ML Practical’s

Name : Pruthvi Rathod

Division: B **Roll No.: 45**

Subject: Machine Learning Practical Assignments.

Practical No. 1

Aim: To perform the cross validation techniques.

Theory:

Cross validation is a technique used in machine learning to evaluate the performance of a model on unseen dataThe main purpose of cross validation is to prevent overfitting, which occurs when a model is trained too well on the training data and performs poorly on new, unseen data. By evaluating the model on multiple validation sets, cross validation provides a more realistic estimate of the model’s generalization performance, i.e., its ability to perform well on new, unseen data.

There are several types as follow:

1. K-fold cross-validation
2. Holdout cross-validation
3. Stratified k-fold cross-validation
4. Leave-P-out cross validation
5. Leave one out
6. Monte Carlo Cross Validation
7. Time Series

Code and Output:

1. K-fold cross-validation

Code -

```
from sklearn.datasets import load_iris
from sklearn.model_selection import cross_val_score, KFold
from sklearn.linear_model import LogisticRegression
iris=load_iris()
X=iris.data
Y=iris.target
lr=LogisticRegression()
k_fold= KFold (n_splits=7)
```

```
score = cross_val_score(lr,X,Y,cv=k_fold)
print("Cross Validation Scores: {}".format(score))
print("Average Cross Validation {}".format(score.mean()))
```

Output -

```
Cross Validation Scores: [1.         1.         1.
0.80952381 0.95238095 0.85714286
0.9047619 ]
Average Cross Validation 0.9319727891156463
```

2. Holdout cross-validation

Code -

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

Irls= load_iris()
X=iris.data
Y=iris.target
Ir=LogisticRegression()
x_train, x_test, y_train , y_test=train_test_split(X, Y,
test_size=0.25, random_state=42)
lr.fit(x_train,y_train)
results=lr.predict(x_test)
print ("Training
accuracy:{}".format(accuracy_score(lr.predict(x_train),y_train)))
print("Testing
accuracy:{}".format(accuracy_score(results,y_test)))
```

Output -

```
Training accuracy:0.9642857142857143
Testing accuracy:1.0
```

3. Stratified k-fold cross-validation

Code -

```
from sklearn.model_selection import ShuffleSplit,cross_val_score
from sklearn.datasets import load_iris
from sklearn.linear_model import LogisticRegression
Ir=LogisticRegression()
shuffle_split_cv=ShuffleSplit(test_size=0.3,
train_size=0.5,n_splits=6)
```

```
scores=cross_val_score(lr,iris.data,iris.target,cv=shuffle_split_
cv)
print("Cross Validation Scores:{}".format(scores))
print("Average Cross Validation score:{}".format(scores .mean()))
```

Output -

```
Cross Validation Scores:[0.97777778 1.         0.91111111 0.
97777778 0.95555556 0.95555556]
Average Cross Validation score:0.9629629629629629
```

4. Leave-P-out cross validation

Code -

```
from sklearn.datasets import load_iris
from sklearn.model_selection import cross_val_score,
StratifiedKFold
from sklearn.linear_model import LogisticRegression
iris=load_iris()
X=iris.data
Y=iris.target
Ir=LogisticRegression()
st_kf = StratifiedKFold(n_splits=3)
score=cross_val_score(lr,X,Y,cv=st_kf)
print("Cross Validation Scores:{}".format(score))
print("Average Cross Validation score:{}".format(score.mean()))
```

Output -

```
Cross Validation Scores:[0.98 0.96 0.98]
Average Cross Validation score:0.9733333333333333
```

5. Leave one out

Code -

```
from sklearn.model_selection import LeavePOut,cross_val_score
from sklearn.datasets import load_iris
from sklearn.ensemble import RandomForestClassifier
iris=load_iris()
X=iris.data
Y=iris.target
lpo=LeavePOut(p=1)
lpo.get_n_splits(X)
tree= RandomForestClassifier(n_estimators=5,max_depth=3,n_jobs=-
1)
score =cross_val_score(tree,X,Y,cv=lpo)
print("Cross Validation Scores-{}".format(score))
print("Average Cross Validation score:{}".format(score.mean()))
```

Output -

