Machine Learning Techniques

Instance Based Learning:

Instance-based learning is a family of learning algorithms that, instead of performing explicit generalization, compares new problem instances with instances seen in training, which have been stored in memory.

They are sometimes referred to as lazy learning methods because they delay processing until a new instance must be classified. The nearest neighbors of an instance are defined in terms of Euclidean distance.

Instance-based learning representation:

It generates classification predictions using only specific instances. Instance-based learning algorithms do not maintain a set of abstractions derived from specific instances. This approach extends the nearest neighbor algorithm, which has large storage requirements.

Performance dimensions used for instance-based learning algorithm:

Time complexity of Instance based learning algorithms depends upon the size of training data. Time complexity of this algorithm in the worst case is O (n), where n is the number of training items to be used to classify a single new instance.

Functions of instance-based learning:

Instance-based learning refers to a family of techniques for classification and regression, which produce a class label/prediction based on the similarity of the query to its nearest neighbor(s) in the training set.
Functions are as follows:

1. **Similarity:** Similarity is a machine learning method that uses a nearest neighbor approach to identify the similarity of two or more objects to each other based on algorithmic distance functions.
2. **Classification:** Process of categorizing a given set of data into classes, It can be performed on both structured and unstructured data. The process starts with predicting the class of given data points. The classes are often referred to as target, label or categories.
3. **Concept Description:** Much of human learning involves acquiring general concepts from past experiences. This description can then be used to predict the class labels of unlabeled cases.

Advantages and disadvantages of instance-based learning:

Advantages of instance-based learning:

- It has the ability to adapt to previously unseen data, which means that one can store a new instance or drop the old instance.

Disadvantages of instance-based learning:

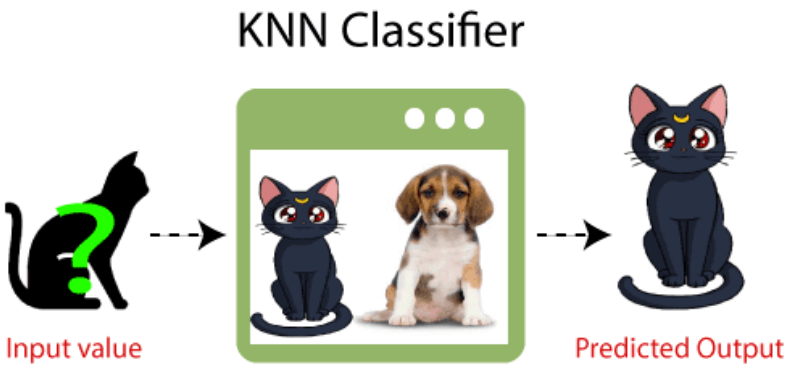
- Classification costs are high.
- Large amount of memory required to store the data, and each query involves starting the identification of a local model from scratch.

K-nearest Neighbors Algorithm:

- K-Nearest Neighbors is one of the simplest Machine Learning algorithms based on Supervised Learning technique.
- K-NN algorithm assumes the similarity between the new case/data and available cases and put the new case into the category that is most similar to the available categories.
- K-NN algorithm stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well suite category by using K- NN algorithm.
- K-NN algorithm can be used for Regression as well as for Classification but mostly it is used for the Classification problems.
- K-NN is a **non-parametric algorithm**, which means it does not make any assumption on underlying data.
- It is also called a **lazy learner algorithm** because it does not learn from the training set immediately instead it stores the dataset and at the time of classification, it performs an action on the dataset.
- KNN algorithm at the training phase just stores the dataset and when it gets new data, then it classifies that data into a category that is much similar to the new data.

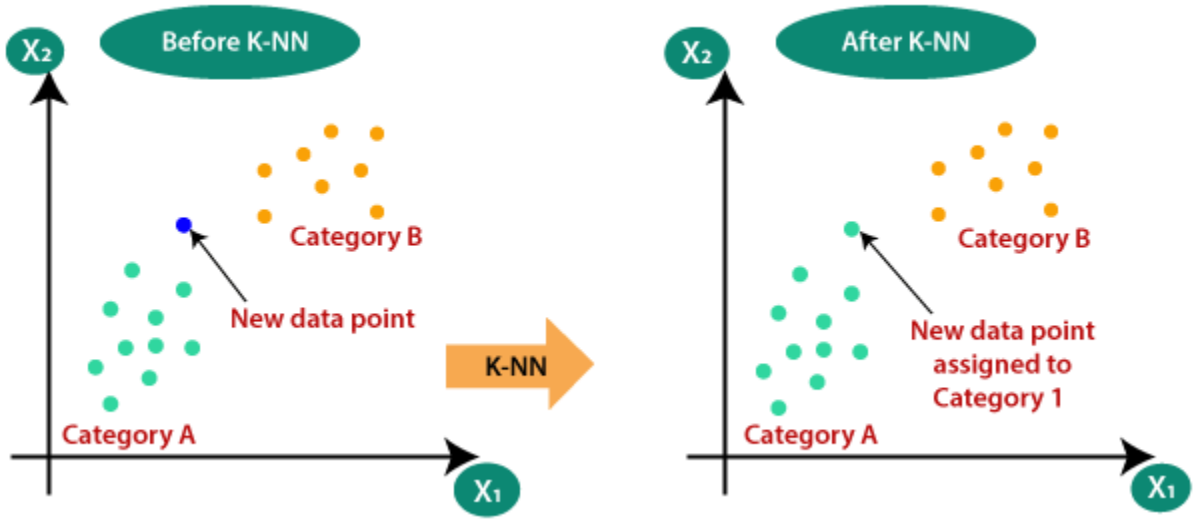
Example:

Suppose, we have an image of a creature that looks similar to cat and dog, but we want to know either it is a cat or dog. So for this identification, we can use the KNN algorithm, as it works on a similarity measure. Our KNN model will find the similar features of the new data set to the cats and dogs images and based on the most similar features it will put it in either cat or dog category.



Why do we need a K-NN Algorithm?

Suppose there are two categories, i.e., Category A and Category B, and we have a new data point x1, so this data point will lie in which of these categories. To solve this type of problem, we need a K-NN algorithm. With the help of K-NN, we can easily identify the category or class of a particular dataset. Consider the below diagram:

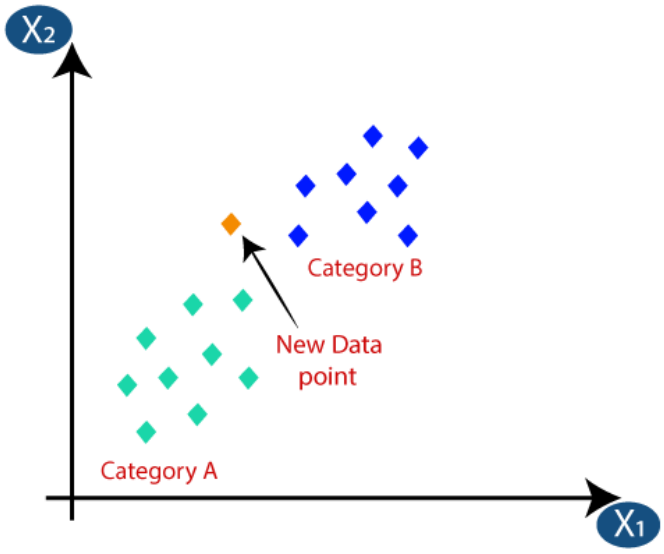


How does K-NN work?

