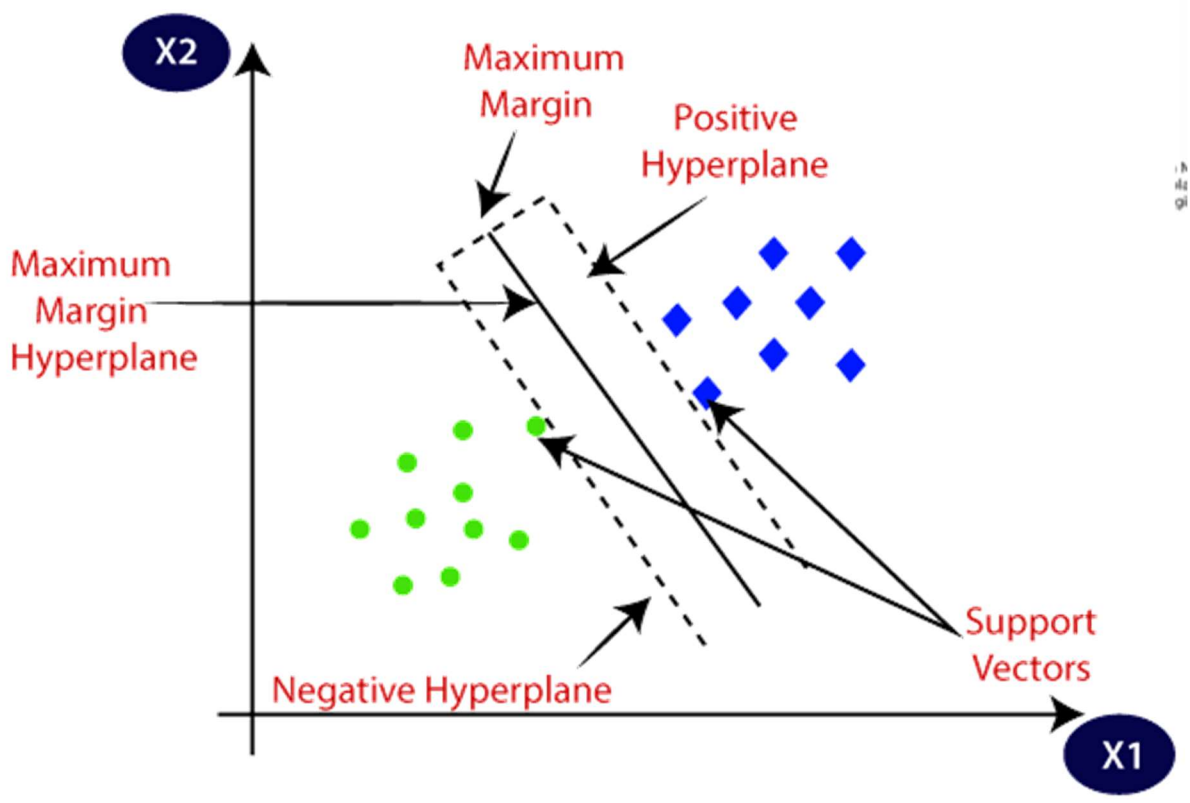
Unit-3

Support Vector Machines

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:



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.
- **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.

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.

Support Vectors:

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.

