Classification

Naive Bayes Classifier

The Naive Bayes classifier is a simple yet powerful supervised machine learning algorithm based on **Bayes' Theorem**. It is widely used for classification tasks, especially in text classification and natural language processing.

Key Concepts

1. **Bayes' Theorem**: $P(A \mid B) = \frac{P(B|A) \cdot P(A)}{P(B)}$
    - Here, (P(A|B)) is the posterior probability of class (A) given feature (B).
    - (P(B|A)) is the likelihood of feature (B) given class (A).
    - (P(A)) is the prior probability of class (A).
    - (P(B)) is the prior probability of feature (B).
2. **Naive Assumption**:
    - It assumes that all features are **conditionally independent** given the class label. While this assumption is rarely true in real-world data, it simplifies computation and often performs well in practice.
3. **Classification**:
    - The algorithm calculates the probability of a data point belonging to each class and assigns it to the class with the highest probability.

Types of Naive Bayes Classifiers

1. **Gaussian Naive Bayes**:
    - Used when features are continuous and assumed to follow a Gaussian (normal) distribution.
2. **Multinomial Naive Bayes**:
    - Suitable for discrete data, often used in text classification (e.g., spam detection).
3. **Bernoulli Naive Bayes**:
    - Designed for binary/boolean features, commonly used in document classification tasks.

Advantages

- Simple and fast to implement.
- Works well with high-dimensional data.

- Effective for small datasets.

Limitations

- Relies on the assumption of feature independence, which may not hold in all cases.
- Struggles with datasets where features are highly correlated.

Applications

- Spam email detection.
- Sentiment analysis.
- Document classification.
- Medical diagnosis.

Dataset & Program:

Dataset:

Social_Network_Ads.csv

Age	EstimateSalary	Purchased
19	49000	0
35	55000	0
32	150000	1
25	53000	0
47	80000	1
48	88000	1
26	75000	0

Like this ……

```
#Naive Bayes

# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Importing the dataset
dataset = pd.read_csv('Social_Network_Ads.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values

# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state
= 0)
print(X_train)
print(y_train)
print(X_test)
print(y_test)

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

# Training the Naive Bayes model on the Training set
```

```
from sklearn.naive_bayes import GaussianNB
classifier = GaussianNB()
classifier.fit(X_train, y_train)

# Predicting a new result
print(classifier.predict(sc.transform([[30,87000]])))

# Predicting the Test set results
y_pred = classifier.predict(X_test)
print(np.concatenate((y_pred.reshape(len(y_pred),1),
y_test.reshape(len(y_test),1)),1))

# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy_score(y_test, y_pred)

# Visualising the Training set results
from matplotlib.colors import ListedColormap
X_set, y_set = sc.inverse_transform(X_train), y_train
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 10, stop = X_set[:,
0].max() + 10, step = 0.25),
                     np.arange(start = X_set[:, 1].min() - 1000, stop = X_set[:, 1].max() +
1000, step = 0.25))
plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(),
X2.ravel()])).T)).reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
```

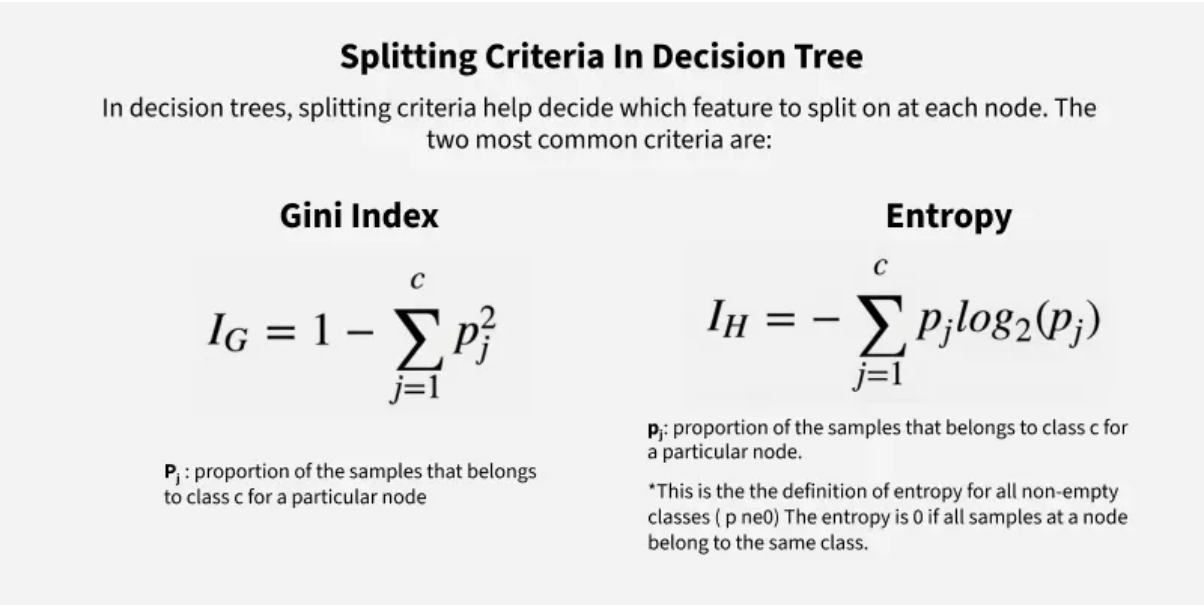
```
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1], c = ListedColormap(('red',
'green'))(i), label = j)
plt.title('Naive Bayes (Training set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()


# Visualising the Test set results
from matplotlib.colors import ListedColormap
X_set, y_set = sc.inverse_transform(X_test), y_test
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 10, stop = X_set[:,
0].max() + 10, step = 0.25),
                     np.arange(start = X_set[:, 1].min() - 1000, stop = X_set[:, 1].max() +
1000, step = 0.25))
plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(),
X2.ravel()]).T)).reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1], c = ListedColormap(('red',
'green'))(i), label = j)
plt.title('Naive Bayes (Test set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

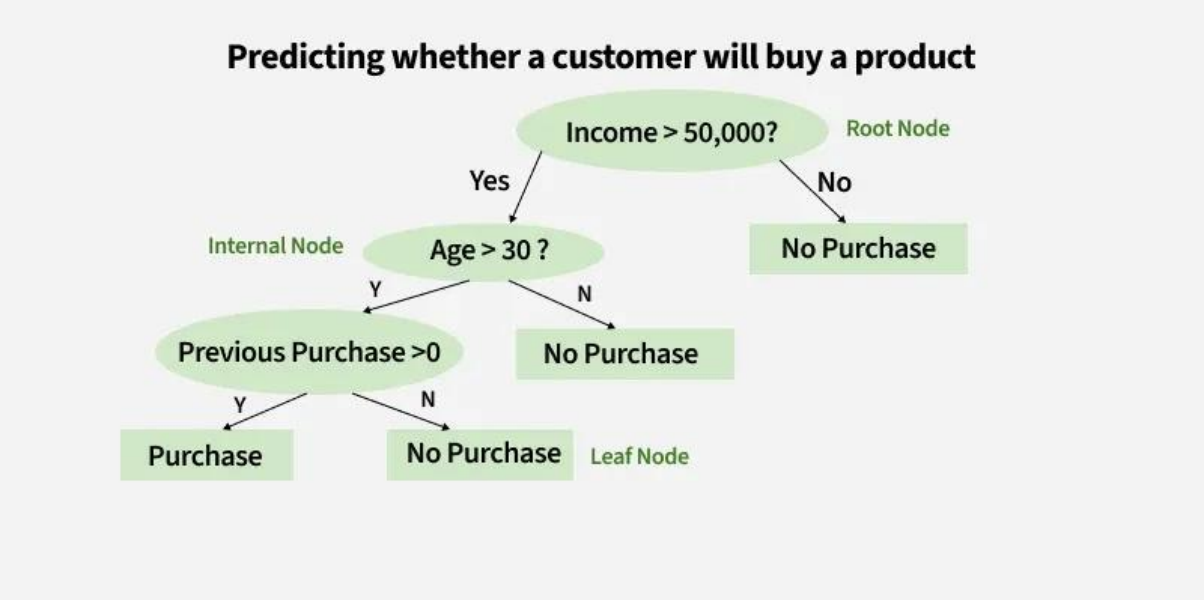
Decision Tree

A decision tree is a graphical representation that outlines the various choices available and the potential outcomes of those choices. It begins with a root node, which represents the initial decision or problem. From this root node, branches extend to represent different options or actions that can be taken. Each branch leads to further decision nodes, where additional choices can be made, and these in turn branch out to show the possible consequences of each decision. This continues until the branches reach leaf nodes, which represent the final outcomes or decisions.

The decision tree structure allows for a clear and organized way to visualize the decision-making process, making it easier to understand how different choices lead to different results. This is particularly useful in complex scenarios where multiple factors and potential outcomes need to be considered. By breaking down the decision process into manageable steps and visually mapping them out, decision trees help decision-makers evaluate the potential risks and benefits of each option, leading to more informed and rational decisions.

Decision trees are useful tools in many fields like business, healthcare, and finance. They help analyze things systematically by providing a simple way to compare different strategies and their likely impacts. This helps organizations and individuals make decisions that are not only based on data but also transparent and justifiable. This ensures that the chosen path aligns with their objectives and constraints.



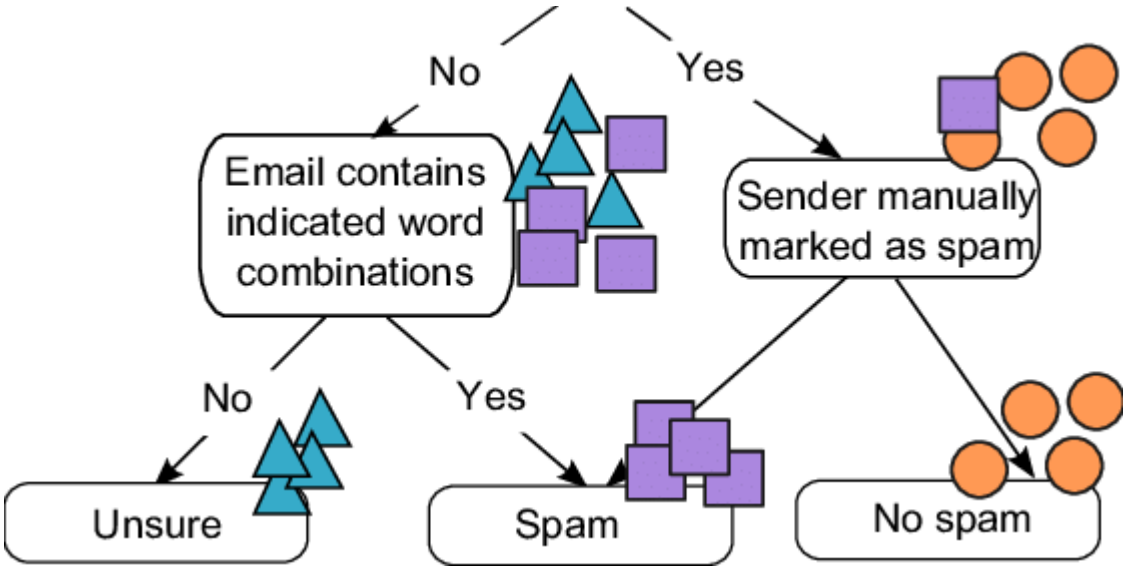


Types of Decision Trees

It's important to remember the different types of decision trees: classification trees and regression trees. Each type has various algorithms, nodes, and branches that make them unique. It's crucial to select the type that best fits the purpose of your decision tree.

Classification Trees

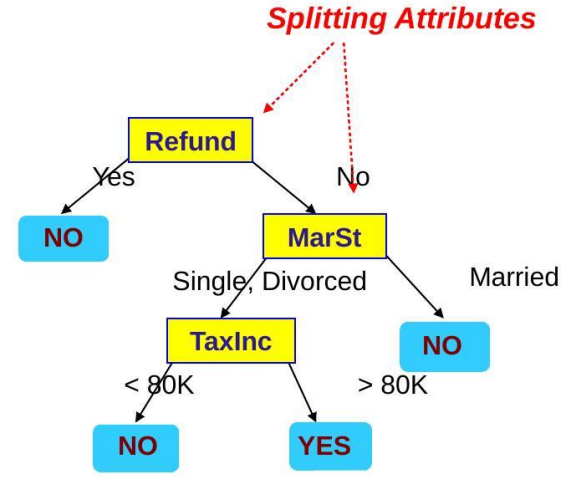
Classification trees are used when the target variable is categorical. The tree splits the dataset into subsets based on the values of attributes, aiming to classify instances into classes or categories. For example, determining whether an email is spam or not spam.



Example of a Decision Tree

Tid	Refund	Marital Status	Taxable Income	Cheat
	categorical	categorical	continuous	class
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

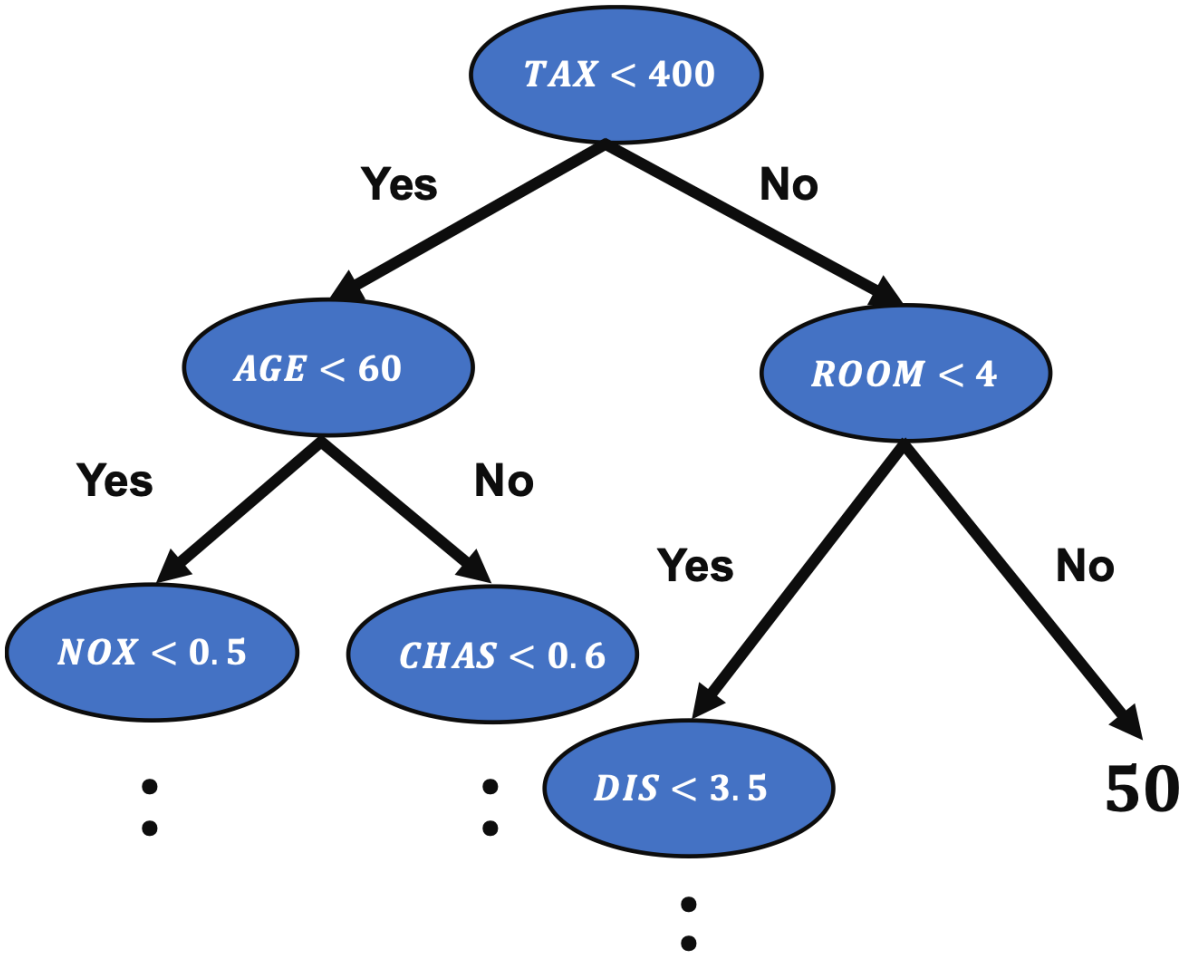
Training Data



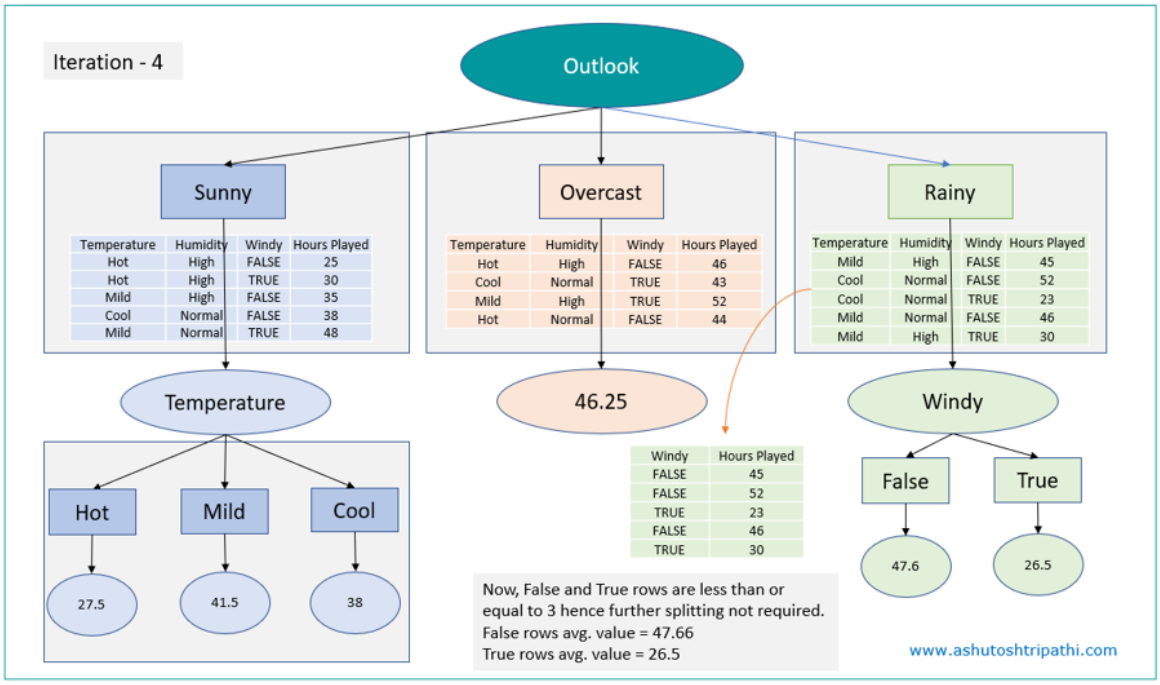
Model: Decision Tree

Regression Trees

Regression trees are employed when the target variable is continuous. They predict outcomes that are real numbers or continuous values by recursively partitioning the data into smaller subsets. For example, predicting the price of a house based on its features.



Weather Forecasting:



Same Dataset & Decision Tree Classification Program:

```
# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Importing the dataset
dataset = pd.read_csv('Social_Network_Ads.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values

# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)
print(X_train)
```

```
print(y_train)
print(X_test)
print(y_test)

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

# Training the Decision Tree Classification model on the Training set
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier(criterion = 'entropy', random_state = 0)
classifier.fit(X_train, y_train)

# Predicting a new result
print(classifier.predict(sc.transform([[30,87000]])))

# Predicting the Test set results
y_pred = classifier.predict(X_test)
print(np.concatenate((y_pred.reshape(len(y_pred),1),
y_test.reshape(len(y_test),1)),1))

# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy_score(y_test, y_pred)
```

```
# Visualising the Training set results
from matplotlib.colors import ListedColormap
X_set, y_set = sc.inverse_transform(X_train), y_train
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 10, stop = X_set[:, 0].max() + 10, step = 0.25),
                     np.arange(start = X_set[:, 1].min() - 1000, stop = X_set[:, 1].max() + 1000, step = 0.25))
plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(), X2.ravel()])).T)).reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1], c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('Decision Tree Classification (Training set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()


# Visualising the Test set results
from matplotlib.colors import ListedColormap
X_set, y_set = sc.inverse_transform(X_test), y_test
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 10, stop = X_set[:, 0].max() + 10, step = 0.25),
                     np.arange(start = X_set[:, 1].min() - 1000, stop = X_set[:, 1].max() + 1000, step = 0.25))
plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(), X2.ravel()])).T)).reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
```

```
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1], c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('Decision Tree Classification (Test set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

Dataset & Decision Tree Regression Programs:

Dataset:

Position	Level	Salary
Business Analyst	**1**	**55000**
Junior Consultant	**2**	**60000**
Senior Consultant	**3**	**70000**
Manager	**4**	**90000**
---	**--**	**--**
C-level	**9**	**520000**
CEO	**10**	**120000**

Programs:

```
# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Importing the dataset
dataset = pd.read_csv('Position_Salaries.csv')
X = dataset.iloc[:, 1:-1].values
y = dataset.iloc[:, -1].values

# Training the Decision Tree Regression model on the whole dataset
from sklearn.tree import DecisionTreeRegressor
regressor = DecisionTreeRegressor(random_state = 0)
regressor.fit(X, y)

# Predicting a new result
regressor.predict([[6.5]])

# Visualising the Decision Tree Regression results (higher resolution)
```

```
X_grid = np.arange(min(X), max(X), 0.01)
X_grid = X_grid.reshape((len(X_grid), 1))
plt.scatter(X, y, color = 'red')
plt.plot(X_grid, regressor.predict(X_grid), color = 'blue')
plt.title('Truth or Bluff (Decision Tree Regression)')
plt.xlabel('Position level')
plt.ylabel('Salary')
plt.show()
```

Support Vector Machine

Support Vector Machines (SVM) are supervised machine learning algorithms used for classification, regression, and outlier detection. The core idea of SVM is to find the **optimal hyperplane** that separates data points of different classes with the maximum margin. This margin is the distance between the hyperplane and the nearest data points, known as **support vectors**.