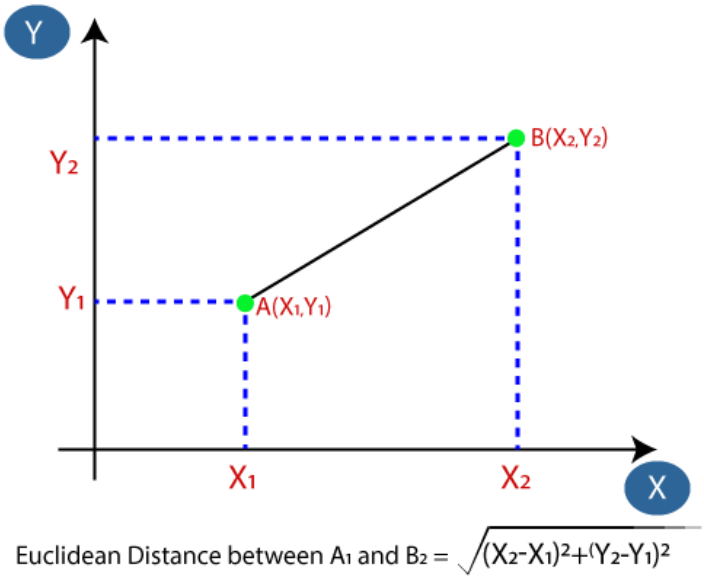
The K-NN working can be explained on the basis of the below algorithm:

- **Step-1:** Select the number K of the neighbors
- **Step-2:** Calculate the Euclidean distance of **K number of neighbors**
- **Step-3:** Take the K nearest neighbors as per the calculated Euclidean distance.
- **Step-4:** Among these k neighbors, count the number of the data points in each category.
- **Step-5:** Assign the new data points to that category for which the number of the neighbor is maximum.
- **Step-6:** Our model is ready.

Suppose we have a new data point and we need to put it in the required category. Consider the below image:



- Firstly, we will choose the number of neighbors, so we will choose the k=5.
- Next, we will calculate the **Euclidean distance** between the data points. The Euclidean distance is the distance between two points, which we have already studied in geometry. It can be calculated as:



- By calculating the Euclidean distance we got the nearest neighbors, as three nearest neighbors in category A and two nearest neighbors in category B. Consider the below image:

- As we can see the 3 nearest neighbors are from category A, hence this new data point must belong to category A.

How to select the value of K in the K-NN Algorithm?

Below are some points to remember while selecting the value of K in the K-NN algorithm:

- There is no particular way to determine the best value for "K", so we need to try some values to find the best out of them. The most preferred value for K is 5.
- A very low value for K such as K=1 or K=2, can be noisy and lead to the effects of outliers in the model.
- Large values for K are good, but it may find some difficulties.

Advantages of KNN Algorithm:

- It is simple to implement.
- It is robust to the noisy training data
- It can be more effective if the training data is large.

Disadvantages of KNN Algorithm:

- Always needs to determine the value of K which may be complex some time.
- The computation cost is high because of calculating the distance between the data points for all the training samples.

Numerical Example of K-NN Algorithm:

Example -1:

Name	Age	Gender	Sports
Ajay	32	M	Football
Manish	40	M	Neither
Vishal	16	M	Cricket
Ankita	34	F	Cricket
Neha	55	F	Neither
Arun	40	M	Cricket
Virendra	20	M	Neither
Mansi	15	F	Cricket
Meghna	55	F	Football
Prem	15	M	Football
Ruchi	5	F	?

Suppose K (Numbers of nearest neighbors) = 3 and Male (M) = 0, Female (F) = 1

Now calculate Euclidean Distance by the formula given below

$D = \text{sqrt}\ [(x2-x1)^2 + (y2-y1)^2]$

For Ajay – M=0 and age = 32

$D = \text{sqrt}\ [(32-5)^2 + (0-1)^2] = 27.02$ => Distance between Ruchi to Ajay

In the same way we calculate the distance between Ruchi to remaining data. So after calculating all the table looks like as:

Name	Age	Gender	Distance	Sports
Ajay	32	0	27.02	Football
Manish	40	0	35.01	Neither
Vishal	16	0	11.00	Cricket
Ankita	34	1	**9.00**	Cricket
Neha	55	1	50.01	Neither
Arun	40	0	35.01	Cricket
Virendra	20	0	15.00	Neither
Mansi	15	1	**10.00**	Cricket
Meghna	55	1	50.00	Football
Prem	15	0	**10.05**	Football
Ruchi	5	1		

Since the maximum closest neighbors are belongs to Cricket so that Ruchi also like the cricket.

Example -2:

We have data from questionnaires survey (to ask people opinion) and objective testing with two attributes (acid durability and strength) to classify whether a special paper tissue is good or not. Here is four training samples

X1=Acid Durability (Seconds)	X2=Strength (kg/square meter)	Y = Classification
7	7	Bad
7	4	Bad
3	4	Good
1	4	Good

Now the factory produces new paper tissue that pass laboratory test with x1 = 3 and x2 = 7. Without another expensive survey, can we guess what the classification of this new tissue is?

Suppose K (Numbers of nearest neighbors) = 3

Now calculate the distance (Coordinate of Query Instance is x1 =3 and x2 =7)

X1=Acid Durability (Seconds)	X2=Strength (kg/square meter)	Distance
7	7	$(7-3)^2 + (7-7)^2 = 16 = 4$
7	4	$(7-3)^2 + (4-7)^2 = 25 = 5$
3	4	$(3-3)^2 + (4-7)^2 = 9 = 3$
1	4	$(1-3)^2 + (4-7)^2 = 13 = 3.60$

Since the maximum closest neighbors are belongs to Good so that the classification of new tissue is good.

Python Code of KNN:

Example-1

Consider a dataset that contains two variables: height (cm) & weight (kg). Each point is classified as normal or underweight.

