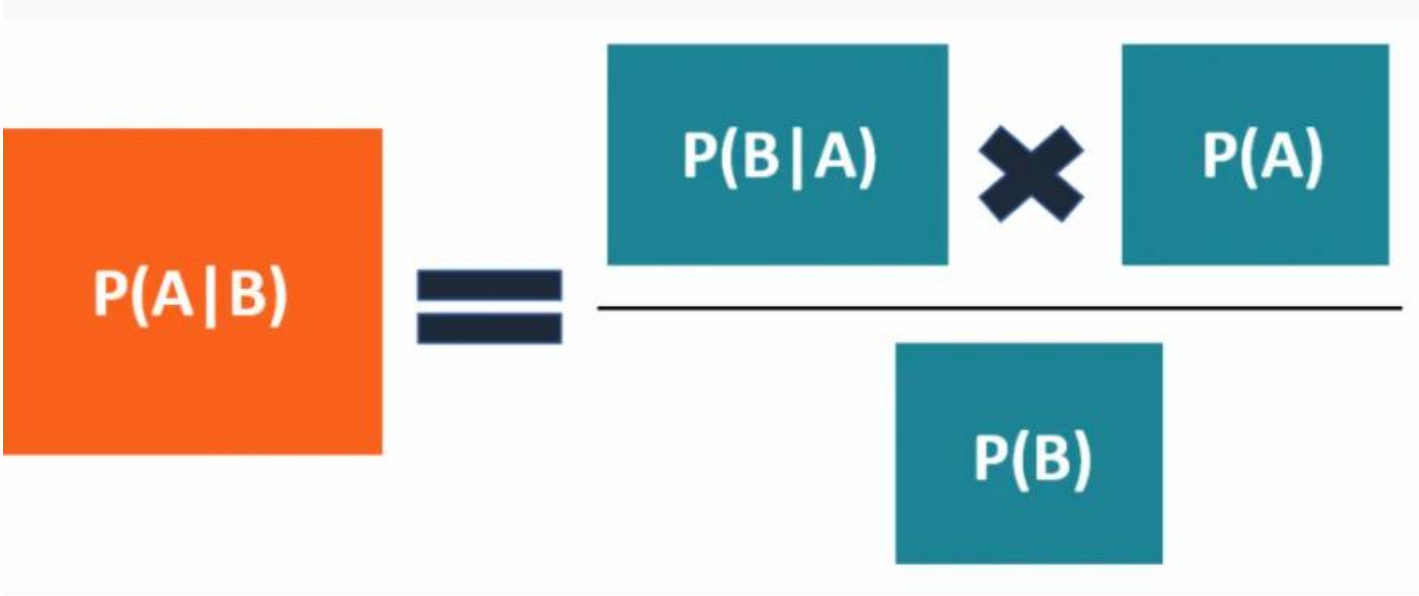
12-Scikit Learn- Classification with Naive Bayes

Naive Bayes methods are a set of supervised learning algorithms based on applying Bayes theorem with a strong assumption that all the predictors are independent to each other.

Bayes theorem is a mathematical formula used to determine the condition probability of events. Essentially, the Bayes theorem describes the **probability** of an event based on **prior knowledge of the conditions** that might be relevant to the event.



where:

- P(A|B): the probability of event A occurring, given event B has occurred.
- P(B|A): the probability of event B occurring, given event A has occurred.
- P(A): the probability of event A
- P(B):the probability of event B

The Scikit-learn provides different naive Bayes classifier models namely **Gaussian, Multinomial, Complement and Bernoulli.**

- Gaussian Naive Bayes: this classifier assumes that the data from each label is drown from a simple Gaussian distribution.
- Multinomial Naive Bayes : It assumes that the features are drown from a simple Multinomial distribution.
- Bernoulli Naive Bayes: Assumption in this model is that features binary (0s and 1a) in nature.
- Complement Naive Bayes: it was designed to correct the severe assumptions made by Multinomial Bayes classifier.

12-1- Building Naive Bayes Classifier

```
 import sklearn
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
data = load_breast_cancer()
label_names=data['target_names']
labels=data['target']
feature_names=data['feature_names']
feature=data['data']
train,test,train_labels,test_labels=train_test_split(feature,labels,test_size=0.40,random_state=42)
from sklearn.naive_bayes import GaussianNB
GNBclf=GaussianNB()
model=GNBclf.fit(train,train_labels)
pred=GNBclf.predict(test)
print(pred)
```

output:

```
[1 0 0 1 1 0 0 0 1 1 1 0 1 0 1 0 1 1 1 0 1 1 0 1 1 1 1 1 1 0 1 1 1 1 1 1 0
 1 0 1 1 0 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 0 0 1 1 0 0 1 1 1 0 0 1 1 0 0 1 0
 1 1 1 1 1 1 0 1 1 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 1 0 0 1 0 0 1 1 1 0 1 1 0
 1 1 0 0 0 1 1 1 0 0 1 1 0 1 0 0 1 1 0 0 0 1 1 1 0 1 1 0 0 1 0 1 1 0 1 0 0
 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 0 1 1 0 1 1 1 1 1 1 0 0
 0 1 1 0 1 0 1 1 1 1 0 1 1 0 1 1 1 0 1 0 0 1 1 1 1 1 1 1 1 0 1 1 1 1 1 0 1
 0 0 1 1 0 1]
```

13- Scikit Learn- Decision Tree

Decision Tree re the most powerful non-parametric supervised earning method. They can be used for classification and regression tasks. The main goal of DTree is to create a model predicting target variable value by learning simple decision rules deduced from data features. Decision Tree have two main entities; one s root node,where the data split, and other is decision nodes or leaves,where we got final output.

