Classification in Python

As a marketing manager, you want a set of customers who are most likely to purchase your product. This is how you can save your marketing budget by finding your audience.

As a loan manager, you need to identify risky loan applications to achieve a lower loan default rate. This process of classifying customers into a group of potential and non-potential customers or safe or risky loan applications is known as a classification problem.

Classification is a two-step process, learning step and prediction step. In the learning step, the model is developed based on given training data. In the prediction step, the model is used to predict the response for given data.

Types Of Classifiers--------

Naive Bayes Classifier

Decision Tree Classifier

Nearest Neighbor Classifier

Support Vector Classifier

Naive Bayes Classifier

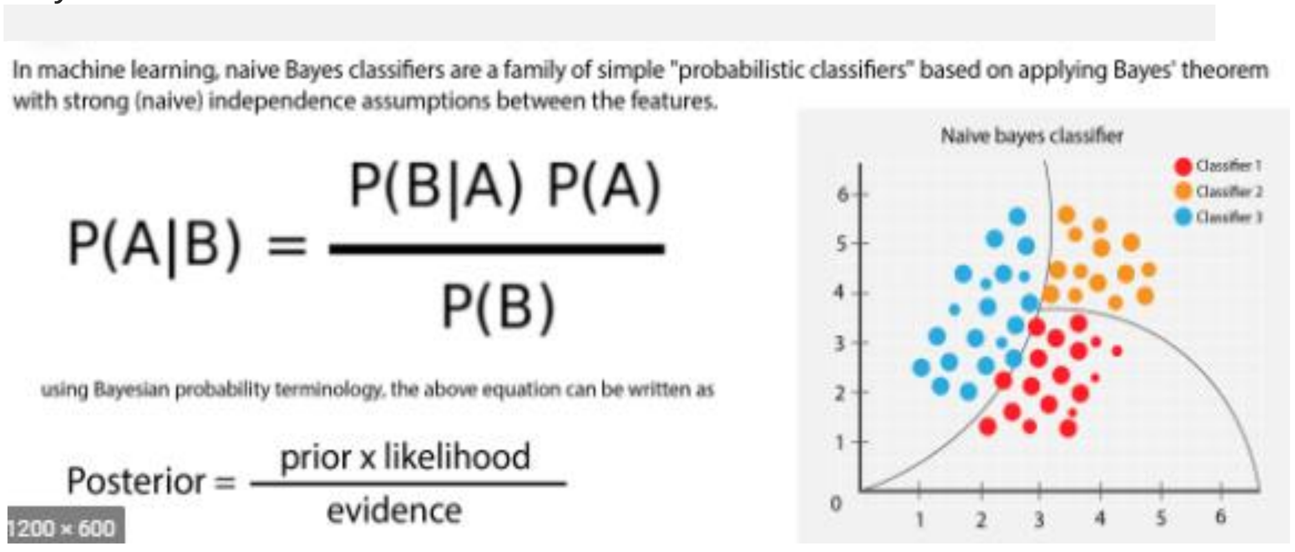
What is a classifier?

A classifier is a machine learning model that is used to discriminate different objects based on certain features.

Principle of Naive Bayes Classifier:

A Naive Bayes classifier is a probabilistic machine learning model that's used for classification task. The crux of the classifier is based on the Bayes theorem.

Bayes Theorem:



In machine learning, naive Bayes classifiers are a family of simple "probabilistic classifiers" based on applying Bayes' theorem with strong (naive) independence assumptions between the features.

$$P(A|B) = \frac{P(B|A)\,P(A)}{P(B)}$$

using Bayesian probability terminology, the above equation can be written as

$$\text{Posterior} = \frac{\text{prior x likelihood}}{\text{evidence}}$$

The variable **y** is the class variable(play golf), which represents if it is suitable to play golf or not given the conditions. Variable **X** represent the parameters/features.

X is given as,

$$X = (x_1, x_2, x_3, ..., x_n)$$

Here x_1,x_2....x_n represent the features, i.e they can be mapped to outlook, temperature, humidity and windy. By substituting for **X** and expanding using the chain rule we get,

$$P(y|x_1, ..., x_n) = \frac{P(x_1|y)P(x_2|y)...P(x_n|y)P(y)}{P(x_1)P(x_2)...P(x_n)}$$

Now, you can obtain the values for each by looking at the dataset and substitute them into the equation. For all entries in the dataset, the denominator does not change, it remain static. Therefore, the denominator can be removed and a proportionality can be introduced.

$$P(y|x_1, ..., x_n) \propto P(y) \prod_{i=1}^{n} P(x_i|y)$$

In our case, the class variable(**y**) has only two outcomes, yes or no. There could be cases where the classification could be multivariate. Therefore, we need to find the class **y** with maximum probability.

$$y = argmax_y P(y) \prod_{i=1}^{n} P(x_i|y)$$

Using the above function, we can obtain the class, given the predictors.

X is given as,

$$X = (x_1, x_2, x_3, \ldots.., x_n)$$

$$P(Class_j \mid x') = P(x'_1 \mid Class_j) \times P(x'_2 \mid Class_j) \times \ldots \times P(x'_k \mid Class_j) \times P(Class_j)$$