Weight (x2)	Height (y2)	Class
51	167	Underweight
62	182	Normal
69	176	Normal
64	173	Normal
65	172	Normal
56	174	Underweight
58	169	Normal
57	173	Normal
55	170	Normal

Based on the above data, you need to classify the following set as normal or underweight using the KNN algorithm.
Now, we have a new data point (x1, y1), and we need to determine its class.

x1 = 57 y1 = 170

```
from math import sqrt
# calculate the Euclidean distance between two vectors
def euclidean_distance(row1, row2):
	distance = 0.0
	for i in range(len(row1)-1):
		distance += (row1[i] - row2[i])**2
	return sqrt(distance)


def get_neighbors(train, test_row, num_neighbors):
	distances = list()
	for train_row in train:
		dist = euclidean_distance(train_row, test_row)
		distances.append((train_row, dist))
	distances.sort(key=lambda tup: tup[1])
	neighbors = list()
	for i in range(num_neighbors):
		neighbors.append(distances[i][0])
	return neighbors
# Test distance function
dataset = [
   [51,167,1],
	[62,182,0],
	[69,176,0],
	[64,173,0],
	[65,172,0],
```

```
        [56,174,1],

        [58,169,0],

        [57,173,0],

        [55,170,0],

        [57,170]]
neighbors = get_neighbors(dataset, dataset[9], 5)
for neighbor in neighbors:
        print(neighbor)
```

Output-

```
[57, 170]
[58, 169, 0]
[55, 170, 0]
[57, 173, 0]
[56, 174, 1]
```

Example No. 2(Numerical shown above)

```
from math import sqrt
from __future__ import print_function  # for python 2 compatibility
import sys
ipython = get_ipython()

def exception_handler(exception_type, exception, traceback):
    print("%s: %s" % (exception_type.__name__, exception), file=sys.stderr)

ipython._showtraceback = exception_handler

def euclidean_distance1(r1, r2):
    di = 0.0
    for i in range(len(r1)-1):
        di += ((r1[i] - r2[i]))**2
    return sqrt(di)

def get_neighbors1(complete_data, first_row, num_neighbors):
    d = list()
    for r in complete_data:
        dist = euclidean_distance1(r, first_row)
        d.append(r)
    d.sort(key=lambda tup: tup[1])
    neighbors = list()
    for i in range(num_neighbors):
        neighbors.append(d[i])
    return neighbors

dataset = [
```

```
    [7,7,0],
    [7,4,0],
    [3,4,1],
    [1,4,1],
    [3,7]]

neighbors1 = get_neighbors1(dataset, dataset[4], 3)
for neighbor1 in neighbors1:
    print(neighbor1)
```

Output

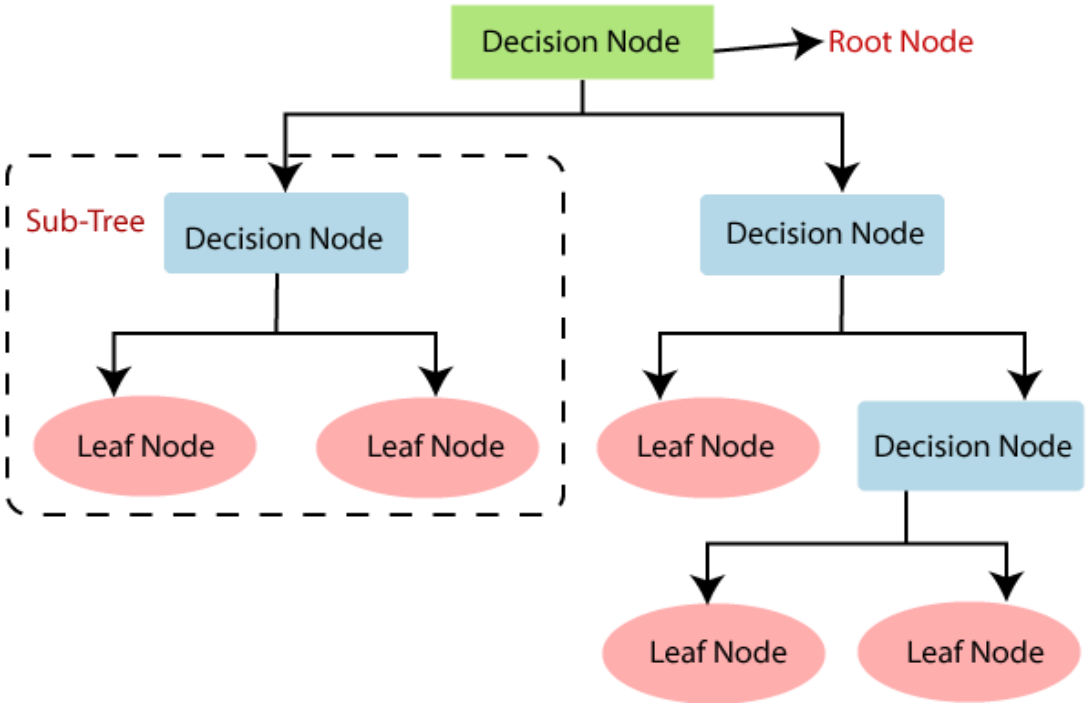
```
[7, 4, 0]
[3, 4, 1]
[1, 4, 1]
```

Decision Tree Learning

Decision Tree:

- Decision Trees are a type of Supervised Machine Learning (that is you explain what the input is and what the corresponding output is in the training data) where the data is continuously split according to a certain parameter.
- The tree can be explained by two entities, namely decision nodes and leaves. The leaves are the decisions or the final outcomes. And the decision nodes are where the data is split.
- Decision Tree can be used to solve both regression and classification problems.
- It is a graphical representation for getting all the possible solutions to a problem/decision based on given conditions.
- It is called a decision tree because, similar to a tree, it starts with the root node, which expands on further branches and constructs a tree-like structure.
- In order to build a tree, we use the **CART algorithm,** which stands for **Classification and Regression Tree algorithm.**
- A decision tree simply asks a question, and based on the answer (Yes/No), it further split the tree into sub trees.

A decision tree can contain categorical data (YES/NO) as well as numeric data.



An example of a decision tree can be explained using below binary tree. Let's say you want to predict whether a person is fit given their information like age, eating habit, and physical activity, etc. The decision nodes here are questions like 'What's the age?', 'Does he exercise?', 'Does he eat a lot of pizzas'? And the leaves, which are outcomes like either 'fit', or 'unfit'. In this case this was a binary classification problem (a yes no type problem).

There are two main types of Decision Trees:

Classification trees (Yes/No types)

What we've seen above is an example of classification tree, where the outcome was a variable like 'fit' or 'unfit'. Here the decision variable is **Categorical**.

Regression trees (Continuous data types)

Here the decision or the outcome variable is **Continuous**, e.g. a number like 123.

Why use Decision Tree:

Below are the two reasons for using the Decision tree:

- Decision Trees usually mimic human thinking ability while making a decision, so it is easy to understand.
- The logic behind the decision tree can be easily understood because it shows a tree-like structure.

Working of Decision Tree Algorithm:

Step-1: Begin the tree with the root node, says S, which contains the complete dataset.

Step-2: Find the best attribute in the dataset using **Attribute Selection Measure (ASM).**

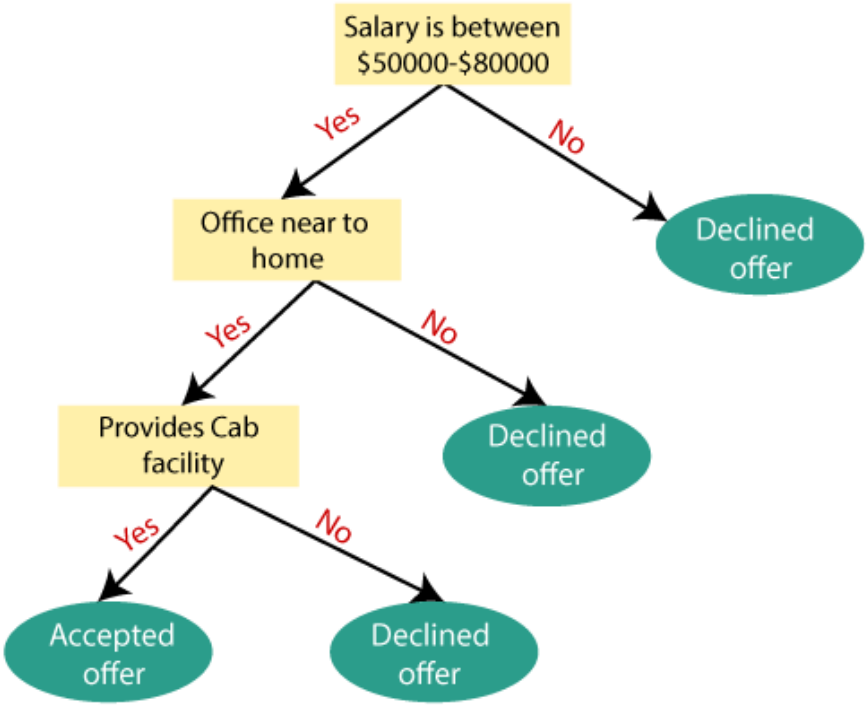
Step-3: Divide the S into subsets that contains possible values for the best attributes.

Step-4: Generate the decision tree node, which contains the best attribute.

Step-5: Recursively make new decision trees using the subsets of the dataset created in step -3. Continue this process until a stage is reached where you cannot further classify the nodes and called the final node as a leaf node.

Example:

Suppose there is a candidate who has a job offer and wants to decide whether he should accept the offer or Not.



The decision tree starts with the root node (Salary attribute by ASM). The root node splits further into the next decision node (distance from the office) and one leaf node based on the corresponding labels. The next decision node further gets split into one decision node (Cab facility) and one leaf node. Finally, the decision node splits into two leaf nodes (Accepted offers and Declined offer).

Attribute Selection Measure:

There are two popular techniques for **Attribute Selection Measure (ASM)** as follows:

- Information Gain
- Gini Index

Information Gain: To understand the concept of **information gain,** first understand the term **Entropy.**

Entropy is metric to measure the impurity in a given attribute. It specifies uncertainty or randomness in data. Entropy can be calculated as:

Entropy(s)= -P(yes)log2 P(yes)- P(no) log2 P(no)

Where, S = total number of samples, P(yes) = Probability of yes and P(no) = Probability of no

Information Gain:

- Information gain is the measurement of changes in entropy after the segmentation of a dataset based on an attribute.
- It calculates how much information a feature provides us about a class.
- According to the value of information gain, we split the node and build the decision tree.
- A decision tree algorithm always tries to maximize the value of information gain, and a node/attribute having the highest information gain is split first. It can be calculated using the below formula:

  Information Gain= Entropy(S)- [(Weighted Avg) *Entropy(each feature)

Gini Index:

- Gini index is a measure of impurity or purity used while creating a decision tree in the CART(Classification and Regression Tree) algorithm.
- An attribute with the low Gini index should be preferred as compared to the high Gini index.
- It only creates binary splits, and the CART algorithm uses the Gini index to create binary splits.
- Gini index can be calculated using the below formula:

$$\text{Gini Index} = 1 - \sum_j P_j^2$$

ID3 Algorithm:

Iterative Dichotomiser 3 or commonly known as ID3. ID3 was invented by Ross Quinlan.

It is a **classification algorithm** that follows a **greedy approach** of **building a decision tree** by selecting a best attribute that yields **maximum Information Gain (IG) or minimum Entropy (H)**.

Decision Tree is most effective if the problem characteristics look like the following points:

Instances can be described by attribute-value pairs.
Target function is discrete-valued.

Steps in ID3 Algorithm:

The **steps in ID3 algorithm** are as follows:

1. Calculate entropy for dataset.

2. For each attribute/feature.
    2.1. Calculate entropy for all its categorical values.
    2.2. Calculate information gain for the feature.

3. Find the feature with maximum information gain.

4. Repeat it until we get the desired tree.

Pruning: Getting an Optimal Decision tree

Pruning is a process of deleting the unnecessary nodes from a tree in order to get the optimal decision tree.

A too-large tree increases the risk of over fitting, and a small tree may not capture all the important features of the dataset. Therefore, a technique that decreases the size of the learning tree without reducing accuracy is known as Pruning. There are mainly two types of tree **pruning** technology used:

Cost Complexity Pruning
Reduced Error Pruning.

Advantages of the Decision Tree

- It is simple to understand as it follows the same process which a human follow while making any decision

in real-life.

- It can be very useful for solving decision-related problems.
- It helps to think about all the possible outcomes for a problem.
- There is less requirement of data cleaning compared to other algorithms.

Disadvantages of the Decision Tree

- The decision tree contains lots of layers, which makes it complex.
- It may have an over fitting issue, which can be resolved using the **Random Forest algorithm.**
- For more class labels, the computational complexity of the decision tree may increase.

The strengths of decision tree methods are:

- Decision trees are able to generate understandable rules.
- Decision trees perform classification without requiring much computation.
- Decision trees are able to handle both continuous and categorical variables.
- Decision trees provide a clear indication of which fields are most important for prediction or classification.

The weakness of decision tree methods is:

- Decision trees are less appropriate for estimation tasks where the goal is to predict the value of a continuous attribute.
- Decision trees are prone to errors in classification problems with many class and relatively small number of training examples.
- Decision tree can be computationally expensive to train. The process of growing a decision tree is computationally expensive. At each node, each candidate splitting field must be sorted before its best split can be found. In some algorithms, combinations of fields are used and a search must be made for optimal combining weights. Pruning algorithms can also be expensive since many candidate sub-trees must be formed and compared.

Issues in Decision Tree:

- Determining how deeply to grow the decision tree,
- Handling continuous attributes,
- Choosing an appropriate attribute selection measure,
- Handling training data with missing attribute values,
- Handling attributes with differing costs, and
- Improving computational efficiency.

Example of Decision Tree:

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

Here, dataset is of binary classes(yes and no), where 9 out of 14 are "yes" and 5 out of 14 are "no".

Complete entropy of dataset is –

H(S) = - p(yes) * log2(p(yes)) - p(no) * log2(p(no))

= - (9/14) * log2(9/14) - (5/14) * log2(5/14)

= - (-0.41) - (-0.53)

= 0.94

First Attribute - Outlook

Categorical values - sunny, overcast and rain

H(Outlook=sunny) = -(2/5)*log(2/5)-(3/5)*log(3/5) =0.971

H(Outlook=rain) = -(3/5)*log(3/5)-(2/5)*log(2/5) =0.971

H(Outlook=overcast) = -(4/4)*log(4/4)-0 = 0

Average Entropy Information for Outlook -

- I(Outlook) = p(sunny) * H(Outlook=sunny) + p(rain) * H(Outlook=rain) + p(overcast) * H(Outlook=overcast)

  = (5/14)*0.971 + (5/14)*0.971 + (4/14)*0

  = 0.693

Information Gain = H(S) - I(Outlook)

= 0.94 - 0.693

= 0.247

Second Attribute - Temperature

Categorical values - hot, mild, cool

H(Temperature=hot) = -(2/4)*log(2/4)-(2/4)*log(2/4) = 1

H(Temperature=cool) = -(3/4)*log(3/4)-(1/4)*log(1/4) = 0.811

H(Temperature=mild) = -(4/6)*log(4/6)-(2/6)*log(2/6) = 0.9179

Average Entropy Information for Temperature -

I(Temperature) = p(hot)*H(Temperature=hot) + p(mild)*H(Temperature=mild) + p(cool)*H(Temperature=cool)

= (4/14)*1 + (6/14)*0.9179 + (4/14)*0.811

= 0.9108

Information Gain = H(S) - I(Temperature)

= 0.94 - 0.9108

= 0.0292

Third Attribute - Humidity

Categorical values - high, normal

H(Humidity=high) = -(3/7)*log(3/7)-(4/7)*log(4/7) = 0.983

H(Humidity=normal) = -(6/7)*log(6/7)-(1/7)*log(1/7) = 0.591

Average Entropy Information for Humidity -

I(Humidity) = p(high)*H(Humidity=high) + p(normal)*H(Humidity=normal)

= (7/14)*0.983 + (7/14)*0.591

= 0.787

Information Gain = H(S) - I(Humidity)

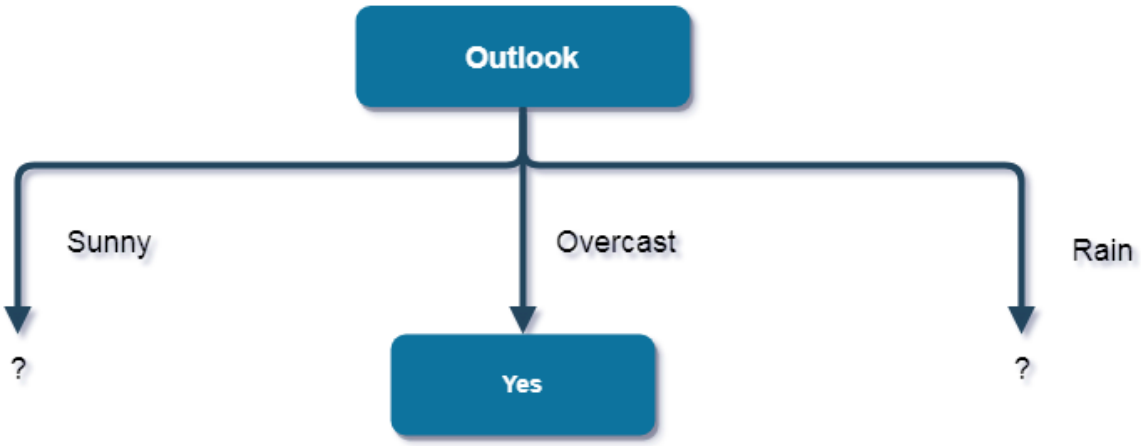
= 0.94 - 0.787

= 0.153

Fourth Attribute - Wind

Categorical values - weak, strong
H(Wind=weak) = -(6/8)*log(6/8)-(2/8)*log(2/8) = 0.811
H(Wind=strong) = -(3/6)*log(3/6)-(3/6)*log(3/6) = 1
Average Entropy Information for Wind -
I(Wind) = p(weak)*H(Wind=weak) + p(strong)*H(Wind=strong)
= (8/14)*0.811 + (6/14)*1
= 0.892
Information Gain = H(S) - I(Wind)
= 0.94 - 0.892
= 0.048

Here, the attribute with maximum information gain is Outlook. So, the decision tree built so far -



Here, when Outlook == overcast, it is of pure class(Yes).

Now, we have to repeat same procedure for the data with rows consist of Outlook value as Sunny and then for Outlook value as Rain

Now, finding the best attribute for splitting the data with Outlook=Sunny values{ Dataset rows = [1, 2, 8, 9, 11]

Complete entropy of Sunny is –

H(S) = - p(yes) * log2(p(yes)) - p(no) * log2(p(no))

= - (2/5) * log2(2/5) - (3/5) * log2(3/5)

= 0.971

First Attribute - Temperature

Categorical values - hot, mild, cool
H(Sunny, Temperature=hot) = -0-(2/2)*log(2/2) = 0
H(Sunny, Temperature=cool) = -(1)*log(1)- 0 = 0
H(Sunny, Temperature=mild) = -(1/2)*log(1/2)-(1/2)*log(1/2) = 1
Average Entropy Information for Temperature -
I(Sunny, Temperature) = p(Sunny, hot)*H(Sunny, Temperature=hot) + p(Sunny, mild)*H(Sunny, Temperature=mild) + p(Sunny, cool)*H(Sunny, Temperature=cool)
= (2/5)*0 + (1/5)*0 + (2/5)*1
= 0.4
Information Gain = H(Sunny) - I(Sunny, Temperature)
= 0.971 - 0.4
= 0.571

Second Attribute - Humidity

Categorical values - high, normal

H(Sunny, Humidity=high) = - 0 - (3/3)*log(3/3) = 0

H(Sunny, Humidity=normal) = -(2/2)*log(2/2)-0 = 0

Average Entropy Information for Humidity -

I(Sunny, Humidity) = p(Sunny, high)*H(Sunny, Humidity=high) + p(Sunny, normal)*H(Sunny, Humidity=normal)

= (3/5)*0 + (2/5)*0

= 0

Information Gain = H(Sunny) - I(Sunny, Humidity)

= 0.971 – 0

= 0.971

Third Attribute - Wind

Categorical values - weak, strong

H(Sunny, Wind=weak) = -(1/3)*log(1/3)-(2/3)*log(2/3) = 0.918

H(Sunny, Wind=strong) = -(1/2)*log(1/2)-(1/2)*log(1/2) = 1

Average Entropy Information for Wind -

I(Sunny, Wind) = p(Sunny, weak)*H(Sunny, Wind=weak) + p(Sunny, strong)*H(Sunny, Wind=strong)

