be found out. After that, the most dominant one is put on the tree as decision node.
- Thereafter, entropy and gain scores would be calculated again among the other attributes. Thus, the next most dominant attribute is found. Finally, this procedure continues until reaching a decision for that branch. That's why, it is called Iterative Dichotomiser.

Entropy:

ID3 algorithm uses entropy to calculate the homogeneity of a sample. If the sample is completely homogeneous the entropy is zero and if the sample is equally divided then it has entropy of one.



$$\text{Entropy} = -0.5 \log_2 0.5 - 0.5 \log_2 0.5 = 1$$

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

Play Golf	
Yes	No
9	5

$$\begin{aligned}\text{Entropy}(\text{PlayGolf}) &= \text{Entropy}(5,9) \\ &= \text{Entropy}(0.36, 0.64) \\ &= -(0.36 \log_2 0.36) - (0.64 \log_2 0.64) \\ &= 0.94\end{aligned}$$

$$E(T, X) = \sum_{c \in X} P(c)E(c)$$

		Play Golf		
		Yes	No	
Outlook	Sunny	3	2	5
	Overcast	4	0	4
	Rainy	2	3	5
				14



$$\begin{aligned}
 E(\text{PlayGolf}, \text{Outlook}) &= P(\text{Sunny}) * E(3,2) + P(\text{Overcast}) * E(4,0) + P(\text{Rainy}) * E(2,3) \\
 &= (5/14) * 0.971 + (4/14) * 0.0 + (5/14) * 0.971 \\
 &= 0.693
 \end{aligned}$$

Information Gain

The information gain is based on the decrease in entropy after a data-set is split on an attribute. Constructing a decision tree is all about finding attribute that returns the highest information gain (i.e., the most homogeneous branches).

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.

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.

		Play Golf	
		Yes	No
Outlook	Sunny	3	2
	Overcast	4	0
	Rainy	2	3
Gain = 0.247			

		Play Golf	
		Yes	No
Temp.	Hot	2	2
	Mild	4	2
	Cool	3	1
Gain = 0.029			

		Play Golf	
		Yes	No
Humidity	High	3	4
	Normal	6	1
Gain = 0.152			

		Play Golf	
		Yes	No
Windy	False	6	2
	True	3	3
Gain = 0.048			

$$Gain(T, X) = Entropy(T) - Entropy(T, X)$$

$$\begin{aligned} G(\text{PlayGolf, Outlook}) &= E(\text{PlayGolf}) - E(\text{PlayGolf, Outlook}) \\ &= 0.940 - 0.693 = 0.247 \end{aligned}$$

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

Entropy(S) =

$$\sum - p(l) \cdot \log_2 p(l)$$

Gain(S, A) =

Entropy(S) –

$$\sum [p(S|A) \cdot \text{Entropy}(S|A)]$$

Entropy

We need to calculate the entropy first. Decision column consists of 14 instances and includes two labels: yes and no. There are 9 decisions labeled yes, and 5 decisions labeled no.

$$\text{Entropy}(\text{Decision}) = - p(\text{Yes}) \cdot \log_2 p(\text{Yes}) - p(\text{No}) \cdot \log_2 p(\text{No})$$

$$\text{Entropy}(\text{Decision}) = - (9/14) \cdot \log_2(9/14) - (5/14) \cdot \log_2(5/14) = 0.940$$

Now, we need to find the most dominant factor for decisioning.

Wind factor on decision

$$\text{Gain}(\text{Decision}, \text{Wind}) = \text{Entropy}(\text{Decision}) - \sum [p(\text{Decision}|\text{Wind}) \cdot \text{Entropy}(\text{Decision}|\text{Wind})]$$

Wind attribute has two labels: weak and strong. We would reflect it to the formula.

$$\text{Gain}(\text{Decision}, \text{Wind}) = \text{Entropy}(\text{Decision}) - [p(\text{Decision}|\text{Wind}=\text{Weak}) \cdot \text{Entropy}(\text{Decision}|\text{Wind}=\text{Weak})] - [p(\text{Decision}|\text{Wind}=\text{Strong}) \cdot \text{Entropy}(\text{Decision}|\text{Wind}=\text{Strong})]$$

Now, we need to calculate (Decision|Wind=Weak) and (Decision|Wind=Strong) respectively.