Maximal Margin Classifier

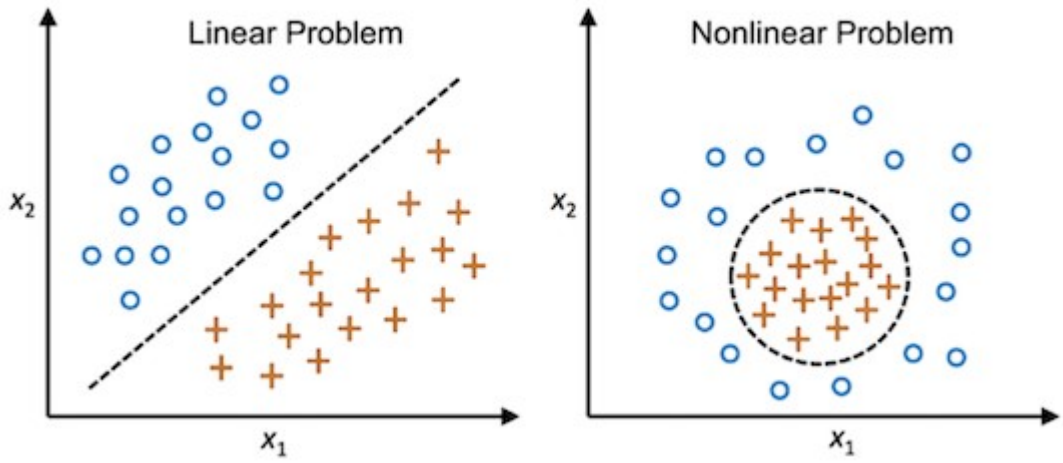
This classifier is designed specifically for linearly separable data, refers to the condition in which data can be separated linearly using a hyperplane.

Linearly separable and non-linearly separable data

Linear and non-linear separable data are described in the diagram below.

Linearly separable data is data that is populated in such a way that it can be

easily classified with a straight line or a hyperplane. Non-linearly separable data, on the other hand, is described as data that cannot be separated using a simple straight line (requires a complex classifier).



Based on the maximum margin, the Maximal-Margin Classifier chooses the optimal hyperplane. The dotted lines, parallel to the hyperplane in the following diagram are the **margins** and the distance between both these dotted lines (Margins) is the Maximum Margin.

Maximal Margin Separators

A Maximal Margin Separator (in a 2-dimensional space) is a hyperplane (in this case a line) that completely separates 2 classes of observations, while giving the most space between the line and the nearest observation. These nearest observations are the support vectors.

Introduction to Kernel Methods

Kernels or kernel methods (also called Kernel functions) are sets of different types of algorithms that are being used for pattern analysis. They are used to solve a non-linear problem by using a linear classifier. Kernels Methods are employed in SVM (Support Vector Machines) which are used in classification and regression problems. The SVM uses what is called a “Kernel Trick” where the data is transformed and an optimal boundary is found for the possible outputs.

SVM Kernels

In practice, SVM algorithm is implemented with kernel that transforms an input data space into the required form. SVM uses a technique called the kernel trick in which kernel takes a low dimensional input space and transforms it into a higher dimensional space. In simple words, kernel converts non-separable problems into separable problems by adding more dimensions to it. It makes SVM more powerful, flexible and accurate. The following are some of the types of kernels used by SVM.

Linear Kernel

It can be used as a dot product between any two observations. The formula of linear kernel is as below −

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

From the above formula, we can see that the product between two vectors say x & xi is the sum of the multiplication of each pair of input values.

Polynomial Kernel

It is more generalized form of linear kernel and distinguish curved or nonlinear input space. Following is the formula for polynomial kernel −

$$k(X,Xi)=1+sum(X*X_i)\text{^}dk(X,Xi)=1+sum(X*Xi)\text{^}d$$

Here d is the degree of polynomial, which we need to specify manually in the learning algorithm.

Radial Basis Function (RBF) Kernel

RBF kernel, mostly used in SVM classification, maps input space in indefinite dimensional space. Following formula explains it mathematically −

$$K(x,xi)=exp(−gamma∗sum(x−xi^2))K(x,xi)=exp(−gamma∗sum(x−xi^2))$$

Here, *gamma* ranges from 0 to 1. We need to manually specify it in the learning algorithm. A good default value of *gamma* is 0.1.

As we implemented SVM for linearly separable data, we can implement it in Python for the data that is not linearly separable. It can be done by using kernels

In [16]:
```
#Data Pre-processing Step
# importing libraries
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd

import warnings
warnings.filterwarnings("ignore")

#importing datasets
data_set= pd.read_csv('user_data.csv')

#Extracting Independent and dependent Variable
x= data_set.iloc[:, [2,3]].values
y= data_set.iloc[:, 4].values
```