= (3/5)*0.918 + (2/5)*1

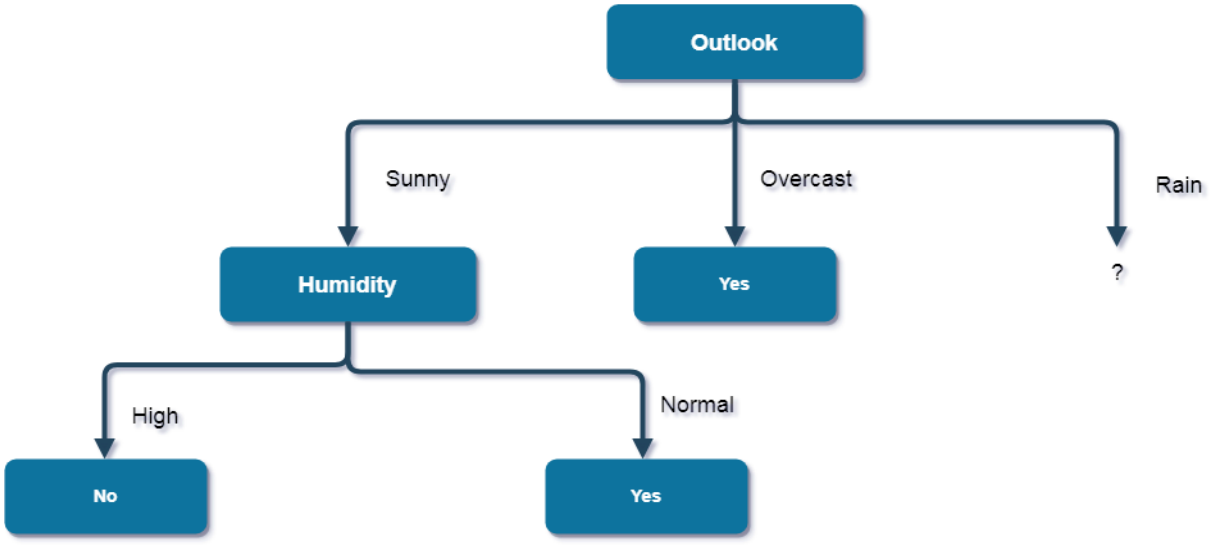
= 0.9508

Information Gain = H(Sunny) - I(Sunny, Wind)

= 0.971 - 0.9508

= 0.0202

Here, the attribute with maximum information gain is Humidity. So, the decision tree built so far



Here, when Outlook = Sunny and Humidity = High, it is a pure class of category "no". And When Outlook = Sunny and Humidity = Normal, it is again a pure class of category "yes". Therefore, we don't need to do further calculations

Now, finding the best attribute for splitting the data with Outlook=Sunny values { Dataset rows = [4, 5, 6, 10, 1]}

Complete entropy of Rain is -
H(S) = - p(yes) * log2(p(yes)) - p(no) * log2(p(no))
= - (3/5) * log(3/5) - (2/5) * log(2/5)
= 0.971

First Attribute – Temperature

Categorical values - mild, cool

H(Rain, Temperature=cool) = -(1/2)*log(1/2)- (1/2)*log(1/2) = 1

H(Rain, Temperature=mild) = -(2/3)*log(2/3)-(1/3)*log(1/3) = 0.918

Average Entropy Information for Temperature -

I(Rain, Temperature) = p(Rain, mild)*H(Rain, Temperature=mild) + p(Rain, cool)*H(Rain, Temperature=cool)
= (2/5)*1 + (3/5)*0.918
= 0.9508

Information Gain = H(Rain) - I(Rain, Temperature)
= 0.971 - 0.9508
= 0.0202

Second Attribute - Wind

Categorical values - weak, strong

H(Wind=weak) = -(3/3)*log(3/3)-0 = 0

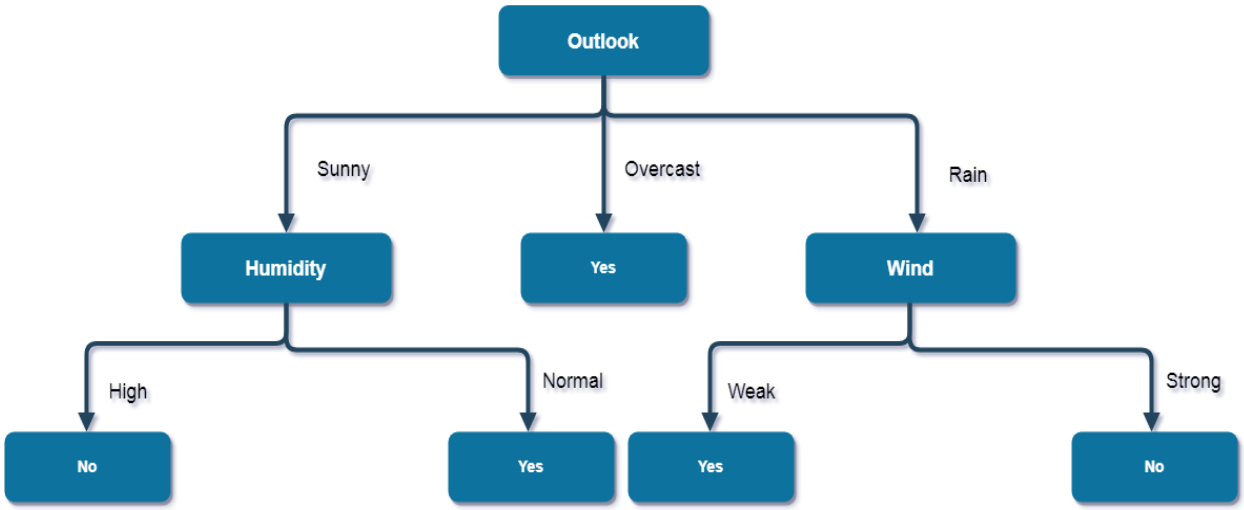
H(Wind=strong) = 0-(2/2)*log(2/2) = 0

Average Entropy Information for Wind -

I(Wind) = p(Rain, weak)*H(Rain, Wind=weak) + p(Rain, strong)*H(Rain, Wind=strong)
= (3/5)*0 + (2/5)*0
= 0

Information Gain = H(Rain) - I(Rain, Wind)
= 0.971 - 0
= 0.971

Here, the attribute with maximum information gain is Wind. So, the decision tree built so far –



Python Code for Decision Tree

```
# Run this program on your local python
# interpreter, provided you have installed
# the required libraries.

# Importing the required packages
import numpy as np
import pandas as pd
from sklearn.metrics import confusion_matrix
from sklearn.cross_validation import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report

# Function importing Dataset
def importdata():
        balance_data = pd.read_csv(
'https://archive.ics.uci.edu/ml/machine-learning-'+
'databases/balance-scale/balance-scale.data',
        sep= ',', header = None)

        # Printing the dataswet shape
        print ("Dataset Length: ", len(balance_data))
        print ("Dataset Shape: ", balance_data.shape)

        # Printing the dataset obseravtions
        print ("Dataset: ",balance_data.head())
```

```
        return balance_data

# Function to split the dataset
def splitdataset(balance_data):

        # Separating the target variable
        X = balance_data.values[:, 1:5]
        Y = balance_data.values[:, 0]

        # Splitting the dataset into train and test
        X_train, X_test, y_train, y_test = train_test_split(
        X, Y, test_size = 0.3, random_state = 100)

        return X, Y, X_train, X_test, y_train, y_test

# Function to perform training with giniIndex.
def train_using_gini(X_train, X_test, y_train):

        # Creating the classifier object
        clf_gini = DecisionTreeClassifier(criterion = "gini",
                        random_state = 100,max_depth=3, min_samples_leaf=5)

        # Performing training
        clf_gini.fit(X_train, y_train)
        return clf_gini

# Function to perform training with entropy.
def tarin_using_entropy(X_train, X_test, y_train):

        # Decision tree with entropy
        clf_entropy = DecisionTreeClassifier(
                        criterion = "entropy", random_state = 100,
                        max_depth = 3, min_samples_leaf = 5)

        # Performing training
        clf_entropy.fit(X_train, y_train)
        return clf_entropy


# Function to make predictions
def prediction(X_test, clf_object):

        # Predicton on test with giniIndex
        y_pred = clf_object.predict(X_test)
```

```
        print("Predicted values:")
        print(y_pred)
        return y_pred

# Function to calculate accuracy
def cal_accuracy(y_test, y_pred):

        print("Confusion Matrix: ",
                confusion_matrix(y_test, y_pred))

        print ("Accuracy : ",
        accuracy_score(y_test,y_pred)*100)

        print("Report : ",
        classification_report(y_test, y_pred))

# Driver code
def main():

        # Building Phase
        data = importdata()
        X, Y, X_train, X_test, y_train, y_test = splitdataset(data)
        clf_gini = train_using_gini(X_train, X_test, y_train)
        clf_entropy = tarin_using_entropy(X_train, X_test, y_train)

        # Operational Phase
        print("Results Using Gini Index:")

        # Prediction using gini
        y_pred_gini = prediction(X_test, clf_gini)
        cal_accuracy(y_test, y_pred_gini)

        print("Results Using Entropy:")
        # Prediction using entropy
        y_pred_entropy = prediction(X_test, clf_entropy)
        cal_accuracy(y_test, y_pred_entropy)


# Calling main function
if __name__=="__main__":
        main()
```

