SVM on Multiclass Dataset

Use RBF, Polynomial and Sigmoid kernel with SVM and compare the performance of the kernels using suitable multiclass data set.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split,GridSearchCV # Import train_test_split function
from sklearn.svm import SVC #Import svm model
from sklearn.metrics import accuracy_score,confusion_matrix,classification_report,precision_score,recall_score
```

```
from google.colab import drive
drive.mount('/content/drive')
```

Show hidden output

```
data = pd.read_csv("/content/drive/MyDrive/heart.csv")
```

```
data.head()
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 303 entries, 0 to 302
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   age       303 non-null    int64  
 1   sex       303 non-null    int64  
 2   cp        303 non-null    int64  
 3   trestbps  303 non-null    int64  
 4   chol      303 non-null    int64  
 5   fbs       303 non-null    int64  
 6   restecg   303 non-null    int64  
 7   thalach   303 non-null    int64  
 8   exang     303 non-null    int64  
 9   oldpeak   303 non-null    float64
 10  slope     303 non-null    int64  
 11  ca        303 non-null    int64  
 12  thal      303 non-null    int64  
 13  target    303 non-null    int64  
dtypes: float64(1), int64(13)
memory usage: 33.3 KB
```

```
x = data.drop('target',axis = 1)
y = data.dtypes
```

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3)
```

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

```
ml.fit(x_train, y_train)
```

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

```
print(ml.support_vectors_)
```

```
[[7.00000e+01 1.52300e+00 1.33100e+01 3.58000e+00 8.20000e-01 7.19900e+01
  1.20000e-01 1.01700e+01 0.00000e+00 3.00000e-02]
 [7.30000e+01 1.51593e+00 1.30900e+01 3.59000e+00 1.52000e+00 7.31000e+01
```

```
 6.70000e-01 7.83000e+00 0.00000e+00 0.00000e+00]
[1.43000e+02 1.51662e+00 1.28500e+01 3.51000e+00 1.44000e+00 7.30100e+01
 6.80000e-01 8.23000e+00 6.00000e-02 2.50000e-01]
[1.62000e+02 1.51934e+00 1.36400e+01 3.54000e+00 7.50000e-01 7.26500e+01
 1.60000e-01 8.89000e+00 1.50000e-01 2.40000e-01]
[1.47000e+02 1.51769e+00 1.36500e+01 3.66000e+00 1.11000e+00 7.27700e+01
 1.10000e-01 8.60000e+00 0.00000e+00 0.00000e+00]
[1.75000e+02 1.52058e+00 1.28500e+01 1.61000e+00 2.17000e+00 7.21800e+01
 7.60000e-01 9.70000e+00 2.40000e-01 5.10000e-01]
[1.76000e+02 1.52119e+00 1.29700e+01 3.30000e-01 1.51000e+00 7.33900e+01
 1.30000e-01 1.12700e+01 0.00000e+00 2.80000e-01]
[1.65000e+02 1.51915e+00 1.27300e+01 1.85000e+00 1.86000e+00 7.26900e+01
 6.00000e-01 1.00900e+01 0.00000e+00 0.00000e+00]
[1.64000e+02 1.51514e+00 1.40100e+01 2.68000e+00 3.50000e+00 6.98900e+01
 1.68000e+00 5.87000e+00 2.20000e+00 0.00000e+00]
[1.85000e+02 1.51115e+00 1.73800e+01 0.00000e+00 3.40000e-01 7.54100e+01
 0.00000e+00 6.65000e+00 0.00000e+00 0.00000e+00]
[1.77000e+02 1.51905e+00 1.40000e+01 2.39000e+00 1.56000e+00 7.23700e+01
 0.00000e+00 9.57000e+00 0.00000e+00 0.00000e+00]
[1.83000e+02 1.51916e+00 1.41500e+01 0.00000e+00 2.09000e+00 7.27400e+01
 0.00000e+00 1.08800e+01 0.00000e+00 0.00000e+00]
[1.92000e+02 1.51602e+00 1.48500e+01 0.00000e+00 2.38000e+00 7.32800e+01
 0.00000e+00 8.76000e+00 6.40000e-01 9.00000e-02]
[1.89000e+02 1.52247e+00 1.48600e+01 2.20000e+00 2.06000e+00 7.02600e+01
 7.60000e-01 9.76000e+00 0.00000e+00 0.00000e+00]
[1.88000e+02 1.52315e+00 1.34400e+01 3.34000e+00 1.23000e+00 7.23800e+01
 6.00000e-01 8.83000e+00 0.00000e+00 0.00000e+00]
[1.86000e+02 1.51131e+00 1.36900e+01 3.20000e+00 1.81000e+00 7.28100e+01
 1.76000e+00 5.43000e+00 1.19000e+00 0.00000e+00]]
```

```
print(ml.n_support_)
```

```
[1 2 2 4 3 4]
```

```
y_pred = ml.predict(x_test)
```

```
print(accuracy_score(y_test,y_pred))
```

```
0.9846153846153847
```

```
print(confusion_matrix(y_test,y_pred))
```

```
[[18  0  0  0  0  0]
 [ 0 27  1  0  0  0]
 [ 0  0  4  0  0  0]
 [ 0  0  0  3  0  0]
 [ 0  0  0  0  1  0]
 [ 0  0  0  0  0 11]]
```

```
print(classification_report(y_test,y_pred))
```

```
                   precision    recall  f1-score   support

                1       1.00      1.00      1.00        18
                2       1.00      0.96      0.98        28
                3       0.80      1.00      0.89         4
                5       1.00      1.00      1.00         3
                6       1.00      1.00      1.00         1
                7       1.00      1.00      1.00        11

         accuracy                           0.98        65
        macro avg       0.97      0.99      0.98        65
     weighted avg       0.99      0.98      0.99        65
```

With Different Kernels

```
model1=SVC(kernel='sigmoid',gamma=0.001)
model2=SVC(kernel='poly',degree=3)
model3=SVC(kernel='rbf')
```

```
model1.fit(x_train,y_train)
model2.fit(x_train,y_train)
model3.fit(x_train,y_train)
```

```
SVC()
```

```
ypred1=model1.predict(x_test)
```

```
ypred2=model2.predict(x_test)

ypred3=model3.predict(x_test)
```

+ Code + Text

```
print(accuracy_score(y_test,ypred1))
```

```
0.6923076923076923
```

```
print(accuracy_score(y_test,ypred2))
```

```
0.9076923076923077
```

```
print(accuracy_score(y_test,ypred3))
```

```
0.8769230769230769
```