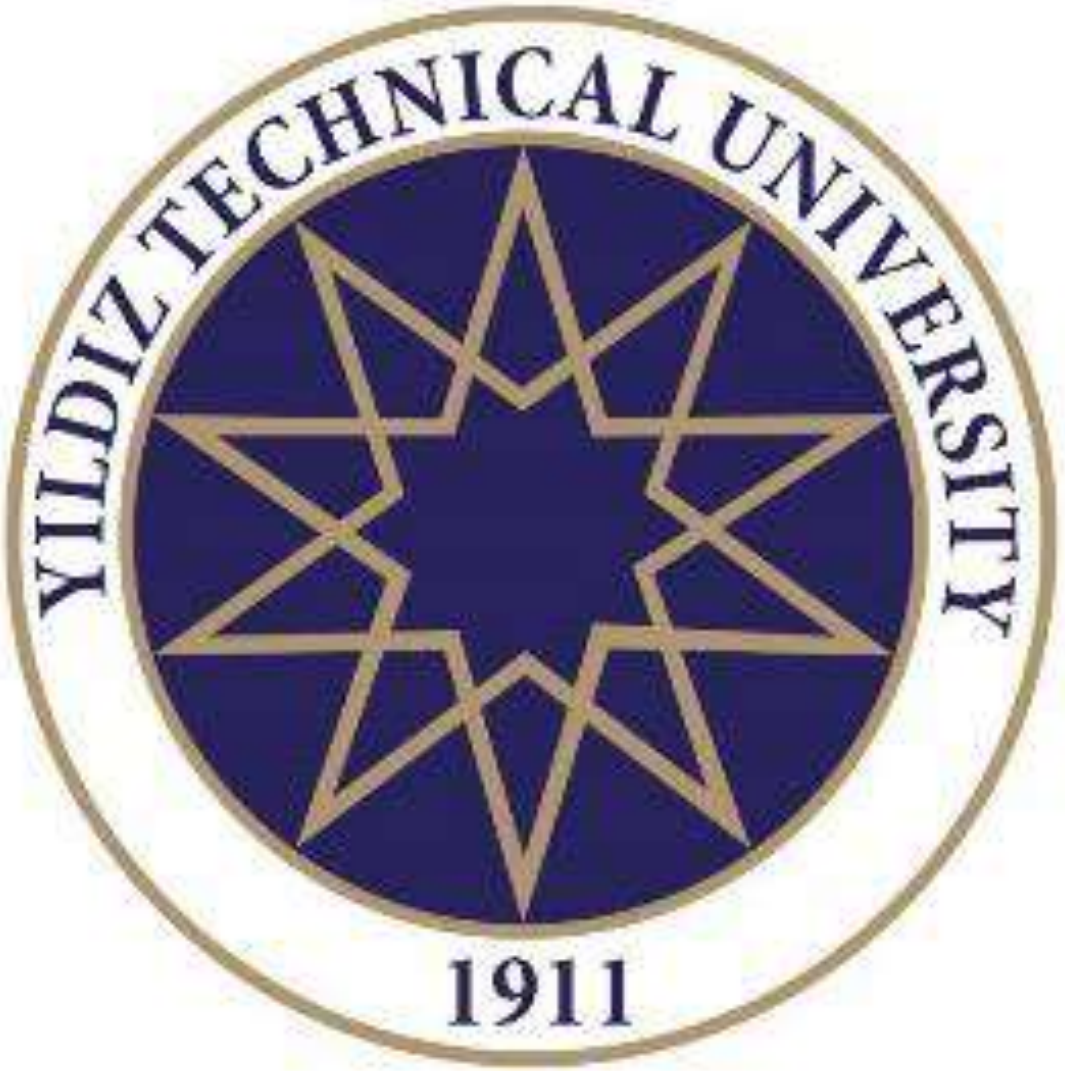



**YILDIZ TECHNICAL UNIVERSITY**
**FACULTY OF MECHANICAL ENGINEERING**
**DEPARTMENT OF INDUSTRIAL ENGINEERING**

# END4991 INDUSTRIAL ENGINEERING DESIGN II

## FINAL REPORT

- **İlyas Uğur Vural 19061069**
- **Özlem Yüksel Bilir 19061070**
- **Nihal Diler 19069606**
- **Özgür Gümüş 19061062**
- **Melis Kamacıoğlu 18061034**

# Table of contents

# Introduction

| | Age | Sex | BP | Cholesterol | Na_to_K | Drug |
|---|---|---|---|---|---|---|
| 0 | 23 | F | HIGH | HIGH | 25.355 | drugY |
| 1 | 47 | M | LOW | HIGH | 13.093 | drugC |
| 2 | 47 | M | LOW | HIGH | 10.114 | drugC |
| 3 | 28 | F | NORMAL | HIGH | 7.798 | drugX |
| 4 | 61 | F | LOW | HIGH | 18.043 | drugY |

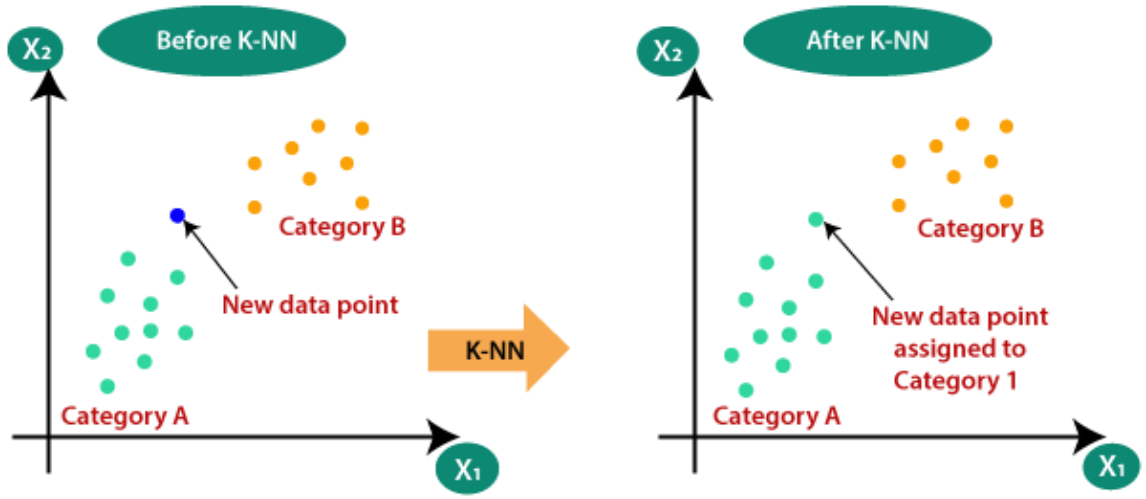
- **Dataset:** The dataset with 200 rows contains information that effects drug type such as Age, Sex, BP, Cholesterol levels, Na to Potassium Ratio. Age shows the age of the patient, sex shows the gender. BP is for blood pressure and it has three values; high, normal and low. Na to potassiom rate is the rate of Sodium to Potassiom in patient's blood.

# KNN

- **Description:** KNN stands for "k-nearest neighbors". It is a type of supervised machine learning algorithm used for classification and regression. Given a new observation, it finds the k-number of training examples that are closest to it in feature space, and then it assigns the class label based on the majority class among those k-nearest examples. It's a simple yet powerful algorithm that can be used for both classification and regression problems.

# KNN

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:



# Implementation of KNN in Python

**Collect and preprocess the data:** This step involves collecting the data that will be used to train and test the algorithm. The data may need to be cleaned and preprocessed to ensure that it is in a format that can be used by the algorithm.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import os
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split,  cros
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import cross_val_score
from sklearn.metrics import roc_curve
from sklearn.metrics import auc
```

```
df = pd.read_csv("drug200.csv")
df.head()
```

| | Age | Sex | BP | Cholesterol | Na_to_K | Drug |
|---|---|---|---|---|---|---|
| **0** | 23 | F | HIGH | HIGH | 25.355 | DrugY |
| **1** | 47 | M | LOW | HIGH | 13.093 | drugC |
| **2** | 47 | M | LOW | HIGH | 10.114 | drugC |
| **3** | 28 | F | NORMAL | HIGH | 7.798 | drugX |
| **4** | 61 | F | LOW | HIGH | 18.043 | DrugY |

**Adding** Age, Sex, BP, Cholesterol and Na_to_K columns to X and Drug column to Y

