Support Vector Machine

Importing Libraries

```python
import pandas as pd
import numpy as np
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split

pd.set_option('max_columns', None)
```

```python
%matplotlib inline
import matplotlib.pyplot as plt
from matplotlib import style
style.use('ggplot')
```

Reading the Data

```python
data = pd.read_csv('data.txt', delimiter=' ', header=None)
```

Exploratory Data Analysis

```python
data
```

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
0	1	6	4	12	5	5	3	4	1	67	3	2	1	2	1	0	0	1	0	0	1	0	0	1	1
1	2	48	2	60	1	3	2	2	1	22	3	1	1	1	1	0	0	1	0	0	1	0	0	1	2
2	4	12	4	21	1	4	3	3	1	49	3	1	2	1	1	0	0	1	0	0	1	0	1	0	1
3	1	42	2	79	1	4	3	4	2	45	3	1	2	1	1	0	0	0	0	0	0	0	0	1	1
4	1	24	3	49	1	3	3	4	4	53	3	2	2	1	1	1	0	1	0	0	0	0	0	1	2
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
795	4	9	2	23	2	2	2	4	2	22	3	1	1	1	1	0	0	1	0	1	0	0	0	1	1
796	1	18	2	75	5	5	3	4	2	51	3	1	2	2	1	0	1	1	0	0	0	0	0	1	2
797	4	12	4	13	1	2	2	4	2	22	3	2	1	1	1	0	0	1	0	1	0	0	1	0	1
798	4	24	3	7	5	5	4	4	3	54	3	2	1	2	1	1	0	1	0	0	1	0	0	1	1
799	2	9	2	15	5	2	3	2	1	35	3	1	1	1	1	1	0	1	0	0	1	1	0	0	1

800 rows × 25 columns

```python
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 800 entries, 0 to 799
Data columns (total 25 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   0       800 non-null    int64
 1   1       800 non-null    int64
 2   2       800 non-null    int64
 3   3       800 non-null    int64
 4   4       800 non-null    int64
 5   5       800 non-null    int64
 6   6       800 non-null    int64
 7   7       800 non-null    int64
 8   8       800 non-null    int64
 9   9       800 non-null    int64
 10  10      800 non-null    int64
 11  11      800 non-null    int64
 12  12      800 non-null    int64
 13  13      800 non-null    int64
 14  14      800 non-null    int64
 15  15      800 non-null    int64
 16  16      800 non-null    int64
 17  17      800 non-null    int64
 18  18      800 non-null    int64
 19  19      800 non-null    int64
 20  20      800 non-null    int64
 21  21      800 non-null    int64
 22  22      800 non-null    int64
 23  23      800 non-null    int64
 24  24      800 non-null    int64
dtypes: int64(25)
memory usage: 156.4 KB
```

In [108]:

```python
# data = data.loc[:,data.apply(pd.Series.nunique) < 5]
```

In [109]:

```python
X, y = data.iloc[:, :-1].values, data.iloc[:, -1].values
```

In [110]:

```python
def svm(X, y, model):
    accuracies = []

    for i in range(10):
        accuracy = 0
        X_train, X_test, y_train, y_test = train_test_split(
            X, y, test_size=0.4, random_state=i)

        model.fit(X_train, y_train)
        y_predict = model.predict(X_test)
        for j in range(len(y_predict)):
            if y_predict[j] == y_test[j]:
                accuracy += 1
        accuracies.append(accuracy / len(y_predict) * 100)
    return accuracies
```

In [121]:

```python
svm_clf = Pipeline((
("linear_svc", SVC(C=0.1, kernel="linear")),
))
not_scaled_accuracies = svm(X, y, svm_clf)
mean_of_not_scaled_accuracies = np.mean(not_scaled_accuracies)

print(not_scaled_accuracies)
print(mean_of_not_scaled_accuracies)
```

```
[74.375, 77.8125, 69.0625, 73.4375, 75.0, 74.6875, 73.125, 75.625, 73.4375, 71.5625]
73.8125
```

In [122]:

```python
svm_clf = Pipeline((
("scaler", StandardScaler()),
("linear_svc", SVC(C=0.1, kernel="linear")),
```

```
))
scaled_accuracies = svm(X, y, svm_clf)
mean_of_scaled_accuracies = np.mean(scaled_accuracies)

print(scaled_accuracies)
print(mean_of_scaled_accuracies)

[75.0, 78.125, 71.5625, 75.625, 75.625, 75.3125, 73.125, 75.0, 73.4375, 73.125]
74.59375
```

Reporting

(A) - Difference Between The Dataset Used in **1** and **2**

```
# Data used in pipeline A
data.describe()
```