Weak wind factor on decision

Day	Outlook	Temp.	Humidity	Wind	Decision
1	Sunny	Hot	High	Weak	No
3	Overcast	Hot	High	Weak	Yes
4	Rain	Mild	High	Weak	Yes
5	Rain	Cool	Normal	Weak	Yes
8	Sunny	Mild	High	Weak	No
9	Sunny	Cool	Normal	Weak	Yes
10	Rain	Mild	Normal	Weak	Yes
13	Overcast	Hot	Normal	Weak	Yes

There are 8 instances for weak wind. Decision of 2 items are no and 6 items are yes as illustrated below.

$$\begin{aligned} &1- \text{Entropy}(\text{Decision}|\text{Wind}=\text{Weak}) \\ &= - p(\text{No}) \cdot \log_2 p(\text{No}) - p(\text{Yes}) \cdot \log_2 p(\text{Yes}) \end{aligned}$$

$$\begin{aligned} &2- \text{Entropy}(\text{Decision}|\text{Wind}=\text{Weak}) \\ &= - (2/8) \cdot \log_2(2/8) - (6/8) \cdot \log_2(6/8) = 0.811 \end{aligned}$$

Strong wind factor on decision

Day	Outlook	Temp.	Humidity	Wind	Decision
2	Sunny	Hot	High	Strong	No
6	Rain	Cool	Normal	Strong	No
7	Overcast	Cool	Normal	Strong	Yes
11	Sunny	Mild	Normal	Strong	Yes
12	Overcast	Mild	High	Strong	Yes
14	Rain	Mild	High	Strong	No

Here, there are 6 instances for strong wind. Decision is divided into two equal parts.

$$\begin{aligned} &1- \text{Entropy}(\text{Decision}|\text{Wind}=\text{Strong}) \\ &= - p(\text{No}) \cdot \log_2 p(\text{No}) - p(\text{Yes}) \cdot \log_2 p(\text{Yes}) \end{aligned}$$

$$\begin{aligned} &2- \text{Entropy}(\text{Decision}|\text{Wind}=\text{Strong}) \\ &= - (3/6) \cdot \log_2(3/6) - (3/6) \cdot \log_2(3/6) = 1 \end{aligned}$$

Now, we can turn back to Gain(Decision, Wind) equation.

Gain(Decision, Wind)

$$\begin{aligned} &= \text{Entropy(Decision)} - [p(\text{Decision}|\text{Wind}=\text{Weak}) \cdot \\ &\text{Entropy(Decision}|\text{Wind}=\text{Weak})] - [\\ &p(\text{Decision}|\text{Wind}=\text{Strong}) \cdot \\ &\text{Entropy(Decision}|\text{Wind}=\text{Strong})] \\ &= 0.940 - [(8/14) \cdot 0.811] - [(6/14) \cdot 1] = 0.048 \end{aligned}$$

Calculations for wind column is over. Now, we need to apply same calculations for other columns to find the most dominant factor on decision.

