Experiment No.6

Title:Implementation of Support Vector Machine

Part A:Implementation of SVM Model on Synthetic data

Importing Libraries

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

Creating and Visualizing synthetic dataset

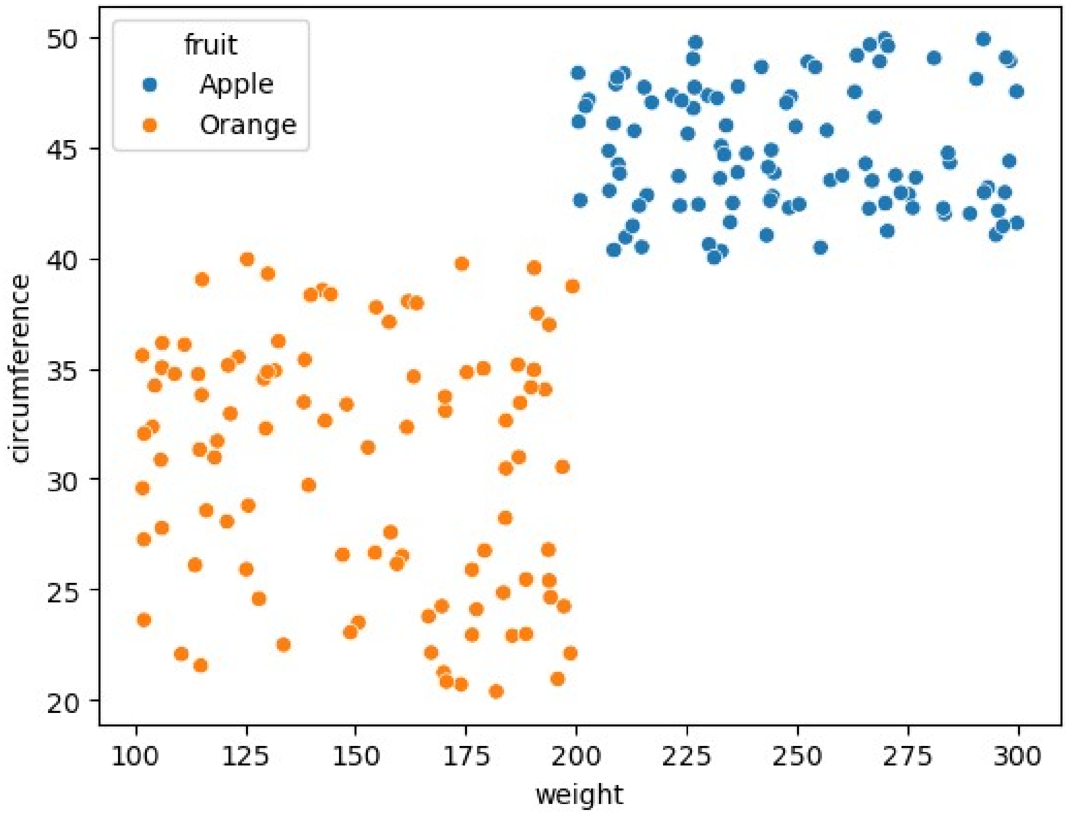
```
# Create a dataframe for apples
apples = pd.DataFrame({
    'weight': np.random.uniform(200.0, 300.0, size=100),
    'circumference': np.random.uniform(40.0, 50.0, size=100),
    'fruit': 'Apple'
})

# Create a dataframe for oranges
oranges = pd.DataFrame({
    'weight': np.random.uniform(100.0, 200.0, size=100),
    'circumference': np.random.uniform(20.0, 40.0, size=100),
    'fruit': 'Orange'
})

# Concatenate the two dataframes
df = pd.concat([apples, oranges])
df.head()
```

	weight	circumference	fruit
0	223.253891	43.690499	Apple
1	298.310891	48.903060	Apple
2	244.214135	44.880593	Apple
3	210.796607	48.338364	Apple
4	263.650477	49.161789	Apple

```
sns.scatterplot(x='weight', y='circumference', hue='fruit', data=df)
plt.show()
```



Separating Features and Target Variables

```
X=df[['weight', 'circumference']]
y=df['fruit']
```

SVM Model

```
from sklearn import svm
model=svm.SVC(kernel='linear')
model.fit(X, y)
```

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

Predicting Values

```
y_pred=model.predict(X)
y_pred
```

```
array(['Apple', 'Apple', 'Apple', 'Apple', 'Apple', 'Apple', 'Apple',
       'Apple', 'Apple', 'Apple', 'Apple', 'Apple', 'Apple', 'Apple',
       'Apple', 'Apple', 'Apple', 'Apple', 'Apple', 'Apple', 'Apple',
       'Apple', 'Apple', 'Apple', 'Apple', 'Apple', 'Apple', 'Apple',
       'Apple', 'Apple', 'Apple', 'Apple', 'Apple', 'Apple', 'Apple',
       'Apple', 'Apple', 'Apple', 'Apple', 'Apple', 'Apple', 'Apple',
       'Apple', 'Apple', 'Apple', 'Apple', 'Apple', 'Apple', 'Apple',
       'Apple', 'Apple', 'Apple', 'Apple', 'Apple', 'Apple', 'Apple',
```

```
       'Apple', 'Apple', 'Apple', 'Apple', 'Apple', 'Apple', 'Apple',
       'Apple', 'Apple', 'Apple', 'Apple', 'Apple', 'Apple', 'Apple',
       'Apple', 'Apple', 'Apple', 'Apple', 'Apple', 'Apple', 'Apple',
       'Apple', 'Apple', 'Apple', 'Apple', 'Apple', 'Apple', 'Apple',
       'Apple', 'Apple', 'Apple', 'Apple', 'Apple', 'Apple', 'Apple',
       'Apple', 'Apple', 'Apple', 'Apple', 'Apple', 'Apple', 'Apple',
       'Apple', 'Apple', 'Orange', 'Orange', 'Orange', 'Orange',
'Orange',
       'Orange', 'Orange', 'Orange', 'Orange', 'Orange', 'Orange',
       'Orange', 'Orange', 'Orange', 'Orange', 'Orange', 'Orange',
       'Orange', 'Orange', 'Orange', 'Orange', 'Orange', 'Orange',
       'Orange', 'Orange', 'Orange', 'Orange', 'Orange', 'Orange',
       'Orange', 'Orange', 'Orange', 'Orange', 'Orange', 'Orange',
       'Orange', 'Orange', 'Orange', 'Orange', 'Orange', 'Orange',
       'Orange', 'Orange', 'Orange', 'Orange', 'Orange', 'Orange',
       'Orange', 'Orange', 'Orange', 'Orange', 'Orange', 'Orange',
       'Orange', 'Orange', 'Orange', 'Orange', 'Orange', 'Orange',
       'Orange', 'Orange', 'Orange', 'Orange', 'Orange', 'Orange',
       'Orange', 'Orange', 'Orange', 'Orange', 'Orange', 'Orange',
       'Orange', 'Orange', 'Orange', 'Orange', 'Orange', 'Orange',
       'Orange', 'Orange', 'Orange', 'Orange', 'Orange', 'Orange',
       'Orange', 'Orange', 'Orange', 'Orange', 'Orange', 'Orange',
       'Orange', 'Orange', 'Orange', 'Orange', 'Orange', 'Orange',
       'Orange', 'Orange', 'Orange', 'Orange', 'Orange'],
dtype=object)
```

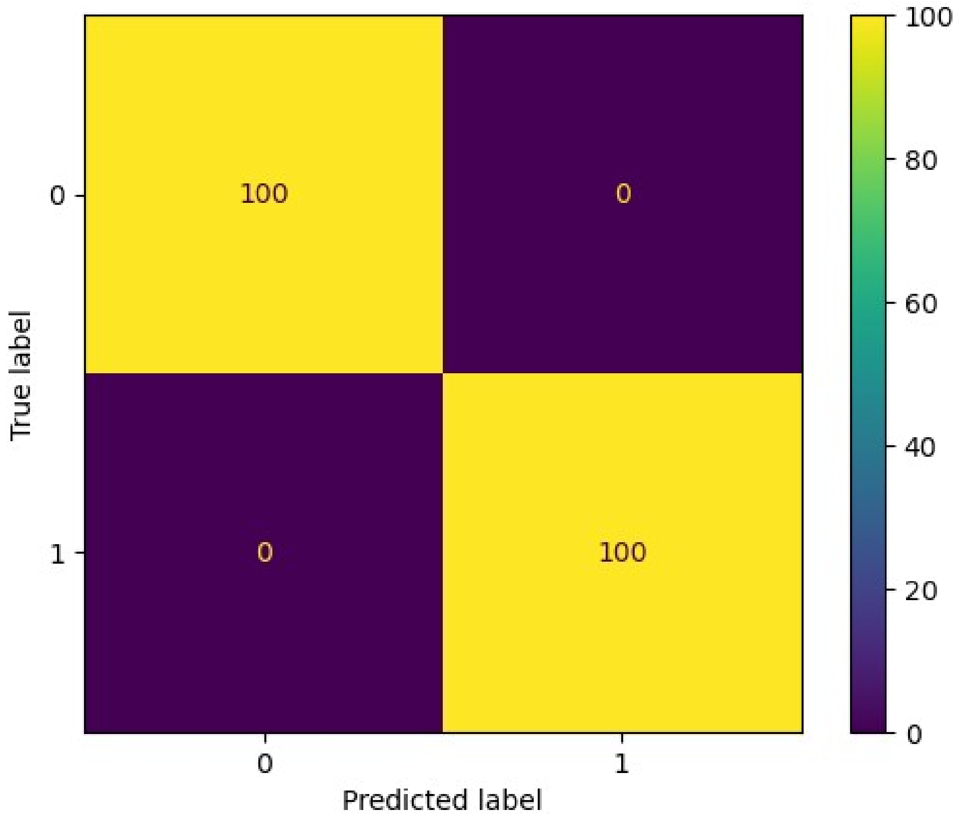
Performance Metrics

```
from sklearn.metrics import accuracy_score, confusion_matrix,
classification_report, ConfusionMatrixDisplay
print(accuracy_score(y, y_pred))
print(confusion_matrix(y, y_pred))
print(classification_report(y, y_pred))
ConfusionMatrixDisplay(confusion_matrix(y, y_pred)).plot()
plt.show()
```

```
1.0
[[100   0]
 [  0 100]]
              precision    recall  f1-score   support

       Apple       1.00      1.00      1.00       100
      Orange       1.00      1.00      1.00       100

    accuracy                           1.00       200
   macro avg       1.00      1.00      1.00       200
weighted avg       1.00      1.00      1.00       200
```



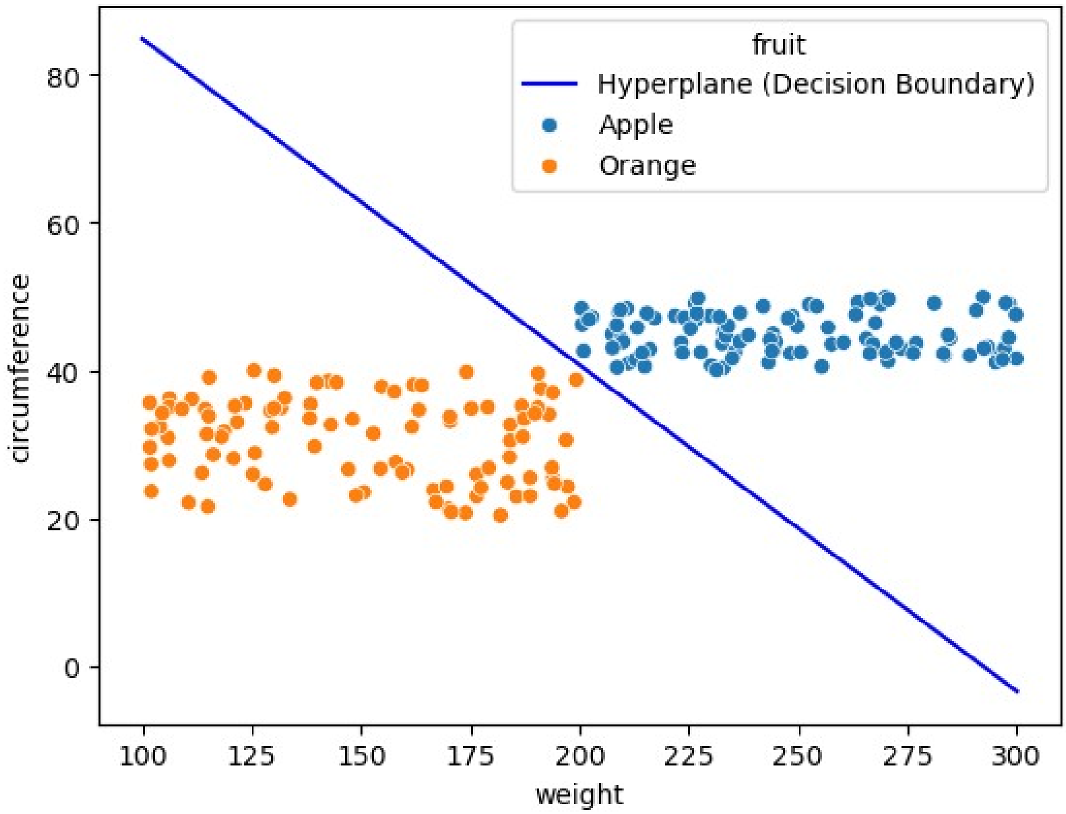
Get the coefficients (w) and intercept (b) of the hyperplane

```
w = model.coef_[0]
b = model.intercept_[0]
print('w:',w)
print('b:',b)

w: [-0.18971444 -0.42958037]
b: 55.41741104047813
```

