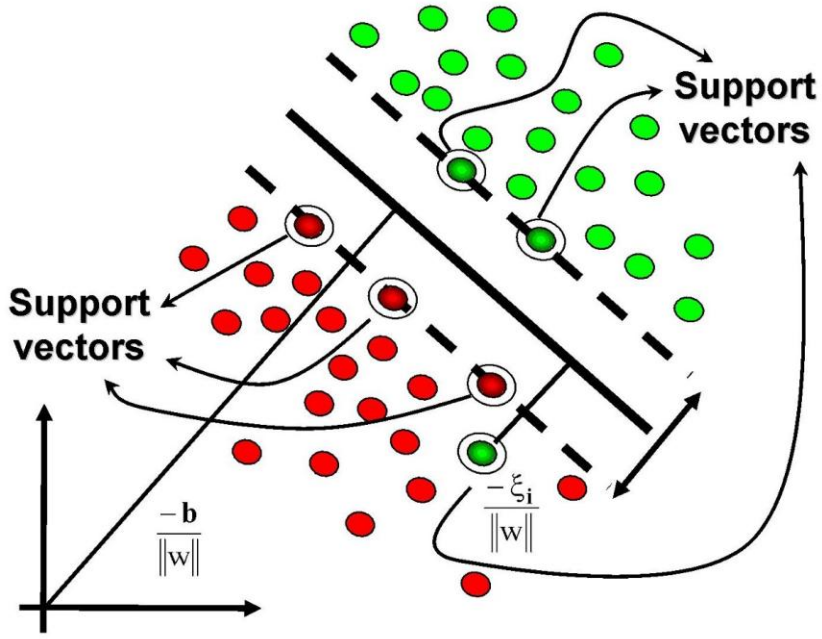
Experiment :

AIM : Write a program to implement Support Vector Machine Algorithm

THEORY :

"Support Vector Machine" (SVM) is a supervised machine learning algorithm which can be used for both classification or regression challenges. However, it is mostly used in classification problems. In this algorithm, we plot each data item as a point in n-dimensional space (where n is number of features you have) with the value of each feature being the value of a particular coordinate. Then, we perform classification by finding the hyper-plane that differentiate the two classes very well.



Support Vectors are simply the co-ordinates of individual observation. Support Vector Machine is a frontier which best segregates the two classes (hyper-plane/ line).

How do Support Vector Machine work?

Above, we got accustomed to the process of segregating the two classes with a hyper-plane. Now the burning question is "How can we identify the right hyper-plane?".

We do it by Tuning parameters: Kernel, Regularization, Gamma and Margin.

Kernel

The learning of the hyperplane in linear SVM is done by transforming the problem using some linear algebra. This is where the kernel plays role.

For **linear kernel** the equation for prediction for a new input using the dot product between the input (x) and each support vector (xi) is calculated as follows:

f(x) = B(0) + sum(ai * (x,xi))

This is an equation that involves calculating the inner products of a new input vector (x) with all support vectors in training data. The coefficients B0 and ai (for each input) must be estimated from the training data by the learning algorithm.

The **polynomial kernel** can be written as *K(x,xi) = 1 + sum(x * xi)^d* and

The **exponential kernel** as *K(x,xi) = exp(-gamma * sum((x—xi²))*.

The **RBF kernel** on two samples **x** and **x'**, represented as feature vectors in some *input space*, is defined as

