Lab#13

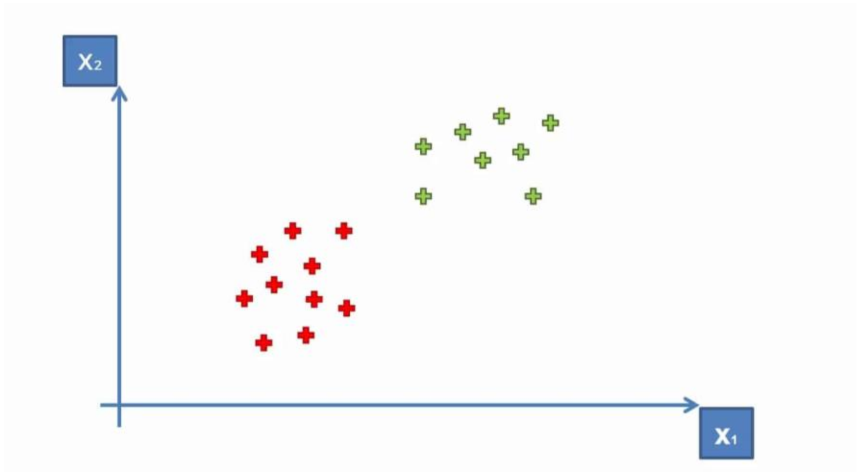
<u>Title:</u> Write a python program to develop Support Vector Machine method and visually manipulate the results.

Support Vector Machine?

SVM was developed in the 1960s and refined in the 1990s. It becomes very popular in the machine learning field because SVM is very powerful compared to other algorithms. SVM (Support Vector Machine) is a supervised machine learning algorithm. That's why training data is available to train the model. SVM uses a classification algorithm to classify a two-group problem. SVM focus on decision boundary and support vectors.

How SVM Works?

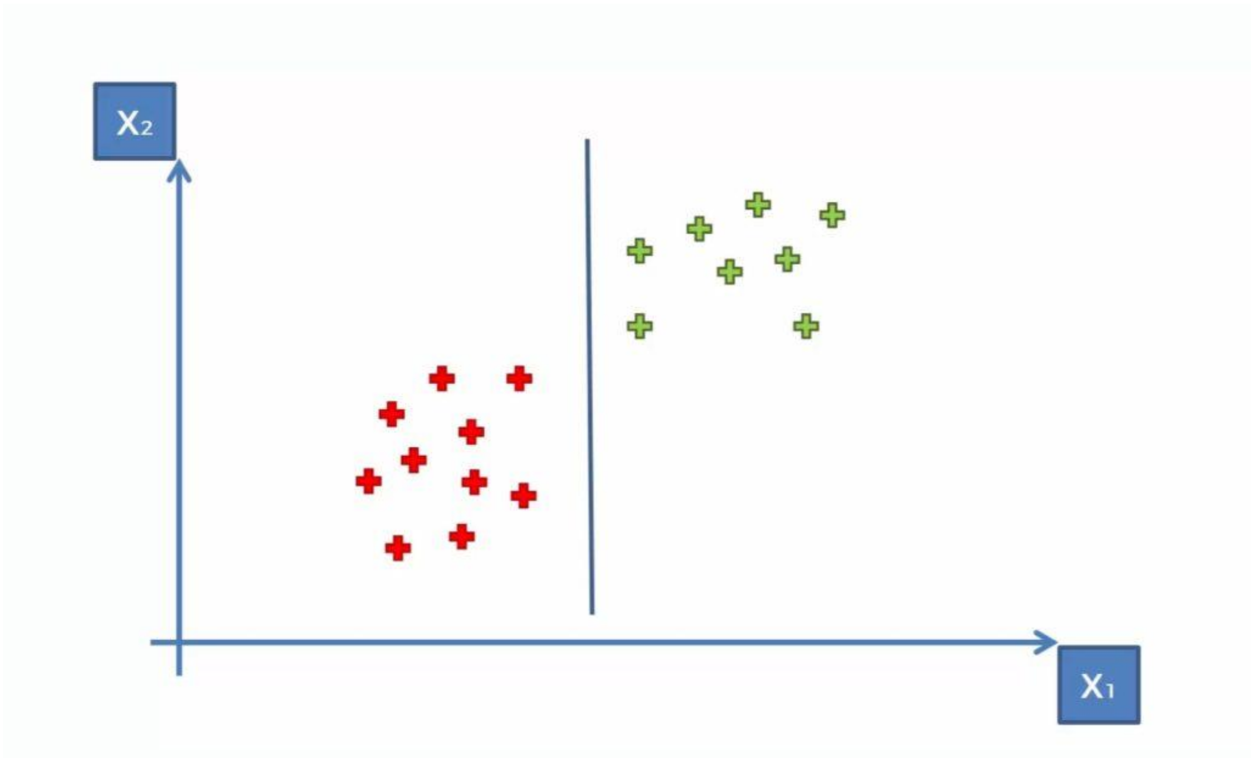
Here, we have two points in two-dimensional space, we have two columns x1 and x2. And we have some observations such as **red and green**, which are already classified. This is linearly separable data.



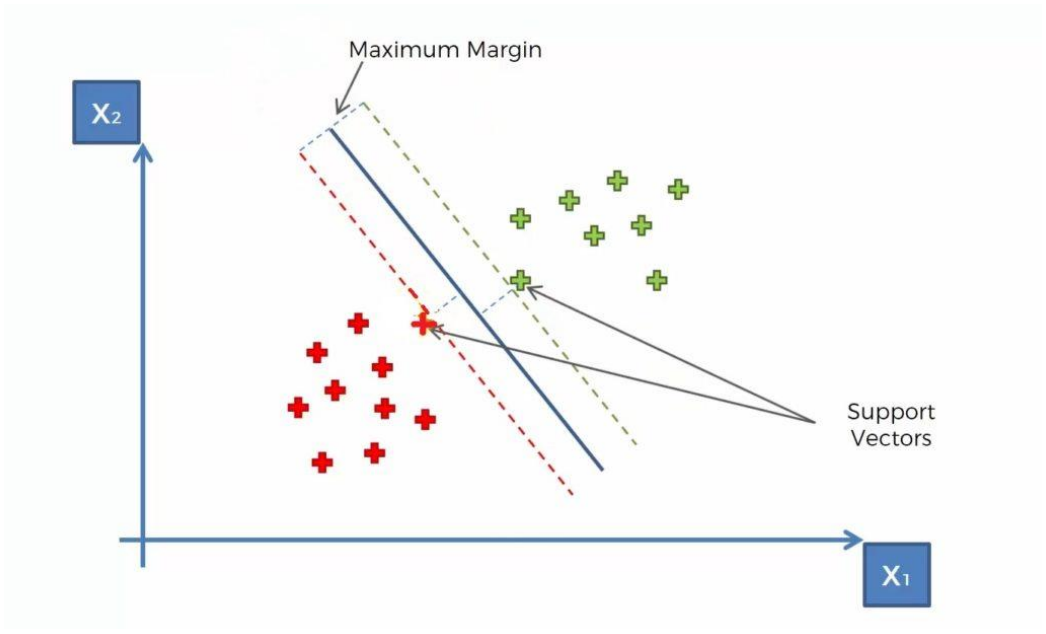
But, now how do we derive a line that separates these points? This means a separation or decision boundary is very important for us when we add new points.
So to classify new points, we need to create a boundary between two categories, and when in the future we will add new points and we want to classify them, then we know where they belong. Either in a Green Area or Red Area.

So how can we separate these points?

One way is to draw a vertical line between two areas, so anything on the right is Red and anything on the left is Green. Something like that-