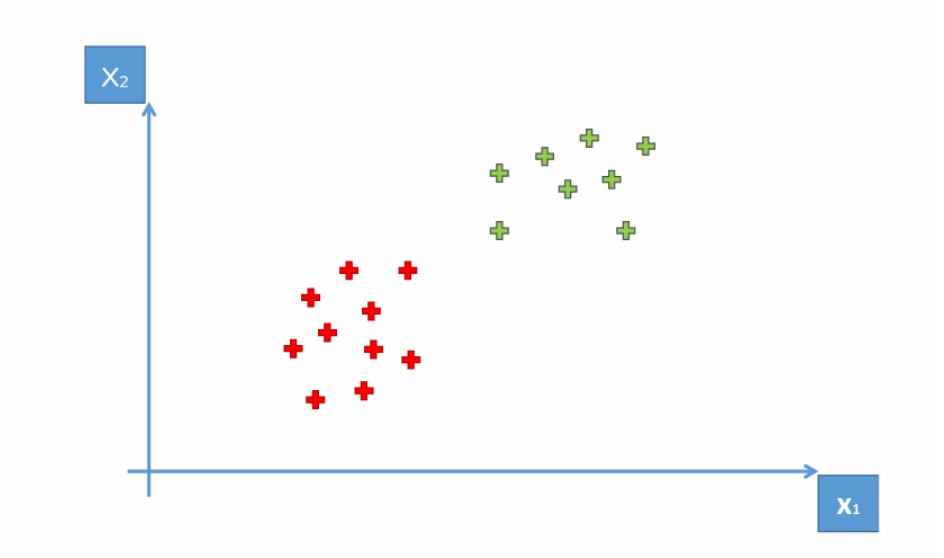
Key Concepts of SVM

- **Hyperplane**: A decision boundary that separates classes in the feature space.
- **Support Vectors**: Data points closest to the hyperplane, crucial for defining the margin.
- **Margin**: The distance between the hyperplane and the nearest support vectors. SVM aims to maximize this margin.
- **Kernel Trick**: A method to transform non-linearly separable data into a higher-dimensional space where it becomes linearly separable. Common kernels include linear, polynomial, and radial basis function (RBF).

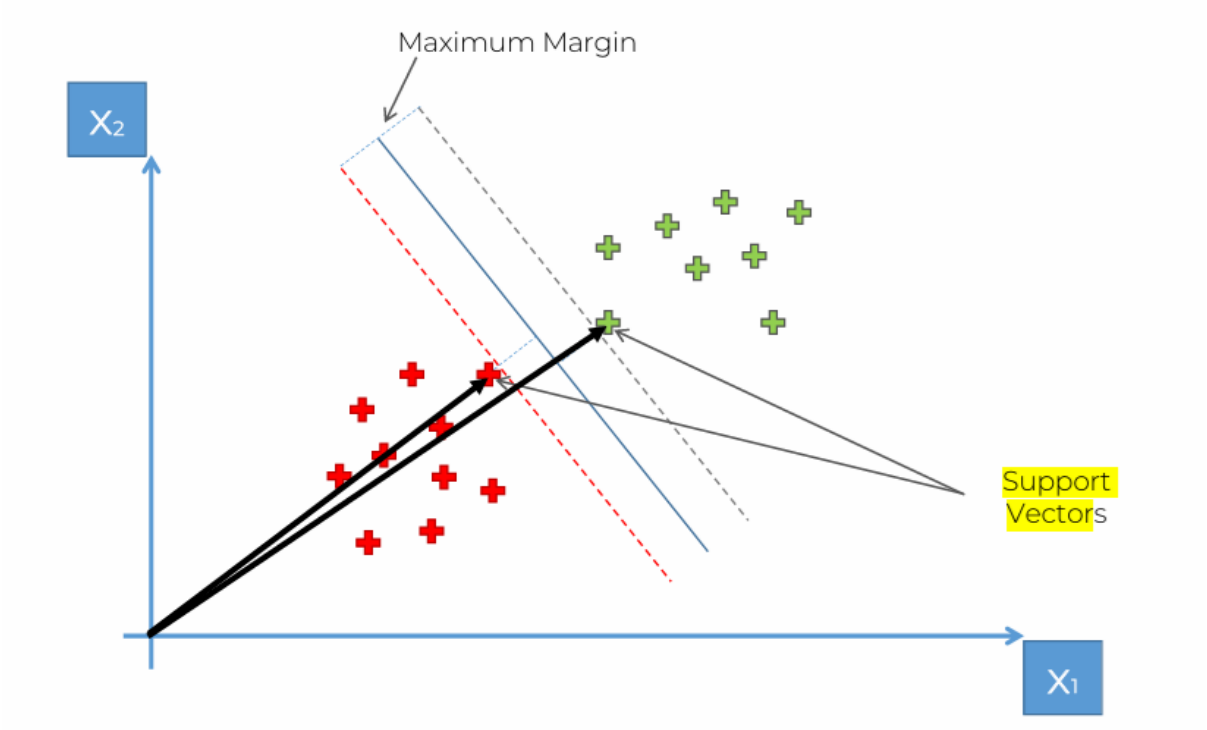
Types of SVM

- **Linear SVM**: Used when data is linearly separable. It finds a straight-line hyperplane in 2D or a flat hyperplane in higher dimensions.
- **Non-Linear SVM**: Handles non-linearly separable data using kernel functions to map data into a higher-dimensional space.