Example

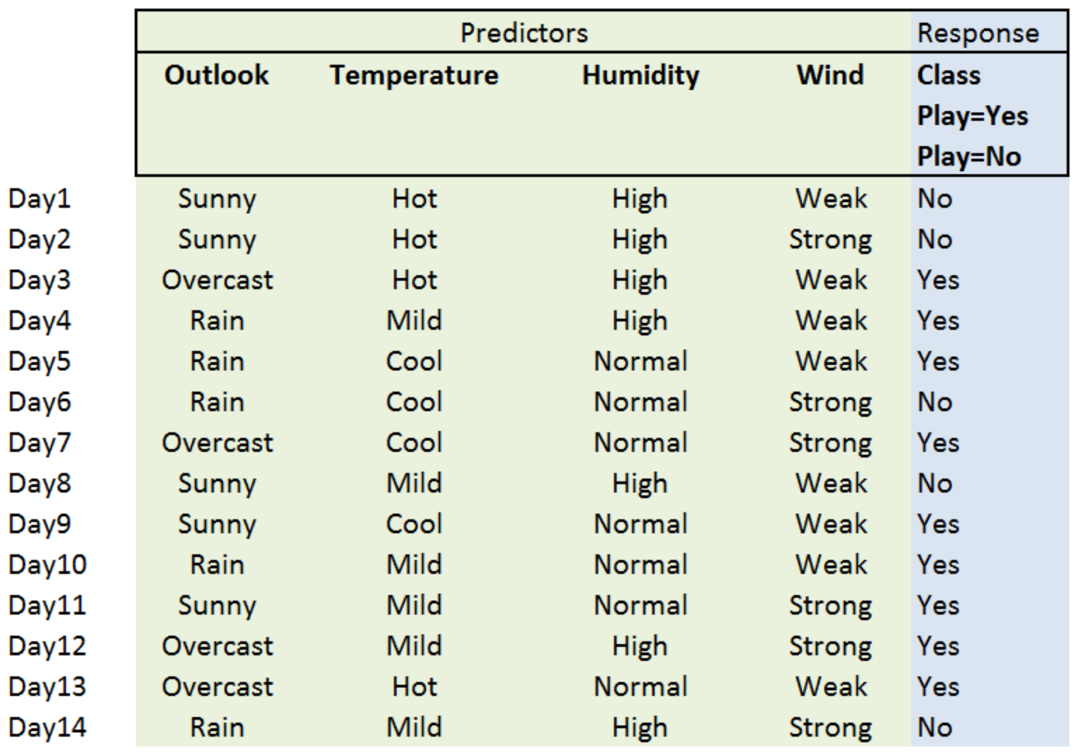
Let’s build a classifier that predicts whether I should play tennis given the forecast. It takes four attributes to describe the forecast; namely, the outlook, the temperature, the humidity, and the presence or absence of wind. Furthermore the values of the four attributes are qualitative (also known as categorical). They take on the values shown below.

Outlook $\in$ [Sunny, Overcast, Rainy]
Temperature $\in$ [Hot, Mild, Cool]
Humidity $\in$ [High, Normal]
Windy $\in$ [Weak, Strong]

The class label is the variable, Play and takes the values yes or no.

Play$\in$ [Yes, No]

We read-in training data below that has been collected over 14 days.



	Predictors				Response
	Outlook	**Temperature**	**Humidity**	**Wind**	**Class** **Play=Yes** **Play=No**
Day1	Sunny	Hot	High	Weak	No
Day2	Sunny	Hot	High	Strong	No
Day3	Overcast	Hot	High	Weak	Yes
Day4	Rain	Mild	High	Weak	Yes
Day5	Rain	Cool	Normal	Weak	Yes
Day6	Rain	Cool	Normal	Strong	No
Day7	Overcast	Cool	Normal	Strong	Yes
Day8	Sunny	Mild	High	Weak	No
Day9	Sunny	Cool	Normal	Weak	Yes
Day10	Rain	Mild	Normal	Weak	Yes
Day11	Sunny	Mild	Normal	Strong	Yes
Day12	Overcast	Mild	High	Strong	Yes
Day13	Overcast	Hot	Normal	Weak	Yes
Day14	Rain	Mild	High	Strong	No

The Learning Phase

In the learning phase, we compute the table of likelihoods (probabilities) from the training data. They are:

P(Outlook=o|Class$_{Play=b}$), where o ∈ [Sunny, Overcast, Rainy] and b ∈ [yes, no]

P(Temperature=t|Class$_{Play=b}$), where t ∈ [Hot, Mild, Cool] and b ∈ [yes, no],

P(Humidity=h|Class$_{Play=b}$), where h∈ [High, Norma] and b ∈ [yes, no],

P(Wind=w|Class$_{Play=b}$), where w ∈ [Weak, Strong] and b ∈ [yes, no].

$P(Outlook=o \mid Class_{Play=Yes \mid No})$	Frequency		Probability in Class	
Outlook =	Play=Yes	Play=No	Play=Yes	Play=No
Sunny	2	3	2/9	3/5
Overcast	4	0	4/9	0/5
Rain	3	2	3/9	2/5
	total= 9	total=5		

$P(Temperature=t \mid Class_{Play=Yes \mid No})$	Frequency		Probability in Class	
Temperature =	Play=Yes	Play=No	Play=Yes	Play=No
Hot	2	2	2/9	2/5
Mild	4	2	4/9	2/5
Cool	3	1	3/9	1/5
	total= 9	total=5		

$P(Humidity=h \mid Class_{Play=Yes \mid No})$	Frequency		Probability in Class	
Humidity =	Play=Yes	Play=No	Play=Yes	Play=No
High	3	4	3/9	4/5
Normal	6	1	6/9	1/5
	total= 9	total=5		

$P(Wind=w \mid Class_{Play=Yes \mid No})$	Frequency		Probability in Class	
Wind =	Play=Yes	Play=No	Play=Yes	Play=No
strong	3	3	3/9	3/5
weak	6	2	6/9	2/5
	total= 9	total=5		

We also calculate $P(Class_{Play=Yes})$ and $P(Class_{Play=No})$.

$P(Class_{Play=Yes})$ & $P(Class_{Play=No})$	Frequency		Probability in Class	
Play	Play=Yes	Play=No	Play=Yes	Play=No
	9	5	9/14	5/14
	total= 9	total=5		

Classification Phase

Let’s say, we get a new instance of the weather condition,

x’=(Outlook=Sunny, Temperature=Cool, Humidity=High, Wind=Strong) that will have to be classified (i.e., are we going to play tennis under the conditions specified by x’).

With the MAP rule, we compute the posterior probabilities. This is easily done by looking up the tables we built in the learning phase.

$P(Class_{Play=Yes}|x') = [P(Sunny|Class_{Play=Yes}) \times P(Cool|Class_{Play=Yes}) \times$

$P(High|Class_{Play=Yes}) \times P(Strong|Class_{Play=Yes})] \times$

$P(Class_{Play=Yes})$

$= 2/9 \times 3/9 \times 3/9 \times 3/9 \times 9/14 = 0.0053$

$P(Class_{Play=No}|x') = [P(Sunny|Class_{Play=No}) \times P(Cool|Class_{Play=No}) \times$

$P(High|Class_{Play=No}) \times P(Strong|Class_{Play=No})] \times$

$P(Class_{Play=No})$

$= 3/5 \times 1/5 \times 4/5 \times 3/5 \times 5/14 = 0.0205$

Since $P(Class_{Play=Yes}|x')$ less than $P(Class_{Play=No}|x')$, we classify the new instance x' to be "No".