13-1: Decision Tree algorithms

- **ID3:** It is also called Iterative Dichotomiser 3. The main goal of this algorithm is to find those categorical features, for every node, that will yield the largest information gain for categorical targets.
- **C4.5:** It defines a discrete attribute that partition the continuous attribute value into a discrete set of intervals. That's the reason it removed the restriction of categorical features.
- **C5:** It works similar as C4.5 but it uses less memory and build smaller rulesets. It is more accurate than C4.5.
- **CART :** It is called Classification and Regression Tree algorithm. It basically generates binary splits by using the features and threshold yielding the largest information gain at each node.

13-2: Implementation Example

```
from sklearn import tree
from sklearn.model_selection import train_test_split
X=[[14,25],[98,52],[987,254],[125,653],[214,410],[235,21],[28,54],[451,712],[111,214],[985,437],[965,439],
  [321,124],[354,842],[85,32],[963,41],[854,654],[960,370],[96,526]]
Y=['Woman','Man','Man','Man','Woman','Man','Woman','Woman','Woman','Man','Man','Woman',
  'Man','Woman','Woman','Man','Man','Man']

data_feture_name=['height','length of hair']
DTclf=tree.DecisionTreeClassifier()
clf=DTclf.fit(X,Y)
prediction=clf.predict([[125,653]])
print(prediction)
```

output:

['Man']

14-Randomized Decision Tree Algorithms

Decision Tree is usually trained by recursively splitting the data, but being prone to overfit, they have been transformed to random forests by training many trees over various sub-samples of the data. **sklearn.ensemble** having two algorithms based on randomized decision trees.

In this algorithm, each decision tree in the ensemble is built from a sample drown with replacement from the training set and then gets the prediction from each of them and finally selects the best solution by means of voting. It can be used for both classification as well as regression tasks.

- **Classification by Random Forest :** For creating a **random forest classifier**, Scikit-learn module provides **sklearn.ensemble.RandomForestClassifier.** Main parameters are **max-features** and **n_estimators. max_features** is the size of the random subsets of features to consider when splitting a node. **n_estimators** are the number of trees in the forest.

```
from sklearn.model_selection import cross_val_score
from sklearn.datasets import make_blobs
from sklearn.ensemble import RandomForestClassifier
X,y=make_blobs(n_samples=10000,n_features=10,centers=100,random_state=0)
RFclf=RandomForestClassifier(n_estimators=10,min_samples_split=2)
score=cross_val_score(RFclf,X,y,cv=5)
print(score.mean())
```

output:

0.9998000000000001

- **Reression with Randm Forest:** For creating a random forest regression, scikit-learn module provides **sklearn.ensemble.RandomForestRegressor.** it will use the sample paramreters as used by sklearn.ensemble.RandomForestClassifier.

```
from sklearn.model_selection import cross_val_score
from sklearn.datasets import make_blobs
from sklearn.ensemble import RandomForestRegressor
X,y=make_blobs(n_samples=10000,n_features=10,centers=100,random_state=0)
RFclf=RandomForestRegressor(n_estimators=10,min_samples_split=2)
score=cross_val_score(RFclf,X,y,cv=5)
print(score.mean())
```

output:

0.9705321938477912

15- Extra Tree Methods

For each feature under consideration, it selects a random value for the split. benefit of using extra tree methods is that it allows to reduce the variance of the model. the disadvantage of using these methods is that slightly increases the bias.

- **Classification with Extra-Tree Method:** For creating a classifier using Extra-Tree method scikit-learn module provides **sklearn.ensemble.ExtraTreesClassifier.** It uses the same parameters as used by sklearn.ensemble.RandomForestClassifier.

```
from sklearn.model_selection import cross_val_score
from sklearn.datasets import make_blobs
from sklearn.ensemble import ExtraTreesClassifier
X,y=make_blobs(n_samples=10000,n_features=10,centers=100,random_state=0)
ETclf=ExtraTreesClassifier(n_estimators=10,min_samples_split=2)
score=cross_val_score(ETclf,X,y,cv=5)
print(score.mean())
```

output:

0.9998000000000001

- **Regression with Extra-Tree Method:** For creating a Extra-Tree regression, scikit-learn module provides sklearn.ensemble.ExtraTreesRegressor. it use the same parameters as used by sklearn.ensemble.ExtraTreesClassifier.

```
from sklearn.model_selection import cross_val_score
from sklearn.datasets import make_blobs
from sklearn.ensemble import ExtraTreesRegressor
X,y=make_blobs(n_samples=10000,n_features=10,centers=100,random_state=0)
ETclf=ExtraTreesRegressor(n_estimators=10,min_samples_split=2)
score=cross_val_score(ETclf,X,y,cv=5)
print(score.mean())
```

output:

0.9914243964799774