G H Raisoni College of Engineering
SY AI Semester-IV AY 2023-24 Division-A
UCAIP210: Machine Learning Algorithms Practicals
Lab Manual
Practical Teacher : Dr Monika Y. Dangore

Experiment No: 8

Aim:

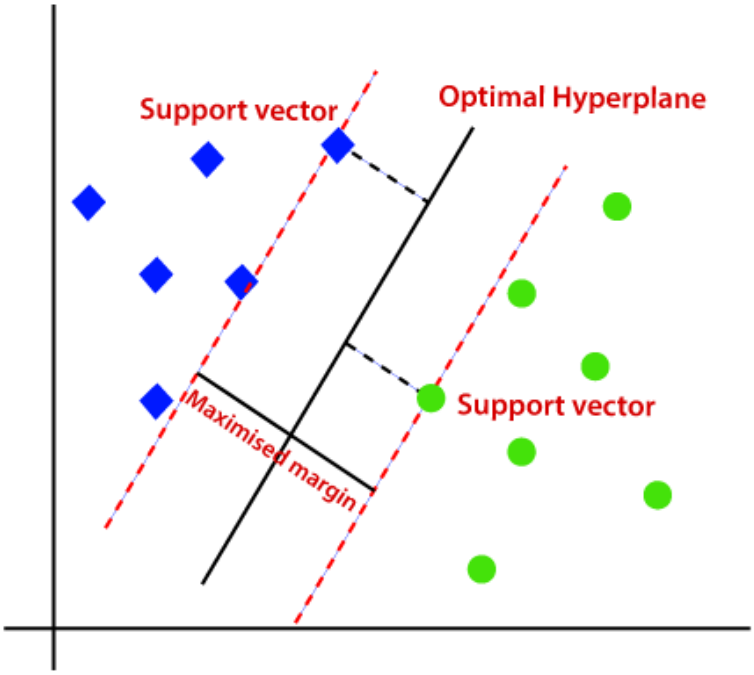
Implement Support Vector Machine (SVM) Algorithm

Introduction to Support Vector Machine Algorithm:

Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems. However, primarily, it is used for Classification Problems in Machine Learning.

The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future. This best decision boundary is called a hyperplane.

SVM chooses the extreme points/vectors that help in creating the hyperplane. These extreme cases are called as support vectors, and hence algorithm is termed as Support Vector Machine. Consider the below diagram in which there are two different categories that are classified using a decision boundary or hyperplane:

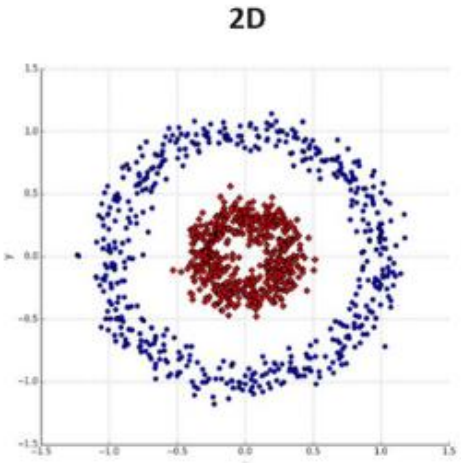


SVM algorithm can be used for **Face detection, image classification, text categorization,** etc.

Types of SVM

SVM can be of two types:

- **Linear SVM:** Linear SVM is used for linearly separable data, which means if a dataset can be classified into two classes by using a single straight line, then such data is termed as linearly separable data, and classifier is used called as Linear SVM classifier. Figure shown above is the example of linear data.
- **Non-linear SVM:** Non-Linear SVM is used for non-linearly separated data, which means if a dataset cannot be classified by using a straight line, then such data is termed as non-linear data and classifier used is called as Non-linear SVM classifier. Example of non linear data is shown below:



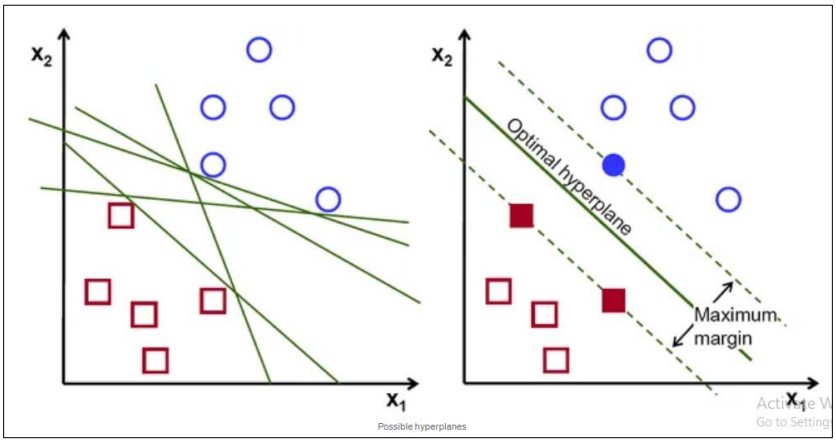
Hyperplane and Support Vectors in the SVM algorithm:

Hyperplane: There can be multiple lines/decision boundaries to segregate the classes in n-dimensional space, but we need to find out the best decision boundary that helps to classify the data points. This best boundary is known as the hyperplane of SVM.

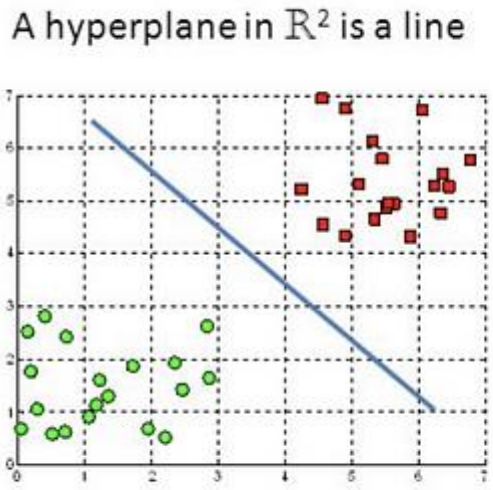
We always create a hyperplane that has a maximum margin, which means the maximum distance between the data points.

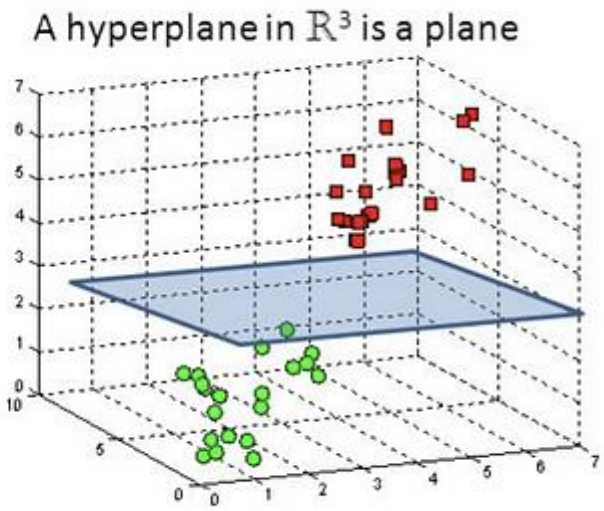
The data points or vectors that are the closest to the hyperplane and which affect the position of the hyperplane are termed as Support Vector. Since these vectors support the hyperplane, hence called a Support vector. So as it is 2-d space so by just using a straight line, we can easily

separate these two classes. But there can be multiple lines that can separate these classes. Consider the below image:



Hyperplanes are decision boundaries that help classify the data points. Data points falling on either side of the hyperplane can be attributed to different classes. Also, the dimension of the hyperplane depends upon the number of features. If the number of input features is 2, then the hyperplane is just a line. If the number of input features is 3, then the hyperplane becomes a two-dimensional plane. It becomes difficult to imagine when the number of features exceeds 3.





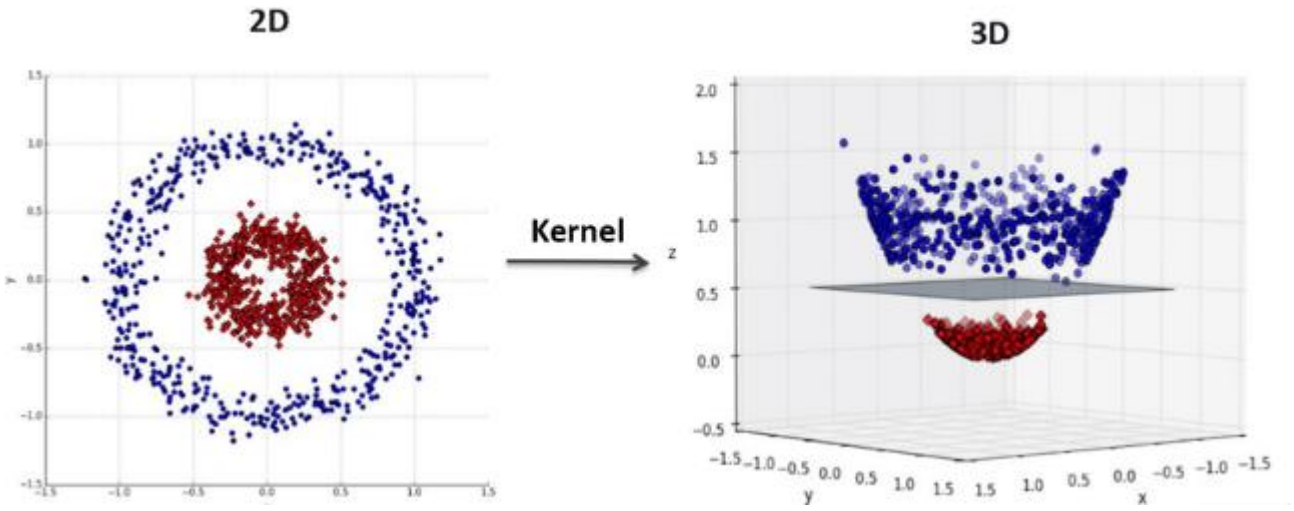
Support Vector Machine Terminology

1. **Hyperplane:** Hyperplane is the decision boundary that is used to separate the data points of different classes in a feature space. In the case of linear classifications, it will be a linear equation i.e. $wx+b = 0$.
2. **Support Vectors:** Support vectors are the closest data points to the hyperplane, which makes a critical role in deciding the hyperplane and margin.
3. **Margin**: Margin is the distance between the support vector and hyperplane. The main objective of the support vector machine algorithm is to maximize the margin. The wider margin indicates better classification performance.
4. **Kernel**: Kernel is the mathematical function, which is used in SVM to map the original input data points into high-dimensional feature spaces, so, that the hyperplane can be easily found out even if the data points are not linearly separable in the original input space. Some of the common kernel functions are linear, polynomial, radial basis function(RBF), and sigmoid.
5. **Hard Margin:** The maximum-margin hyperplane or the hard margin hyperplane is a hyperplane that properly separates the data points of different categories without any misclassifications.
6. **Soft Margin:** When the data is not perfectly separable or contains outliers, SVM permits a soft margin technique. Each data point has a slack (loose / relaxed) variable introduced by the soft-margin SVM formulation, which softens the strict margin requirement and permits certain misclassifications or violations. It discovers a compromise between increasing the margin and reducing violations.
7. **C:** Margin maximisation and misclassification fines are balanced by the regularisation parameter C in SVM. The penalty for going over the margin or misclassifying data items is decided by it. A stricter penalty is imposed with a greater value of C, which results in a smaller margin and perhaps fewer misclassifications.

Kernel Functions in SVM:

SVM utilizes kernel functions to map the input data points into a higher-dimensional space where the separation between the two classes becomes easier.

The most widely used kernels in SVM are the linear kernel, polynomial kernel, and Gaussian (radial basis function) kernel. The choice of kernel relies on the nature of the data and the job at hand. The linear kernel is used when the data is roughly linearly separable, whereas the polynomial kernel is used when the data has a complicated curved border. The Gaussian kernel is employed when the data has no clear boundaries and contains complicated areas of overlap. We can see this in the plot below, where the red and blue data points have been separated by a hyperplane in the 3D space:



As we can see, the kernel trick has helped us find a solution for a non-linearly separable dataset.

The Iris Flower Dataset:

Objective:

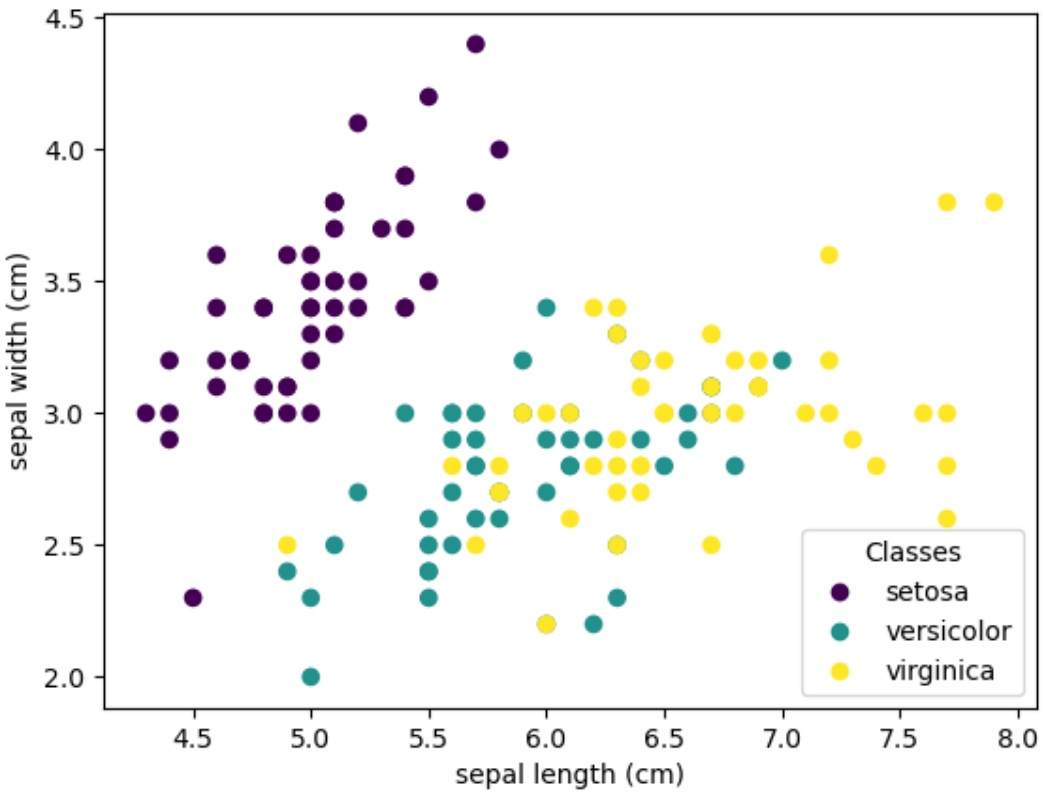
Classify the Iris flower into its correct species (Iris Versicolor, Iris Setosa, Iris Virginica) based on the sepal length, sepal width, petal length and petal width.

