In [1]:

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

In [2]:

```
bankdata = pd.read_csv("C:\\Users\\khanw\\Documents\\Github Repository\\AI_Spring2022\\Data
```

In [3]:

```
bankdata.head()
```

Out[3]:

	Variance	Skewness	Curtosis	Entropy	Class
0	3.62160	8.6661	-2.8073	-0.44699	0
1	4.54590	8.1674	-2.4586	-1.46210	0
2	3.86600	-2.6383	1.9242	0.10645	0
3	3.45660	9.5228	-4.0112	-3.59440	0
4	0.32924	-4.4552	4.5718	-0.98880	0

All of the attributes in the dataset are numeric. The label is also numeric i.e. 0 and 1.

In [4]:

```
bankdata.shape
```

Out[4]:

```
(1372, 5)
```

Data Preprocessing

(1) Dividing the data into attributes and labels
(2) dividing the data into training and testing sets.

In [5]:

```
X = bankdata.drop('Class', axis=1)
y = bankdata['Class']
```

In [6]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20)
```

Imporing SVC Model

In [7]:

```
from sklearn.svm import SVC
svclassifier = SVC(kernel='linear')
```

Training the Algorithm

In [8]:

```
svclassifier.fit(X_train, y_train)
```

Out[8]:

```
SVC(kernel='linear')
```

Making Prediction or Testing

The predict() method always expects a 2D array of shape [n_samples, n_features]. This means that if you want to predict even for a single data point, you will have to convert it into a 2D array.

In [9]:

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

In [10]:

```
print(X_test)
```

```
      Variance   Skewness  Curtosis  Entropy
473    2.82090   7.310800  -0.81857 -1.87840
1202  -0.70346   2.957000  -3.59470 -3.14570
1140  -0.53072  -0.097265  -0.21793  1.04260
315    0.32920  -4.455200   4.57180 -0.98880
1060  -4.84260  -4.993200  10.40520 -0.53104
...        ...        ...       ...      ...
264    2.48300   6.615500  -0.79287 -0.90863
463   -1.94580  11.221700   1.90790 -3.44050
609    3.40920   5.404900  -2.52280 -0.89958
951   -1.69360   2.785200  -2.18350 -1.92760
511    3.94330   2.501700   1.52150  0.90300

[275 rows x 4 columns]
```

Evaluating the Algorithm

Confusion matrix, precision, recall, and F1 measures are the most commonly used metrics for classification tasks.

In [11]:

```
from sklearn.metrics import classification_report, confusion_matrix
print(confusion_matrix(y_test,y_pred))
print(classification_report(y_test,y_pred))
```

```
[[141   1]
 [  0 133]]
              precision    recall  f1-score   support

           0       1.00      0.99      1.00       142
           1       0.99      1.00      1.00       133

    accuracy                           1.00       275
   macro avg       1.00      1.00      1.00       275
weighted avg       1.00      1.00      1.00       275
```

Kernel SVM (Non-Linear SVM)

Using Iris Dataset

In [12]:

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

In [13]:

```
# Read dataset to pandas dataframe
irisdata = pd.read_csv("C:\\Users\\khanw\\Documents\\Github Repository\\AI_Spring2022\\Data
```

In [14]:

```
irisdata.head()
```

Out[14]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa

Preprocessing

In [15]:

```
X = irisdata.drop(['Species', 'Id'], axis=1)
y = irisdata['Species']
```

Train Test Split

In [16]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20)
```

Training the Algorithm

To train the kernel SVM, we use the same SVC class of the Scikit-Learn's svm library. The difference lies in the value for the kernel parameter of the SVC class. In the case of the simple SVM we used "linear" as the value for the kernel parameter. However, for kernel SVM you can use Gaussian, polynomial, sigmoid, or computable kernel.

1. Polynomial Kernel

In the case of polynomial kernel, you also have to pass a value for the degree parameter of the SVC class. This basically is the degree of the polynomial.

Importing and fitting the model

In [17]:

```
from sklearn.svm import SVC
svclassifier = SVC(kernel='poly', degree=8)
svclassifier.fit(X_train, y_train)
```

Out[17]:

```
SVC(degree=8, kernel='poly')
```

Testing the Model

In [18]:

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

Evaluating the Model

In [19]:

```
from sklearn.metrics import classification_report, confusion_matrix
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
[[10  0  0]
 [ 0 10  1]
 [ 0  2  7]]
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        10
Iris-versicolor       0.83      0.91      0.87        11
 Iris-virginica       0.88      0.78      0.82         9

       accuracy                           0.90        30
      macro avg       0.90      0.90      0.90        30
   weighted avg       0.90      0.90      0.90        30
```