```
X = df[["Age","Sex","BP","Cholesterol","Na_to_K"]]
X.head()
```

| | Age | Sex | BP | Cholesterol | Na_to_K |
|---|---|---|---|---|---|
| **0** | 23 | F | HIGH | HIGH | 25.355 |
| **1** | 47 | M | LOW | HIGH | 13.093 |
| **2** | 47 | M | LOW | HIGH | 10.114 |
| **3** | 28 | F | NORMAL | HIGH | 7.798 |
| **4** | 61 | F | LOW | HIGH | 18.043 |

```
y = df['Drug']
y.head()
```

```
0    DrugY
1    drugC
2    drugC
3    drugX
4    DrugY
Name: Drug, dtype: object
```

**Split the data into training and test sets:** The data is typically split into two sets, one for training and one for testing. The training set is used to train the algorithm, while the test set is used to evaluate the performance of the trained model. Then StandardScaler is used to standardize the features of a dataset by removing the mean and scaling to unit variance.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0, test_size = 0.3)
```

```
X_train = pd.get_dummies(X_train)
X_test = pd.get_dummies(X_test)
X_train.head()
```

| | Age | Na_to_K | Sex_F | Sex_M | BP_HIGH | BP_LOW | BP_NORMAL | Cholesterol_HIGH | Cholesterol_NORMAL |
|---|---|---|---|---|---|---|---|---|---|
| **131** | 52 | 32.922 | 0 | 1 | 0 | 1 | 0 | 0 | 1 |
| **96** | 58 | 38.247 | 1 | 0 | 0 | 1 | 0 | 1 | 0 |
| **181** | 59 | 13.884 | 1 | 0 | 0 | 0 | 1 | 1 | 0 |
| **19** | 32 | 25.974 | 1 | 0 | 1 | 0 | 0 | 0 | 1 |
| **153** | 72 | 14.642 | 1 | 0 | 0 | 1 | 0 | 0 | 1 |

```
sc=StandardScaler()

X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

**Define the distance metric:** The distance metric is a function that is used to calculate the distance between two observations. Common distance metrics include Euclidean distance, Manhattan distance and Minkowski distance. After trying this 3 distances it has been seen that manhattan is the best distance metric out of them.

```
knn = KNeighborsClassifier(n_neighbors=30, metric='manhattan')
knn.fit(X_train,y_train)
```

```
KNeighborsClassifier(metric='manhattan', n_neighbors=30)
```

```
print(knn.score(X_train, y_train),knn.score(X_test, y_test))
```

```
0.8571428571428571 0.85
```

```
y_pred = knn.predict(X_test)
```

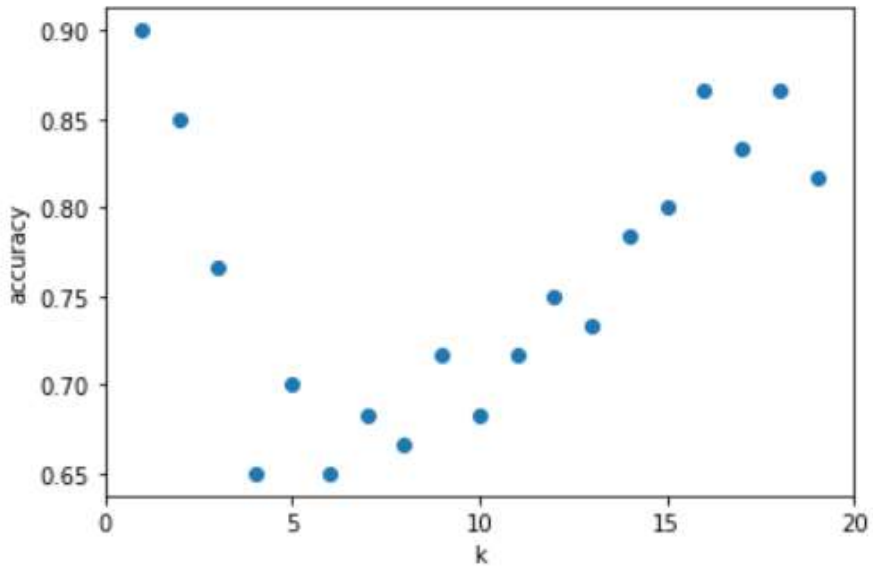
```
### Use the trained k-NN classifier model to classify new, previously unseen objects
knn_prediction = knn.predict([[52, 32.922, 0, 1, 0, 1, 0, 0, 1]])

knn_prediction[0]
```

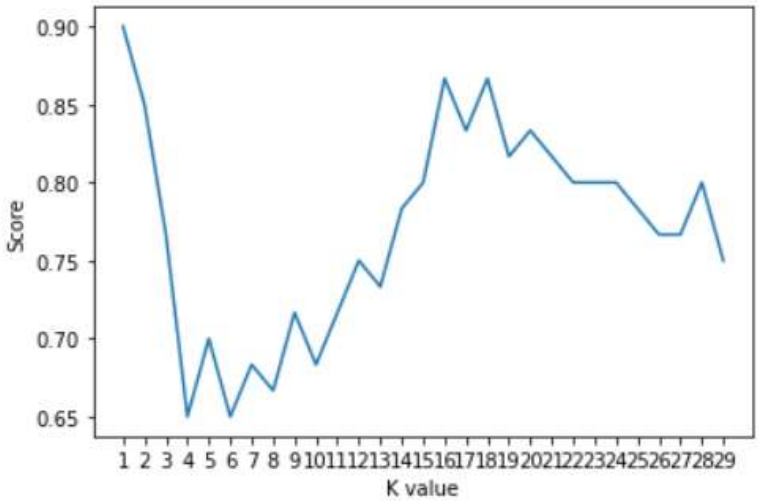
```
'DrugY'
```

# Results

The sensitivity of k-NN classification accuracy to the choice of the 'k' parameter value is determined and displayed in the graph below




The max KNN Acccuracy on every ‘k’ value is determined and displayed in the graph below.




KNN Acc Max 90.00%

# Confusion Matrix

- The confusion matrix is used to show the number of correct predictions and incorrect predictions made by the classifier. It is typically represented as a table, with the true class labels on one axis and the predicted class labels on the other axis.
- A confusion matrix allows to understand how well the classifier is doing by providing a clear picture of how many observations are being correctly classified and how many are not. It also helps to identify common mistakes made by the classifier.

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

```
              precision    recall  f1-score   support

       DrugY       0.75      0.80      0.77        30
       drugA       1.00      0.80      0.89         5
       drugB       0.50      0.33      0.40         3
       drugC       1.00      1.00      1.00         4
       drugX       0.72      0.72      0.72        18

    accuracy                           0.77        60
   macro avg       0.79      0.73      0.76        60
weighted avg       0.77      0.77      0.76        60

[[24  0  1  0  5]
 [ 1  4  0  0  0]
 [ 2  0  1  0  0]
 [ 0  0  0  4  0]
 [ 5  0  0  0 13]]
```

# ROC AUC

- ROC AUC (Receiver Operating Characteristic - Area Under the Curve) is a performance evaluation metric for binary classification problems. It is implemented for measure of the trade-off between the true positive rate (sensitivity) and the false positive rate (1-specificity) of a classifier.

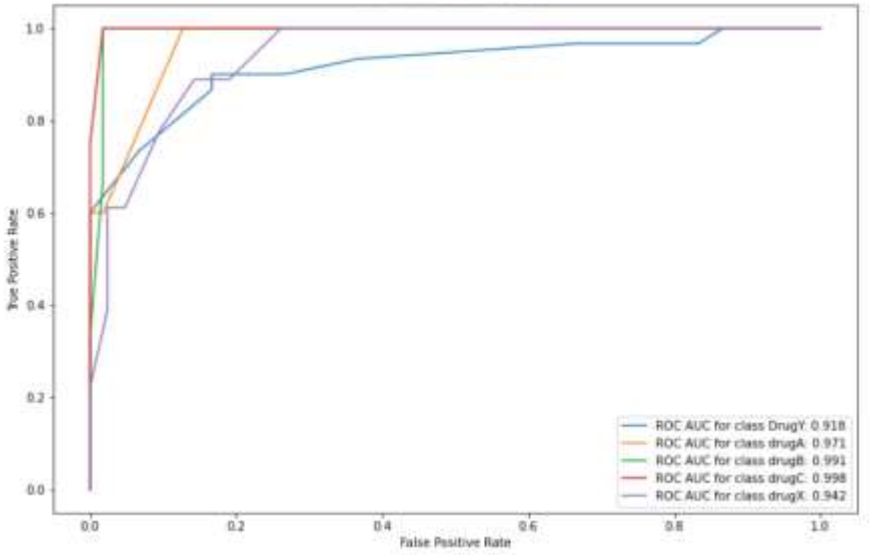
```
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve, auc
y_probs = knn.predict_proba(X_test)
roc_auc = roc_auc_score(y_test, y_probs, multi_class='ovr')
print("Overall ROC AUC: {:.3f}".format(roc_auc))
```