In [17]:
```
x
```

Out[17]:
```
array([[    19,  19000],
       [    35,  20000],
       [    26,  43000],
       [    27,  57000],
       [    19,  76000],
       [    27,  58000],
       [    27,  84000],
       [    32, 150000],
       [    25,  33000],
       [    35,  65000],
       [    26,  80000],
       [    26,  52000],
       [    20,  86000],
       [    32,  18000],
       [    18,  82000],
       [    29,  80000],
       [    47,  25000],
       [    45,  26000],
       [    46,  28000],
```

In [18]:
```
y
```

Out[18]:
```
array([0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1,
       0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0,
       1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0,
       1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1,
       0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1,
       1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1,
       0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0,
       1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1,
       0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1,
       1, 1, 0, 1], dtype=int64)
```

In [19]:
```
# Splitting the dataset into training 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)
#feature Scaling
from sklearn.preprocessing import StandardScaler
st_x= StandardScaler()
x_train= st_x.fit_transform(x_train)
x_test= st_x.transform(x_test)
```

In [20]:
```
x_train
```

Out[20]:
```
array([[ 0.58164944, -0.88670699],
       [-0.60673761,  1.46173768],
       [-0.01254409, -0.5677824 ],
       [-0.60673761,  1.89663484],
       [ 1.37390747, -1.40858358],
       [ 1.47293972,  0.99784738],
       [ 0.08648817, -0.79972756],
       [-0.01254409, -0.24885782],
       [-0.21060859, -0.5677824 ],
       [-0.21060859, -0.19087153],
       [-0.30964085, -1.29261101],
       [-0.30964085, -0.5677824 ],
       [ 0.38358493,  0.09905991],
       [ 0.8787462 , -0.59677555],
       [ 2.06713324, -1.17663843],
       [ 1.07681071, -0.13288524],
       [ 0.68068169,  1.78066227],
       [-0.70576986,  0.56295021],
       [ 0.77971394,  0.35999821],
```

In [21]:
```
x_test
```

Out[21]:
```
array([[-0.80480212,  0.50496393],
       [-0.01254409, -0.5677824 ],
       [-0.30964085,  0.1570462 ],
       [-0.80480212,  0.27301877],
       [-0.30964085, -0.5677824 ],
       [-1.10189888, -1.43757673],
       [-0.70576986, -1.58254245],
       [-0.21060859,  2.15757314],
       [-1.99318916, -0.04590581],
       [ 0.8787462 , -0.77073441],
       [-0.80480212, -0.59677555],
       [-1.00286662, -0.42281668],
       [-0.11157634, -0.42281668],
       [ 0.08648817,  0.21503249],
       [-1.79512465,  0.47597078],
       [-0.60673761,  1.37475825],
       [-0.11157634,  0.21503249],
       [-1.89415691,  0.44697764],
       [ 1.67100423,  1.75166912],
       [-0.30964085, -1.37959044],
       [-0.30964085, -0.65476184],
       [ 0.8787462 ,  2.15757314],
       [ 0.28455268, -0.53878926],
       [ 0.8787462 ,  1.02684052],
       [-1.49802789, -1.20563157],
       [ 1.07681071,  2.07059371],
       [-1.00286662,  0.50496393],
       [-0.90383437,  0.30201192],
       [-0.11157634, -0.21986468],
       [-0.60673761,  0.47597078],
       [-1.6960924 ,  0.53395707],
       [-0.11157634,  0.27301877],
       [ 1.86906873, -0.27785096],
       [-0.11157634, -0.48080297],
       [-1.39899564, -0.33583725],
       [-1.99318916, -0.50979612],
       [-1.59706014,  0.33100506],
       [-0.4086731 , -0.77073441],
       [-0.70576986, -1.03167271],
       [ 1.07681071, -0.97368642],
       [-1.10189888,  0.53395707],
       [ 0.28455268, -0.50979612],
       [-1.10189888,  0.41798449],
       [-0.30964085, -1.43757673],
       [ 0.48261718,  1.22979253],
       [-1.10189888, -0.33583725],
       [-0.11157634,  0.30201192],
       [ 1.37390747,  0.59194336],
       [-1.20093113, -1.14764529],
       [ 1.07681071,  0.47597078],
       [ 1.86906873,  1.51972397],
       [-0.4086731 , -1.29261101],
       [-0.30964085, -0.3648304 ],
       [-0.4086731 ,  1.31677196],
       [ 2.06713324,  0.53395707],
       [ 0.68068169, -1.089659  ],
       [-0.90383437,  0.38899135],
       [-1.20093113,  0.30201192],
       [ 1.07681071, -1.20563157],
       [-1.49802789, -1.43757673],
       [-0.60673761, -1.49556302],
       [ 2.1661655 , -0.79972756],
       [-1.89415691,  0.18603934],
       [-0.21060859,  0.85288166],
```

```
[-1.89415691, -1.26361786],
[ 2.1661655 ,  0.38899135],
[-1.39899564,  0.56295021],
[-1.10189888, -0.33583725],
[ 0.18552042, -0.65476184],
[ 0.38358493,  0.01208048],
[-0.60673761,  2.331532  ],
[-0.30964085,  0.21503249],
[-1.59706014, -0.19087153],
[ 0.68068169, -1.37959044],
[-1.10189888,  0.56295021],
[-1.99318916,  0.35999821],
[ 0.38358493,  0.27301877],
[ 0.18552042, -0.27785096],
[ 1.47293972, -1.03167271],
[ 0.8787462 ,  1.08482681],
[ 1.96810099,  2.15757314],
[ 2.06713324,  0.38899135],
[-1.39899564, -0.42281668],
[-1.20093113, -1.00267957],
[ 1.96810099, -0.91570013],
[ 0.38358493,  0.30201192],
[ 0.18552042,  0.1570462 ],
[ 2.06713324,  1.75166912],
[ 0.77971394, -0.8287207 ],
[ 0.28455268, -0.27785096],
[ 0.38358493, -0.16187839],
[-0.11157634,  2.21555943],
[-1.49802789, -0.62576869],
[-1.29996338, -1.06066585],
[-1.39899564,  0.41798449],
[-1.10189888,  0.76590222],
[-1.49802789, -0.19087153],
[ 0.97777845, -1.06066585],
[ 0.97777845,  0.59194336],
[ 0.38358493,  0.99784738]])
```