This data sets consists of 3 different types of irises' (Setosa, Versicolour, and Virginica), the rows being the samples and the columns being: Sepal Length, Sepal Width, Petal Length and Petal Width and variety (species).

The below plot uses the first two features.

```
import matplotlib.pyplot as plt


_, ax = plt.subplots()
scatter = ax.scatter(iris.data[:, 0], iris.data[:, 1], c=iris.target)
ax.set(xlabel=iris.feature_names[0], ylabel=iris.feature_names[1])
_ = ax.legend(
    scatter.legend_elements()[0], iris.target_names, loc="lower right", title="Classes"
)
```



Each point in the scatter plot refers to one of the 150 iris flowers in the dataset, with the color indicating their respective type (Setosa, Versicolour, and Virginica). You can already see a pattern regarding the Setosa type, which is easily identifiable based on its short and wide sepal. Only considering these 2 dimensions, sepal width and length, there's still overlap between the Versicolor and Virginica types

The figures above show that Iris setosa can be easily distinguished from other Iris flower species (thus, very easily classified) but Iris versicolor and Iris virginica are quite similar in terms of petal length , petal width, sepal length and sepal width. The objective is to classify the Iris flower based on the sepal length, sepal width petal length and petal width into correct category.

Program:

Attach the printouts of the program.

Result:

The concept Support Vector Machine is studied and program is executed successfully.

Following Content can be skipped in the Write-up:

iloc() in Python:

The iloc() function in Python is a method provided by the pandas library, which is widely used for data analysis and manipulation. It stands for "integer location" and is primarily used for accessing and retrieving data from pandas DataFrame objects using integer-based indexing regardless of the labels or names associated with the rows and columns.

The syntax of the iloc function in Python is as follows:

```
df.iloc[row_start:row_end, column_start:column_end]
```

In this syntax, “df” is the DataFrame that we want to select data from. The “row_start” and “row_end” arguments specify the starting and ending positions of the rows that we want to select. The “column_start” and “column_end” arguments specify the starting and ending positions of the columns that we want to select.

Here’s an overview of the different parameters of the iloc function:

- **row_start:** This argument specifies the integer position of the starting row for the selection. If this parameter is not specified, it defaults to 0, which is the first row of the DataFrame.
- **row_end:** This argument specifies the integer position of the ending row for the selection. If this parameter is not specified, it defaults to the last row of the DataFrame.

- **column_start:** This argument specifies the integer position of the starting column for the selection. If this parameter is not specified, it defaults to 0, which is the first column of the DataFrame.
- **column_end:** This argument specifies the integer position of the ending column for the selection. If this parameter is not specified, it defaults to the last column of the DataFrame.

Note that the row_end and column_end parameters are non-inclusive, meaning that the final row or column specified in the range is not included in the selection.

Return value of iloc() Function in Python

The iloc function in Python returns a view of the selected rows and columns from a Pandas DataFrame. This view can be used to access, modify, or delete the selected data.

One trick that works on Python lists is negative indexing. For example, the second to last item in a list could be found at index -2. The same goes for the second to last row of a DataFrame using the .iloc

scikit-learn's train_test_split() Function

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25)
```

The train_test_split() method is used to split our data into train and test sets. First, we need to divide our data into features (X) and labels (y).

The dataframe gets divided into X_train, X_test, y_train, and y_test. X_train and y_train sets are used for training and fitting the model.

The X_test and y_test sets are used for testing the model whether it is predicting the right outputs/labels or not. We can explicitly test the size of the train and test sets. It is suggested to keep our train sets larger than the test sets.

accuracy_score function in Sklearn:

In Python, the **accuracy_score function** of the sklearn.metrics package calculates the accuracy score for a set of predicted labels against the true labels (actual values in the dataset).

The accuracy_score function accepts the following parameters:

- **y_test or y_true**: These are the true labels.
- **y_pred**: These are the predicted labels.

This function returns either the fraction of the correct predictions or the number of correct predictions