```
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc, roc_auc_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelBinarizer

lb = LabelBinarizer()
y_test_bin = lb.fit_transform(y_test)
```

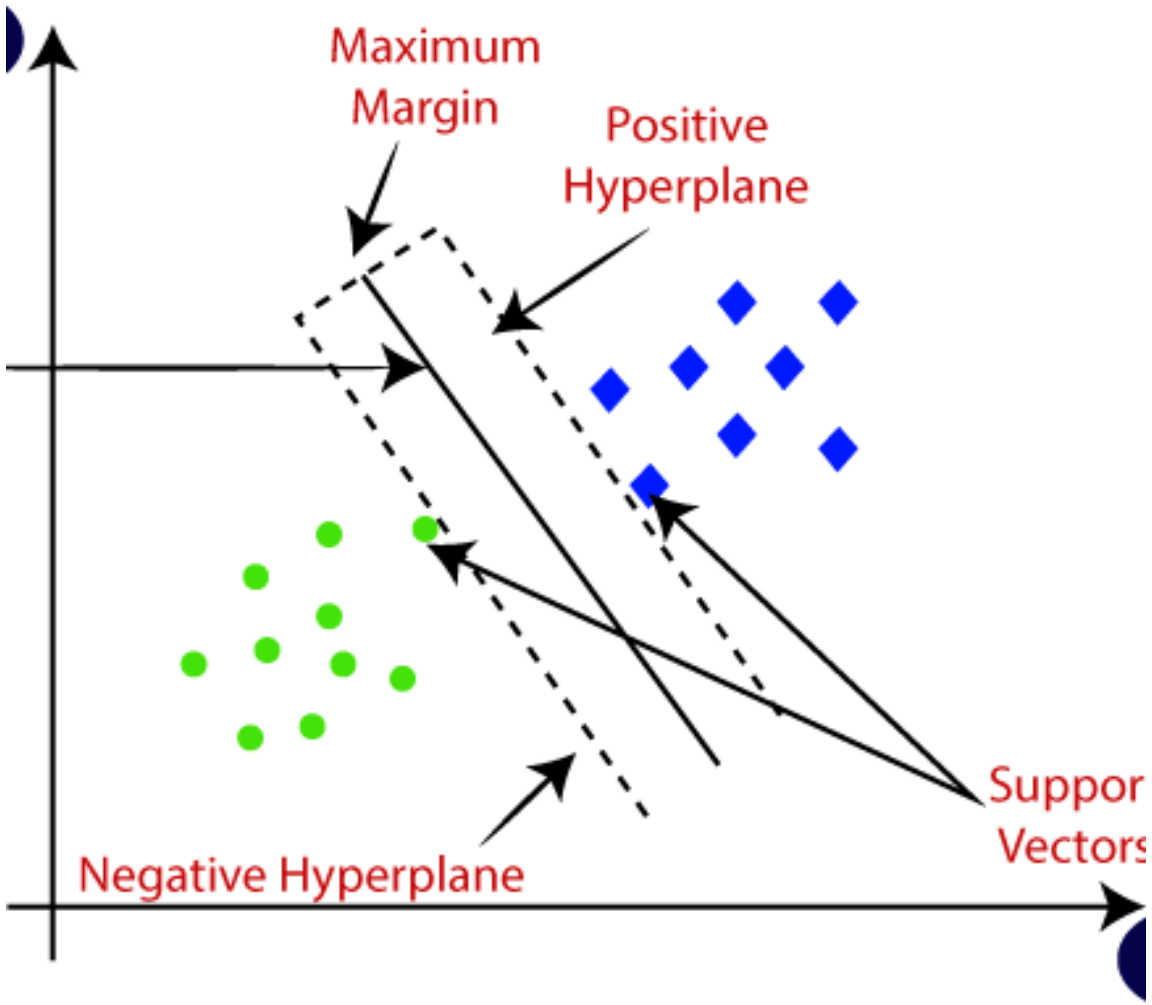
```
y_test_labels = lb.classes_
```

```
plt.figure(figsize=(12,8))
for i in range(y_test_bin.shape[1]):
    fpr, tpr, _ = roc_curve(y_test_bin[:, i], y_probs[:, i])
    roc_auc = auc(fpr, tpr)
    plt.plot(fpr, tpr, label='ROC AUC for class {}: {:.3f}'.format(y_test_labels[i], roc_auc))
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Multiclass KNN ROC AUC')
plt.legend()
plt.show()
```



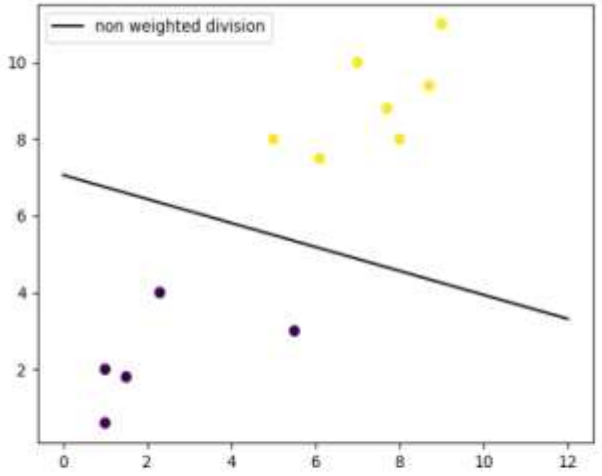
# SVM

**Description:** SVM stands for Support Vector Machine, it is a type of supervised learning algorithm that can be used for classification or regression tasks. The main idea behind SVM is to find a hyperplane (a line or a plane in high-dimensional space) that separates different classes of data points with the largest margin possible. This hyperplane is called the decision boundary. SVM algorithm is useful in high dimensional spaces and also effective in cases where the number of dimensions is greater than the number of samples.



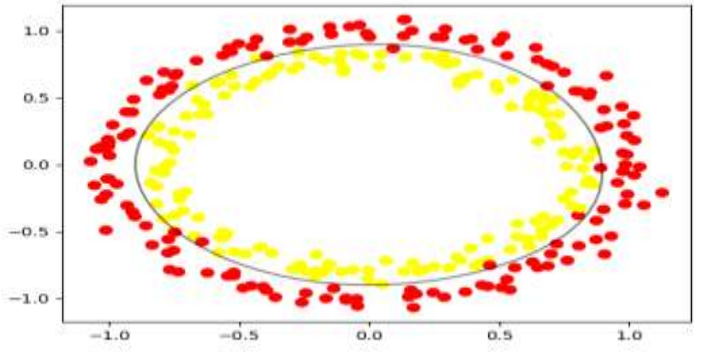
# There are two main types of SVM:

**Linear SVM**: Linear SVM is used for linear classification problems, where the data can be separated by a straight line or a plane. A linear SVM finds the hyperplane that separates the different classes of data points with the largest margin.




linear SVM

**Non-Linear SVM:** Non-Linear SVM is used for non-linear classification problems, where the data cannot be separated by a straight line or a plane. Non-Linear SVM uses a technique called the kernel trick, which maps the input data into a higher-dimensional space where it becomes linearly separable. Some examples of Non-linear kernel functions include polynomial, radial basis function (RBF) and sigmoid.




non-linear SVM using RBF kernel

Benefits of SVM:

1. SVM can handle high dimensional data effectively, even when the number of dimensions is greater than the number of samples.
2. SVM can be used for both classification and regression tasks.
3. SVM can effectively handle non-linear decision boundaries using kernel tricks.
4. SVM is less prone to overfitting compared to some other algorithms.

# Implementation of SVM

- SVM model is set with kernel = linear and calculate precision, accuracy, recall, f1-score and support values.

In [255]:

```
from sklearn.svm import SVC
SVCclassifier1= SVC(kernel='linear', max_iter=251, probability=True)
SVCclassifier1.fit(X_train, y_train)

y_pred = SVCclassifier1.predict(X_test)

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

from sklearn.metrics import accuracy_score
SVCAcc = accuracy_score(y_pred,y_test)
print('SVC accuracy is: {:.2f}%'.format(SVCAcc*100))
```