Types of Naive Bayes Classifier:

Multinomial Naive Bayes:

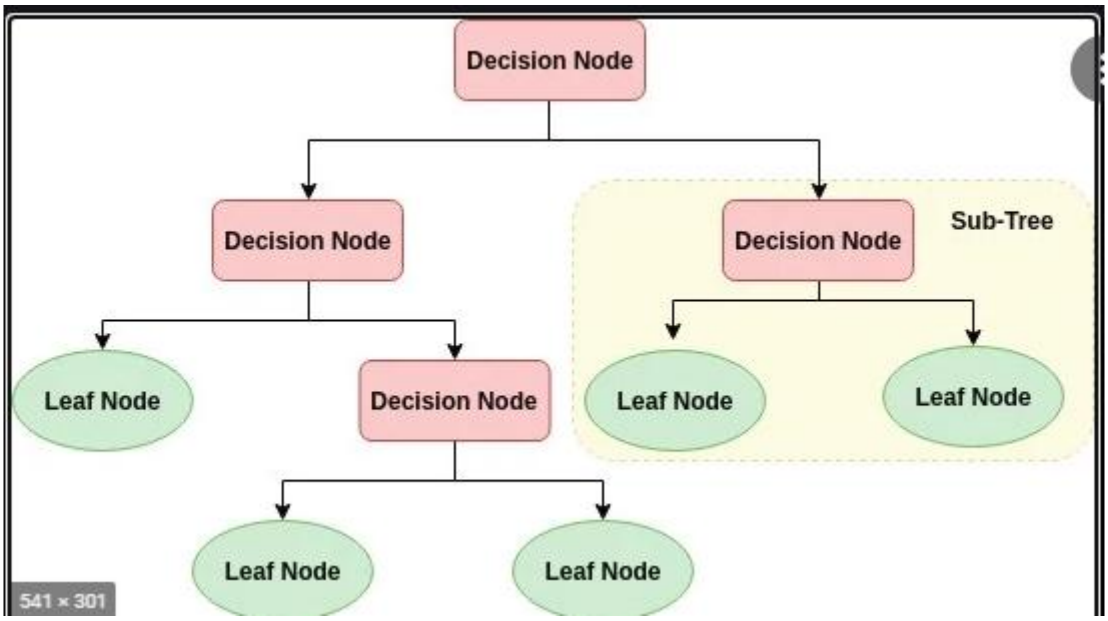
This is mostly used for document classification problem, i.e whether a document belongs to the category of sports, politics, technology etc. The features/predictors used by the classifier are the frequency of the words present in the document.

Gaussian Naive Bayes:

When the predictors take up a continuous value and are not discrete, we assume that these values are sampled from a gaussian distribution.

Decision Tree Classifier

A decision tree is a flowchart-like tree structure where an internal node represents feature(or attribute), the branch represents a decision rule, and each leaf node represents the outcome. The topmost node in a decision tree is known as the root node. It learns to partition on the basis of the attribute value. It partitions the tree in recursively manner call recursive partitioning. This flowchart-like structure helps you in decision making. It's visualization like a flowchart diagram which easily mimics the human level thinking. That is why decision trees are easy to understand and interpret.



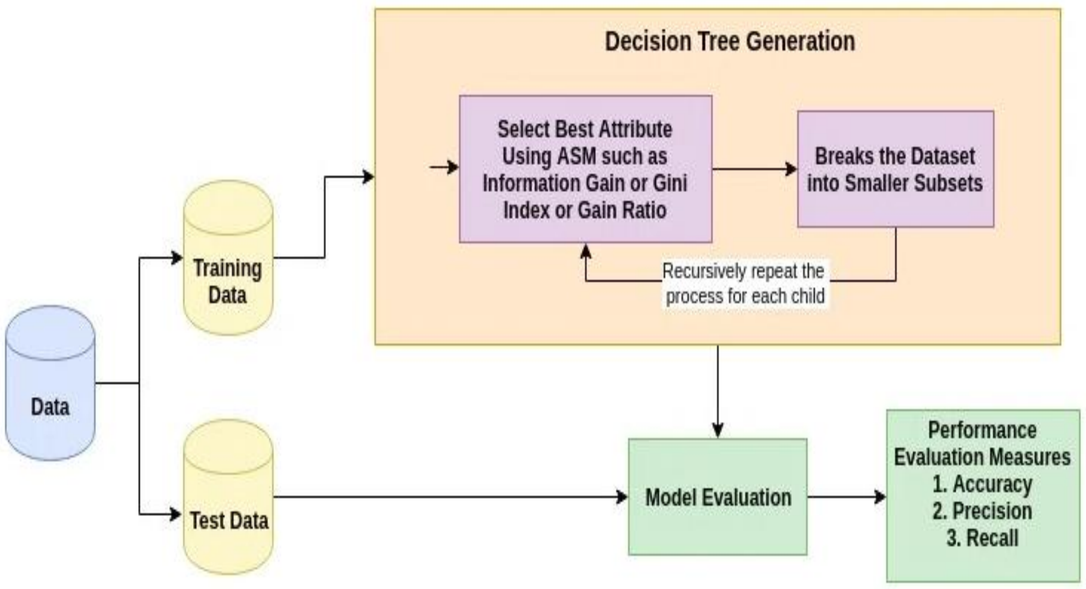
- **root node** that has no incoming edges and zero or more outgoing edges.
- **Internal nodes**, each of which has exactly one incoming edge and two or more outgoing edges.
- **Leaf or terminal nodes**, each of which has exactly one incoming edge and no outgoing edges.

Decision trees can handle high dimensional data with good accuracy.

The basic idea behind any decision tree algorithm is as follows:

1. Select the best attribute using Attribute Selection Measures(ASM) to split the records.
2. Make that attribute a decision node and breaks the dataset into smaller subsets.
3. Starts tree building by repeating this process recursively for each child until one of the condition will match:

- All the tuples belong to the same attribute value.
- There are no more remaining attributes.
- There are no more instances.