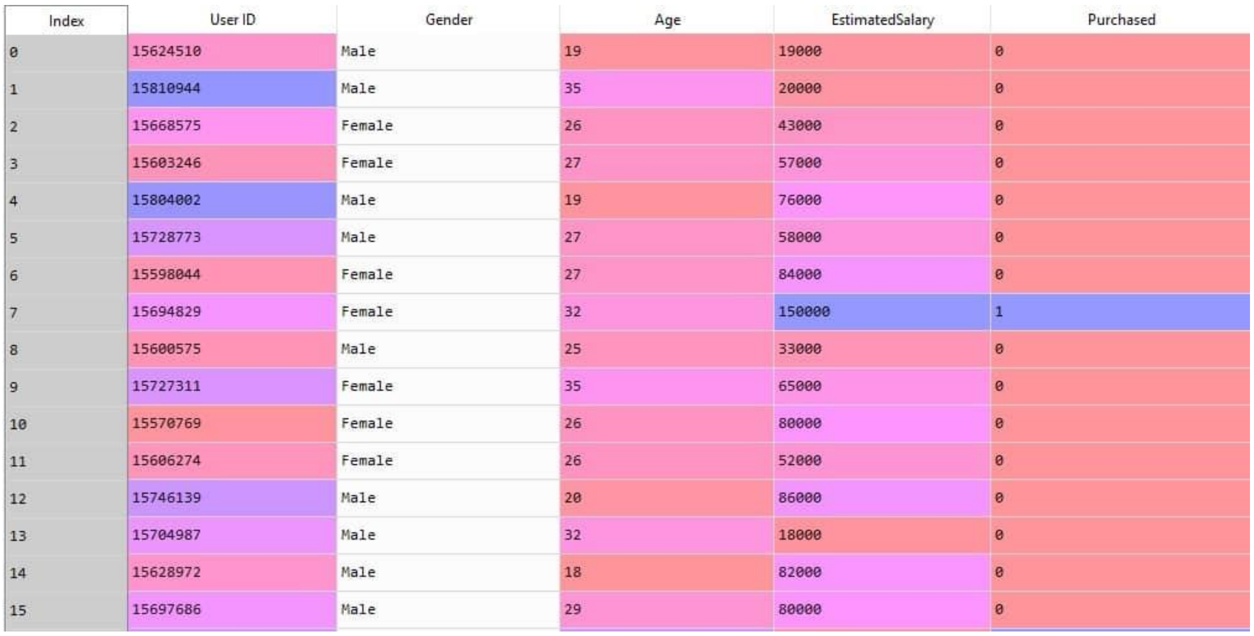
However, there is one more way, draw a horizontal line or diagonal line. You can create multiple diagonal lines, which achieve similar results to separate our points into two classes. But our main task is to find the optimal line or best decision boundary. And for this SVM is used. SVM finds the best decision boundary, which helps us to separate points into different spaces. SVM finds the best or optimal line through the maximum margin, which means it has max distance and equidistance from both classes or spaces. The sum of these two classes has to be maximized to make this line the maximum margin.



These, two vectors are support vectors. In SVM, only support vectors are contributing. That's why these points or vectors are known as support vectors. Due to support vectors, this algorithm is called a Support Vector Algorithm(SVM). In the above figure, the line in the middle is a maximum margin hyperplane or classifier. In a two-dimensional plane, it looks like a line, but in a multi-dimensional, it is a hyperplane.

SVM Implementation in Python

For implementation, I am gonna use Social Network Ads Dataset. You can download the dataset from Kaggle. This dataset has two independent variables customer age and salary and one dependent variable whether the customer purchased SUVs or not. 1 means purchase the SUV and 0 means not purchase the SUV. And we have to train the SVM model with this dataset and after training, our model has to classify whether a customer purchased the SUV or not based on the customer's age and salary.



Index	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0
5	15728773	Male	27	58000	0
6	15598044	Female	27	84000	0
7	15694829	Female	32	150000	1
8	15600575	Male	25	33000	0
9	15727311	Female	35	65000	0
10	15570769	Female	26	80000	0
11	15606274	Female	26	52000	0
12	15746139	Male	20	86000	0
13	15704987	Male	32	18000	0
14	15628972	Male	18	82000	0
15	15697686	Male	29	80000	0

The first step is **Data Pre-processing** but before data pre-processing, we need to **import the libraries**.

1. Import the Libraries-

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

2. Load the Dataset

```
dataset = pd.read_csv('Social_Network_Ads.csv')
```

3. Split Dataset into X and Y

```
X = dataset.iloc[:, [2, 3]].values
y = dataset.iloc[:, 4].values
```

When you run these lines, you get two separate tables X and Y.

4. Split the X and Y Dataset into the Training set and Test set

For building a machine learning model, we need to train our model on the training set. And for checking the performance of our model, we use a Test set. That’s why we have to split the X and Y datasets 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)
```

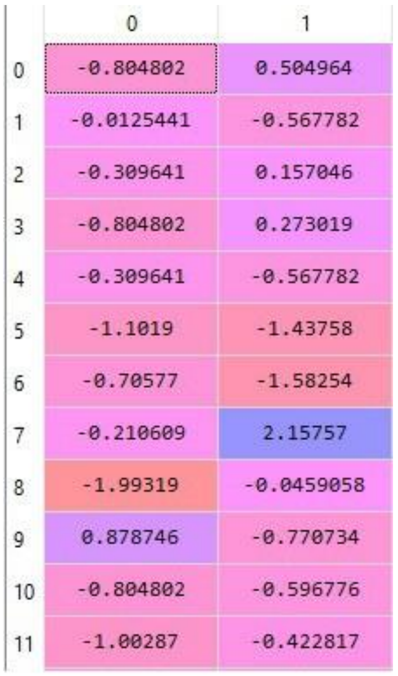
While splitting into training and test set, you have to remember that, 80%-90% of your data should be in the training tests. And that’s why I write test_size = 0.25.
Now we have split our dataset into X_train, X_test, y-train, and y_test. The next step is-

5. Perform Feature Scaling

As you can see in the dataset, all values are not in the same range. And that requires a lot of time for calculation. So to overcome this problem, we perform feature scaling.
Feature scaling helps us to normalize the data within a particular range.

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_tes
```

After performing feature scaling, all values are normalized and looks something like this-



	0	1
0	-0.804802	0.504964
1	-0.0125441	-0.567782
2	-0.309641	0.157046
3	-0.804802	0.273019
4	-0.309641	-0.567782
5	-1.1019	-1.43758
6	-0.70577	-1.58254
7	-0.210609	2.15757
8	-1.99319	-0.0459058
9	0.878746	-0.770734
10	-0.804802	-0.596776
11	-1.00287	-0.422817

Now, we are done with the data preprocessing steps. It's time to fit SVM into the training set.

5. Fit SVM to the Training set

```
from sklearn.svm import SVC
classifier = SVC(kernel = 'rbf', random_state = 0)
classifier.fit(X_train, y_train)
```

This **SVC class** allows us to build a **kernel SVM model (linear as well as non-linear)**, The default value of the kernel is **'rbf'**. Why 'rbf', because it is nonlinear and gives better results as compared to linear.

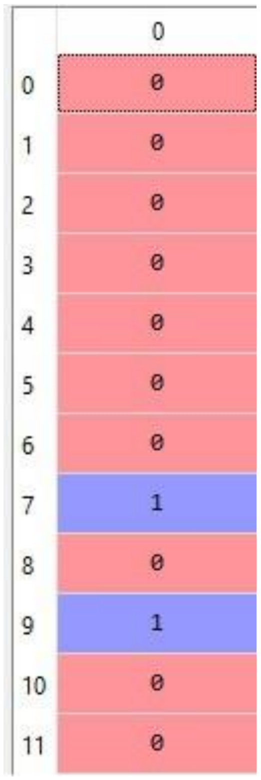
The classifier.fit(X_train, y_train) fits the SVM algorithm to the training set- X_train and y_train.

Now, all done. It's time to predict the Test set. So the next step is-

6. Predict the Test Set Results

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

When you run this line of code, you will get y_pred, something like this-



	0
0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	1
8	0
9	1
10	0
11	0

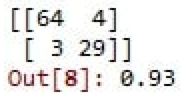
But can you explain by looking at these predicted values, how many values are predicted right, and how many values are predicted wrong?

For a small dataset, you can. But when we have a large dataset, it’s quite impossible. And that’s why we use a confusion matrix, to clear our confusion.