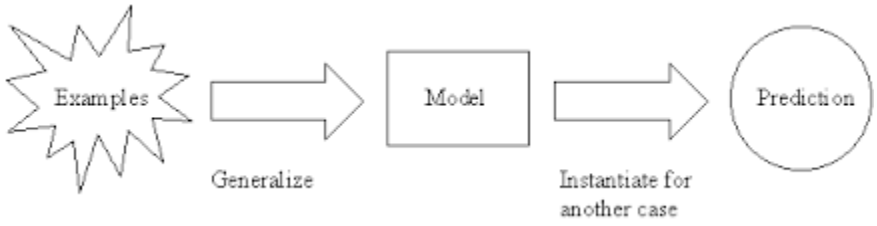
Inductive Bias

The inductive bias of a learning algorithm is the set of assumptions that the learner uses to predict outputs of given inputs that it has not encountered. In machine learning, one aims to construct algorithms that are able to learn to predict a certain target output.

The **inductive bias** (also known as **learning bias**) of a learning algorithm is the set of assumptions that the learner uses to predict outputs of given inputs that it has not encountered.

Inductive Inference

Inductive inference is **the process of reaching a general conclusion from specific examples**. The general conclusion should apply to unseen examples.



BAYESIAN LEARNING

Bayes Theorem

Bayes theorem is given by an English statistician, philosopher, and Presbyterian minister named Mr. Thomas Bayes in 17th century. Bayes provides their thoughts in decision theory which is extensively used in important mathematics concepts as Probability.

Bayes theorem is also widely used in Machine Learning where we need to predict classes precisely and accurately.

An important concept of Bayes theorem named **Bayesian method** is used to calculate conditional probability in Machine Learning application that includes classification tasks.

Further, a simplified version of Bayes theorem (Naïve Bayes classification) is also used to reduce computation time and average cost of the projects.

Bayes theorem is also known with some other name such as **Bayes rule or Bayes Law.**

Bayes theorem helps to determine the probability of an event with random knowledge.

It is used to calculate the probability of occurring one event while other one already occurred. It is a best method to relate the condition probability and marginal probability.

Bayes' theorem can be derived using product rule and conditional probability of event X with known event Y:

According to the product rule we can express as the probability of event X with known event Y as follows;

P(X ? Y)= P(X|Y) P(Y) {equation 1}

Further, the probability of event Y with known event X:

P(X ? Y)= P(Y|X) P(X) {equation 2}

Mathematically, Bayes theorem can be expressed by combining both equations on right hand side. We will get:

$$P(X|Y) = \frac{P(Y|X).P(X)}{P(Y)}$$

Here, both events X and Y are independent events which means probability of outcome of both events does not depends one another.

The above equation is called as Bayes Rule or Bayes Theorem.

- P(X|Y) is called as **posterior**, which we need to calculate. It is defined as updated probability after considering the evidence.
- P(Y|X) is called the likelihood. It is the probability of evidence when hypothesis is true.
- P(X) is called the **prior probability**, probability of hypothesis before considering the evidence
- P(Y) is called marginal probability. It is defined as the probability of evidence under any consideration.

Hence, Bayes Theorem can be written as:

posterior = likelihood * prior / evidence

While studying the Bayes theorem, we need to understand few important concepts. These are as follows:

1. Experiment

An experiment is defined as the planned operation carried out under controlled condition such as tossing a coin, drawing a card and rolling a dice, etc.

2. Sample Space

During an experiment what we get as a result is called as possible outcomes and the set of all possible outcome of an event is known as sample space.

For example, if we are rolling a dice, sample space will be:

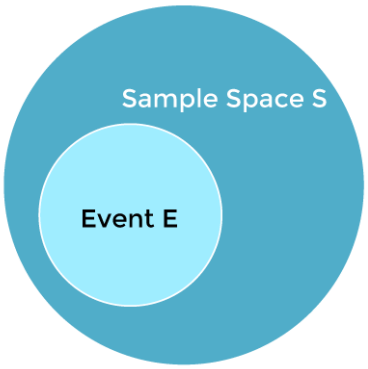
S1 = {1, 2, 3, 4, 5, 6}

Similarly, if our experiment is related to toss a coin and recording its outcomes, then sample space will be:

S2 = {Head, Tail}

3. Event

Event is defined as subset of sample space in an experiment. Further, it is also called as set of outcomes.



Advantages of Naïve Bayes Classifier in Machine Learning:

- It is one of the simplest and effective methods for calculating the conditional probability and text classification problems.
- A Naïve-Bayes classifier algorithm is better than all other models where assumption of independent predictors holds true.

- It is easy to implement than other models.
- It requires small amount of training data to estimate the test data which minimize the training time period.
- It can be used for Binary as well as Multi-class Classifications.

Disadvantages of Naïve Bayes Classifier in Machine Learning:

The main disadvantage of using Naïve Bayes classifier algorithms is, it limits the assumption of independent predictors because it implicitly assumes that all attributes are independent or unrelated but in real life it is not feasible to get mutually independent attributes.

Why it is called Naïve Bayes

The Naïve Bayes algorithm is comprised of two words Naïve and Bayes, Which can be described as:

Naïve: It is called Naïve because it assumes that the occurrence of a certain feature is independent of the occurrence of other features. Such as if the fruit is identified on the bases of color, shape, and taste, then red, spherical, and sweet fruit is recognized as an apple. Hence each feature individually contributes to identify that it is an apple without depending on each other.

Bayes: It is called Bayes because it depends on the principle of Bayes' Theorem.