Decision Tree Generation
Select Best Attribute Using ASM such as Information Gain or Gini Index or Gain Ratio
Breaks the Dataset into Smaller Subsets
Recursively repeat the process for each child
Training Data
Data
Test Data
Model Evaluation
Performance Evaluation Measures
1. Accuracy
2. Precision
3. Recall

Importing Required Libraries

Let's first load the required libraries.

```
# Import Decision Tree Classifier


fromsklearn.treeimportDecisionTreeClassifier
```

https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html

Optimizing Decision Tree Performance

- **criterion : optional (default=”gini”) or Choose attribute selection measure**: This parameter allows us to use the different-different attribute selection measure. Supported criteria are “gini” for the Gini index and “entropy” for the information gain.
- **splitter : string, optional (default=”best”) or Split Strategy**: This parameter allows us to choose the split strategy. Supported strategies are “best” to choose the best split and “random” to choose the best random split.
- **max_depth :int or None, optional (default=None) or Maximum Depth of a Tree**: The maximum depth of the tree. If None, then nodes are expanded until all the leaves contain less than min_samples_split samples. The higher value of maximum depth causes overfitting, and a lower value causes underfitting

Attribute Selection Measures

Attribute selection measure is a heuristic for selecting the splitting criterion that partition data into the best possible manner. It is also known as splitting rules because it helps us to determine breakpoints for tuples on a given node. ASM provides a rank to each feature(or attribute) by explaining the given dataset. Best score attribute will be selected as a splitting attribute (Source). In the case of a continuous-valued attribute, split points for branches also need to define. Most popular selection measures are Information Gain, Gain Ratio, and Gini Index.

Information Gain

Shannon invented the concept of entropy, which measures the impurity of the input set. In physics and mathematics, entropy referred as the randomness or the impurity in the system. In information theory, it refers to the impurity in a group of examples. Information gain is the decrease in entropy. Information gain computes the difference between entropy before split and average entropy after split of the dataset based on given attribute values. ID3 (Iterative Dichotomiser) decision tree algorithm uses information gain.

$$\mathrm{Info(D)}= - \sum_{i=1}^{m} pi \log_2 pi$$

Where, Pi is the probability that an arbitrary tuple in D belongs to class Ci.

$$\mathrm{Info}_A\mathrm{(D)}=\sum_{j=1}^{V} \frac{|Dj|}{|D|} \quad \mathrm{X\ Info}(\mathrm{D}_j)$$

$$\mathrm{Gain(A)=Info(D)-\ Info}_A\mathrm{(D)}$$

Where,

- Info(D) is the average amount of information needed to identify the class label of a tuple in D.
- |Dj|/|D| acts as the weight of the jth partition.
- InfoA(D) is the expected informa-tion required to classify a tuple from D based on the partitioning by A.

The attribute A with the highest information gain, Gain(A), is chosen as the splitting attribute at node N().

Gain Ratio

Information gain is biased for the attribute with many outcomes. It means it prefers the attribute with a large number of distinct values. For instance, consider an attribute with a unique identifier such as customer_ID has zero info(D) because of pure partition. This maximizes the information gain and creates useless partitioning.

C4.5, an improvement of ID3, uses an extension to information gain known as the gain ratio. Gain ratio handles the issue of bias by normalizing the information gain using Split Info. Java implementation of the C4.5 algorithm is known as J48, which is available in WEKA data mining tool.

$$SplitInfo_A(D) = -\sum_{j=1}^{v} \frac{|D_j|}{|D|} \times \log_2\left(\frac{|D_j|}{|D|}\right)$$

Where,

- |Dj|/|D| acts as the weight of the jth partition.
- v is the number of discrete values in attribute A.

The gain ratio can be defined as

$$GainRatio(A) = \frac{Gain(A)}{SplitInfo_A(D)}$$

The attribute with the highest gain ratio is chosen as the splitting attribute (Source).

Gini index

Another decision tree algorithm CART (Classification and Regression Tree) uses the Gini method to create split points.

$$\text{Gini(D)}=1-\sum_{i=1}^{m} \text{Pi}^2$$

Where, pi is the probability that a tuple in D belongs to class Ci.

The Gini Index considers a binary split for each attribute. You can compute a weighted sum of the impurity of each partition. If a binary split on attribute A partitions data D into D1 and D2, the Gini index of D is:

$$\text{Gini}_A(\text{D})=\frac{|D1|}{|D|}\text{Gini}(\text{D}_1)+\frac{|D2|}{|D|}\text{Gini}(\text{D}_2)$$

In case of a discrete-valued attribute, the subset that gives the minimum gini index for that chosen is selected as a splitting attribute. In the case of continuous-valued attributes, the strategy is to select each pair of adjacent values as a possible split-point and point with smaller gini index chosen as the splitting point.

$$\Delta Gini(A) = Gini(D) - Gini_A(D).$$

The attribute with minimum Gini index is chosen as the splitting attribute.

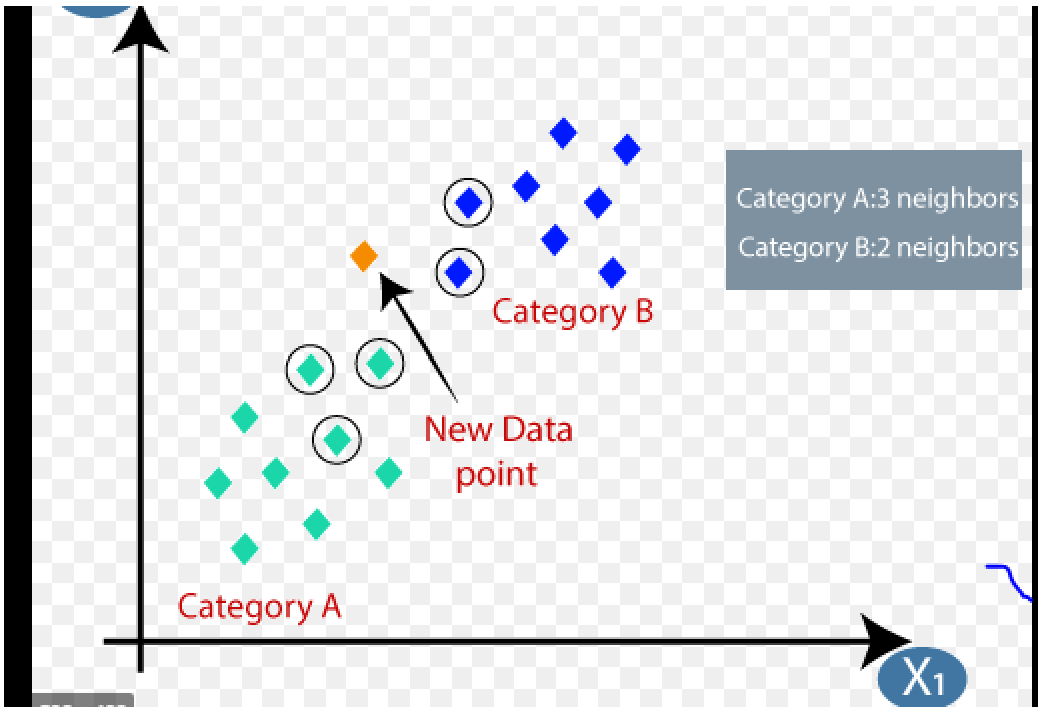
Decision Tree Classifier Building in Scikit-learn

KNearestNeighbors Classifier

K Nearest Neighbor(KNN) is a very simple, easy to understand, versatile and one of the topmost machine learning algorithms. KNN used in the variety of applications such as finance, healthcare, political science, handwriting detection, image recognition and video recognition.

In Credit ratings, financial institutes will predict the credit rating of customers. In loan disbursement, banking institutes will predict whether the loan is safe or risky. In political science, classifying potential voters in two classes will vote or won’t vote. KNN algorithm used for both classification and regression problems. KNN algorithm based on feature similarity approach.

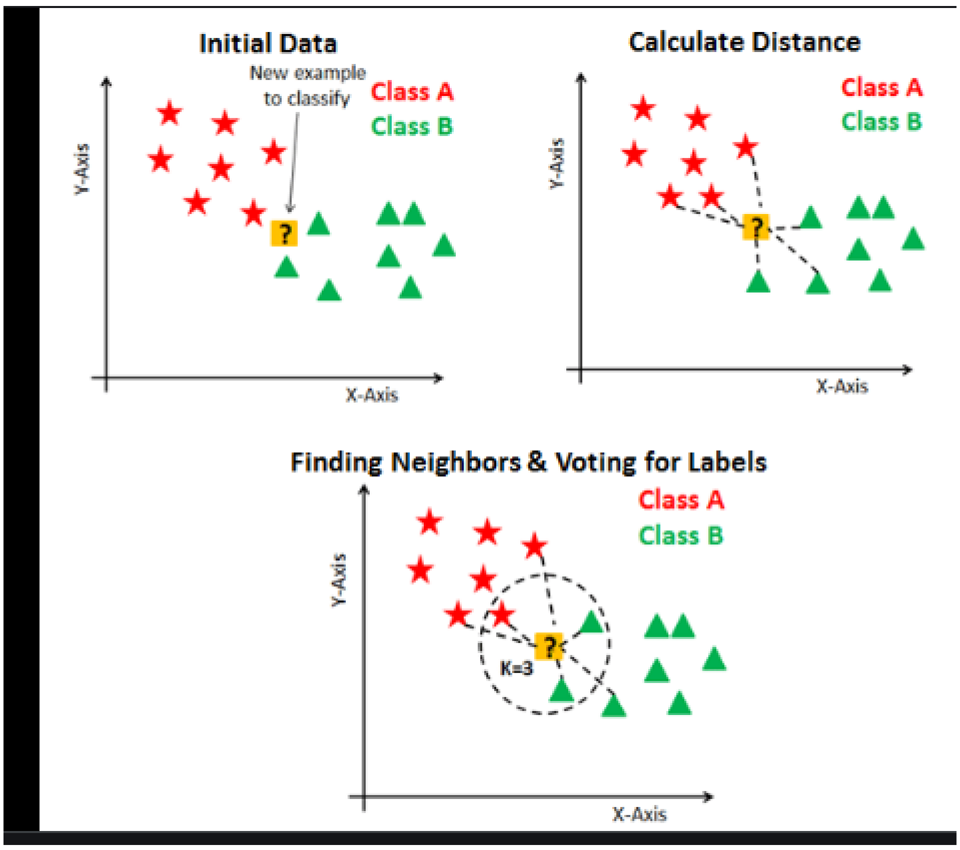
In KNN, K is the number of nearest neighbors. The number of neighbors is the core deciding factor. K is generally an odd number if the number of classes is 2. When K=1, then the algorithm is known as the nearest neighbor algorithm. This is the simplest case. Suppose P1 is the point, for which label needs to predict. First, you find the one closest point to P1 and then the label of the nearest point assigned to P1.



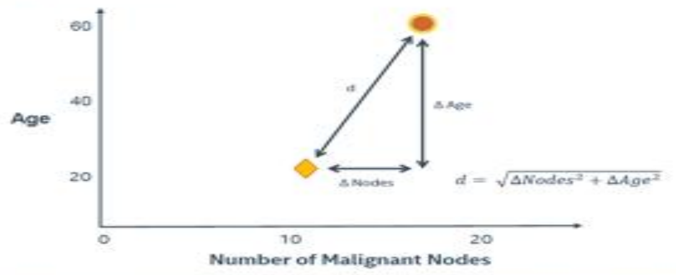
Suppose P1 is the point, for which label needs to predict. First, you find the k closest point to P1 and then classify points by majority vote of its k neighbors. Each object votes for their class and the class with the most votes is taken as the prediction. For finding closest similar points, you find the distance between points using distance measures such as Euclidean distance, Hamming distance, Manhattan distance and Minkowski distance. KNN has the following basic steps:

1. Calculate distance
2. Find closest neighbors
3. Vote for labels

Notice in the image above that most of the time, similar data points are close to each other. The KNN algorithm hinges on this assumption being true enough for the algorithm to be useful. KNN captures the idea of similarity (sometimes called distance, proximity, or closeness) with some mathematics we might have learned in our childhood— calculating the distance between points on a graph.

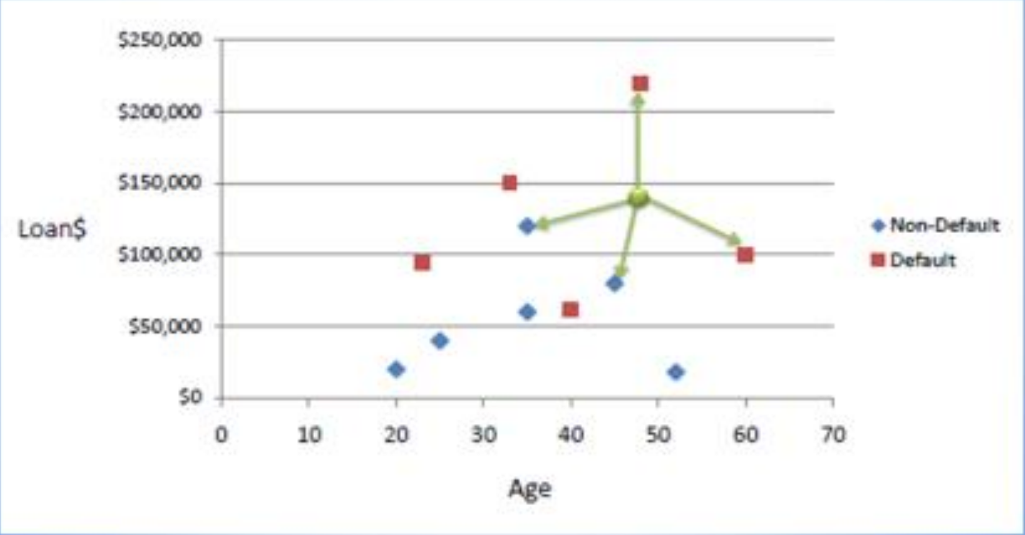


EUCLIDEAN DISTANCE (L2 DISTANCE)