In [20]:

```
print(X_test)
```

```
     SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm
86             6.7           3.1            4.7           1.5
93             5.0           2.3            3.3           1.0
78             6.0           2.9            4.5           1.5
70             5.9           3.2            4.8           1.8
130            7.4           2.8            6.1           1.9
44             5.1           3.8            1.9           0.4
125            7.2           3.2            6.0           1.8
32             5.2           4.1            1.5           0.1
6              4.6           3.4            1.4           0.3
54             6.5           2.8            4.6           1.5
21             5.1           3.7            1.5           0.4
73             6.1           2.8            4.7           1.2
92             5.8           2.6            4.0           1.2
3              4.6           3.1            1.5           0.2
40             5.0           3.5            1.3           0.3
38             4.4           3.0            1.3           0.2
47             4.6           3.2            1.4           0.2
46             5.1           3.8            1.6           0.2
122            7.7           2.8            6.7           2.0
117            7.7           3.8            6.7           2.2
134            6.1           2.6            5.6           1.4
55             5.7           2.8            4.5           1.3
56             6.3           3.3            4.7           1.6
110            6.5           3.2            5.1           2.0
22             4.6           3.6            1.0           0.2
143            6.8           3.2            5.9           2.3
133            6.3           2.8            5.1           1.5
57             4.9           2.4            3.3           1.0
50             7.0           3.2            4.7           1.4
140            6.7           3.1            5.6           2.4
```

In [21]:

```
print(y_pred)
```

```
['Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor' 'Iris-virginica'
 'Iris-virginica' 'Iris-setosa' 'Iris-virginica' 'Iris-setosa'
 'Iris-setosa' 'Iris-versicolor' 'Iris-setosa' 'Iris-versicolor'
 'Iris-versicolor' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa'
 'Iris-setosa' 'Iris-virginica' 'Iris-virginica' 'Iris-versicolor'
 'Iris-versicolor' 'Iris-versicolor' 'Iris-virginica' 'Iris-setosa'
 'Iris-virginica' 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor'
 'Iris-virginica']
```

In [22]:

```
svclassifier.predict(np.reshape([[5.0, 2.0, 3.5, 1.0]],(1,-1)))
```

```
C:\Users\khanw\.conda\envs\tf-gpu\lib\site-packages\sklearn\base.py:450: Use
rWarning: X does not have valid feature names, but SVC was fitted with featu
re names
  warnings.warn(
```

Out[22]:

```
array(['Iris-versicolor'], dtype=object)
```

2. Gaussian Kernel

To use Gaussian kernel, you have to specify 'rbf' as value for the Kernel parameter of the SVC class.

In [23]:

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

Out[23]:

```
SVC()
```

In [24]:

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

In [25]:

```
from sklearn.metrics import classification_report, confusion_matrix
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
[[10  0  0]
 [ 0 10  1]
 [ 0  0  9]]
                precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        10
Iris-versicolor       1.00      0.91      0.95        11
 Iris-virginica       0.90      1.00      0.95         9

       accuracy                           0.97        30
      macro avg       0.97      0.97      0.97        30
   weighted avg       0.97      0.97      0.97        30
```

3. Sigmoid Kernel

To use the sigmoid kernel, you have to specify 'sigmoid' as value for the kernel parameter of the SVC class.

In [26]:

```
from sklearn.svm import SVC
svclassifier = SVC(kernel='sigmoid')
svclassifier.fit(X_train, y_train)
```

Out[26]:

```
SVC(kernel='sigmoid')
```

In [27]:

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

In [29]:

```
from sklearn.metrics import classification_report, confusion_matrix
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
[[ 0  0 10]
 [ 0  0 11]
 [ 0  0  9]]
                 precision    recall  f1-score   support

    Iris-setosa       0.00      0.00      0.00        10
Iris-versicolor       0.00      0.00      0.00        11
 Iris-virginica       0.30      1.00      0.46         9

       accuracy                           0.30        30
      macro avg       0.10      0.33      0.15        30
   weighted avg       0.09      0.30      0.14        30


C:\Users\khanw\.conda\envs\tf-gpu\lib\site-packages\sklearn\metrics\_classif
ication.py:1318: UndefinedMetricWarning: Precision and F-score are ill-defin
ed and being set to 0.0 in labels with no predicted samples. Use `zero_divis
ion` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\khanw\.conda\envs\tf-gpu\lib\site-packages\sklearn\metrics\_classif
ication.py:1318: UndefinedMetricWarning: Precision and F-score are ill-defin
ed and being set to 0.0 in labels with no predicted samples. Use `zero_divis
ion` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\khanw\.conda\envs\tf-gpu\lib\site-packages\sklearn\metrics\_classif
ication.py:1318: UndefinedMetricWarning: Precision and F-score are ill-defin
ed and being set to 0.0 in labels with no predicted samples. Use `zero_divis
ion` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

Comparison of Kernel Performance

If we compare the performance of the different types of kernels we can clearly see that the sigmoid kernel performs the worst. This is due to the reason that sigmoid function returns two values, 0 and 1, therefore it is more suitable for binary classification problems. However, in our case we had three output classes.

Amongst the Gaussian kernel and polynomial kernel, we can see that Gaussian kernel achieved a perfect 100% prediction rate while polynomial kernel misclassified one instance. Therefore the Gaussian kernel performed slightly better. However, there is no hard and fast rule as to which kernel performs best in every scenario. It is all about testing all the kernels and selecting the one with the best results on your test dataset.