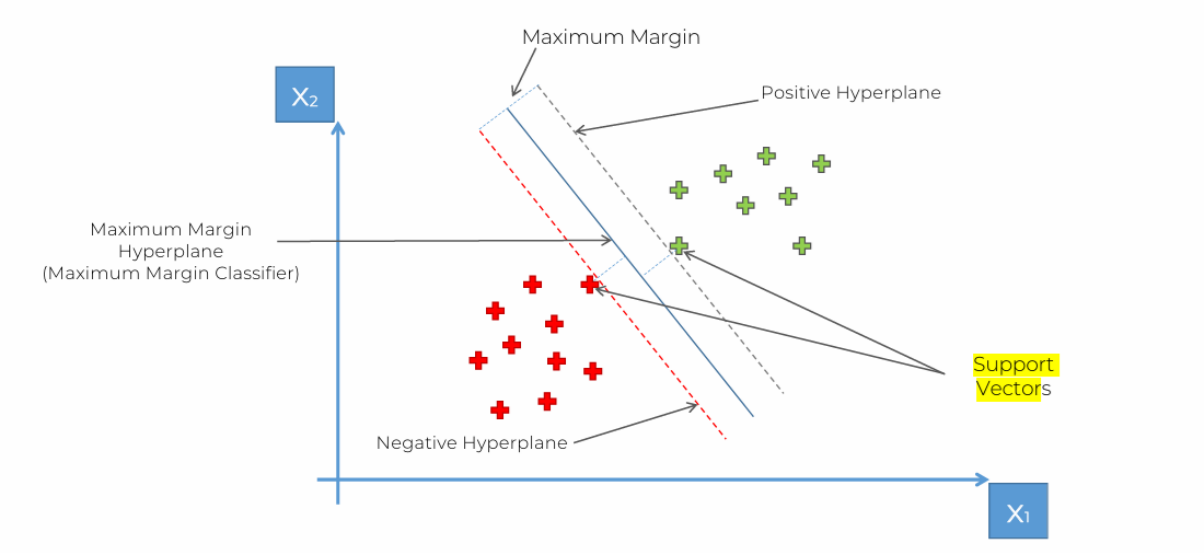
How to separate these points?



Support Vector:



Hyperplanes:



Mathematical Computation of SVM

Consider a binary classification problem with two classes, labeled as +1 and -1. We have a training dataset consisting of input feature vectors X and their corresponding class labels Y. The equation for the linear hyperplane can be written as:

$wTx+b=0$

Where:

- w is the normal vector to the hyperplane (the direction perpendicular to it).
- b is the offset or bias term representing the distance of the hyperplane from the origin along the normal vector w.

Distance from a Data Point to the Hyperplane

- The distance between a data point x_i and the decision boundary can be calculated as:

$$di=(w^T x_i+b)/ ||w||$$

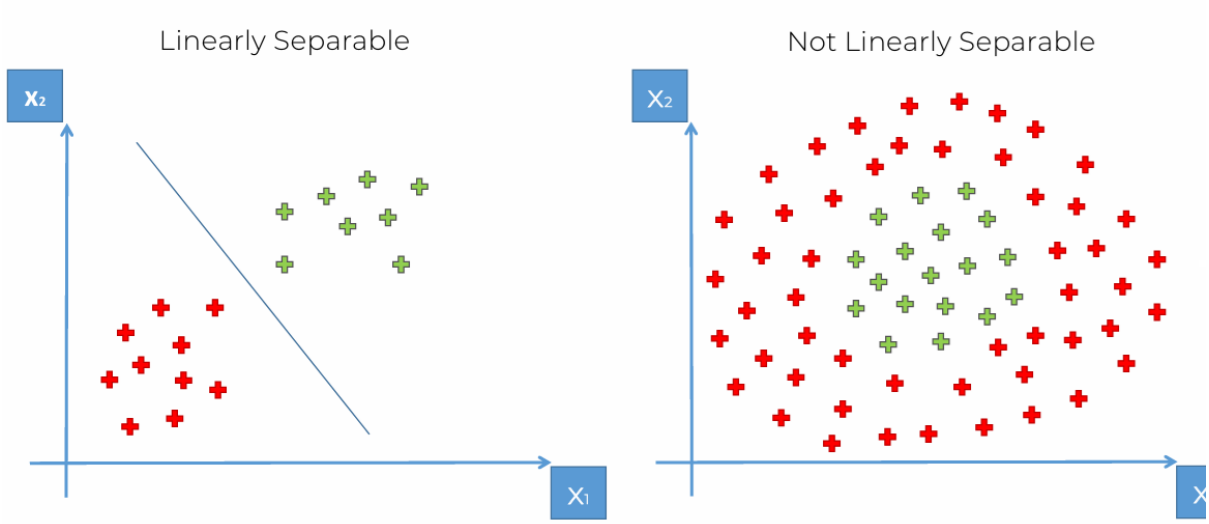
where ||w|| represents the Euclidean norm of the weight vector w.

Linear SVM Classifier

Distance from a Data Point to the Hyperplane:

$$\hat{y} = \begin{cases} 1 & : \ w^T x + b \geq 0 \\ -1 & : \ w^T x + b < 0 \end{cases}$$

Where $\hat{y}$ is the predicted label of a data point.



Support Vector Regression (SVR)

Support Vector Regression (SVR) is a machine learning technique derived from Support Vector Machines (SVM) and is used for regression tasks. Unlike traditional regression models, SVR focuses on predicting continuous outputs by finding a hyperplane that best fits the data within a specified margin of tolerance, known as epsilon.

Key Concepts in SVR

SVR operates by transforming input features into a high-dimensional space using kernel functions. This allows it to handle both linear and non-linear relationships effectively. The primary goal is to maximize the number of data points within the epsilon margin while minimizing prediction errors outside this range.

Kernels in SVR

Kernels are mathematical functions that define the similarity between data points. Common kernel types include:

- Linear Kernel: Suitable for linearly separable data.
- Polynomial Kernel: Captures more complex relationships by introducing non-linearity.
- Radial Basis Function (RBF) Kernel: Handles intricate patterns and is widely used for non-linear data.

Hyperparameters

SVR has several hyperparameters that influence its performance:

- C: Controls the trade-off between achieving a low error on the training data and maintaining a smooth model. Higher values of C focus on minimizing training errors.
- Epsilon (ε): Defines the margin of tolerance within which predictions are not penalized.
- Gamma: Determines the influence of a single training example in non-linear kernels like RBF.

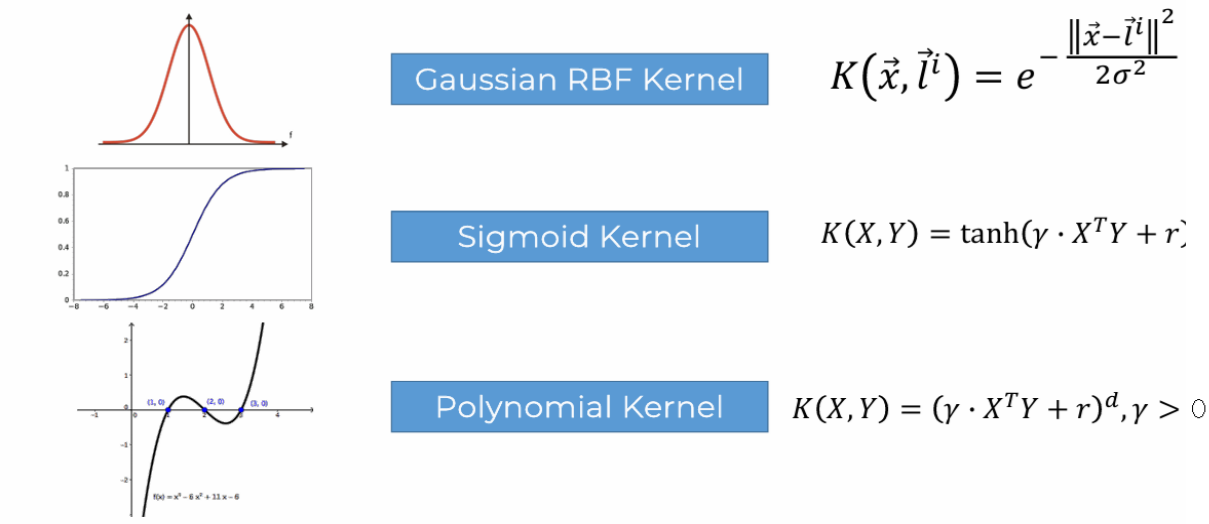
Key Concepts of Non-Linear SVM

Non-linear SVMs rely on the kernel trick, which computes the dot product of data points in a higher-dimensional space without explicitly transforming the data. This allows SVMs to efficiently handle complex decision boundaries without incurring high computational costs.

Popular Kernel Functions

- Radial Basis Function (RBF): Ideal for circular or spherical patterns, it measures the distance between points and maps them into a higher-dimensional space.

Types of Kernel Function:



The Gaussian RBF Kernal:

The Radial Basis Function (RBF) kernel, also known as the Gaussian kernel, is a widely used kernel function in machine learning. It measures the similarity between two data points based on their Euclidean distance in the input space. The RBF kernel is mathematically defined as:

$K(x, x') = \exp(-||x - x'||^2 / (2\sigma^2))$

Here:

- $||x - x'||^2$ is the squared Euclidean distance between two points.
- σ (sigma) is the bandwidth parameter that controls the smoothness of the decision boundary.

Key Characteristics of RBF and Gaussian Kernels

1. Infinite-Dimensional Projection: The RBF kernel implicitly maps data into an infinite-dimensional feature space. This allows linear algorithms applied in this space to model highly non-linear relationships in the original input space.
2. Local Similarity: The kernel assigns higher similarity values to points that are closer in the input space and lower values to points farther apart. This

behavior creates a "bump" or "hill" around each data point, decaying exponentially with distance.

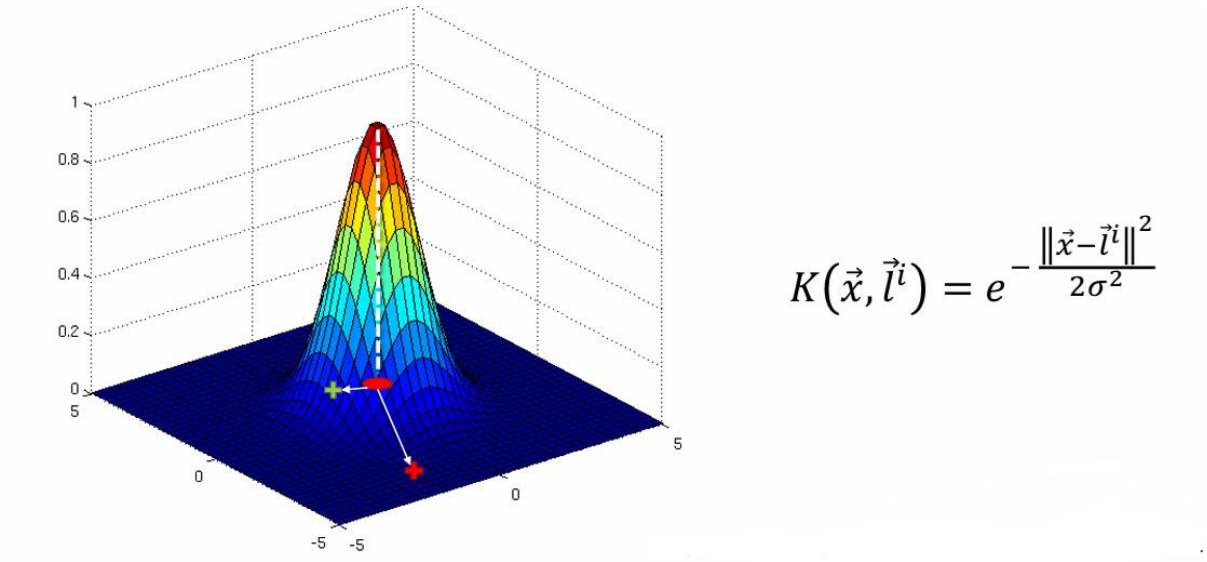
3. Parameter σ: The parameter σ determines the width of the Gaussian curve. A smaller σ results in narrower bumps, capturing finer details, while a larger σ smoothens the decision boundary.

The RBF kernel on two samples $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^k$, represented as feature vectors in some *input space*, is defined as[2]

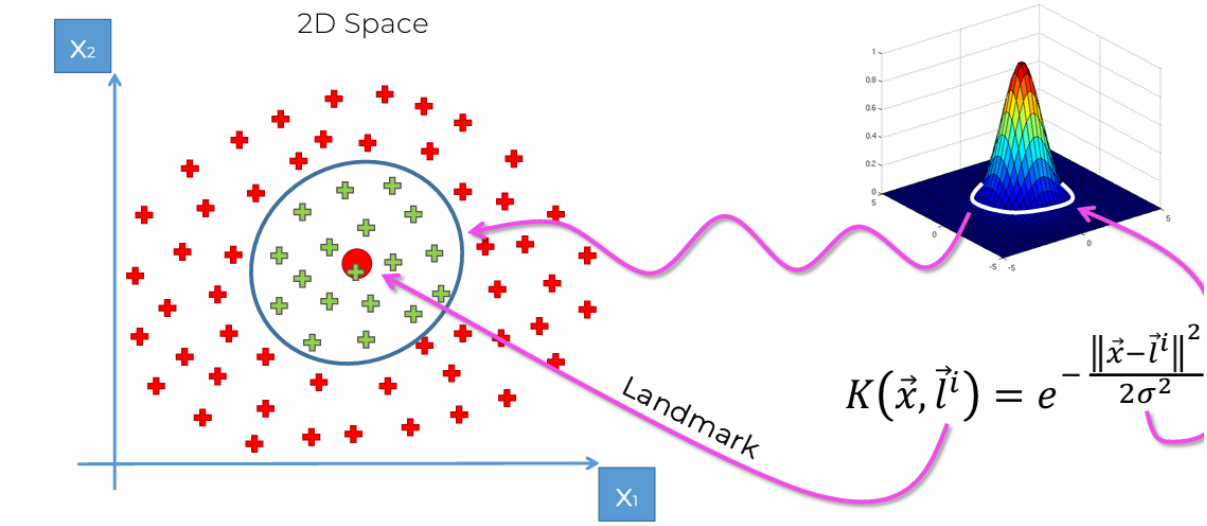
$$K(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{2\sigma^2}\right)$$

$\|\mathbf{x} - \mathbf{x}'\|^2$ may be recognized as the squared Euclidean distance between the two feature vectors. σ is a free parameter. equivalent definition involves a parameter $\gamma = \frac{1}{2\sigma^2}$:

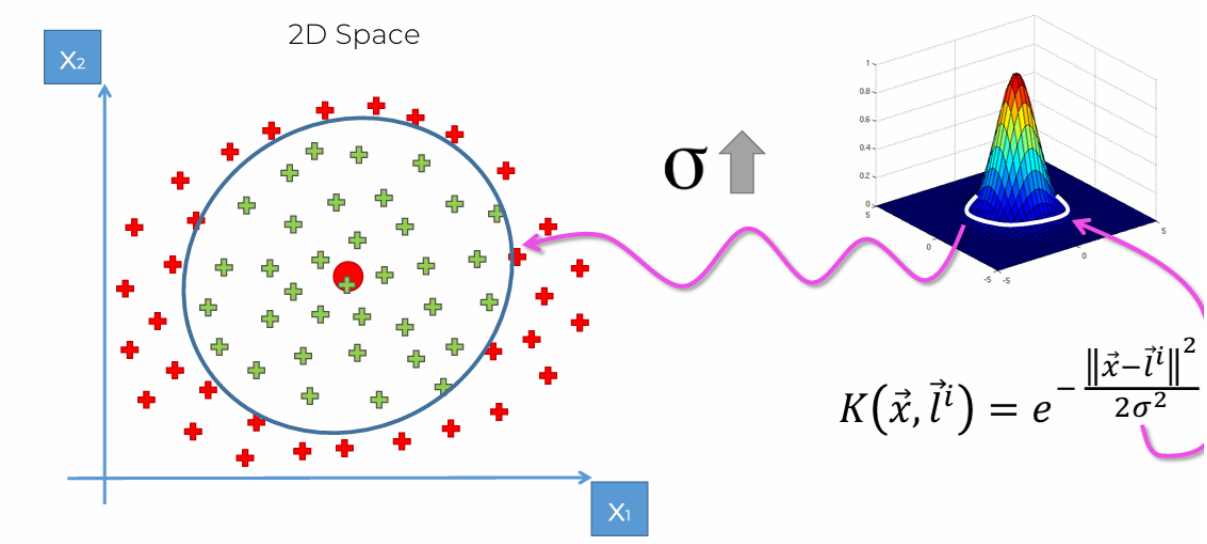
$$K(\mathbf{x}, \mathbf{x}') = \exp(-\gamma\|\mathbf{x} - \mathbf{x}'\|^2)$$

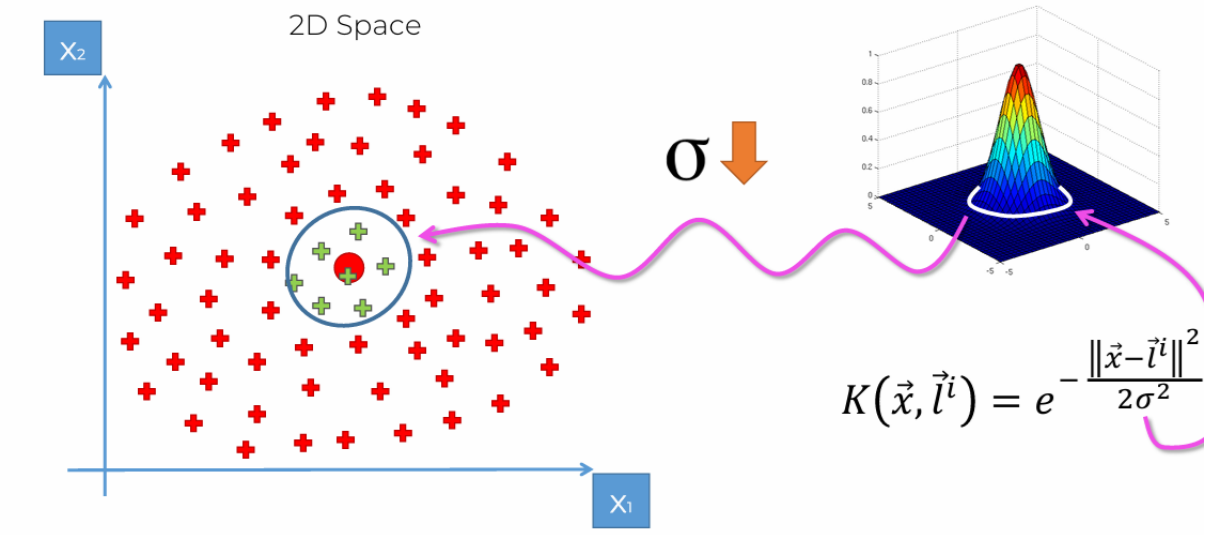


$$K(\vec{x}, \vec{l}^i) = e^{-\frac{\|\vec{x}-\vec{l}^i\|^2}{2\sigma^2}}$$



$$K(\vec{x}, \vec{l}^i) = e^{-\frac{\|\vec{x}-\vec{l}^i\|^2}{2\sigma^2}}$$


2D Space
X2
X1
σ
$K(\vec{x}, \vec{l}^i) = e^{-\frac{\|\vec{x}-\vec{l}^i\|^2}{2\sigma^2}}$


2D Space
X2
X1
σ
$K(\vec{x}, \vec{l}^i) = e^{-\frac{\|\vec{x}-\vec{l}^i\|^2}{2\sigma^2}}$

Dataset & Program:

Age	EstimatedSalary	Purchased
19	19000	0
35	20000	0
26	43000	0
27	57000	0
19	76000	0
27	58000	0
27	84000	0
32	150000	1
25	33000	0
--	--	--

PROGRAMS:

```
# Kernel SVM


# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd


# Importing the dataset
dataset = pd.read_csv('Social_Network_Ads.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values


# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state
= 0)
print(X_train)
print(y_train)
```

```
print(X_test)
print(y_test)

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

# Training the Kernel SVM model on the Training set
from sklearn.svm import SVC
classifier = SVC(kernel = 'rbf', random_state = 0)
classifier.fit(X_train, y_train)

# Predicting a new result
print(classifier.predict(sc.transform([[30,87000]])))

# Predicting the Test set results
y_pred = classifier.predict(X_test)
print(np.concatenate((y_pred.reshape(len(y_pred),1),
y_test.reshape(len(y_test),1)),1))

# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy_score(y_test, y_pred)

# Visualising the Training set results
```

```
from matplotlib.colors import ListedColormap

X_set, y_set = sc.inverse_transform(X_train), y_train

X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 10, stop = X_set[:,
0].max() + 10, step = 0.25),

                     np.arange(start = X_set[:, 1].min() - 1000, stop = X_set[:, 1].max() +
1000, step = 0.25))

plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(),
X2.ravel()]).T)).reshape(X1.shape),

             alpha = 0.75, cmap = ListedColormap(('red', 'green')))

plt.xlim(X1.min(), X1.max())

plt.ylim(X2.min(), X2.max())

for i, j in enumerate(np.unique(y_set)):

    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1], c = ListedColormap(('red',
'green'))(i), label = j)

plt.title('Kernel SVM (Training set)')

plt.xlabel('Age')

plt.ylabel('Estimated Salary')

plt.legend()

plt.show()


# Visualising the Test set results

from matplotlib.colors import ListedColormap

X_set, y_set = sc.inverse_transform(X_test), y_test

X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 10, stop = X_set[:,
0].max() + 10, step = 0.25),

                     np.arange(start = X_set[:, 1].min() - 1000, stop = X_set[:, 1].max() +
1000, step = 0.25))

plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(),
X2.ravel()]).T)).reshape(X1.shape),

             alpha = 0.75, cmap = ListedColormap(('red', 'green')))

plt.xlim(X1.min(), X1.max())

plt.ylim(X2.min(), X2.max())

for i, j in enumerate(np.unique(y_set)):
```

```
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1], c = ListedColormap(('red',
'green'))(i), label = j)

plt.title('Kernel SVM (Test set)')

plt.xlabel('Age')

plt.ylabel('Estimated Salary')

plt.legend()

plt.show()
```

Random Forest

Random forests are a combination of tree predictors such that each tree depends on the values of a random vector sampled independently and with the same distribution for all trees in the forest.