Fitting the SVM classifier to the training set:

Now the training set will be fitted to the SVM classifier. To create the SVM classifier, we will import SVC class from Sklearn.svm library. Below is the code for it:

In [22]:
```
from sklearn.svm import SVC # "Support vector classifier"
classifier = SVC(kernel='linear', random_state=0)
classifier.fit(x_train, y_train)
```

Out[22]:
```
SVC(kernel='linear', random_state=0)
```

In the above code, we have used kernel='linear', as here we are creating SVM for linearly separable data. However, we can change it for non-linear data. And then we fitted the classifier to the training dataset(x_train, y_train)

The model performance can be altered by changing the value of C(Regularization factor), gamma, and kernel.

Predicting the test set result: Now, we will predict the output for test set. For this, we will create a new vector y_pred. Below is the code for it:

In [23]:
```
#Predicting the test set result
y_pred= classifier.predict(x_test)
```

In [24]:
```
y_pred
```

Out[24]:
```
array([0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1,
       0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
       1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1,
       0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1], dtype=int64)
```

Creating the confusion matrix: Now we will see the performance of the SVM classifier that how many incorrect predictions are there as compared to the Logistic regression classifier. To create the confusion matrix, we need to import the confusion_matrix function of the sklearn library. After importing the function, we will call it using a new variable cm. The function takes two parameters, mainly y_true(the actual values) and y_pred (the targeted value return by the classifier). Below is the code for it:

In [25]:
```
#Creating the Confusion matrix
from sklearn.metrics import confusion_matrix
cm= confusion_matrix(y_test, y_pred)
cm
```

