

Fundamentals of Machine Learning Lab Assignment-6

Name:Hitha Choudhary G

USN:22BTRAD015

Branch:CSE in AI&DE

Implementing SVM algorithm.

- Strong 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 relationships, they are versatile and effective in a wide range of applications.
- A supervised machine learning approach called Support Vector Machine (SVM) is used for regression as well as classification. Even yet, classification problems are the most appropriate use for regression problems.
- 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 possible buffer between the nearest points of various classes.

SVM Kernels

In real-world applications, the SVM algorithm is carried out using a kernel. An input data space is transformed into the necessary form using a kernel.

By introducing additional dimensions, it transforms non separable issues into separable problems. It works best in problems with non-linear separation.

Using a kernel trick, you can create a classifier that is more accurate.

Linear Kernel: Any two supplied observations can be normalized to a dot product using a linear kernel. The total of the multiplications of each pair of input values is the product between two vectors.

Polynomial Kernel: An expanded version of the linear kernel is called a polynomial kernel. Curved or nonlinear input space can be distinguished using the polynomial kernel.

Classifier Building in Scikit-Learn

You can use the cancer dataset, a well-known multi-class classification problem, in the model's building section.

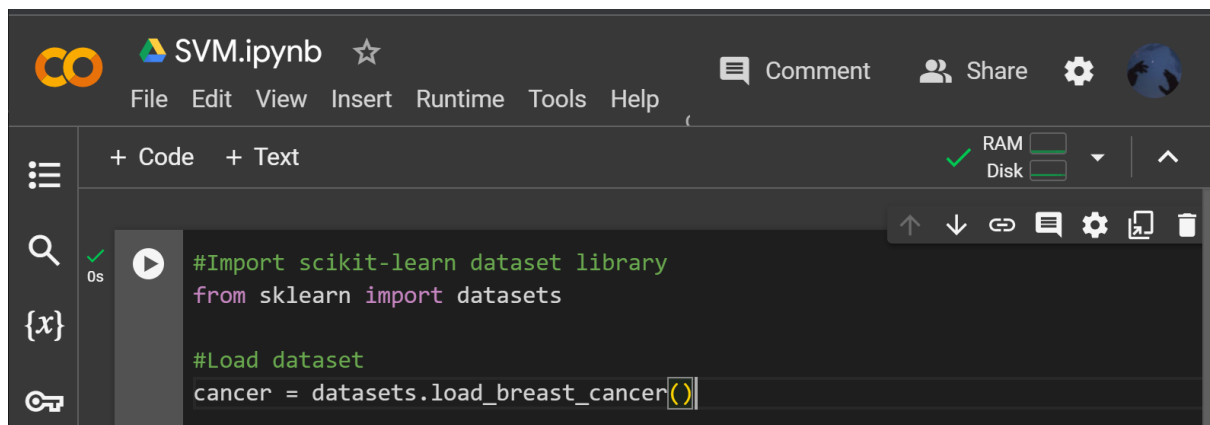
There are two categories of cancer in this data: benign (not harmful) and malignant (destructive). You can construct a model to categorize the kind of cancer here.

Loading the Data: We will first load the required dataset.

Code:

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

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

A screenshot of a Jupyter Notebook interface. The top bar shows the file name 'SVM.ipynb' and various icons for file operations, comment, share, settings, and a globe. Below the top bar is a menu bar with 'File', 'Edit', 'View', 'Insert', 'Runtime', 'Tools', and 'Help'. The main area of the notebook shows a code cell with the following text:

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

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

 The code is written in a dark-themed editor with syntax highlighting. On the left side of the code cell, there is a search icon, a play button, and a variable inspector icon. On the right side, there are icons for up/down arrows, a link, a message, a settings gear, and a trash can. The RAM and Disk usage is shown in the top right corner of the code cell.

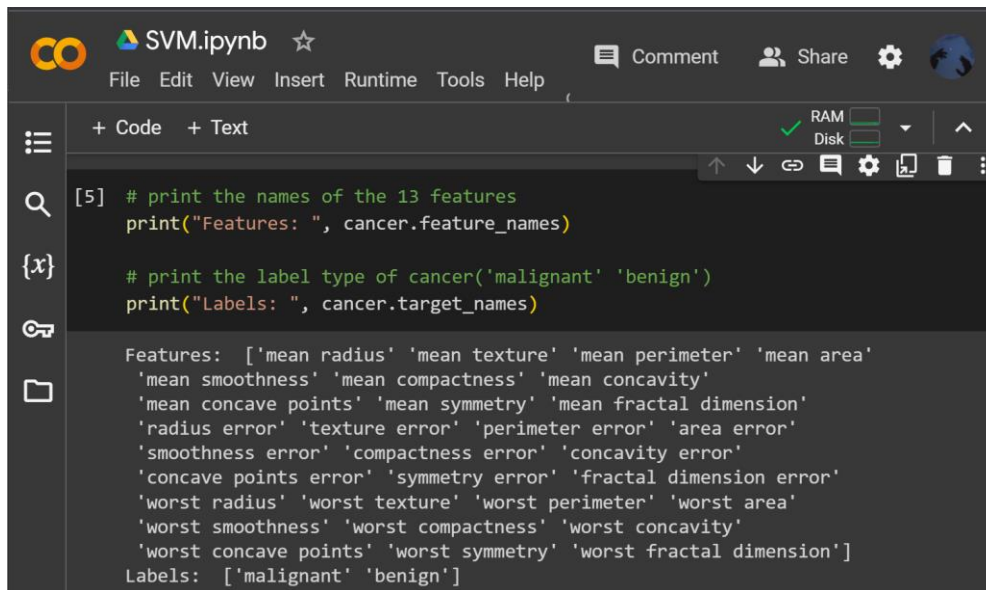
Exploring the Data: We will now check the features and target names of the data.

Code:

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

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

Output:



```
SVM.ipynb
File Edit View Insert Runtime Tools Help

[5] # print the names of the 13 features
    print("Features: ", cancer.feature_names)

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

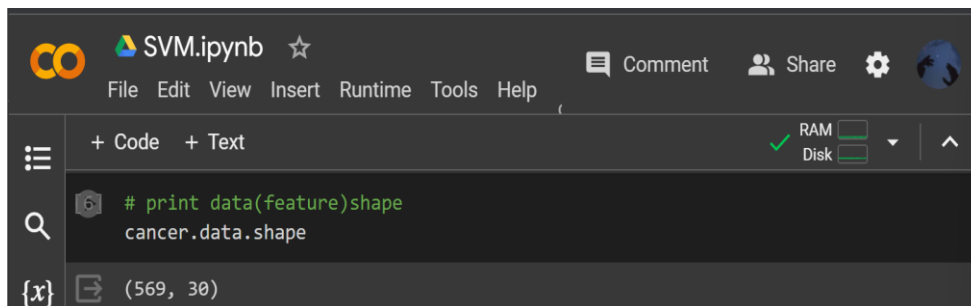
Features: ['mean radius' 'mean texture' 'mean perimeter' 'mean area'
          'mean smoothness' 'mean compactness' 'mean concavity'
          'mean concave points' 'mean symmetry' 'mean fractal dimension'
          'radius error' 'texture error' 'perimeter error' 'area error'
          'smoothness error'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

The dataset comprises of 30 features and a target(type of cancer).

Code:

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

```
cancer.data.shape
```



```
SVM.ipynb
File Edit View Insert Runtime Tools Help

# print data(feature)shape
cancer.data.shape

(569, 30)
```

We can also get the shape of the dataset using “.data.shape”.

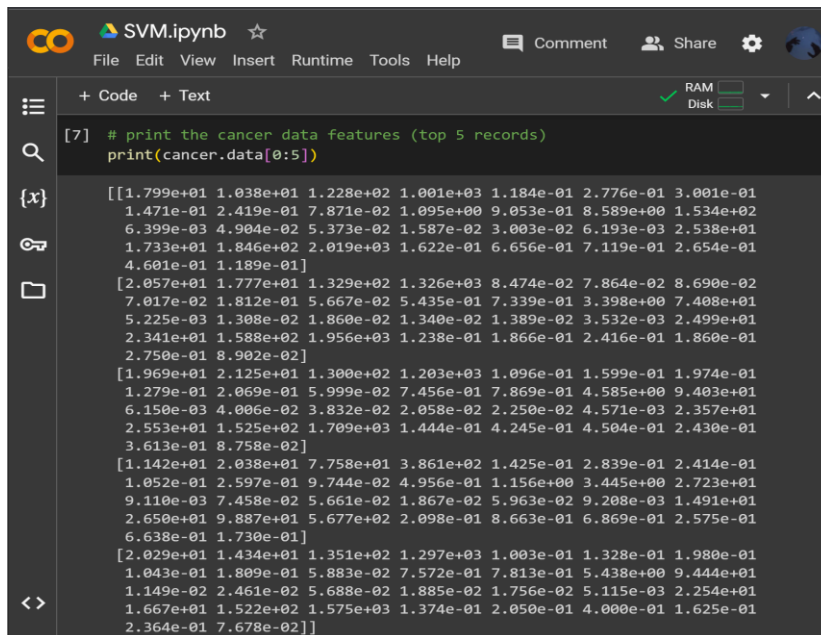
Next, let’s check the top 5 records of the feature set.

Code:

```
# print the cancer data features (top 5 records)
```

```
print(cancer.data[0:5])
```

Output:



```
[7] # print the cancer data features (top 5 records)
print(cancer.data[0:5])
```

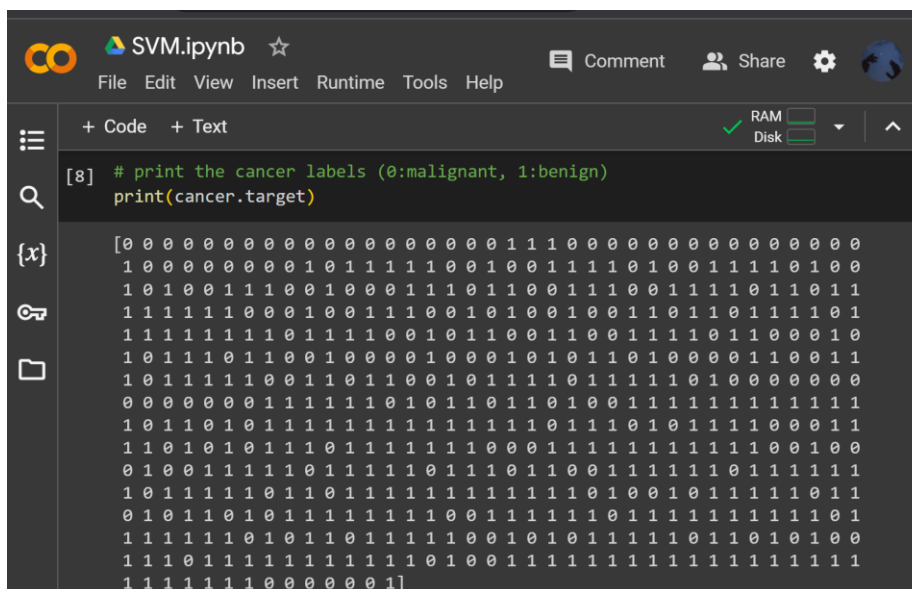
```
[[1.799e+01 1.038e+01 1.228e+02 1.001e+03 1.184e-01 2.776e-01 3.001e-01
 1.471e-01 2.419e-01 7.871e-02 1.095e+00 9.053e-01 8.589e+00 1.534e+02
 6.399e-03 4.904e-02 5.373e-02 1.587e-02 3.003e-02 6.193e-03 2.538e+01
 1.733e+01 1.846e+02 2.019e+03 1.622e-01 6.656e-01 7.119e-01 2.654e-01
 4.601e-01 1.189e-01]
[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]]
```

Taking a look at the target set.

Code:

```
# print the cancer labels (0:malignant, 1:benign)
```

```
print(cancer.target)
```



```
[8] # print the cancer labels (0:malignant, 1:benign)
print(cancer.target)
```

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

Splitting the Data: Partitioning the dataset into training and test sets is a sensible approach to comprehend model performance.

Utilizing the `train_test_split()` function, divide the dataset. Three options must be passed: `test_set` size, `target`, and `features`. Furthermore, you can choose records at random by using `random_state`.

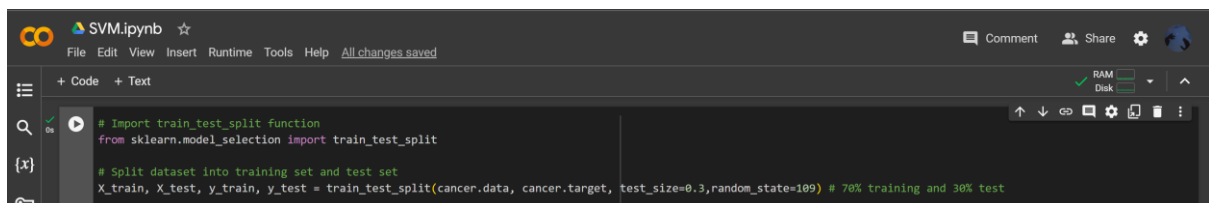
Code:

```
# 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(cancer.data, cancer.target,
test_size=0.3, random_state=109) # 70% training and 30% test
```



Generating Model: The support vector machine model needs to be built. To create a support vector classifier object, first import the SVM module. Then, use the `SVC()` function to supply the linear kernel as an input.

Next, use `fit()` to fit your model on the train set and `predict()` to make predictions on the test set.

Code:

```
#Import svm model
```

```
from sklearn import svm
```

```
#Create a svm Classifier
```

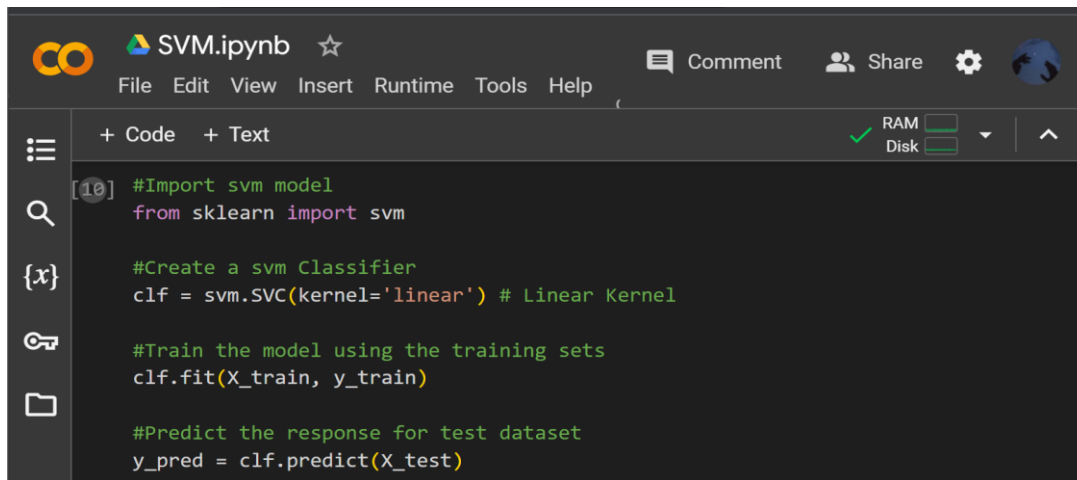
```
clf = svm.SVC(kernel='linear') # Linear Kernel
```

```
#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)
```

A screenshot of a Jupyter Notebook interface. The top bar shows the Colab logo, the notebook name 'SVM.ipynb', and a star icon. Below the bar are tabs for 'File', 'Edit', 'View', 'Insert', 'Runtime', 'Tools', and 'Help'. On the right, there are icons for 'Comment', 'Share', 'Settings', and a globe. The left sidebar contains icons for a menu, search, variables, keys, and files. The main area shows a code cell with the following Python code:

```
[10] #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)
```

Evaluating the Model: Let's calculate the classifier or model's predictive accuracy for breast cancer in patients.

By contrasting the actual test set values with the projected values, accuracy may be calculated.

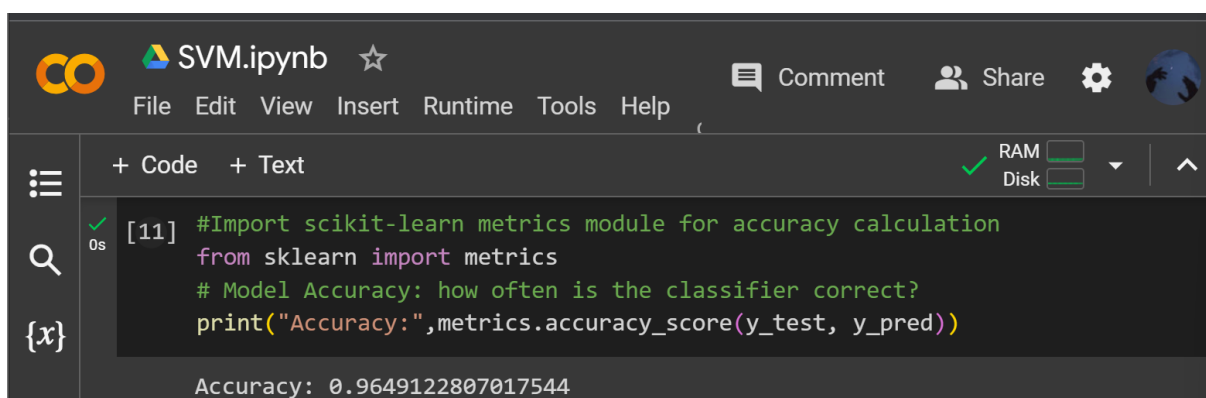
Code:

```
#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:

A screenshot of a Jupyter Notebook interface showing the execution of the accuracy calculation code. The top bar and sidebar are the same as the previous screenshot. The code cell is now labeled '[11]' and contains the same code as before. Below the code cell, the output is displayed: 'Accuracy: 0.9649122807017544'. The RAM and Disk usage indicators are visible in the top right corner of the code cell area.

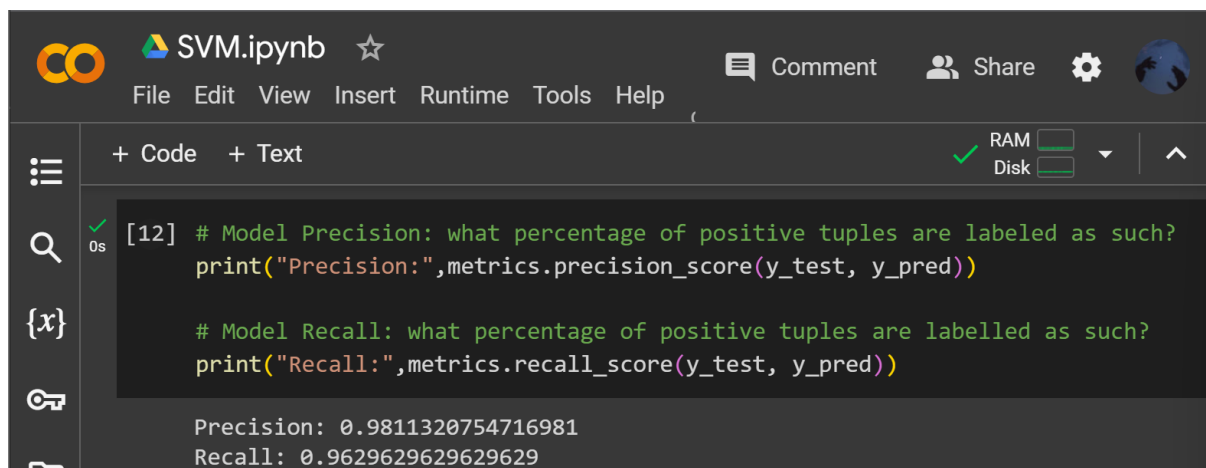
We got a classification rate of 96.49% which is considered as a very good accuracy. For further evaluation we will also be checking the precision and recall of the model.

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:

A screenshot of a Jupyter Notebook interface. The top bar shows the 'SVM.ipynb' file name and various icons for commenting, sharing, and settings. Below the top bar is a menu bar with 'File', 'Edit', 'View', 'Insert', 'Runtime', 'Tools', and 'Help'. The main area is divided into a left sidebar with icons for file explorer, search, and other functions, and a central code editor. The code editor shows two lines of Python code: a comment and a print statement for precision, followed by another comment and a print statement for recall. Below the code, the output is displayed: 'Precision: 0.9811320754716981' and 'Recall: 0.9629629629629629'. The interface also shows RAM and Disk usage indicators in the top right corner of the code editor area.

```
[12] # 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
```

Here we got a precision of 98% and recall of 96% which are pretty accurate values.

Advantages and Disadvantages

- They only employ a portion of the training points during the choice phase, they also consume less memory. SVM works best with a clear margin of separation and with high dimensional space.
- Due to its lengthy training period and high training time, SVM is not appropriate for huge datasets. It is also sensitive to the kind of kernel being used, and performs poorly with overlapping classes.

Github Link: <https://github.com/hithachoudhary/machinelearning>