```
Cross Validation Scores-[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. 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. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
0. 1.
 0. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
1. 0.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 0. 0. 1. 1. 1. 1. 1. 1. 1.
1. 1.
 1. 1. 1. 1. 1. 1.]
Average Cross Validation score:0.94
```

6. Monte Carlo Cross Validation

Code –

```
from sklearn.datasets import load_iris
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import LeaveOneOut, cross_val_score
iris=load_iris()
X=iris.data
Y=iris.target
leave_one_out=LeaveOneOut()
rfc=RandomForestClassifier(n_estimators=7,max_depth=4,n_jobs=-1)
score=cross_val_score(rfc,X,Y,cv=leave_one_out)
print("Cross Validation Scores:{}".format(score))
print("Average Cross Validation score:{}".format(score.mean()))
```

Output –

```
Cross Validation Scores:[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. 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. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
0. 1.
 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
1. 0.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 1. 1. 1. 0. 1. 1. 1.
1. 1.
 1. 1. 1. 1. 1. 1.]
Average Cross Validation score:0.94
```

7. Time Series

Code –

```
import numpy as np
from sklearn.model_selection import TimeSeriesSplit
```

```
X = np.array([[1,2],[3,4],[1,2],[3,4],[1,2],[3,4],[1,1],[2,2]])
Y = np.array([1,2,3,4,5,6,7,8])
tscv = TimeSeriesSplit(n_splits=7)
print(tscv)
i=1
for train_index, test_index in tscv.split(X):
  print(f"TRAIN {i}:",train_index,"TEST:",test_index)
  X_train, X_test = X[train_index], X[test_index]
  y_train, y_test =Y[train_index], Y[test_index]
  i+=1
```

<u>Output-</u>