Example-1:

Suppose we have a dataset of **weather conditions** and corresponding target variable "**Play**". So using this dataset we need to decide that whether we should play or not on a particular day according to the weather conditions. So to solve this problem, we need to follow the below steps:

1. Convert the given dataset into frequency tables.
2. Generate Likelihood table by finding the probabilities of given features.
3. Now, use Bayes theorem to calculate the posterior probability.

Problem: If the weather is sunny, then the Player should play or not?

Solution: To solve this, first consider the below dataset:

Weather	Play
Sunny	No
Overcast	Yes
Rainy	Yes
Sunny	Yes
Sunny	Yes
Overcast	Yes
Rainy	No
Rainy	No
Sunny	Yes
Rainy	Yes
Sunny	No
Overcast	Yes
Overcast	Yes
Rainy	No

Frequency Table		
Weather	No	Yes
Overcast		4
Rainy	3	2
Sunny	2	3
Grand Total	5	9

Likelihood table				
Weather	No	Yes		
Overcast		4	=4/14	0.29
Rainy	3	2	=5/14	0.36
Sunny	2	3	=5/14	0.36
All	5	9		
	=5/14	=9/14		
	0.36	0.64		

We can solve it using above discussed method of posterior probability.

- P(Yes | Sunny) = P(Sunny | Yes) * P(Yes) / P (Sunny)
- Here we have P (Sunny |Yes) = 3/9 = 0.33, P(Sunny) = 5/14 = 0.36, P(Yes)= 9/14 = 0.64
- Now, P (Yes | Sunny) = 0.33 * 0.64 / 0.36 = 0.60, which has higher probability.

Thus, A player can play on a sunny day.

Example-2:

Consider the car theft problem with attributes Color, Type, Origin, and the target, Stolen can be either Yes or No.

Example No.	Color	Type	Origin	Stolen?
1	Red	Sports	Domestic	Yes
2	Red	Sports	Domestic	No
3	Red	Sports	Domestic	Yes
4	Yellow	Sports	Domestic	No
5	Yellow	Sports	Imported	Yes
6	Yellow	SUV	Imported	No
7	Yellow	SUV	Imported	Yes
8	Yellow	SUV	Domestic	No
9	Red	SUV	Imported	No
10	Red	Sports	Imported	Yes

According to Color Attribute

Frequency Table

		Stolen?	
		Yes	No
Color	Red	3	2
	Yellow	2	3

Likelihood Table

		Stolen?	
		P(Yes)	P(No)
Color	Red	3/5	2/5
	Yellow	2/5	3/5

According to Type Attribute

Frequency Table

		Stolen?	
		Yes	No
Type	Sports	4	2
	SUV	1	3

Likelihood Table

		Stolen?	
		P(Yes)	P(No)
Type	Sports	4/5	2/5
	SUV	1/5	3/5

<u>According to Origin Attribute</u>

Frequency Table

		Stolen?	
		Yes	No
Origin	Domestic	2	3
	Imported	3	2

Likelihood Table

		Stolen?	
		P(Yes)	P(No)
Origin	Domestic	2/5	3/5
	Imported	3/5	2/5

So in this example, we have 3 predictors X.

Color	Type	Origin	Stolen
Red	SUV	Domestic	?

As per the equations discussed above, we can calculate the posterior probability P (Yes | X) as :

$$P(Yes \mid X) = P(Red \mid Yes) * P(SUV \mid Yes) * P(Domestic \mid Yes) * P(Yes)$$

$$= 3/5 * 1/5 * 2/5 * 1$$
$$= 0.048$$

And, P (No | X):

$$P(No \mid X) = P(Red \mid No) * P(SUV \mid No) * P(Domestic \mid No) * P(No)$$

$$= 2/5 * 3/5 * 3/5 * 1$$
$$= 0.144$$

Since 0.144 > 0.048, Which means given the features RED SUV and Domestic, our example gets classified as 'NO' the car is not stolen.

Pros and Cons of Naive Bayes

Pros:

- It is easy and fast to predict class of test data set. It also perform well in multi class prediction
- When assumption of independence holds, a Naive Bayes classifier performs better compare to other models like logistic regression and you need less training data.
- It perform well in case of categorical input variables compared to numerical variable(s). For numerical variable, normal distribution is assumed (bell curve, which is a strong assumption).

Cons:

- If categorical variable has a category (in test data set), which was not observed in training data set, then model will assign a 0 (zero) probability and will be unable to make a prediction. This is often known as "Zero Frequency". To solve this, we can use the smoothing technique. One of the simplest smoothing techniques is called Laplace estimation.
- On the other side naive Bayes is also known as a bad estimator, so the probability outputs from predict proba are not to be taken too seriously.
- Another limitation of Naive Bayes is the assumption of independent predictors. In real life, it is almost impossible that we get a set of predictors which are completely independent.

To build a basic model using Naive Bayes in Python

Again, scikit learn (python library) will help here to build a Naive Bayes model in Python. There are three types of Naive Bayes model under the scikit-learn library:

Gaussian: It is used in classification and it assumes that features follow a normal distribution.

Multinomial: It is used for discrete counts. For example, let's say, we have a text classification problem. Here we can consider Bernoulli trials which is one step further and instead of "word occurring in the document", we have "count how often word occurs in the document", you can think of it as "number of times outcome number x_i is observed over the n trials".

Bernoulli: The binomial model is useful if your feature vectors are binary (i.e. zeros and ones). One application would be text classification with 'bag of words' model where the 1s & 0s are "word occurs in the document" and "word does not occur in the document" respectively.

Now, we look at an implementation of Gaussian Naive Bayes classifier using scikit-learn.

```
# load the iris dataset
from sklearn.datasets import load_iris
iris = load_iris()

# store the feature matrix (X) and response vector (y)
X = iris.data
y = iris.target

# splitting X and y 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=0.4, random_state=1)

# training the model on training set
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()
gnb.fit(X_train, y_train)

# making predictions on the testing set
y_pred = gnb.predict(X_test)

# comparing actual response values (y_test) with predicted response values (y_pred)
from sklearn import metrics
print("Gaussian Naive Bayes model accuracy(in %):", metrics.accuracy_score(y_test,
y_pred)*100)
```

Feature Engineering and Feature Selection

Feature engineering enables you to build more complex models than you could with only raw data. It also allows you to build interpretable models from any amount of data. Feature selection will help you limit these features to a manageable number.

Feature selection is **the process of reducing the number of input variables when developing a predictive model**. It is desirable to reduce the number of input variables to both reduce the computational cost of modeling and, in some cases, to improve the performance of the model.

Instance-based learning:

(sometimes called *memory-based learning*) is a family of learning algorithms that, **instead of performing explicit generalization, compares new problem instances with instances seen in training, which have been stored in memory.**
Ex- k-nearest neighbor, decision tree

Model-based learning:

Machine learning models that are **parameterized** with a certain number of parameters that **do not change as the size of training data changes.**
If you don't assume any distribution with a fixed number of parameters over your data, for example, in k-nearest neighbor, or in a decision tree, where the number of parameters grows with the size of the training data, then you are **not model-based**, or *nonparametric*.