Age
60
40
20
0
10
20
Number of Malignant Nodes
d
Δ Age
Δ Nodes
$d = \sqrt{\Delta Nodes^2 + \Delta Age^2}$

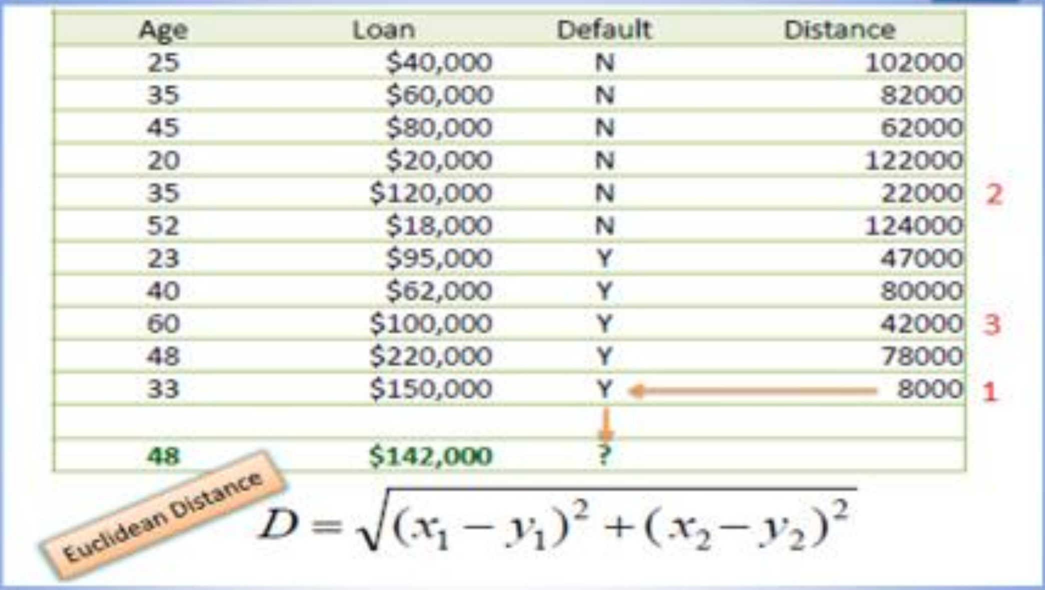
$$d = \sqrt{(node1-node2)^2 + (age1-age2)^2}$$

Consider the following data concerning credit default. Age and Loan are two numerical variables (predictors) and Default is the target.



We can now use the training set to classify an unknown case (Age=48 and Loan=$142,000) using Euclidean distance. If K=1 then the nearest neighbor is the last case in the training set with Default=Y.

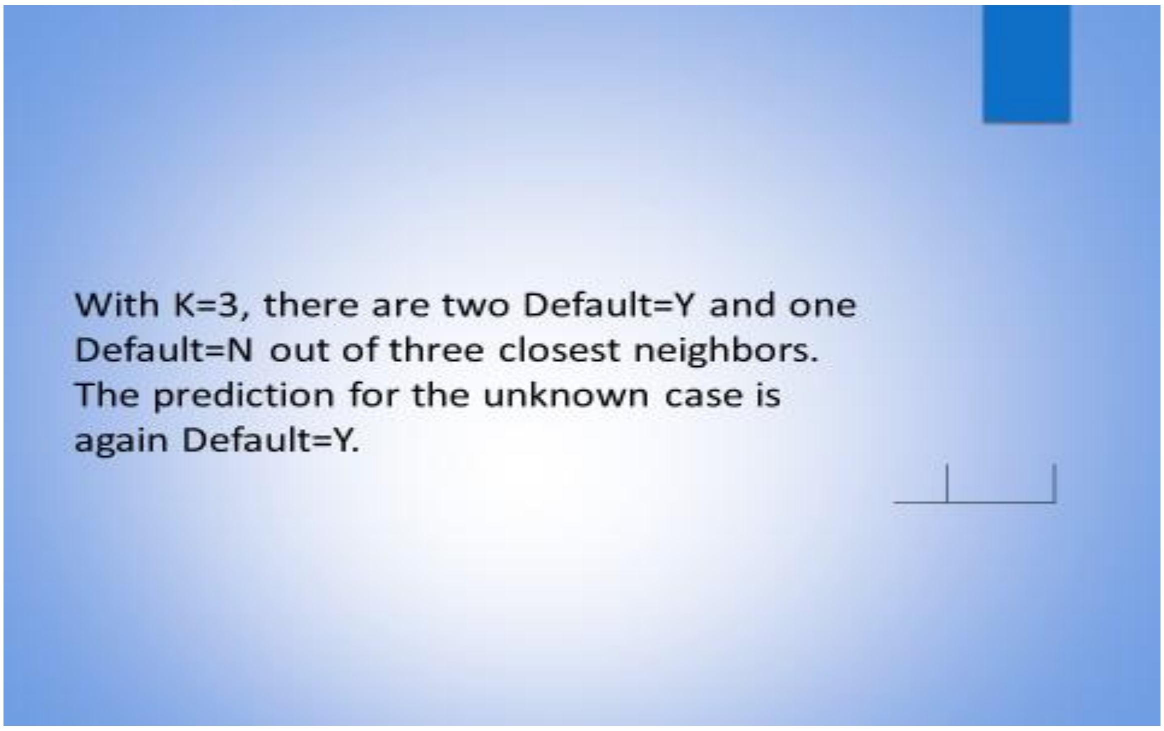
- D = Sqrt[(48-33)^2 + (142000-150000)^2] = 8000.01 >> Default=Y



Age	Loan	Default	Distance	
25	$40,000	N	102000	
35	$60,000	N	82000	
45	$80,000	N	62000	
20	$20,000	N	122000	
35	$120,000	N	22000	2
52	$18,000	N	124000	
23	$95,000	Y	47000	
40	$62,000	Y	80000	
60	$100,000	Y	42000	3
48	$220,000	Y	78000	
33	$150,000	Y	8000	1
48	**$142,000**	?		