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

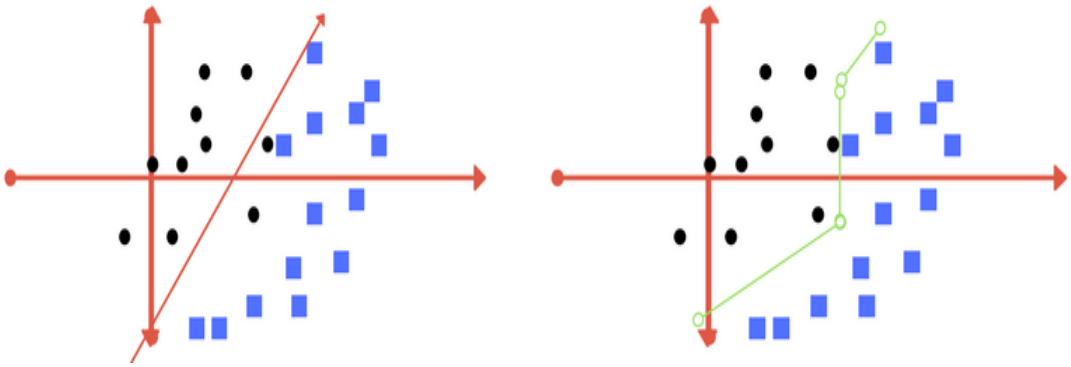
Polynomial and exponential kernels calculates separation line in higher dimension. This is called **kernel trick**

Regularization

The Regularization parameter (often termed as C parameter in python's sklearn library) tells the SVM optimization how much you want to avoid misclassifying each training example.

For large values of C, the optimization will choose a smaller-margin hyperplane if that hyperplane does a better job of getting all the training points classified correctly. Conversely, a very small value of C will cause the optimizer to look for a larger-margin separating hyperplane, even if that hyperplane misclassifies more points.

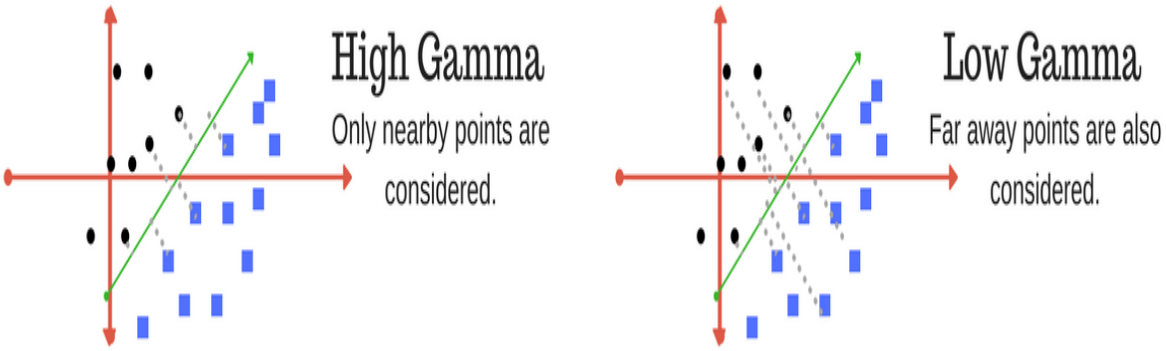
The images below (same as image 1 and image 2 in section 2) are example of two different regularization parameter. Left one has some misclassification due to lower regularization value. Higher value leads to results like right one.



Left: low regularization value, right: high regularization value

Gamma

The gamma parameter defines how far the influence of a single training example reaches, with low values meaning ‘far’ and high values meaning ‘close’. In other words, with low gamma, points far away from plausible seperation line are considered in calculation for the seperation line. Where as high gamma means the points close to plausible line are considered in calculation.