Other factors on decision

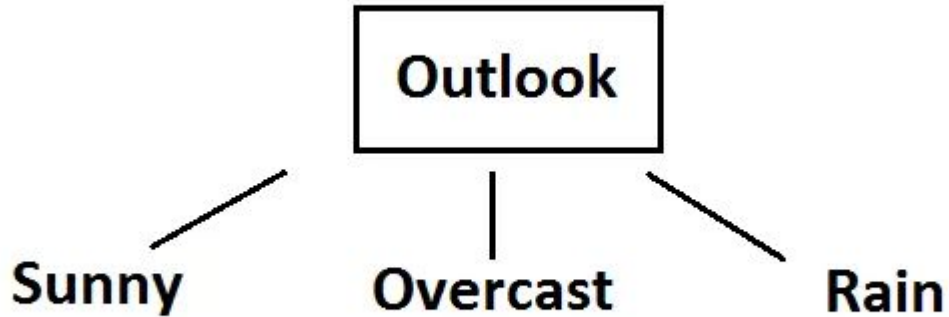
We have applied similar calculation on the other columns.

1- $\text{Gain(Decision, Outlook)} = 0.246$

2- $\text{Gain(Decision, Temperature)} = 0.029$

3- $\text{Gain(Decision, Humidity)} = 0.151$

As seen, outlook factor on decision produces the highest score. That's why, outlook decision will appear in the root node of the tree.



Overcast outlook on decision

Basically, decision will always be yes if outlook were overcast.

Day	Outlook	Temp.	Humidity	Wind	Decision
3	Overcast	Hot	High	Weak	Yes
7	Overcast	Cool	Normal	Strong	Yes
12	Overcast	Mild	High	Strong	Yes
13	Overcast	Hot	Normal	Weak	Yes

Sunny outlook on decision

Day	Outlook	Temp.	Humidity	Wind	Decision
1	Sunny	Hot	High	Weak	No
2	Sunny	Hot	High	Strong	No
8	Sunny	Mild	High	Weak	No
9	Sunny	Cool	Normal	Weak	Yes
11	Sunny	Mild	Normal	Strong	Yes

Here, there are 5 instances for sunny outlook. Decision would be probably 3/5 percent no, 2/5 percent yes.

1- $\text{Gain}(\text{Outlook}=\text{Sunny}|\text{Temperature}) = 0.570$

2- $\text{Gain}(\text{Outlook}=\text{Sunny}|\text{Humidity}) = 0.970$

3- $\text{Gain}(\text{Outlook}=\text{Sunny}|\text{Wind}) = 0.019$

Now, humidity is the decision because it produces the highest score if outlook were sunny.