Euclidean Distance

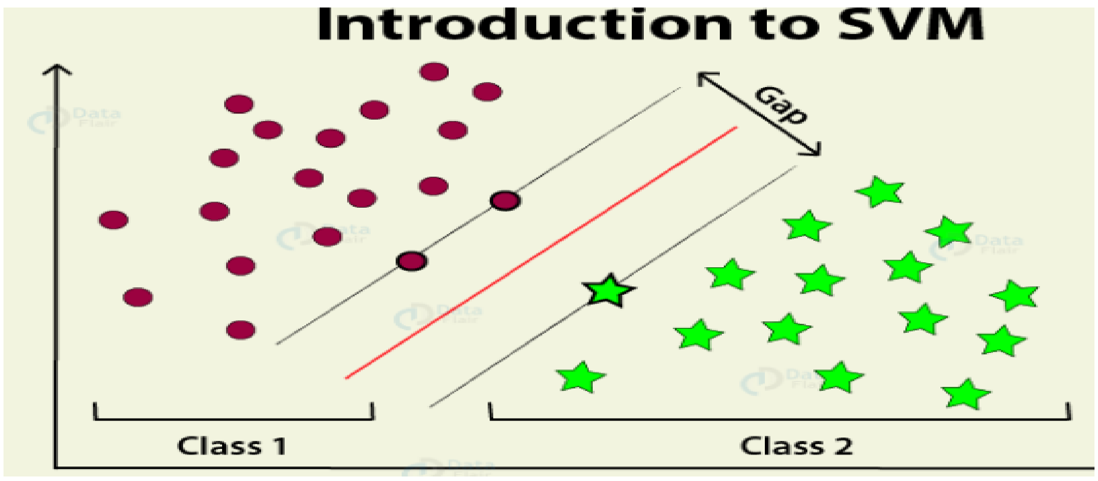
$$D = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2}$$

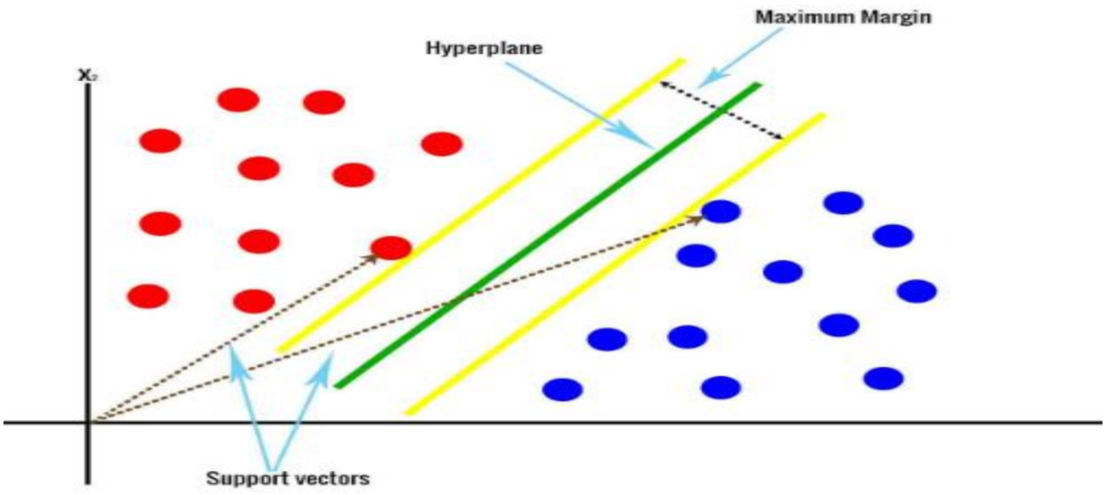

With K=3, there are two Default=Y and one Default=N out of three closest neighbors. The prediction for the unknown case is again Default=Y.

```
fromsklearn.neighborsimportKNeighborsClassifier
```

- from sklearn.neighbors import KNeighborsClassifier
- neigh = KNeighborsClassifier(n_neighbors=3)

Support Vector Machine Classifier



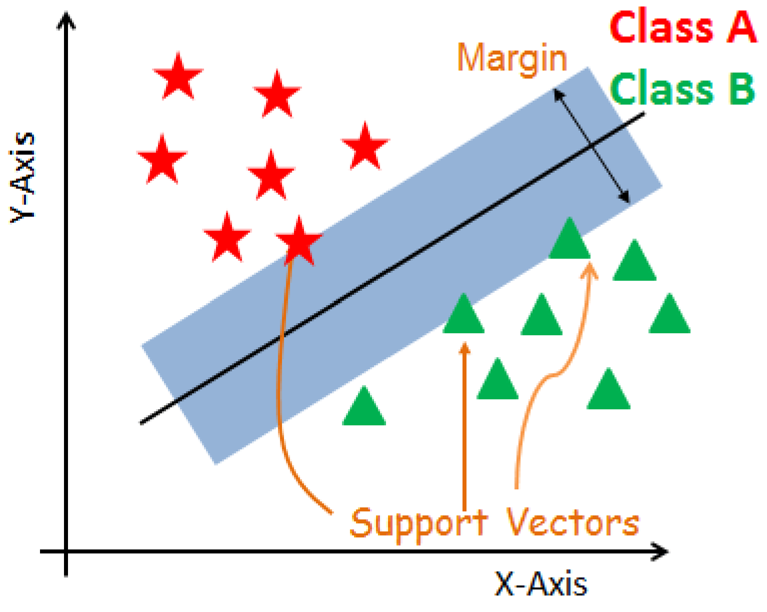


SVM offers very high accuracy compared to other classifiers such as logistic regression, and decision trees. It is known for its kernel trick to handle nonlinear input spaces. It is used in a variety of applications such as face detection, intrusion detection, classification of emails, news articles and web pages, classification of genes, and handwriting recognition.

SVM is an exciting algorithm and the concepts are relatively simple. The classifier separates data points using a hyperplane with the largest amount of margin. That's why an SVM classifier is also known as a discriminative classifier. SVM finds an optimal hyperplane which helps in classifying new data points.

Support Vector Machines

Generally, Support Vector Machines is considered to be a classification approach, it but can be employed in both types of classification and regression problems. It can easily handle multiple continuous and categorical variables. SVM constructs a hyperplane in multidimensional space to separate different classes. SVM generates optimal hyperplane in an iterative manner, which is used to minimize an error. The core idea of SVM is to find a maximum marginal hyperplane(MMH) that best divides the dataset into classes.



Support Vectors

Support vectors are the data points, which are closest to the hyperplane. These points will define the separating line better by calculating margins. These points are more relevant to the construction of the classifier.

Hyperplane

A hyperplane is a decision plane which separates between a set of objects having different class memberships.
SVM takes all the data points in consideration and gives out a line that is called '***Hyperplane***' which divides both the classes. This line is termed as '***Decision boundary***'.

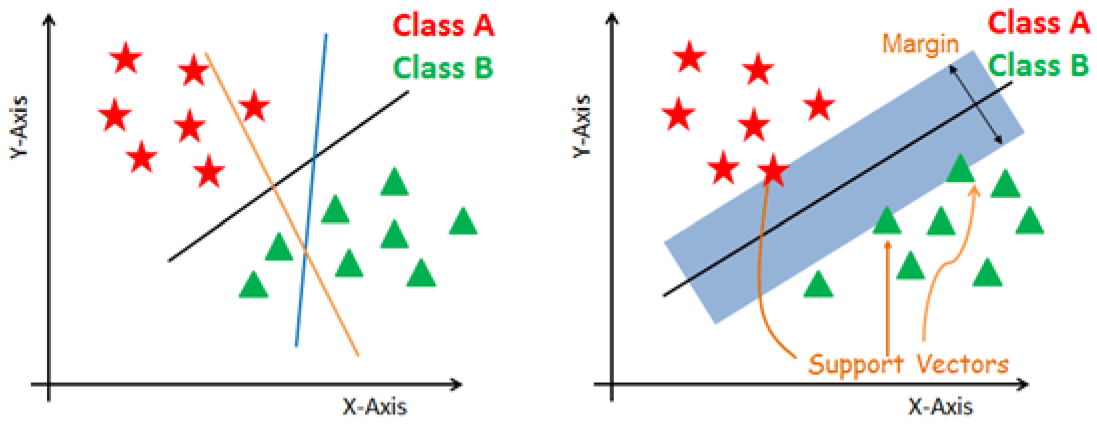
Margin

A margin is a gap between the two lines on the closest class points. This is calculated as the perpendicular distance from the line to support vectors or closest points. If the margin is larger in between the classes, then it is considered a good margin, a smaller margin is a bad margin.

How does SVM work?

The main objective is to segregate the given dataset in the best possible way. The distance between the either nearest points is known as the margin. The objective is to select a hyperplane with the maximum possible margin between support vectors in the given dataset. SVM searches for the maximum marginal hyperplane in the following steps:

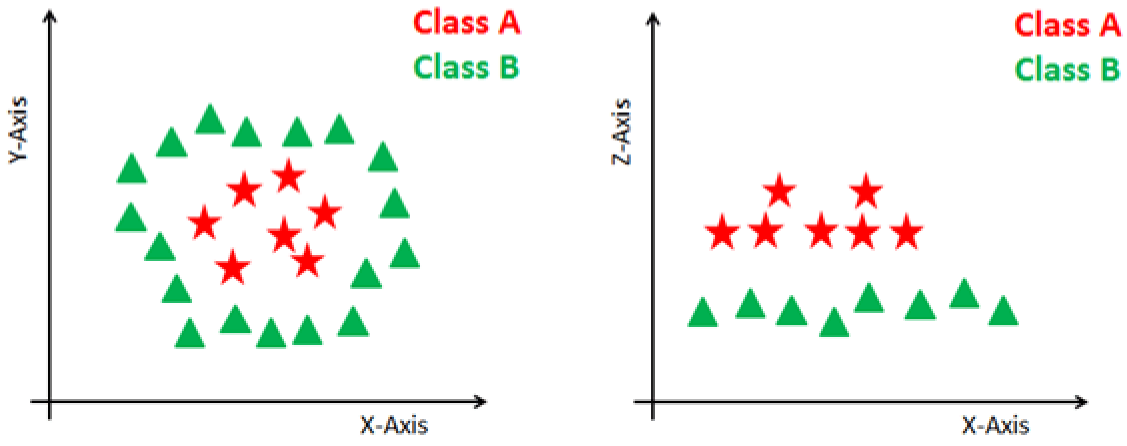
1. Generate hyperplanes which segregates the classes in the best way. Left-hand side figure showing three hyperplanes black, blue and orange. Here, the blue and orange have higher classification error, but the black is separating the two classes correctly.
2. Select the right hyperplane with the maximum segregation from the either nearest data points as shown in the right-hand side figure



Dealing with non-linear and inseparable planes

Some problems can’t be solved using linear hyperplane, as shown in the figure below (left-hand side).

In such situation, SVM uses a kernel trick to transform the input space to a higher dimensional space as shown on the right.



SVM Kernels

The SVM algorithm is implemented in practice using a kernel. A kernel transforms an input data space into the required form. SVM uses a technique called the kernel trick. Here, the kernel takes a low-dimensional input space and transforms it into a higher dimensional space. In other words, you can say that it converts nonseparable problem to separable problems by adding more dimension to it. It is most useful in non-linear separation problem. Kernel trick helps you to build a more accurate classifier.

- **Linear Kernel** A linear kernel can be used as normal dot product any two given observations. The product between two vectors is the sum of the multiplication of each pair of input values.

```
K(x, xi) = sum(x * xi)
```

- **Polynomial Kernel** A polynomial kernel is a more generalized form of the linear kernel. The polynomial kernel can distinguish curved or nonlinear input space.

```
K(x,xi) = 1 + sum(x * xi)^d
```

Where d is the degree of the polynomial. d=1 is similar to the linear transformation. The degree needs to be manually specified in the learning algorithm.

- **Radial Basis Function Kernel** The Radial basis function kernel is a popular kernel function commonly used in support vector machine classification. RBF can map an input space in infinite dimensional space.

```
K(x,xi) = exp(-gamma * sum((x – xi^2))
```

Here gamma is a parameter, which ranges from 0 to 1. A higher value of gamma will perfectly fit the training dataset, which causes over-fitting. Gamma=0.1 is considered to be a good default value. The value of gamma needs to be manually specified in the learning algorithm.

Classifier Building in Scikit-learn

```
fromsklearn.svm import SVC
```