	0	6	7	8	10	11	12	13	14	15	16	17	
count	800.000000	800.000000	800.000000	800.00000	800.000000	800.000000	800.000000	800.000000	800.000000	800.000000	800.000000	800.000000	8(
mean	2.582500	2.673750	2.841250	2.36625	2.676250	1.396250	1.147500	1.398750	1.033750	0.230000	0.101250	0.910000	
std	1.242023	0.700303	1.106833	1.06114	0.706796	0.569773	0.354825	0.489947	0.180698	0.421096	0.301848	0.286361	
min	1.000000	1.000000	1.000000	1.00000	1.000000	1.000000	1.000000	1.000000	1.000000	0.000000	0.000000	0.000000	
25%	1.000000	2.000000	2.000000	1.00000	3.000000	1.000000	1.000000	1.000000	1.000000	0.000000	0.000000	1.000000	
50%	2.000000	3.000000	3.000000	2.00000	3.000000	1.000000	1.000000	1.000000	1.000000	0.000000	0.000000	1.000000	
75%	4.000000	3.000000	4.000000	3.00000	3.000000	2.000000	1.000000	2.000000	1.000000	0.000000	0.000000	1.000000	
max	4.000000	4.000000	4.000000	4.00000	3.000000	4.000000	2.000000	2.000000	2.000000	1.000000	1.000000	1.000000	

```
# Data used in pipeline B
scaler = StandardScaler()

scaler.fit(data)

scaled_features = scaler.transform(data)
pd.DataFrame(scaled_features, index=data.index, columns=data.columns).describe()
```

	0	6	7	8	10	11	12	13	14	15
count	8.000000e+02	8.000000e+02	8.000000e+02	8.000000e+02	8.000000e+02	8.000000e+02	8.000000e+02	8.000000e+02	8.000000e+02	8.000000e+02
mean	1.554312e-17	-4.440892e-17	-3.663736e-17	1.443290e-17	-9.992007e-18	3.774758e-17	1.398881e-16	8.659740e-17	2.753353e-16	-6.661338e-17
std	1.000626e+00	1.000626e+00	1.000626e+00	1.000626e+00	1.000626e+00	1.000626e+00	1.000626e+00	1.000626e+00	1.000626e+00	1.000626e+00
min	-1.274928e+00	-2.391533e+00	-1.664570e+00	-1.288336e+00	-2.373102e+00	-6.958870e-01	-4.159574e-01	-8.143719e-01	-1.868926e-01	-5.465357e-01
25%	-1.274928e+00	-9.626857e-01	-7.605267e-01	-1.288336e+00	4.583396e-01	-6.958870e-01	-4.159574e-01	-8.143719e-01	-1.868926e-01	-5.465357e-01
50%	-4.692862e-01	4.661614e-01	1.435169e-01	-3.453635e-01	4.583396e-01	-6.958870e-01	-4.159574e-01	-8.143719e-01	-1.868926e-01	-5.465357e-01
75%	1.141997e+00	4.661614e-01	1.047561e+00	5.976086e-01	4.583396e-01	1.060295e+00	-4.159574e-01	1.227940e+00	-1.868926e-01	-5.465357e-01
max	1.141997e+00	1.895008e+00	1.047561e+00	1.540581e+00	4.583396e-01	4.572658e+00	2.404093e+00	1.227940e+00	5.350666e+00	1.829707e+00

(A) Here we can see that columns [1, 3, and 9] have a very huge region compared to other columns so after scaling all columns it helped in the accuracy measure **Not By Much Though (Just 0.6%) Which is not much**

But if we tried to drop columns that have very wide range and a lot of unique values (say > 5) we can see that the accuracy of the scaled version of the data is actually nearly the same as the not scaled one if not even less accurate

(B) The Averaged Accuracy of both pipelines ranges in a not small range of accuracy (about 6%) and that might be due to the fact that some of the points might be outliers or have noise in them which affects the learning of the linear svm algorithm

(C) Again the Difference in the averaged accuracies between the two pipelines is not that noticeable, maybe due to the fact that most of the data range is very close If we tried to do feature selection to pick only the most important features to learn form, then we might get a noticeable difference

(D) \ 1 - Reading the data and parse it into X, and y arrays \ 2 - create the svm function which takes the X, and y with the training model pipeline and run the learning algorithm for 10 times and then return all of them as an array \ 3 - take the mean of these accuracies to compare between the two pipelines \ 4 - for the second pipeline, I've applied a StandardScaler for normalizing the input data. I didn't apply any extra preprocessing though for the second pipeline \ 5 - I've choosen C hyperparameter to be 0.1 just after some trials

---

```python
from sklearn import datasets
from svm import SVM
from sklearn.model_selection import train_test_split

iris = datasets.load_iris()

X = iris.data[:, :2]  # we only take the first two features.
y = iris.target

# split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y == 0)

svm_clf = SVM()
svm_clf.fit(X_train, y_train)

y_predict = svm_clf.predict(X_test)

accuracy = 0

for i in range(len(X_test)):
    if (y_predict[i] == 1 and y_test[i] == True) or (y_predict[i] == -1 and y_test[i] == False):
        accuracy += 1

print(accuracy / len(X_test) * 100)
```

```
100.0
```