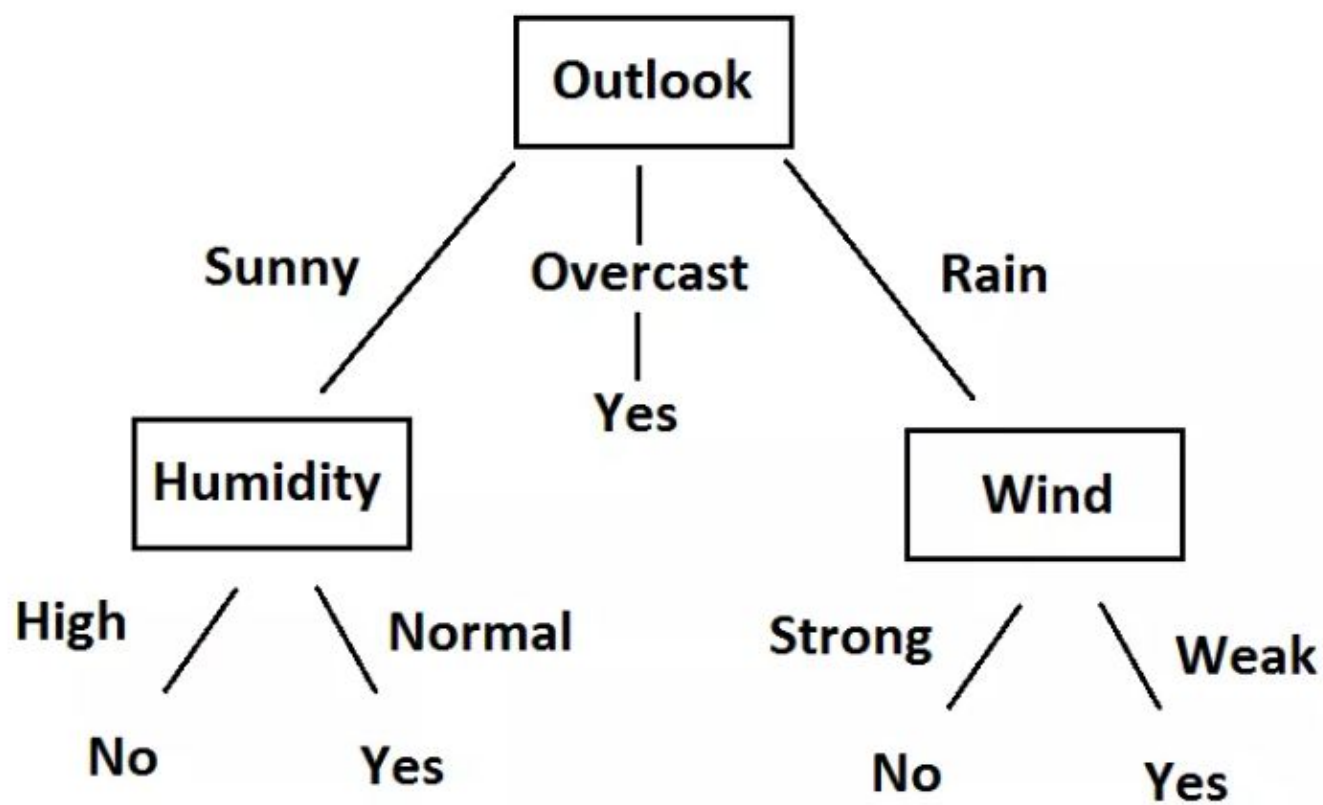
At this point, decision will always be no if humidity were high.

Day	Outlook	Temp.	Humidity	Wind	Decision
1	Sunny	Hot	High	Weak	No
2	Sunny	Hot	High	Strong	No
8	Sunny	Mild	High	Weak	No

On the other hand, decision will always be yes if humidity were normal

Day	Outlook	Temp.	Humidity	Wind	Decision
9	Sunny	Cool	Normal	Weak	Yes
11	Sunny	Mild	Normal	Strong	Yes

Finally, it means that we need to check the humidity and decide if outlook were sunny.



Conclusion

So, decision tree algorithms transform the raw data into rule based mechanism.

They can use nominal attributes whereas most of common machine learning algorithms cannot. However, it is required to transform numeric attributes to nominal in ID3.

Besides, its evolved version C4.5 exists which can handle nominal data. Even though decision tree algorithms are powerful, they have long training time.

On the other hand, they tend to fall over-fitting. Besides, they have evolved versions named **random forests** which tend not to fall over-fitting issue and have shorter training times.

Credits: <https://sefiks.com/2017/11/20/a-step-by-step-id3-decision-tree-example/>

Random Forests

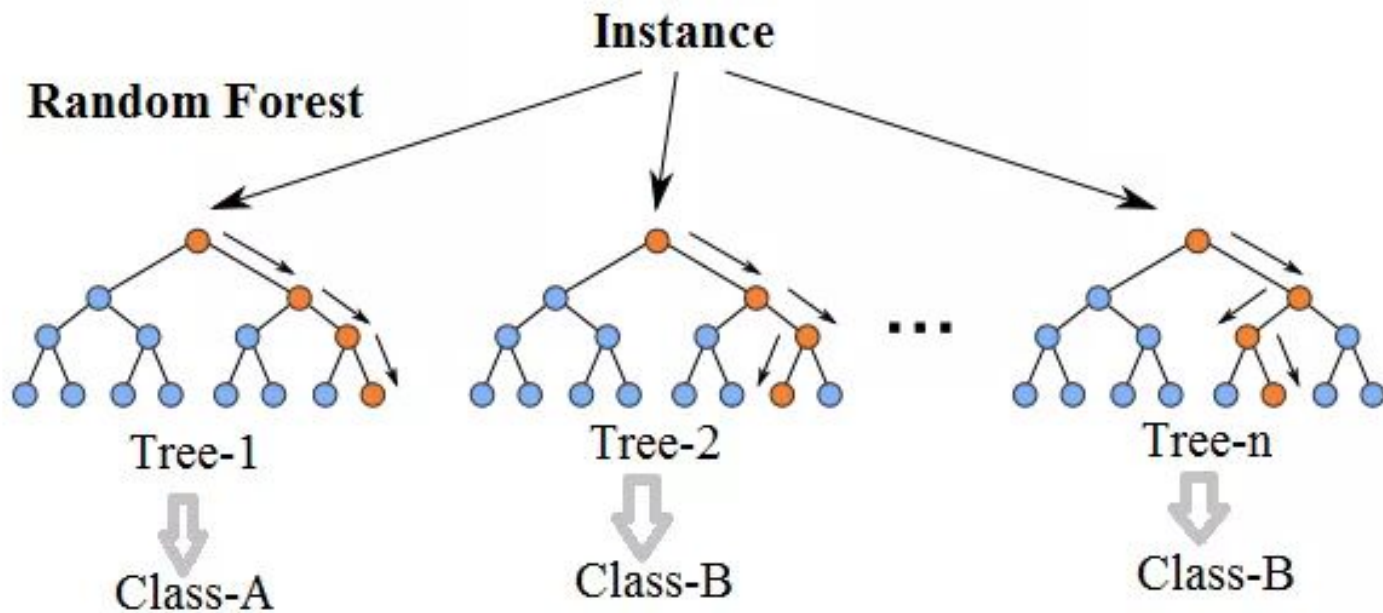
- Life cycle of a tree begins with a seed.
- Then seed grows and becomes a young plant.
- Young plant is transformed to a tree in years.
- After that, trees convert their areas to forest.
- Giant forests all came from a single seed at first.

The rule based systems follow similar steps.

Data would be a seed in this lifecycle. Growing data becomes dataset and that would be a young plant. Decision tree algorithms transform datasets to rule based trees. They are applied to dataset and it becomes to a tree.

Herein, random forest is a new algorithm derived from decision trees. Instead of applying decision tree algorithm on all dataset, dataset would be separated into subsets and same decision tree algorithm would be applied to these subsets. Decision would be made by the highest number of subset results.

Random Forest Simplified



Wikipedia says:

Random forests or **random decision forests** are an ensemble learning method for classification, regression and other tasks that operates by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees.

To say it in simple words:

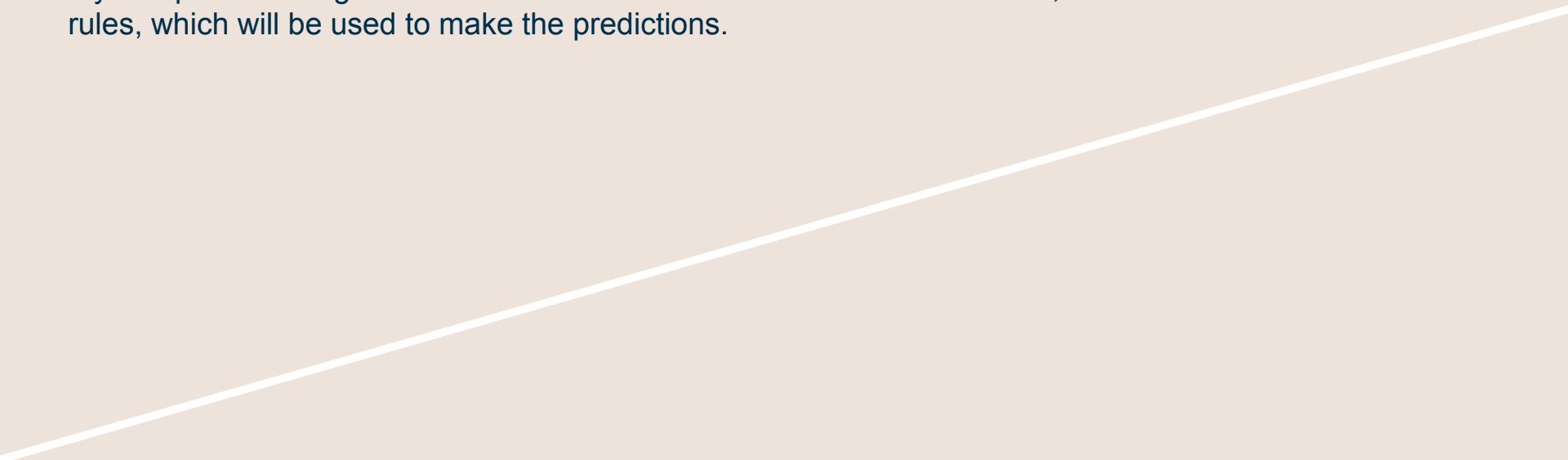
Random forest builds multiple decision trees and merges them together to get a more accurate and stable prediction.

The training algorithm for random forests applies the general technique of bootstrap aggregating, or bagging, to tree learners.

Difference between Decision Trees and Random Forests:

Like I already mentioned, Random Forest is a collection of Decision Trees, but there are some differences.

If you input a training dataset with features and labels into a decision tree, it will formulate some set of rules, which will be used to make the predictions.



For example, if you want to predict whether a person will click on an online advertisement, you could collect the ad's the person clicked in the past and some features that describe his decision.

- 1) If you put the features and labels into a decision tree, it will generate some rules. Then you can predict whether the advertisement will be clicked or not.

In comparison, the Random Forest algorithm randomly selects observations and features to build several decision trees and then averages the results.

- 2) Another difference is that “deep” decision trees might suffer from overfitting.

Random Forest prevents overfitting most of the time, by creating random subsets of the features and building smaller trees using these subsets. Afterwards, it combines the subtrees.

Decision Tree is Good, but Random Forests are Better

So, why traditional decision tree algorithm evolved into random forests?

Working on all dataset may cause overfitting. In other words, it might cause memorizing instead of learning.

In this case, you would have high accuracy on training set, but you would fail on new instances. What's more, random forests work on multiple and small datasets. Increasing the dataset size would increase the learning time exponentially in decision tree.

So, you can parallelize the learning procedure in random forest. In this way, learning time would last less than decision tree.

If decision tree algorithm were the wise / sage person around you, then random forests would be multiple smart people. Wise one might know every domain but each smart person can be expert on different domains. Wise one would most probably respond the correct answer but you might not always have a opportunity to ask him. However, bringing smart people to gather would most probably be acceptable.

How the Random Forest Algorithm Works

The following are the basic steps involved in performing the random forest algorithm:

1. Pick N random records from the dataset.
2. Build a decision tree based on these N records.
3. Choose the number of trees you want in your algorithm and repeat steps 1 and 2.
4. In case of a regression problem, for a new record, each tree in the forest predicts a value for Y (output). The final value can be calculated by taking the average of all the values predicted by all the trees in forest. Or, in case of a classification problem, each tree in the forest predicts the category to which the new record belongs. Finally, the new record is assigned to the category that wins the majority vote.

Using Random Forest for Classification

Problem Definition

The task here is to predict whether a bank currency note is authentic or not based on four attributes i.e. variance of the image wavelet transformed image, skewness, entropy, and kurtosis of the image.

Solution

This is a binary classification problem and we will use a random forest classifier to solve this problem.