```
TimeSeriesSplit(gap=0, max_train_size=None, n_splits=7, test_size=No
ne)
TRAIN 1: [0] TEST: [1]
TRAIN 2: [0 1] TEST: [2]
TRAIN 3: [0 1 2] TEST: [3]
TRAIN 4: [0 1 2 3] TEST: [4]
TRAIN 5: [0 1 2 3 4] TEST: [5]
TRAIN 6: [0 1 2 3 4 5] TEST: [6]
TRAIN 7: [0 1 2 3 4 5 6] TEST: [7]
```

Practical No. 2

Aim: To perform the K- Nearest Neigbhors Algorithm

Theory:

K-Nearest Neighbours is one of the most basic yet essential classification algorithms in Machine Learning. It belongs to the supervised learning domain and finds intense application in pattern recognition, data mining, and intrusion detection.

Distance metrics used in KNN

- Euclidean Distance
- Manhattan Distance
- Minkowski Distance.

How to choose the value of k for KNN Algorithm?

The value of k is very crucial in the KNN algorithm to define the number of neighbors in the algorithm. The value of k in the k-nearest neighbors (k-NN) algorithm should be chosen based on the input data. If the input data has more outliers or noise, a higher value of k would be better. It is recommended to choose an odd value for k to avoid ties in classification. Cross-validation methods can help in selecting the best k value for the given dataset.

Code & Output:

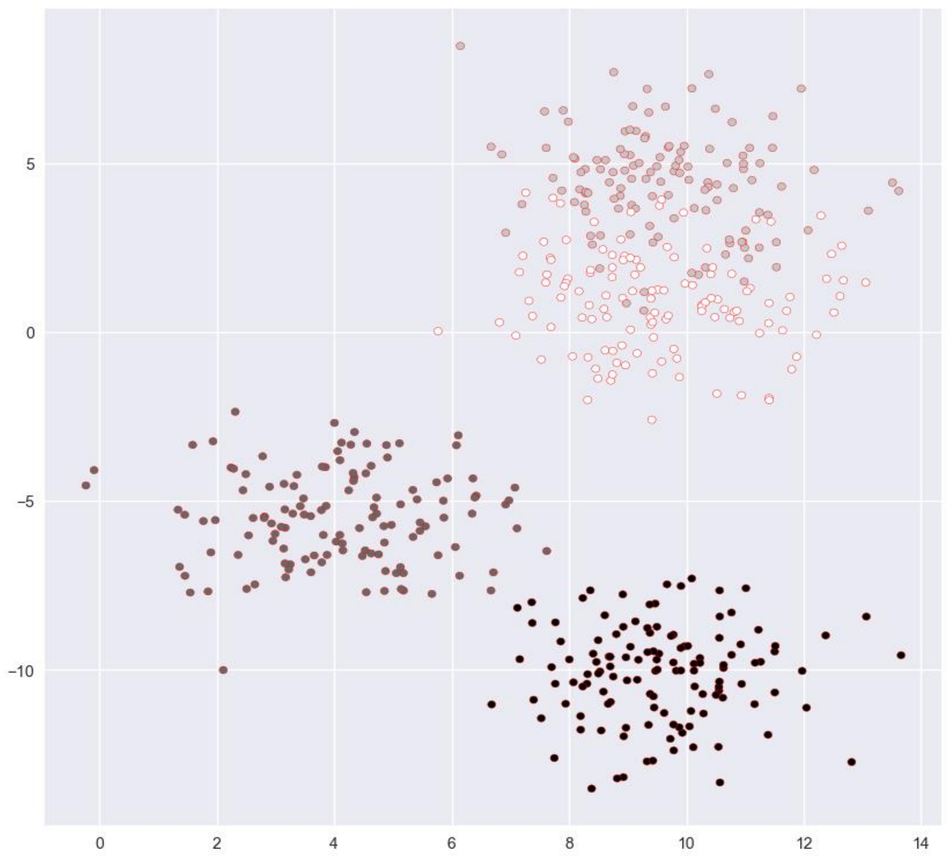
Code -

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
from sklearn.neighbors import KneighborsClassifier
from sklearn.model_selection import train_test_split
x,y=make_blobs(n_samples=500,n_features=2,centers=4,cluster_std=1
.5, random_state=4)
plt.style.use("seaborn")
plt.figure(figsize=(10,10))
plt.scatter(x[:,0],x[:,1],c=y,marker='.',s=100,edgecolor='red')
plt.show()
x_train, x_test, y_train,
y_test=train_test_split(x,y,random_state=0)
knn5=KneighborsClassifier(n_neighbors=5)
knn1=KneighborsClassifier(n_neighbors=1)
knn5.fit(x_train,y_train)
knn1.fit(x_train,y_train)
y_pred_5=knn5.predict(x_test)
y_pred_1=knn1.predict(x_test)
from sklearn.metrics import accuracy_score
```

```
print(“Accuracy score with 5:
“,accuracy_score(y_test,y_pred_5)*100)
print(“Accuracy score with 1:
“,accuracy_score(y_test,y_pred_1)*100)
```

Output -

```
Accuracy score with 5:  93.60000000000001
Accuracy score with 1:  90.4
```



Practical No. 3

Aim: To perform the simple linear regression

Theory:

Linear regression is a type of supervised machine learning algorithm that computes the linear relationship between a dependent variable and one or more independent features. When the number of the independent feature, is 1 then it is known as Univariate Linear regression, and in the case of more than one feature, it is known as multivariate linear regression. The goal of the algorithm is to find the best linear equation that can predict the value of the dependent variable based on the independent variables. The equation provides a straight line that represents the relationship between the dependent and independent variables. The slope of the line indicates how much the dependent variable changes for a unit change in the independent variable(s).

Code & Output:

Code-

```
import numpy as np
from sklearn.linear_model import LinearRegression
x=np.array([5,15,25,35,45,55]).reshape((-1,1))
y=np.array([5, 20, 14, 32,22,38])
model=LinearRegression()
model.fit(x,y)
r_sq=model.score(x,y)
print (f” coefficient of determination: {r_sq}”)
print (f”intercept: {model.intercept_}”)
print (f”scope: {model.coef_}”)
y_pred=model.predict(x)
print (f”predicted response: \n{y_pred}”)
y_pred=model.intercept_ + model.coef_*x
print (f”predicted response: \n{y_pred}”)
```

Output -

```
coefficient of determination: 0.7158756137479542
intercept: 5.633333333333329
scope: [0.54]
predicted response:
[ 8.33333333 13.73333333 19.13333333 24.53333333 29.93333333 3
5.33333333]
predicted response:
[[ 8.33333333][13.73333333][19.13333333]
[24.53333333][29.93333333][35.33333333]]
```

Practical No. 4

Aim: To perfom the multiple linear regression

Code & Output:

Code -

```
import numpy as np
from sklearn.linear_model import LinearRegression
x=[[0,1],[5,1],[15,2],[25,5],[35,11],[45,15],[55,18]]
y=[4,5,20,14,32,22,38]
x,y = np.array(x),np.array(y)
model= LinearRegression().fit(x,y)
r_sq = model.score(x,y)
print(f"Coeffiecent of determinantion : {r_sq}")
print(f"Intercept : {model.intercept_}")
print(f"Score : {model.coef_}")
y_pred = model.predict(x)
print(f"Predicted response : \t{y_pred}")
```

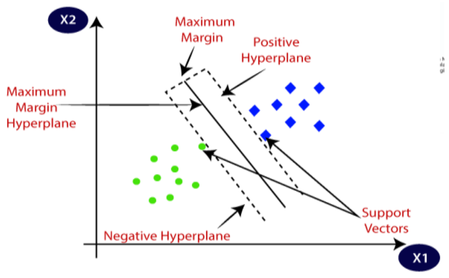
Output-

```
Coeffiecent of determinantion : 0.7979631287564747
Intercept : 4.355292792792792
Score : [ 0.75056306 -0.57713964]
Predicted response :  [ 3.77815315  7.53096847 14.45945946 20.23367117 2
4.27646396 29.47353604
 35.24774775]
```

Practical No. 5

Aim: To perform the SVM algorithm on given dataset (Social Networks Ads)

Theory:
Support Vector Machine(SVM) is a supervised machine learning algorithm used for both classification and regression. Though we say regression problems as well it's best suited for classification. The objective of the SVM algorithm is to find a hyperplane in an N-dimensional space that distinctly classifies the data points. The dimension of the hyperplane depends upon the number of features. If the number of input features is two, then the hyperplane is just a line. If the number of input features is three, then the hyperplane becomes a 2-D plane. It becomes difficult to imagine when the number of features exceeds three.



Use Of kernel function
The SVM kernel is a function that takes low-dimensional input space and transforms it into higher-dimensional space, ie it converts nonseparable problems to separable problems. It is mostly useful in non-linear separation problems.

Code & Output:
Code –
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
dataset=pd.read_csv ('Social_Network_ads.csv')
x=dataset.iloc[:,[2,3]].values
y=dataset.iloc[:, 4].values
x_train,x_test,y_train,y_test=train_test_split(x,y, test_size=0.25,
random_state=0)
sc=StandardScaler()
x_train=sc.fit_transform(x_train)
x_test=sc. transform(x_test)
classifier=SVC (kernel='rbf', random_state=0)
classifier.fit (x_train,y_train)
y_pred=classifier.predict(x_test)
```