Random forest is a commonly-used machine learning algorithm, trademarked by Leo Breiman and Adele Cutler, that combines the output of multiple decision trees to reach a single result. Its ease of use and flexibility have fuelled its adoption, as it handles both classification and regression problems.

Random Forest Classification

Random Forest is a popular machine learning algorithm used for classification and regression tasks. It is an **ensemble learning technique** that builds multiple decision trees during training and combines their outputs to improve accuracy and reduce overfitting.

How It Works

1. **Data Sampling**: Random subsets of the training data are created using a technique called **bootstrap sampling**.
2. **Tree Construction**: For each subset, a decision tree is built. At each split, a random subset of features is considered to determine the best split.
3. **Prediction Aggregation**:
    - For **classification**, the final prediction is made by **majority voting** across all trees.
    - For **regression**, the output is the **average** of predictions from all trees.

Ensemble methods

Ensemble learning methods are made up of a set of classifiers—e.g. decision trees—and their predictions are aggregated to identify the most popular result. The most well-known ensemble methods are bagging, also known as bootstrap aggregation, and boosting. In 1996, Leo Breiman introduced the bagging method; in this method, a random sample of data in a training set is selected with replacement—meaning that the individual data points can be chosen more than once. After several data samples are generated, these models are then trained independently, and depending on the type of task—i.e. regression or classification—the average or majority of those predictions yield a more accurate estimate. This approach is commonly used to reduce variance within a noisy dataset.

Random forest algorithm

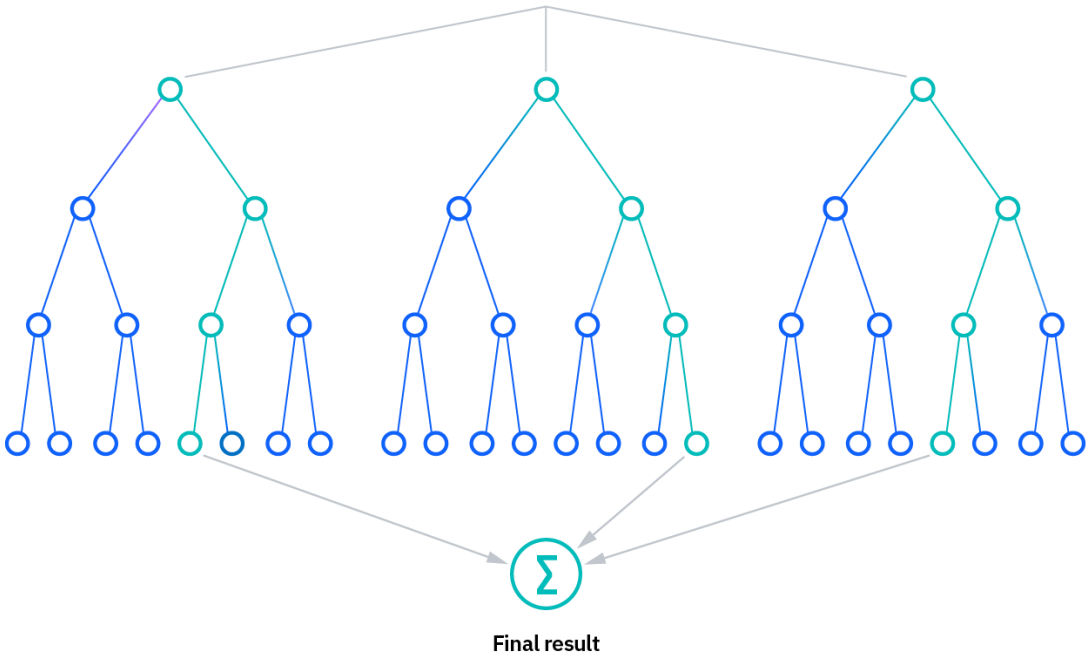
The random forest algorithm is an extension of the bagging method as it utilizes both bagging and feature randomness to create an uncorrelated forest of decision trees. Feature randomness, also known as feature bagging or “the random subspace

method”, generates a random subset of features, which ensures low correlation among decision trees. This is a key difference between decision trees and random forests. While decision trees consider all the possible feature splits, random forests only select a subset of those features.

How it works

Random forest algorithms have three main hyperparameters, which need to be set before training. These include node size, the number of trees, and the number of features sampled. From there, the random forest classifier can be used to solve for regression or classification problems.

The random forest algorithm is made up of a collection of decision trees, and each tree in the ensemble is comprised of a data sample drawn from a training set with replacement, called the bootstrap sample. Of that training sample, one-third of it is set aside as test data, known as the out-of-bag (oob) sample, which we’ll come back to later. Another instance of randomness is then injected through feature bagging, adding more diversity to the dataset and reducing the correlation among decision trees. Depending on the type of problem, the determination of the prediction will vary. For a regression task, the individual decision trees will be averaged, and for a classification task, a majority vote—i.e. the most frequent categorical variable—will yield the predicted class. Finally, the oob sample is then used for cross-validation, finalizing that prediction.



Dataset & Program:

Age	EstimatedSalary	Purchased
19	19000	0
35	20000	0
26	43000	0
27	57000	0
19	76000	0
27	58000	0
27	84000	0
32	150000	1
25	33000	0
--	--	--

```
# Random Forest Classification

# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Importing the dataset
dataset = pd.read_csv('Social_Network_Ads.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values

# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state
= 0)
print(X_train)
print(y_train)
print(X_test)
print(y_test)

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

# Training the Random Forest Classification model on the Training set
```

```
from sklearn.ensemble import RandomForestClassifier
classifier = RandomForestClassifier(n_estimators = 10, criterion = 'entropy',
random_state = 0)
classifier.fit(X_train, y_train)

# Predicting a new result
print(classifier.predict(sc.transform([[30,87000]])))

# Predicting the Test set results
y_pred = classifier.predict(X_test)
print(np.concatenate((y_pred.reshape(len(y_pred),1),
y_test.reshape(len(y_test),1)),1))

# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy_score(y_test, y_pred)

# Visualising the Training set results
from matplotlib.colors import ListedColormap
X_set, y_set = sc.inverse_transform(X_train), y_train
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 10, stop = X_set[:,
0].max() + 10, step = 0.25),
                     np.arange(start = X_set[:, 1].min() - 1000, stop = X_set[:, 1].max() +
1000, step = 0.25))
plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(),
X2.ravel()])).T)).reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1], c = ListedColormap(('red',
'green'))(i), label = j)
plt.title('Random Forest Classification (Training set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()

# Visualising the Test set results
from matplotlib.colors import ListedColormap
X_set, y_set = sc.inverse_transform(X_test), y_test
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 10, stop = X_set[:,
0].max() + 10, step = 0.25),
                     np.arange(start = X_set[:, 1].min() - 1000, stop = X_set[:, 1].max() +
```

```
1000, step = 0.25))
plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(),
X2.ravel()]).T)).reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1], c = ListedColormap(('red',
'green'))(i), label = j)
plt.title('Random Forest Classification (Test set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

Random Forest Intuition: