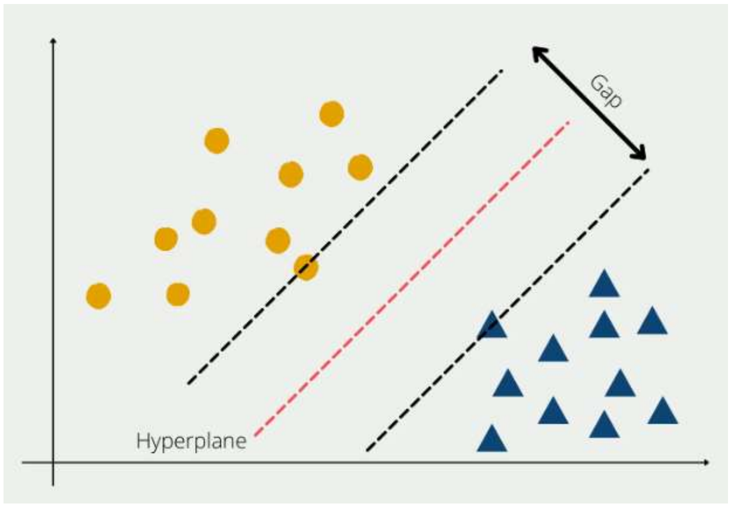
Fundamentals of Machine Learning
Lab Assignment – 6

Name: Niranjana J
USN: 22BTRAD027
Branch: CSE- AI & DE

Implementing Support Vector Machine (SVM) algorithm

- Advanced supervised machine learning machine learning algorithms like Support Vector Machine (SVM) are utilized for tasks including regression, outlier identification, and linear or nonlinear classification.
- Text classification, picture classification, handwriting recognition, spam detection, face detection, gene expression analysis, and anomaly detection are just a few of the many applications for SVMs.
- Because SVMs can handle high-dimensional data and nonlinear connections, they are versatile and effective in a wide range of applications.
- **The SVM algorithm's primary goal is to locate the best hyperplane in an N-dimensional space that may be used to divide data points into various feature space classes.**
- The hyperplane attempts to maintain the largest feasible buffer between the nearest points of various classes.
- The number of features determines the hyperplane's dimension.
- The hyperplane is essentially a line if there are just two input characteristics.
- The hyperplane transforms into a 2-D plane if there are three input characteristics.
- If there are more than three characteristics, it gets hard to envision.



In this example, we are working with the load_breast_cancer dataset that determines whether a tumour size is malignant or benign.

Code

```
#Import scikit-learn dataset library
from sklearn import datasets

#Load dataset
df = datasets.load_breast_cancer()
```

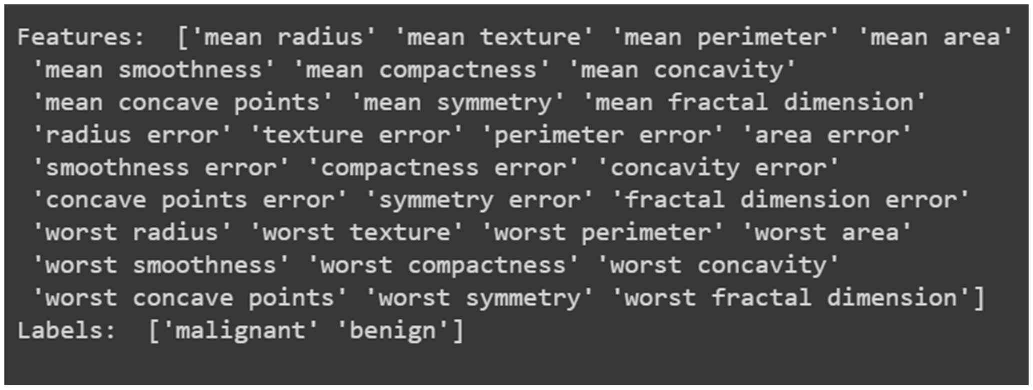
Explanation

To begin with, we are loading the dataset we are going to load from the datasets module in the scikit-learn library. A free machine learning library for the Python programming language is called scikit-learn. From the mentioned library, using load_breast_cancer() function, the required dataset is loaded into the variable df.

Code

```
# print the names of the 13 features
print("Features: ", df.feature_names)

# print the label type of cancer('malignant' 'benign')
print("Labels: ", df.target_names)
```

Output



```
Features:  ['mean radius' 'mean texture' 'mean perimeter' 'mean area'
 'mean smoothness' 'mean compactness' 'mean concavity'
 'mean concave points' 'mean symmetry' 'mean fractal dimension'
 'radius error' 'texture error' 'perimeter error' 'area error'
 'smoothness error' 'compactness error' 'concavity error'
 'concave points error' 'symmetry error' 'fractal dimension error'
 'worst radius' 'worst texture' 'worst perimeter' 'worst area'
 'worst smoothness' 'worst compactness' 'worst concavity'
 'worst concave points' 'worst symmetry' 'worst fractal dimension']
Labels:  ['malignant' 'benign']
```

Explanation

Once the dataset is loaded, we will start the operations on the dataset. In the below code snippet, the remark in the first line gives a quick explanation of what the code does. It tells us that the names of the 13 characteristics that are in the dataset will be printed.

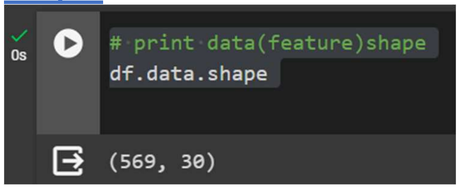
The actual code that outputs the feature names is found in the second line. The result of df.feature_names is shown after the text "Features: " using the print()

method. The dataset object is denoted by df in this instance, and the feature names are stored in feature_names, an attribute.

Code

print data(feature)shape
df.data.shape

Output



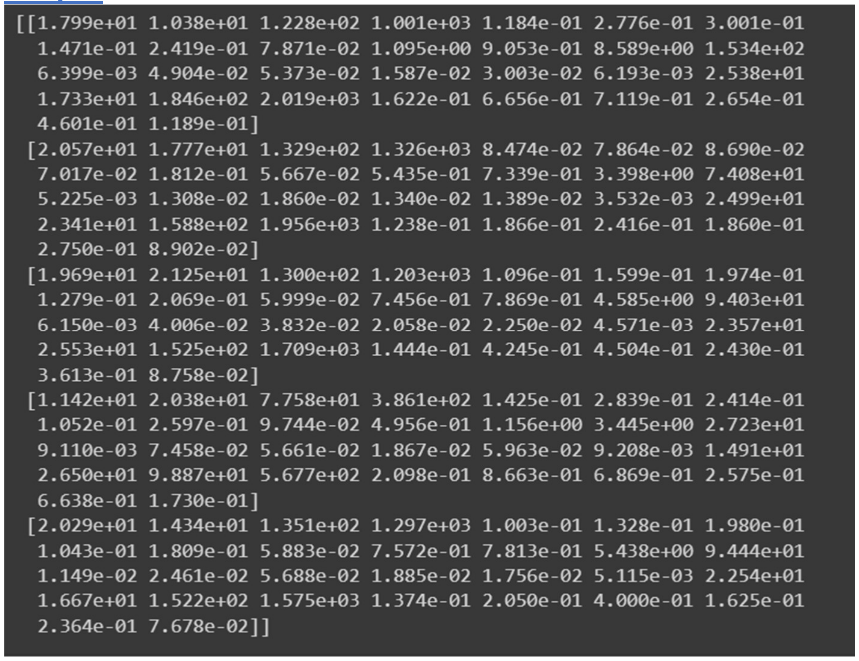
```
# print data(feature)shape
df.data.shape

(569, 30)
```

The shape of the datset can be found using the .shape function. Here the dataset has 569 rows and 30 columns (attributes).

Code

# print the cancer data features (top 5 records)
print(df.data[:5])

Output



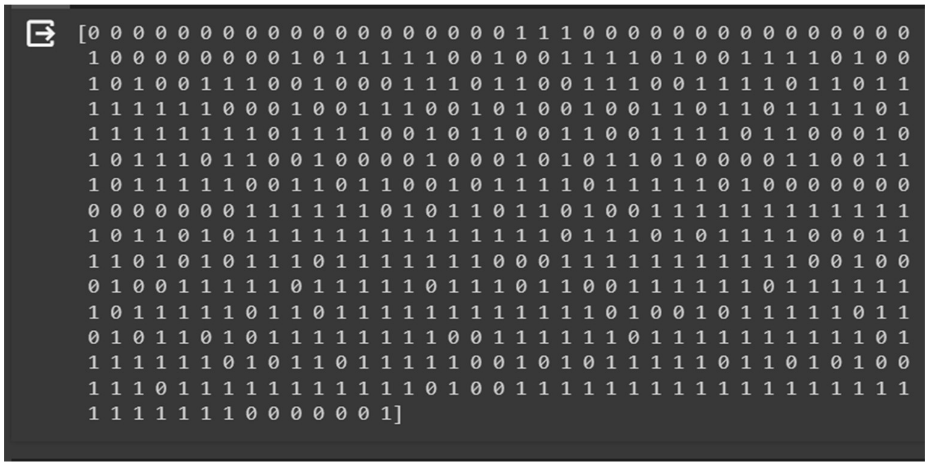
```
[[1.799e+01 1.038e+01 1.228e+02 1.001e+03 1.184e-01 2.776e-01 3.001e-01
  1.471e-01 2.419e-01 7.871e-02 1.095e+00 9.053e-01 8.589e+00 1.534e+02
  6.399e-03 4.904e-02 5.373e-02 1.587e-02 3.003e-02 6.193e-03 2.538e+01
  1.733e+01 1.846e+02 2.019e+03 1.622e-01 6.656e-01 7.119e-01 2.654e-01
  4.601e-01 1.189e-01]
 [2.057e+01 1.777e+01 1.329e+02 1.326e+03 8.474e-02 7.864e-02 8.690e-02
  7.017e-02 1.812e-01 5.667e-02 5.435e-01 7.339e-01 3.398e+00 7.408e+01
  5.225e-03 1.308e-02 1.860e-02 1.340e-02 1.389e-02 3.532e-03 2.499e+01
  2.341e+01 1.588e+02 1.956e+03 1.238e-01 1.866e-01 2.416e-01 1.860e-01
  2.750e-01 8.902e-02]
 [1.969e+01 2.125e+01 1.300e+02 1.203e+03 1.096e-01 1.599e-01 1.974e-01
  1.279e-01 2.069e-01 5.999e-02 7.456e-01 7.869e-01 4.585e+00 9.403e+01
  6.150e-03 4.006e-02 3.832e-02 2.058e-02 2.250e-02 4.571e-03 2.357e+01
  2.553e+01 1.525e+02 1.709e+03 1.444e-01 4.245e-01 4.504e-01 2.430e-01
  3.613e-01 8.758e-02]
 [1.142e+01 2.038e+01 7.758e+01 3.861e+02 1.425e-01 2.839e-01 2.414e-01
  1.052e-01 2.597e-01 9.744e-02 4.956e-01 1.156e+00 3.445e+00 2.723e+01
  9.110e-03 7.458e-02 5.661e-02 1.867e-02 5.963e-02 9.208e-03 1.491e+01
  2.650e+01 9.887e+01 5.677e+02 2.098e-01 8.663e-01 6.869e-01 2.575e-01
  6.638e-01 1.730e-01]
 [2.029e+01 1.434e+01 1.351e+02 1.297e+03 1.003e-01 1.328e-01 1.980e-01
  1.043e-01 1.809e-01 5.883e-02 7.572e-01 7.813e-01 5.438e+00 9.444e+01
  1.149e-02 2.461e-02 5.688e-02 1.885e-02 1.756e-02 5.115e-03 2.254e+01
  1.667e+01 1.522e+02 1.575e+03 1.374e-01 2.050e-01 4.000e-01 1.625e-01
  2.364e-01 7.678e-02]]
```

The code given above prints out top 5 rows of the dataset.

Code

# print the cancer labels (0:malignant, 1:benign)
print(df.target)

Output



```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 1 0 0 0 0 0 0 0 0 1 0 1 1 1 1 1 0 0 1 0 0 1 1 1 1 0 1 0 0 1 1 1 1 0 1 0 0
 1 0 1 0 0 1 1 1 0 0 1 0 0 0 1 1 1 0 1 1 0 0 1 1 1 0 0 1 1 1 1 0 1 1 0 1 1
 1 1 1 1 1 1 0 0 0 1 0 0 1 1 1 0 0 1 0 1 0 0 1 0 0 1 1 0 1 1 0 1 1 1 1 0 1
 1 1 1 1 1 1 1 1 0 1 1 1 1 0 0 1 0 1 1 0 0 1 1 0 0 1 1 1 1 0 1 1 0 0 0 1 0
 1 0 1 1 1 0 1 1 0 0 1 0 0 0 0 1 0 0 0 1 0 1 0 1 1 0 1 0 0 0 0 1 1 0 0 1 1
 1 0 1 1 1 1 1 0 0 1 1 0 1 1 0 0 1 0 1 1 1 1 0 1 1 1 1 1 0 1 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 1 1 1 1 1 1 0 1 0 1 1 0 1 1 0 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1
 1 0 1 1 0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 0 1 0 1 1 1 1 0 0 0 1 1
 1 1 0 1 0 1 0 1 1 1 0 1 1 1 1 1 1 1 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 1 0 0
 0 1 0 0 1 1 1 1 1 0 1 1 1 1 1 0 1 1 1 0 1 1 0 0 1 1 1 1 1 1 0 1 1 1 1 1 1
 1 0 1 1 1 1 1 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 0 1 0 0 1 0 1 1 1 1 1 0 1 1
 0 1 0 1 1 0 1 0 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 0 1
 1 1 1 1 1 1 0 1 0 1 1 0 1 1 1 1 1 0 0 1 0 1 0 1 1 1 1 1 0 1 1 0 1 0 1 0 0
 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 0 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 0 0 0 0 0 0 1]
```

Explanation

To printout the labels of the dataset, target function is used where 0 represents that the tumour is malignant and 1 represents that the tumour is benign.

Code

```
# Import train_test_split function
from sklearn.model_selection import train_test_split

# Split dataset into training set and test set
X_train, X_test, y_train, y_test = train_test_split(df.data, df.target,
test_size=0.3,random_state=109) # 70% training and 30% test
```

Explanation

The given code sample shows how to use the train_test_split function to divide a dataset into training and test sets. Open the sklearn.model_selection module and import the train_test_split function. Pass the target labels (y) and input features (X) as parameters when using the train_test_split method. Use the test_size argument to specify the split ratio. Test_size=0.3, for instance, allots 30% of the data to the test set. Set the random_state argument if you want to make sure the split may be repeated.

Code

```
#Import svm model
from sklearn import svm

#Create a svm Classifier
clf = svm.SVC(kernel='linear') # Linear Kernel

#Train the model using the training sets
clf.fit(X_train, y_train)
```

```
#Predict the response for test dataset
y_pred = clf.predict(X_test)
```

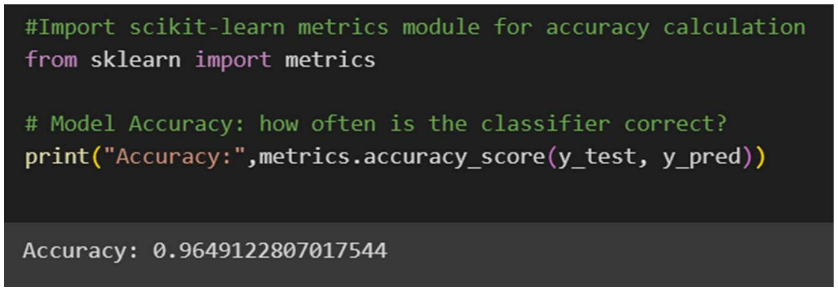
Explanation

The SVM module from the scikit-learn library is imported. By initializing the svm.SVC class with the kernel='linear' argument, we establish an instance of the SVM classifier with a linear kernel. X_train and y_train are the training sets we use to train the SVM classifier. The input characteristics are represented by the X_train variable, while the matching class labels are represented by y_train. We forecast the class labels for the test dataset X_test using the training model. The y_pred variable contains the expected labels.

Code

```
#Import scikit-learn metrics module for accuracy calculation
from sklearn import metrics

# Model Accuracy: how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

Output



```
#Import scikit-learn metrics module for accuracy calculation
from sklearn import metrics

# Model Accuracy: how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.9649122807017544
```

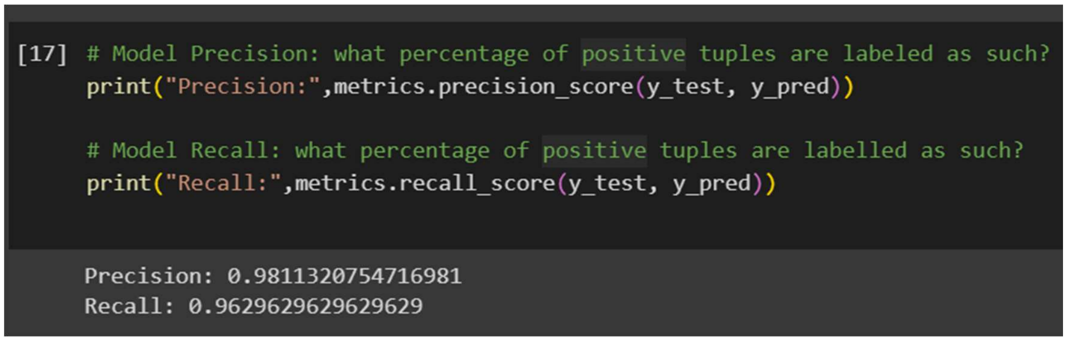
Explanation

The metrics module is imported from the scikit-learn package. This module offers a range of indicators for assessing how well machine learning models work. The accuracy is then determined by supplying the real labels (y_test) and predicted labels (y_pred) as parameters to the accuracy_score function. We print the accuracy score at the end.

Code

```
# Model Precision: what percentage of positive tuples are labeled as such?
print("Precision:",metrics.precision_score(y_test, y_pred))

# Model Recall: what percentage of positive tuples are labelled as such?
print("Recall:",metrics.recall_score(y_test, y_pred))
```

Output



```
[17] # Model Precision: what percentage of positive tuples are labeled as such?
     print("Precision:",metrics.precision_score(y_test, y_pred))

     # Model Recall: what percentage of positive tuples are labelled as such?
     print("Recall:",metrics.recall_score(y_test, y_pred))
```

```
Precision: 0.9811320754716981
Recall: 0.9629629629629629
```

Explanation

The sklearn.metrics module's precision_score and recall_score methods may be used to compute precision and recall in Python. These routines return the accuracy and recall scores, respectively, after receiving as inputs the real labels (y_test) and predicted labels (y_pred).

References

- https://databasecamp.de/en/ml/svm-explained
- https://www.datacamp.com/tutorial/svm-classification-scikit-learn-python
- https://www.geeksforgeeks.org/support-vector-machine-algorithm/

Github link:
https://github.com/niranjana628/ML