Make 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)
```

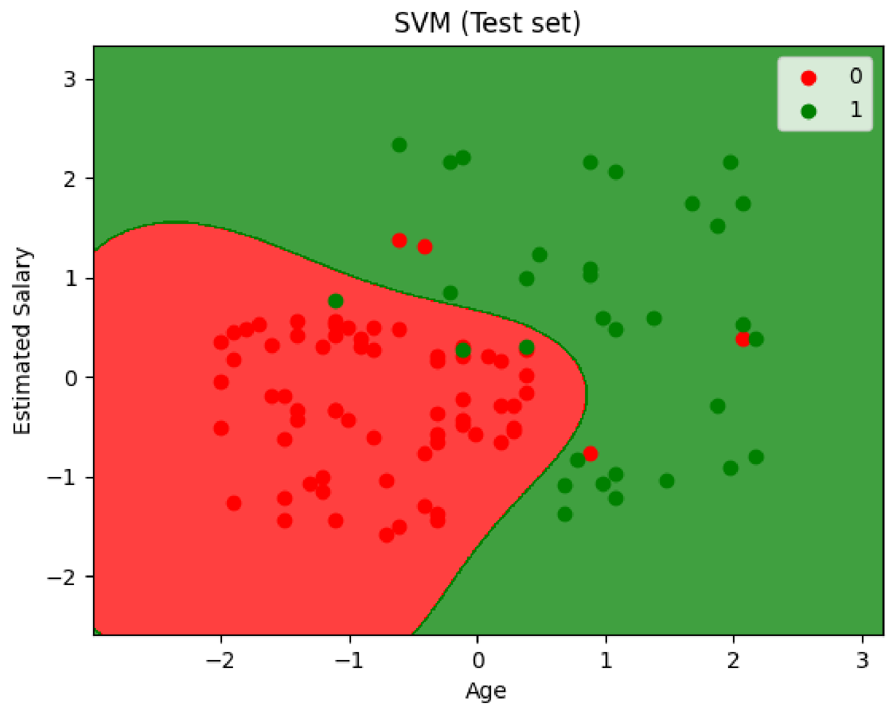
And we got 93% accuracy.



```
[[64  4]
 [ 3 29]]
Out[8]: 0.93
```

8. Visualize the Test set results

```
from matplotlib.colors import ListedColormap
X_set, y_set = X_test, y_test
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop =
X_set[:, 0].max() + 1, step = 0.01),
                     np.arange(start = X_set[:, 1].min() - 1, stop =
X_set[:, 1].max() + 1, step = 0.01))
plt.contourf(X1, X2, classifier.predict(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('SVM (Test set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```



Lab task # 01:

Select a dataset suitable for a classification task (e.g., from scikit-learn datasets or external sources). Preprocess the dataset, handling missing values, encoding categorical variables, and splitting it into features and target variables. Implement an SVM model using Scikit-learn's SVC (Support Vector Classifier). Train the SVM model on the training dataset using the fit method. Visualize the decision boundary or separating hyperplane created by the SVM using suitable plotting libraries (e.g., Matplotlib). Plot the data points and display how the SVM separates different classes or groups.

Experiment with different hyperparameters of the SVM (e.g., kernel type, regularization parameter) to observe their effects on the decision boundary. Visualize how changing these parameters affects the SVM's decision boundary. Evaluate the performance of the SVM model on the testing dataset using appropriate evaluation metrics (e.g., accuracy, precision, recall, F1-score) from sklearn.metrics. Discuss and analyze the results obtained from the visualization and evaluation, focusing on the SVM's ability to separate classes.

Lab Task # 02:

Select a dataset suitable for a multi-class classification task (e.g., Iris dataset, digits dataset from scikit-learn). Preprocess the dataset, handle missing values, encode categorical variables (if any), and split it into features and target variables. Implement an SVM model using Scikit-learn's SVC (Support Vector Classifier) for multi-class classification. Train the SVM model on the training dataset using the fit method. Visualize the decision boundaries or separating hyperplanes created by the SVM for multiple classes. Plot the data points and display how the SVM separates different classes or groups using suitable visualization libraries (e.g., Matplotlib).

Analyze and interpret the multi-class separation achieved by the SVM. Experiment with different SVM hyperparameters (e.g., kernel type, regularization parameter) to observe their impact on multi-class classification. Evaluate the model's performance on the testing dataset using appropriate evaluation metrics (e.g., accuracy, precision, recall, F1-score) from sklearn.metrics. Compare the performance of the SVM model for multi-class classification with different hyperparameter settings. Summarize findings, insights gained, and observations about the SVM's ability to classify multiple classes effectively in a detailed report.