```
from sklearn.metrics import confusion_matrix, accuracy_score
cm =confusion_matrix(y_test,y_pred)
print (cm)
accuracy_score(y_test,y_pred)
```

Output -

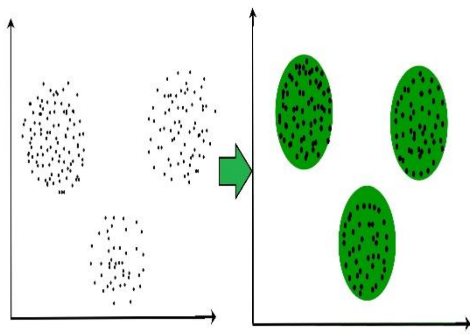
```
[[64  4]
 [ 3 29]]
0.93
```

Practical No. 6

Aim: To perform K Means Clustering algorithm on income dataset.

Theory:

Clustering:- Clustering is the task of dividing the population or data points into a number of groups such that data points in the same groups are more similar to other data points in the same group and dissimilar to the data points in other groups.



K-Means Clustering:- It is an iterative algorithm that divides the unlabeled dataset into k different clusters in such a way that each dataset belongs only one group that has similar properties. It select K centroids, assign each data point to their closest centroid, for K clusters and then calculate the variance and place a new centroid of each cluster.

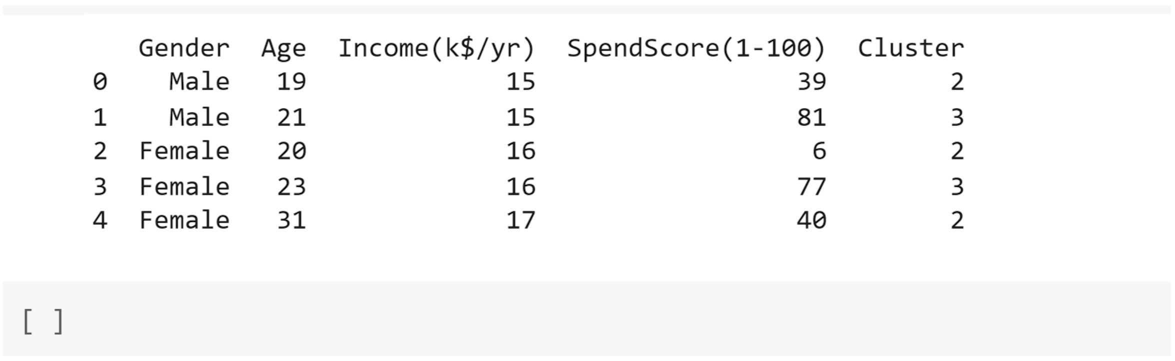
Code & Output:

Code -

```
import numpy as np

import pandas as pd

from sklearn.preprocessing import StandardScaler

from sklearn.preprocessing import MinMaxScaler

from sklearn.cluster import KMeans

from sklearn.cluster import AgglomerativeClustering

df=pd.read_csv("/content/Mall_Customers.csv")

df.head()

df=df.drop(['CustomerID'],axis=1)

df.rename(columns = {'Annual Income (k$)' : 'Income(k$/yr)',
'Spending Score (1-100)' : 'SpendScore(1-100)'}, inplace=True)

X = df[['Income(k$/yr)', 'SpendScore(1-100)']].copy()

X.head()

scaler = StandardScaler()

X_kmeans =  scaler.fit_transform(X)
```

```
wcss = []

for cluster in range(1,11):

    kmeans = KMeans(n_clusters = cluster, init = 'k-means++',
random_state = 42)

    kmeans.fit(X_kmeans)

    wcss.append(kmeans.inertia_)


clusters = KMeans(n_clusters = 5, init = 'k-means++', random_state =
42)

y_kmeans = clusters.fit_predict(X_kmeans)

y_kmeans

df_Kmeans = df.copy()            ## making a copy of original
dataframe

df_Kmeans['Cluster'] = y_kmeans  ## appending the cluster column

print(df_Kmeans.head(5))
```

Output -



```
   Gender  Age  Income(k$/yr)  SpendScore(1-100)  Cluster
0    Male   19             15                 39        2
1    Male   21             15                 81        3
2  Female   20             16                  6        2
3  Female   23             16                 77        3
4  Female   31             17                 40        2
```

```
[ ]
```

Practical No. 7

Aim: Stock Price Prediction using MLPClassifier.

Theory:

Multilayer Perceptron (MLP) is the most fundamental type of neural network architecture when compared to other major types such as Convolutional Neural Network (CNN), Recurrent Neural Network (RNN), Autoencoder (AE) and Generative Adversarial Network (GAN).

In an MLP, perceptrons (neurons) are stacked in multiple layers. Every node on each layer is connected to all other nodes on the next layer. There is no connection between nodes within a single layer. An MLP is a Fully (Densely) Connected Neural Network (FCNN).

Code & Output:

Code –

```
import numpy as np

import pandas as pd

data_train = pd.read_csv("/content/NFLX.csv")

data_train.head()

isna_df = data_train.isnull().sum()

isna_df[isna_df > 0]

print("Max:", isna_df.max())

print(isna_df[isna_df == isna_df.max()])

isna_df.max()/data_train.size

X=data_train.drop(['Date','Adj Close'],axis=1)

y=data_train['Adj Close']

from sklearn.model_selection import train_test_split

x_train,x_test,y_train,y_test=train_test_split(X,y,test_size=0.2)

from sklearn.neural_network import MLPRegressor

regr = MLPRegressor(random_state=1, max_iter=500).fit(x_train,
y_train)

regr.predict(x_test[:2])
```

Output:

```
array([385.00662251, 548.14237429])
```

Practical No. 8

Aim: To perform LSTM algorithm to predict stock price.

Theory:

Long Short-Term Memory (LSTM) networks are a type of recurrent neural network capable of learning order dependence in sequence prediction problems. This is a behavior required in complex problem domains like machine translation, speech recognition, and more. LSTMs are a complex area of deep learning. It can be hard to get your hands around what LSTMs are, and how terms like bidirectional and sequence-to-sequence relate to the field.

The Long Short Term Memory architecture was motivated by an analysis of error flow in existing RNNs which found that long time lags were inaccessible to existing architectures, because backpropagated error either blows up or decays exponentially. An LSTM layer consists of a set of recurrently connected blocks, known as memory blocks. These blocks can be thought of as a differentiable version of the memory chips in a digital computer.

Code & Output:

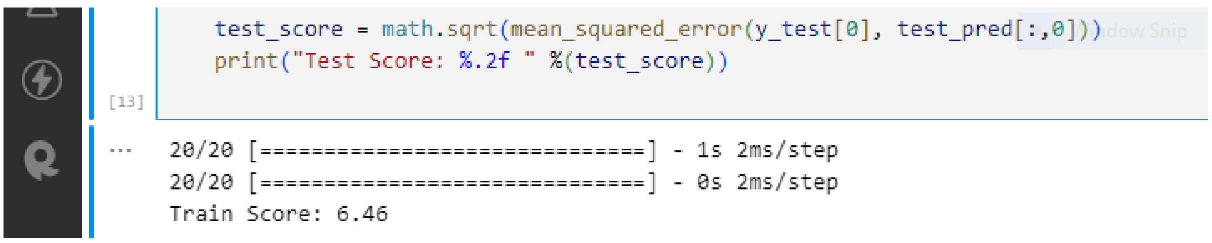
Code -

```
# Import Required Libraries
import math
from keras.models import Sequential
from keras.layers import Dense, LSTM, Dropout
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
import pandas as pd
import numpy as np
# importing the training set
data_train = pd.read_csv("/content/Google_Stock_Price_Train.csv")
data_train.head(10)
# Preprocessing Data
data = data_train.loc[:,["Open"]].values
print(data)
# Reshape
data = data.reshape(-1,1)
data = data.astype("float32")
```

```
data.shape

# scaling

scaler = MinMaxScaler(feature_range=(0,1))

data = scaler.fit_transform(data)

# train test split

train_size = int(len(data) * 0.5)

test_size = len(data) - train_size

train = data[0:train_size, :]

test = data[train_size:len(data), :]

print("train size: {}, test size: {} ".format(len(train),
len(test)))

# timesteps

timesteps = 5

x_data = []

y_data = []

for i in range(len(train) - timesteps -1):

    a = train[i:(i+timesteps), 0]

    x_data.append(a)

    y_data.append(train[i + timesteps, 0])

x_train = np.array(x_data)

y_train = np.array(y_data)

x_data = []

y_data = []

for i in range(len(test) - timesteps -1):

    a = test[i:(i+timesteps), 0]

    x_data.append(a)

    y_data.append(test[i + timesteps, 0])

x_test = np.array(x_data)

y_test = np.array(y_data)
```

```
x_train = np.reshape(x_train, (x_train.shape[0], 1,
x_train.shape[1]))

x_test = np.reshape(x_test, (x_test.shape[0], 1 , x_test.shape[1]))

## Create LSTM model

model = Sequential()

model.add(LSTM(100, input_shape = (1, timesteps)))  # 50 LSTM neuron
(block)

model.add(Dense(units=1))

model.compile(loss = "mean_squared_error", optimizer= "adam")

model.fit(x_train, y_train, epochs = 50, batch_size = 1)

# predictions

train_pred = model.predict(x_train)

test_pred = model.predict(x_test)

# invert predictions

train_pred = scaler.inverse_transform(train_pred)

y_train = scaler.inverse_transform([y_train])

test_pred = scaler.inverse_transform(test_pred)

y_test = scaler.inverse_transform([y_test])

# calculate root mean squared error

train_score = math.sqrt(mean_squared_error(y_train[0],
train_pred[:,0]))

print("Train Score: %.2f " %(train_score))

test_score = math.sqrt(mean_squared_error(y_test[0],
test_pred[:,0]))

print("Test Score: %.2f " %(test_score))
```

Output:



```
test_score = math.sqrt(mean_squared_error(y_test[0], test_pred[:,0]))
print("Test Score: %.2f " %(test_score))
[13]

20/20 [==============================] - 1s 2ms/step
20/20 [==============================] - 0s 2ms/step
Train Score: 6.46
```

Practical No.9

Aim: Find the information gained from each column for the given dataset.

Theory:

Entropy is a scientific concept as well as a measurable physical property that is most commonly associated with a state of disorder, randomness, or uncertainty.

Gini Index or Impurity measures the probability for a random instance being misclassified when chosen randomly. The lower the Gini Index, the better the lower the likelihood of misclassification.

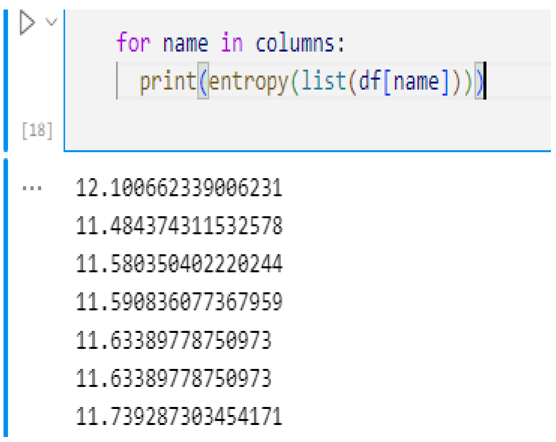
Code & Output:

Code -

```
# Import libraries of python
import pandas as pd
import os
import math
# Read the CSV dataset using pandas
df=pd.read_csv('data.csv')
# Entropy
def entropy(labels):
  # Initialize the entropy to 0.
  entropy = 0.0
  # Iterate over all possible labels.
  for label in set(labels):
    # Calculate the probability of the label.
    probability = labels.count(label) / len(labels)
    # Calculate the contribution of the label to the entropy.
    entropy -= probability * math.log2(probability)
  # Return the entropy.
  return entropy
# Print entropy for each columns of dataset
for name in columns:
  print(entropy(list(df[name])))
```

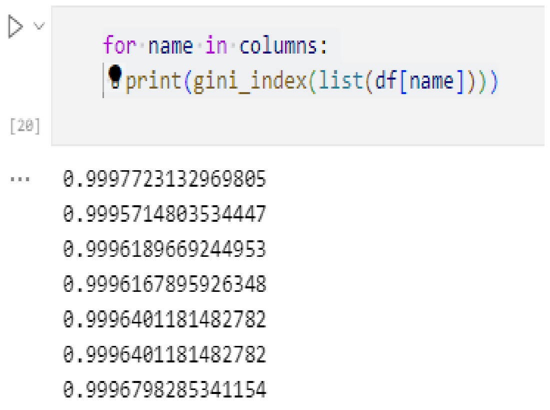
```
# Gini Index
def gini_index(labels):
  # Initialize the Gini impurity to 1.
  gini = 1.0
  # Iterate over all possible labels.
  for label in set(labels):
    # Calculate the probability of the label.
    probability = labels.count(label) / len(labels)
    # Calculate the contribution of the label to the Gini impurity.
    gini -= probability ** 2
  # Return the Gini impurity.
  return gini
# Print Gini index for each columns of dataset
for name in columns:
  print(gini_index(list(df[name])))
```

Output:



```
[18]
for name in columns:
    print(entropy(list(df[name])))
```

```
12.100662339006231
11.484374311532578
11.580350402220244
11.590836077367959
11.63389778750973
11.63389778750973
11.739287303454171
```



```
[20]
for name in columns:
    print(gini_index(list(df[name])))
```

```
0.9997723132969805
0.9995714803534447
0.9996189669244953
0.9996167895926348
0.9996401181482782
0.9996401181482782
0.9996798285341154
```

Practical No. 10

Aim: To predict Stock Price using SVR algorithm.

Theory:

_SVR gives us the flexibility to define how much error is acceptable in our model and will find an appropriate line (or hyperplane in higher dimensions) to fit the data.

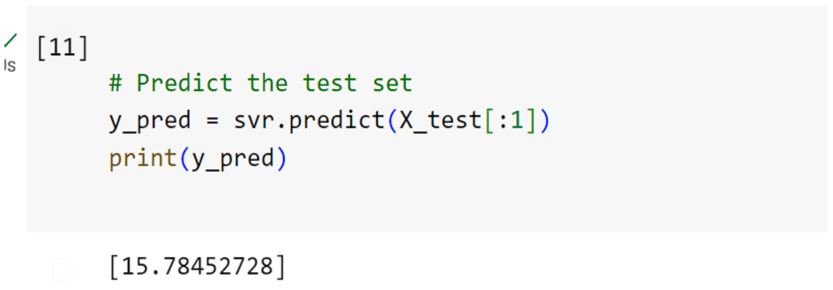
In contrast to OLS, the objective function of SVR is to minimize the coefficients — more specifically, the l2-norm of the coefficient vector — not the squared error.

Code & Output:

Code -

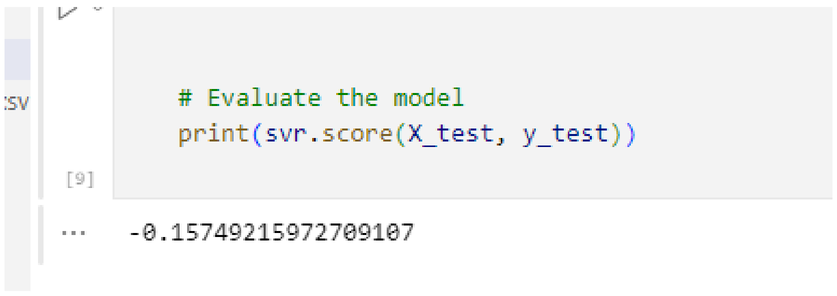
```
# Import needed libraries

import pandas as pd

import numpy as np

from sklearn.svm import SVR

from sklearn.model_selection import train_test_split

# Load the data

df = pd.read_csv("EW-MAX.csv")

df.head()

# Prepare the data

X = df[["Volume", "High", "Low", "Close","Adj_Close"]]

y = df["Open"]

# Split the data into training and test sets

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.25)

# Create the SVR model

svr = SVR(kernel='rbf')

# Train the model

svr.fit(X_train, y_train)

# Predict on the test set

y_pred = svr.predict(X_test)

# Evaluate the model

print(svr.score(X_test, y_test))
```

Output:



```
[11]
# Predict the test set
y_pred = svr.predict(X_test[:1])
print(y_pred)
```

```
[15.78452728]
```



```
[9]
# Evaluate the model
print(svr.score(X_test, y_test))
```

```
-0.15749215972709107
```