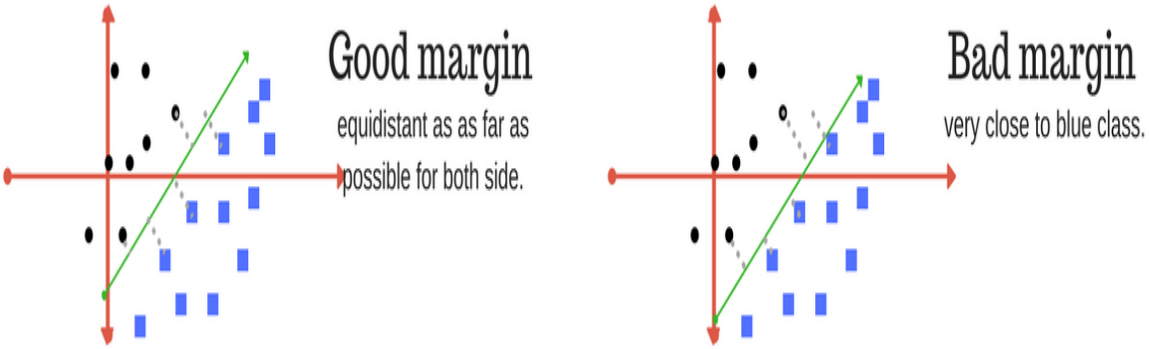


Margin

And finally last but very importrant characteristic of SVM classifier. SVM to core tries to achieve a good margin.

A margin is a separation of line to the closest class points.

A **good margin** is one where this separation is larger for both the classes. Images below gives to visual example of good and bad margin. A good margin allows the points to be in their respective classes without crossing to other class.



ALGORITHM :

The equation for making a prediction for a new input using the dot product between the input (x) and each support vector (xi) is calculated as follows:

f(x) = B0 + sum(ai * (x,xi))

This is an equation that involves calculating the inner products of a new input vector (x) with all support vectors in training data. The coefficients B0 and ai (for each input) must be estimated from the training data by the learning algorithm.

Linear Kernel SVM

The dot-product is called the kernel and can be re-written as:

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

The kernel defines the similarity or a distance measure between new data and the support vectors. The dot product is the similarity measure used for linear SVM or a linear kernel because the distance is a linear combination of the inputs.

Other kernels can be used that transform the input space into higher dimensions such as a Polynomial Kernel and a Radial Kernel. This is called the Kernel Trick.

It is desirable to use more complex kernels as it allows lines to separate the classes that are curved or even more complex. This in turn can lead to more accurate classifiers.

Polynomial Kernel SVM

Instead of the dot-product, we can use a polynomial kernel, for example:

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

where the degree of the polynomial must be specified by hand to the learning algorithm. When d=1, this is the same as the linear kernel. The polynomial kernel allows for curved lines in the input space.

Radial Kernel SVM

Finally, we can also have a more complex radial kernel. For example:

K(x,xi) = exp(-gamma * sum((x – xi2))

Where gamma is a parameter that must be specified to the learning algorithm. A good default value for gamma is 0.1, where gamma is often 0 < gamma < 1. The radial kernel is very local and can create complex regions within the feature space, like closed polygons in two-dimensional space.

CODE :

```
# Importing the libraries

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd


# Importing the dataset

dataset = pd.read_csv('sales.csv')

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

y = dataset.iloc[:, 4].values


# Splitting the dataset into the Training set and Test set

from sklearn.cross_validation import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)


# Feature Scaling

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

X_train = sc.fit_transform(X_train)

X_test = sc.transform(X_test)


# Fitting SVM to the Training set

from sklearn.svm import SVC

classifier = SVC(kernel = 'rbf', random_state = 0)

classifier.fit(X_train, y_train)
```

```
# Predicting the Test set results
y_pred = classifier.predict(X_test)

# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
#Printing the classification report
from sklearn.metrics import classification_report,accuracy_score
print(classification_report(y_test, y_pred))
print("Accuracy is " ,accuracy_score(y_test,y_pred)*100)
# Visualising the Training set results
from matplotlib.colors import ListedColormap
X_set, y_set = X_train, y_train
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(('yellow', 'brown')))
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(('yellow', 'brown'))(i), label = j)
plt.title('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 = 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(('yellow', 'brown')))
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(('yellow', 'brown'))(i), label = j)
plt.title('SVM (Test set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

OUTPUT :

User ID	Gender	Age	EstimatedSalary	Purchased
15624510	Male	19	19000	0
15810944	Male	35	20000	0
15668575	Female	26	43000	0
15603246	Female	27	57000	0
15804002	Male	19	76000	0
15728773	Male	27	58000	0
15598044	Female	27	84000	0
15694829	Female	32	150000	1
15600575	Male	25	33000	0
15727311	Female	35	65000	0
15570769	Female	26	80000	0

CONFUSION MATRIX

	0	1
0	64	4
1	3	29

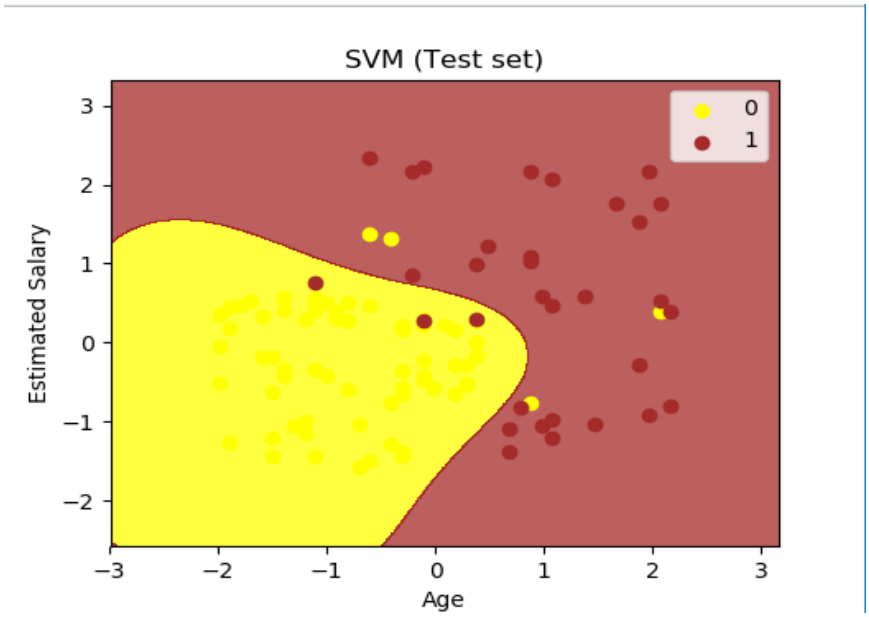
```
In [2]: from sklearn.metrics import classification_report
   ...: print(classification_report(y_test, y_pred))
             precision    recall  f1-score   support

          0       0.96      0.94      0.95        68
          1       0.88      0.91      0.89        32

avg / total       0.93      0.93      0.93       100
```

Accuracy is 93.0


SVM (Training set)
Estimated Salary
Age
0
1
3
2
1
0
−1
−2
−3
−2
−1
0
1
2
3


SVM (Test set)
Estimated Salary
Age
0
1
3
2
1
0
−1
−2
−3
−2
−1
0
1
2
3

DISCUSSION :

After implementing SVM we have come to conclude the following things,

1. It is more accurate that it's competitors.

2. It is more robust i.e. due to optimal margin gap between separating hyper planes, it could do pr edictions better with test data.

- **Pros:**
    - It works really well with clear margin of separation
    - It is effective in high dimensional spaces.
    - It is effective in cases where number of dimensions is greater than the number of samples.
    - It uses a subset of training points in the decision function (called support vectors), so it is also memory efficient.
- **Cons:**
    - It doesn’t perform well, when we have large data set because the required training time is higher
    - It also doesn’t perform very well, when the data set has more noise i.e. target classes are overlapping
    - SVM doesn’t directly provide probability estimates, these are calculated using an expensive five-fold cross-validation. It is related SVC method of Python scikit-learn library.

FINDING AND LEARNING :

In this experiment we implemented support vector machine algorithm(with rbf(radial basis function)kernel) using sklearn library in python on a sales dataset which classifies whether a person will by a certain product given his/her age and salary. And We got an accuracy of 93%.