CODE:

1. Import Libraries

Execute the following code to import the necessary libraries:

```
import pandas as pd
import numpy as np
```

2. Importing Dataset

The dataset for this problem is available at:

<https://drive.google.com/file/d/1mVmGNx6cbfvRHCDvF12ZL3wGLSHD9f/view>

Execute the following command to import the dataset:

```
dataset = pd.read_csv('D:\Datasets\petrol_consumption.csv')
```

To get a high-level view of what the dataset looks like, execute the following command:

```
dataset.head()
```

	Variance	Skewness	Curtosis	Entropy	Class
0	3.62160	8.6661	-2.8073	-0.44699	0
1	4.54590	8.1674	-2.4586	-1.46210	0
2	3.86600	-2.6383	1.9242	0.10645	0
3	3.45660	9.5228	-4.0112	-3.59440	0
4	0.32924	-4.4552	4.5718	-0.98880	0

We can see that the values in our dataset are not very well scaled. We will scale them down before training the algorithm.

3. Preparing Data For Training

Two tasks will be performed in this section. The first task is to divide data into 'attributes' and 'label' sets. The resultant data is then divided into training and test sets.

The following script divides data into attributes and labels:

```
X = dataset.iloc[:, 0:4].values  
y = dataset.iloc[:, 4].values
```

Finally, let's divide the data into training and testing sets:

```
from sklearn.model_selection import train_test_split  
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
```

4. Feature Scaling

```
# Feature Scaling  
from sklearn.preprocessing import StandardScaler  
sc = StandardScaler()  
X_train = sc.fit_transform(X_train)  
X_test = sc.transform(X_test)
```

5. Training the Algorithm

Now that we have scaled our dataset, it is time to train our random forest algorithm to solve this regression problem. Execute the following code:

```
from sklearn.ensemble import RandomForestClassifier
classifier = RandomForestClassifier(n_estimators=20, random_state=0)
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
```

The `RandomForestClassifier` class of the `sklearn.ensemble` library is used to solve classification problems via random forest. The most important parameter of the `RandomForestClassifier` class is the `n_estimators` parameter. This parameter defines the number of trees in the random forest. We will start with `n_estimators=20` to see how our algorithm performs.

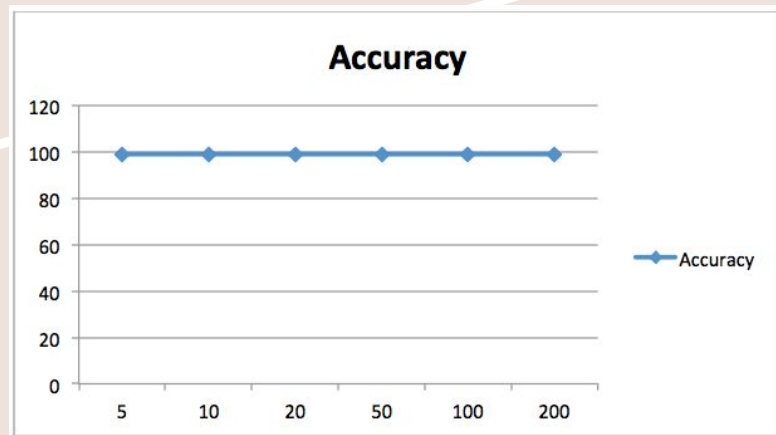
6. Evaluating the Algorithm

For classification problems the metrics used to evaluate an algorithm are accuracy, confusion matrix, precision recall, and F1 values. Execute the following script to find these values:

```
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
```

```
print(confusion_matrix(y_test,y_pred))  
print(classification_report(y_test,y_pred))  
print(accuracy_score(y_test, y_pred))
```

98.90% is a pretty good accuracy, so there isn't much point in increasing our number of estimators anyway. We can see that increasing the number of estimators did not further improve the accuracy.



Credits: <https://stackabuse.com/random-forest-algorithm-with-python-and-scikit-learn/>
https://medium.com/@rishabhjain_22692/decision-trees-it-begins-here-93ff54ef134

Thanks for your time!

:"")