Visualizing the SVM Model Hyperplane

```
x_vals = np.linspace(100, 300, 100)
y_vals = -(w[0]/w[1])*x_vals - b/w[1]
plt.plot(x_vals, y_vals, color='blue', label='Hyperplane (Decision
Boundary)')
sns.scatterplot(x='weight', y='circumference', hue='fruit', data=df)
plt.show()
```



Part B:Implementation of SVM Model on a dataset

```
# Import necessary libraries
import numpy as np
import pandas as pd
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import classification_report,
confusion_matrix,ConfusionMatrixDisplay
import matplotlib.pyplot as plt
import seaborn as sns

#Import the dataset
iris = sns.load_dataset('iris')
iris.head()
```

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

```
#Separating feature and target variables
X=iris[['sepal_width','petal_width']]
Y=iris['species']

#Split data into train and test set
X_train, X_test, Y_train, Y_test = train_test_split(X, Y,
test_size=0.2, random_state=42)

X_train.shape, X_test.shape, Y_train.shape, Y_test.shape
```

```
((120, 2), (30, 2), (120,), (30,))
```

```
#Create SVM Model
svm_model = SVC(kernel='linear', random_state=42)  # You can use other
kernels like 'rbf', 'poly'
svm_model.fit(X_train, Y_train)
```

```
SVC(kernel='linear', random_state=42)
```

```
#Make Predictions
Y_pred = svm_model.predict(X_test)
Y_pred
```

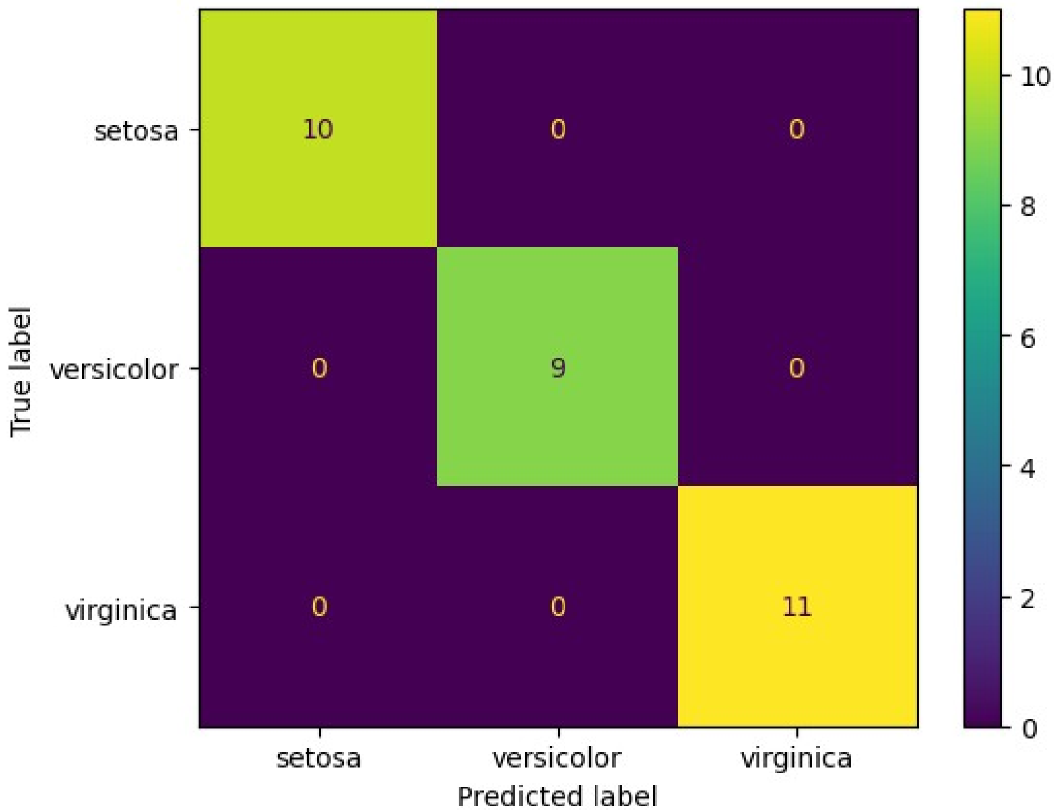
```
array(['versicolor', 'setosa', 'virginica', 'versicolor',
'versicolor',
       'setosa', 'versicolor', 'virginica', 'versicolor',
'versicolor',
       'virginica', 'setosa', 'setosa', 'setosa', 'setosa',
'versicolor',
       'virginica', 'versicolor', 'versicolor', 'virginica', 'setosa',
       'virginica', 'setosa', 'virginica', 'virginica', 'virginica',
       'virginica', 'virginica', 'setosa', 'setosa'], dtype=object)
```

```
#Evaluate the model performance
from sklearn.metrics import accuracy_score, confusion_matrix,
classification_report, ConfusionMatrixDisplay
labels = iris['species'].unique()  # The unique class labels for the
target
print(accuracy_score(Y_test, Y_pred))
print(confusion_matrix(Y_test, Y_pred))
print(classification_report(Y_test, Y_pred))
ConfusionMatrixDisplay(confusion_matrix(Y_test,
Y_pred),display_labels=labels).plot()
plt.show()
```

```
1.0
[[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]
              precision    recall  f1-score   support
```

```
      setosa       1.00      1.00      1.00        10
  versicolor       1.00      1.00      1.00         9
   virginica       1.00      1.00      1.00        11

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30
```



Get the coefficients (w) and intercept (b) of the hyperplanes

```
w0 = svm_model.coef_[0]
w1 = svm_model.coef_[1]
w2 = svm_model.coef_[2]
b = svm_model.intercept_[0]
print('w0:',w[0])
print('w1:',w[1])
print('w2:',w[2])
print('b:',b)
```

```
w0: [ 0.88988726 -1.94381961]
w1: [ 0.45917166 -1.85798801]
w2: [ 0.90852965 -3.63658848]
b: -1.1919105788982298
```

Conclusion:

Merits and Demerits of Support Vector Machine (SVM)

Merits of SVM

1. **Effective in High-Dimensional Spaces**:
   - SVM is particularly effective in high-dimensional spaces and is still effective when the number of dimensions exceeds the number of samples.
2. **Versatile Kernel Trick**:
   - SVM can be adapted to various classification tasks using different kernel functions (linear, polynomial, radial basis function, etc.), allowing it to capture complex relationships.
3. **Robust to Overfitting**:
   - Due to the concept of maximizing the margin between classes, SVM is less prone to overfitting, especially in high-dimensional spaces.
4. **Clear Margin of Separation**:
   - SVM provides a clear margin of separation between classes, making the classification process intuitive and interpretable.
5. **Memory Efficiency**:
   - SVM uses a subset of training points in the decision function (support vectors), making it memory efficient compared to other algorithms that might require all training data.
6. **Works Well with Unbalanced Data**:
   - SVM can handle unbalanced datasets by adjusting the penalty parameter, which helps in improving model performance.

Demerits of SVM

1. **Computationally Intensive**:
   - Training SVMs can be time-consuming, especially with large datasets, as the algorithm's complexity grows with the number of samples.
2. **Choice of Kernel and Parameters**:
   - Selecting the appropriate kernel and tuning parameters can be challenging. Poor choices can lead to underfitting or overfitting.
3. **Sensitivity to Noisy Data**:
   - SVM can be sensitive to noise in the data, especially when there are overlapping classes. This can affect the placement of the decision boundary.
4. **Difficulties with Large Datasets**:
   - While SVMs are effective in high-dimensional spaces, they struggle with very large datasets, where algorithms like logistic regression or decision trees might perform better.
5. **Binary Classification**:
   - SVM is inherently a binary classifier. While it can be extended to multi-class classification (using techniques like one-vs-one or one-vs-all), this adds complexity and can reduce performance.
6. **Interpretability**:

- While SVM provides a clear margin of separation, understanding the model's decisions can be less interpretable compared to simpler models like linear regression or decision trees.