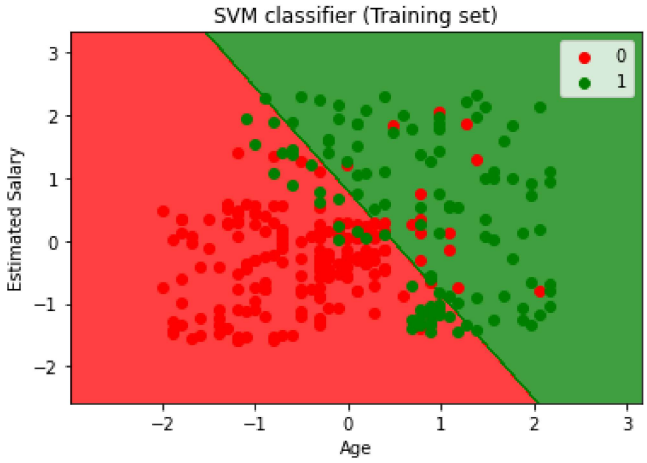
Out[25]:
```
array([[66,  2],
       [ 8, 24]], dtype=int64)
```

As we can see in the above output image, there are 66+24= 90 correct predictions and 8+2= 10 correct predictions. Therefore we can say that our SVM model improved as compared to the Logistic regression model.

Visualizing the training set result: Now we will visualize the training set result, below is the code for it:

In [28]:
```
from matplotlib.colors import ListedColormap
x_set, y_set = x_train, y_train
x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].max() + 1, step
nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))
mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape),
alpha = 0.75, cmap = ListedColormap(('red', 'green')))
mtp.xlim(x1.min(), x1.max())
mtp.ylim(x2.min(), x2.max())
for i, j in enumerate(nm.unique(y_set)):
    mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
        c = ListedColormap(('red', 'green'))(i), label = j)
mtp.title('SVM classifier (Training set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()
```

```
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-
mapping will have precedence in case its length matches with *x* & *y*.  Please use the *color*
keyword-argument or provide a 2D array with a single row if you intend to specify the same RGB o
r RGBA value for all points.
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-
mapping will have precedence in case its length matches with *x* & *y*.  Please use the *color*
keyword-argument or provide a 2D array with a single row if you intend to specify the same RGB o
r RGBA value for all points.
```

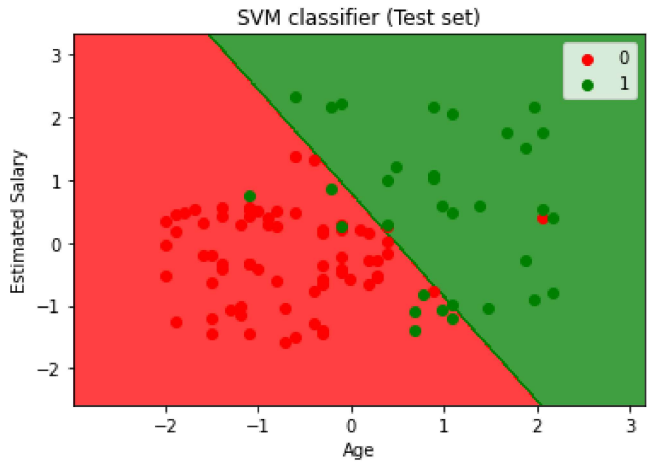


As we can see, the above output is appearing similar to the Logistic regression output. In the output, we got the straight line as hyperplane because we have used a linear kernel in the classifier. And we have also discussed above that for the 2d space, the hyperplane in SVM is a straight line.

Visualizing the test set result:

In [29]:
```
#Visulaizing the test set result
from matplotlib.colors import ListedColormap
x_set, y_set = x_test, y_test
x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].max() + 1, step
nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))
mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape),
alpha = 0.75, cmap = ListedColormap(('red','green' )))
mtp.xlim(x1.min(), x1.max())
mtp.ylim(x2.min(), x2.max())
for i, j in enumerate(nm.unique(y_set)):
    mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
        c = ListedColormap(('red', 'green'))(i), label = j)
mtp.title('SVM classifier (Test set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()
```

```
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-
mapping will have precedence in case its length matches with *x* & *y*.  Please use the *color*
keyword-argument or provide a 2D array with a single row if you intend to specify the same RGB o
r RGBA value for all points.
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-
mapping will have precedence in case its length matches with *x* & *y*.  Please use the *color*
keyword-argument or provide a 2D array with a single row if you intend to specify the same RGB o
r RGBA value for all points.
```



In [ ]: