Untitled

April 20, 2024

[81]:
```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.svm import SVC
from sklearn.datasets import make_moons
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from IPython.display import display
from PIL import Image
from sklearn.model_selection import GridSearchCV as GridSearch
```

[82]:
```python
wine_data = np.genfromtxt('winequality-red-3.csv', delimiter=';',dtype=None,␣
 ↪encoding=None)

header = wine_data[0]
wine_data = wine_data[1:].astype(float)

print("Header:", header)
print("Data:", wine_data)
```

```
Header: ['"fixed acidity' '""volatile acidity""' '""citric acid""'
 '""residual sugar""' '""chlorides""' '""free sulfur dioxide""'
 '""total sulfur dioxide""' '""density""' '""pH""' '""sulphates""'
 '""alcohol""' '""quality"""']
Data: [[ 7.4    0.7    0.    … 0.56   9.4    5.   ]
 [ 7.8    0.88   0.    … 0.68   9.8    5.   ]
 [ 7.8    0.76   0.04  … 0.65   9.8    5.   ]
 …
 [ 6.3    0.51   0.13  … 0.75  11.     6.   ]
 [ 5.9    0.645  0.12  … 0.71  10.2    5.   ]
 [ 6.     0.31   0.47  … 0.66  11.     6.   ]]
```

[83]:
```python
seed = 420
X = wine_data[:,:11]
y = wine_data[:,11]

print("X here: ",X)
print("Y here: ",y)
```

```
X_train,X_test,Y_train,Y_test = train_test_split(X,y,test_size=0.
 ↪3,random_state=seed)

print("Y shape: ",y.shape)
print("X shape: ",X.shape)
```

```
X here:  [[ 7.4    0.7    0.    …  3.51   0.56   9.4  ]
 [ 7.8    0.88   0.    …  3.2    0.68   9.8  ]
 [ 7.8    0.76   0.04  …  3.26   0.65   9.8  ]
 …
 [ 6.3    0.51   0.13  …  3.42   0.75  11.   ]
 [ 5.9    0.645  0.12  …  3.57   0.71  10.2  ]
 [ 6.     0.31   0.47  …  3.39   0.66  11.   ]]
Y here:  [5. 5. 5. … 6. 5. 6.]
Y shape:  (1599,)
X shape:  (1599, 11)
```

[101]:

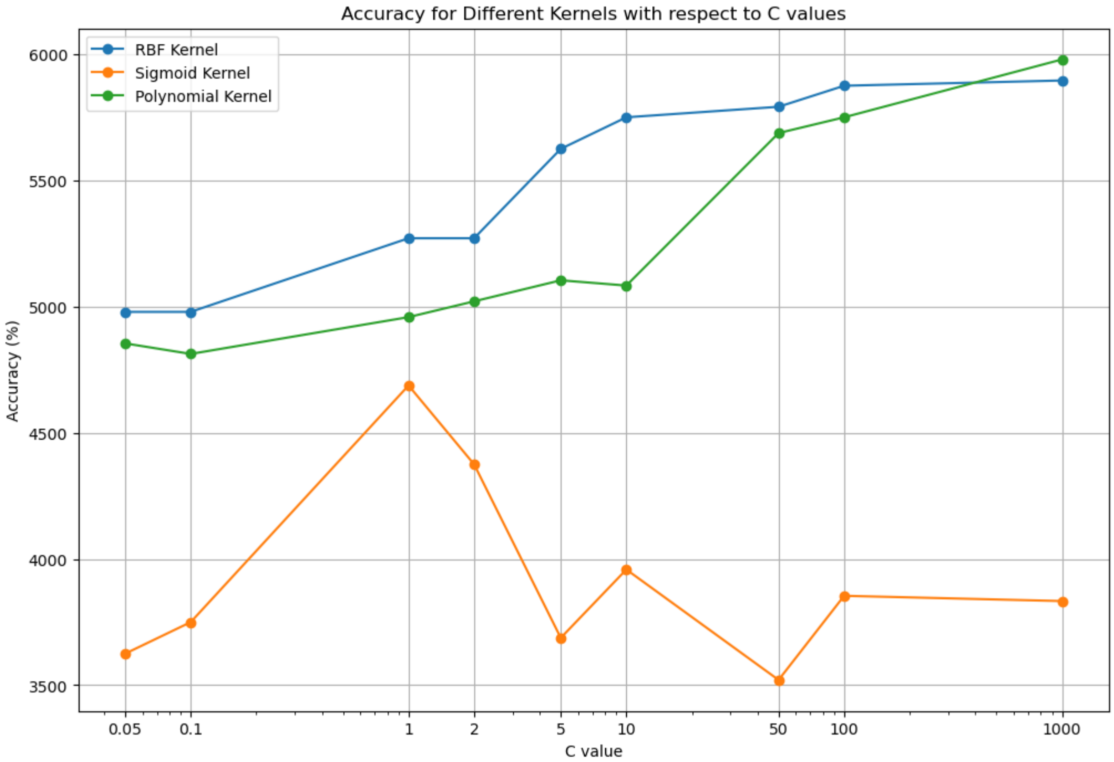
```
params = {
    "kernel": ['rbf', 'sigmoid', 'poly'], # didn't use linear becauseit sucks
    "C": [0.05, 0.1, 1, 2, 5, 10, 50, 100, 1000]
}

grid_search = GridSearchCV(SVC(), params, cv=5)
grid_search.fit(X_train, Y_train)

rbf_Acc = []
sigmoid_Acc = []
poly_Acc = []


# Extracting parameters and mean test scores
params = grid_search.cv_results_['params']
mean_test_scores = grid_search.cv_results_['mean_test_score']

best_acc = 0;
best_ker = None;
best_C = 0;


for param, score in zip(params, mean_test_scores):
    kernel = param['kernel']
    C = param['C']

    # Creating a new SVC instance with the given parameters
    estimator = SVC(kernel=kernel, C=C)
    estimator.fit(X_train, Y_train)
```

```python
    predictions = estimator.predict(X_test)
    accuracy = accuracy_score(Y_test, predictions)

    if accuracy > best_acc:
        best_acc = accuracy
        best_C = C
        best_ker = kernel

    if kernel == "rbf":
        rbf_Acc.append(accuracy)
    elif kernel == "poly":
        poly_Acc.append(accuracy)
    else:
        sigmoid_Acc.append(accuracy)

    print(f"Kernel: {kernel}, C: {C}, Mean Test Score: {score:.4f}, Accuracy:␣
 ↪{accuracy * 100:.2f}%")

print("\n")
print("Best Parameters Found >>>> ", best_ker, best_C, best_acc)
```

```
Kernel: rbf, C: 0.05, Mean Test Score: 0.4835, Accuracy: 49.79%
Kernel: sigmoid, C: 0.05, Mean Test Score: 0.3807, Accuracy: 36.25%
Kernel: poly, C: 0.05, Mean Test Score: 0.4728, Accuracy: 48.54%
Kernel: rbf, C: 0.1, Mean Test Score: 0.4880, Accuracy: 49.79%
Kernel: sigmoid, C: 0.1, Mean Test Score: 0.3431, Accuracy: 37.50%
Kernel: poly, C: 0.1, Mean Test Score: 0.4737, Accuracy: 48.12%
Kernel: rbf, C: 1, Mean Test Score: 0.4996, Accuracy: 52.71%
Kernel: sigmoid, C: 1, Mean Test Score: 0.4235, Accuracy: 46.88%
Kernel: poly, C: 1, Mean Test Score: 0.4772, Accuracy: 49.58%
Kernel: rbf, C: 2, Mean Test Score: 0.5076, Accuracy: 52.71%
Kernel: sigmoid, C: 2, Mean Test Score: 0.4083, Accuracy: 43.75%
Kernel: poly, C: 2, Mean Test Score: 0.4853, Accuracy: 50.21%
Kernel: rbf, C: 5, Mean Test Score: 0.5309, Accuracy: 56.25%
Kernel: sigmoid, C: 5, Mean Test Score: 0.3842, Accuracy: 36.88%
Kernel: poly, C: 5, Mean Test Score: 0.4969, Accuracy: 51.04%
Kernel: rbf, C: 10, Mean Test Score: 0.5460, Accuracy: 57.50%
Kernel: sigmoid, C: 10, Mean Test Score: 0.3851, Accuracy: 39.58%
Kernel: poly, C: 10, Mean Test Score: 0.5103, Accuracy: 50.83%
Kernel: rbf, C: 50, Mean Test Score: 0.5657, Accuracy: 57.92%
Kernel: sigmoid, C: 50, Mean Test Score: 0.3735, Accuracy: 35.21%
Kernel: poly, C: 50, Mean Test Score: 0.5300, Accuracy: 56.88%
Kernel: rbf, C: 100, Mean Test Score: 0.5782, Accuracy: 58.75%
Kernel: sigmoid, C: 100, Mean Test Score: 0.3887, Accuracy: 38.54%
Kernel: poly, C: 100, Mean Test Score: 0.5460, Accuracy: 57.50%
Kernel: rbf, C: 1000, Mean Test Score: 0.5791, Accuracy: 58.96%
Kernel: sigmoid, C: 1000, Mean Test Score: 0.3824, Accuracy: 38.33%
Kernel: poly, C: 1000, Mean Test Score: 0.5719, Accuracy: 59.79%
```

```
Best Parameters Found >>>>  poly 1000 0.5979166666666667
```

[108]:
```
rbf_Acc = [accuracy * 100 for accuracy in rbf_Acc]  # Convert to percentage
sigmoid_Acc = [accuracy * 100 for accuracy in sigmoid_Acc]  # Convert to␣
 ↪percentage
poly_Acc = [accuracy * 100 for accuracy in poly_Acc]  # Convert to percentage

# C values
C_values = [0.05, 0.1, 1, 2, 5, 10, 50, 100, 1000]

# Plotting
plt.figure(figsize=(12, 8))

# Plotting RBF
plt.plot(C_values, rbf_Acc, marker='o', label='RBF Kernel')

# Plotting Sigmoid
plt.plot(C_values, sigmoid_Acc, marker='o', label='Sigmoid Kernel')

# Plotting Polynomial
plt.plot(C_values, poly_Acc, marker='o', label='Polynomial Kernel')

plt.xscale('log')  # Using log scale for better visualization of C values
plt.xticks(C_values, C_values)
plt.title('Accuracy for Different Kernels with respect to C values')
plt.xlabel('C value')
plt.ylabel('Accuracy (%)')
plt.grid(True)
plt.legend()
plt.show()

"""
we deduce from here and conclude that the best kernel is the POLY kernel and␣
 ↪the best
"""
```



[108]: '\nwe deduce from here and conclude that the best kernel is the POLY kernel and the best \n'

[ ]:

[ ]:

[ ]: