Sansar
Page No.:
Date

Experiment - 1.

Aim- How to implement Find-S algo in ML?
In ML, it termed as "a problem of searching through pre-defined space of potential hypothesis for hypothesis that best fits training ex."

What is Find-S algo in ML?

(a) Concept learning - There are large set of features for vehicles. These features differentiate set of cars, trucks, etc, i.e. known as concepts. Machines can also learn from concepts to identify whether obj. belongs to specific category or not.

- Training data
- Target concept
- Actual data objects.

(b) General Hypothesis - Hypothesis is explanation for something. It states general relationship b/w major variables. Ex- for ordering food would be I want a burger.

$$g = \{'?', '?', '?', \ldots, '?'\}$$

(c) specific Hypothesis - It fills in all imp. details about variables given in this. Ex- I want a cheeseburger with chicken pepperoni filling with a lot of lettuce.

$$s = \{'\phi', '\phi', '\phi', \ldots, '\phi'\}$$

Sansar
Page No.:
Date

Steps:-

1. Initialize 'h' to most specific hypothesis.
2. Find-S algo only considers (+)ve ex & eliminates (-)ve ex. For (+)ve ex, algo checks for each attribute in ex. If attribute value is same as hypothesis value, algos moves on w/o any changes. But if attribute value is diff. than hypothesis value, algo changes it to '?'.

How it works?

1. Initialize 'h' with most specific hypothesis. It is first (+)ve ex. in dataset.
2. Check for (+)ve ex, if it is (-)ve, we will move to next.
3. We will check if each attribute in ex is equal to hypothesis value.
4. If value matches, no changes are made.
5. If value doesn't match, value is changed to '?'.
6. We do this until we reach last (+)ve ex in dataset.

Limitations:-

1. Consistent throughout data.
2. Inconsistent sets mislead, since it ignore (-)ve ex.
3. No backtracking technique.

Implementation:-

step1- $h_0 = \{$'Morning', 'sunny', 'warm', 'yes', 'Mild', 'strong'$\}$.
Let's take (+)ve ex,

h1 = {'morning', 'sunny', '?', 'yes', '?', '?'}
h2 = {'?', 'sunny', '?', 'yes', '?', '?'}

Step 2- Import libraries
1. import pandas as pd
2. import numpy as np.

Step 3- Read csv file
3. d1 = pd.read_csv("exp1.csv")

Step 4- Making an array of all attributes
4. d2 = np.array(d1)[:, :-1]

Step 5- Segregating target that has (+)ve & (-)ve ex.
5. t1 = np.array(d1)[:, -1]

Step 6- f^n to implement Find-S algo.

```
def finds(c, t):
for i, val in enumerate(t):
    if val == "yes":
    spec_hypo = c[i].copy()
      break
for i, val in enumerate(c):
    if t[i] == "yes":
    for x in range(len(spec_hypo)):
    if val[x] != spec_hypo[x]:
  spec_hypo[x] = '?'
```

```
else :
    pass
return spec_hypo
```

step 7- Obtaining final hypothesis.

```
print(" Final hypothesis is :", finds(d2, t1))
```

Sansar
Page No.:
Date

Experiment-2

Aim - For given set of trained data ex stored in .CSV file, implement & demonstrate Candidate-Elimination algo to o/p, a description of set of all hypotheses consistent with training ex.

Candidate Elimination algo - It searches incomplete set of hypotheses. It finds every hypothesis i.e. consistent with training data, meaning it searches hypothesis space completely.
It performs bidir" search in hypothesis space. It maintains set S of most specific hypothesis that are consistent with training data & set G, of most general hypothesis consistent with training data.

1. Initialize G to set max. general hypothesis in H.
2. Initialize S to set max specific hypothesis in H.
3. For each training ex d, do
   1) If d is (+)ve ex.
   - Remove from G any hypothesis inconsistent with d.
   - For each hypothesis s in S i.e. not consistent with d.
     1. Remove s from S.
     2. Add to S all min. general" h of s such that.
     - h is consistent with d & some member of G is more general than h.
     3. Remove from S any hypothesis i.e. more general than another hypothesis in S.

2) If d is (-)ve ex.
- Remove from S any hypothesis inconsistent with d.
- For each hypo. g in G i.e. not consistent with d.
  1. Remove g from G.
  2. Add to G all min. specialization h of g such that
     - h is consistent with d & some member of S is more specific than h.
  3. Remove from G any hypothesis i.e. less general than another hypothesis in G.

Implementation :-

Step 1 - Create csv file using given data set.

Step 2 - Import libraries.

```
import pandas as pd
import numpy as np.
```

Step 3 - Read csv file

```
d1 = pd.read_csv("exp2.csv")
```

Step 4 - Making array of all attributes.

```
d2 = np.array(d1)[:,:-1]
```

Step 5 - Segregating target that has (+)ve & (-)ve ex.

```
t1 = np.array(d1)[:,-1]
```

Step 6 - Fn to implement candidate-elimination algo.

```
def candel(c, t):
spec_h = c[0].copy()
print("initialize of specific_h & general_h")
print(spec_h)
gene_h = [["?" for i in range(len(spec_h))]
```

```
for i in range(len(spec_h))] print(gene_h)
for i, h in enumerate(c):
    if t[i] == "yes":
print("If instance is (+)ve"):
 for x in range(len(spec_h)):
    if h[x] != spec_h[x]:
spec_h[x] = '?'    gene_h[x][x] = '?',
if t[i] == "no":
print("If instance is (-)ve"):
 for x in range(len(spec_h)):
    if h[x] != spec_h[x]:  gene_h[x][x] = spec_h[x]
 else:    gene_h[x][x] = '?'
print(spec_h)        print(gene_h)
indices = [i for i, val in enumerate(gene_h) if val
          == ['?', '?', '?', '?', '?', '?']]
for i in indices:
    gene_h.remove(['?', '?', '?', '?', '?', '?'])
 return spec_h, gene_h
```

Step 7- Obtaining final hypothesis.

```
s_final, g_final = candel(d2, t1)
print("Final spec_h=", s_final, sep = "\n")
print("Final gene_h:", g_final, sep = "\n")
```

Sansar
Page No.:
Date

Experiment - 3.

Aim- WAP to demonstrate working of decision tree based ID3 algo.

ID3 is ML algo for building classification trees. It is a greedy, recursive algo that partitions data set on attribute that max. info gain.

Decision trees are supervised learning algo used for classification & regression.

DT are assigned to info based learning algo which use diff. measures of info gain for learning. we can use this for issue i.e. continuous but also categorical i/p & target features.

This process of finding "most informative" feature is done until we accomplish stopping criteria where we then finally end up in so called leaf nodes.

DT contains root, interior & leaf nodes which are connected by branches.

Entropy - $H(x) = -p+ \log_2 p+ - p- \log_2 p-$

Info gain - $G(x,y) = H(x) - \sum_{i \in value(y)} \frac{|\Delta y_i|}{|\Delta y|} H(y_i)$

Types :- i) Iterative Dichotomizer 3 (ID3) ii) C4.5
iii) Classification & Regression Tree.

Entropy of given data set :-

$$H(S) = \sum_{c \in C} -p(c) \log_2 p(c) \qquad C = \{yes, no\}$$

Out of 14 instances, 9 are classified as yes & 5 as no

$p_{yes} = -(9/14) * \log 2(9/14) = 0.41$

$p_{no} = -(5/14) * \log 2(5/14) = 0.53$

$H(S) = p_{yes} + p_{no} = 0.94$

Info gain -

$E(outlook = sunny) = -2/5 \log(2/5) - 3/5 \log(3/5) = 0.971$

$E(outlook = overcast) = -1 \log(1) - 0 \log(0) = 0$

$E(outlook = rainy) = -3/5 \log(3/5) - 2/5 \log(2/5) = 0.971$

Avg, $I(outlook) = 5/14 * 0.971 + 4/14 * 0 + 5/14 * 0.971 = 0.693$

$Gain(outlook) = E(S) - I(outlook) = 0.94 - 0.693 = 0.247.$

$E(windy = false) = -6/8 \log(6/8) - 2/8 \log(2/8) = 0.811$

$E(windy = true) = -3/6 \log(3/6) - 3/6 \log(3/6) = 1$

Avg, $I(windy) = 8/14 * 0.811 + 6/14 * 1 = 0.892$

$Gain(windy) = E(S) - I(windy) = 0.94 - 0.892 = 0.048$

Gain(Temperature) = 0.571 bits

Gain (Humidity) = 0.971 bits

Gain (windy) = 0.020 bits.

Implementation :-

step 1. Create csv file using given data set.

step 2 Import libraries.

```
import numpy as np   import pandas as pd.
from sklearn.model_selection import train_test_split
from sklearn import tree.
from sklearn.metrics import accuracy_score
```

Sansar
Page No.:
Date

Step 3 Read csv file d1 = pd.read_csv("exp3.csv")

Step 4 Encode attributes in numerical values.

```
from sklearn import preprocessing
l_e = preprocessing.LE()
d1['outlook'] = l_e.fit_transform(d1['outlook'])
out_enc = d1['outlook'].unique()
print(out_enc)
d1['temp'] = l_e.fittransform(d1['temp'])
d1['temp'].unique()
```

Step 5 Separate dataset into data & target.

```
d2 = np.array(d1)[:,:-1]    t1 = np.array(d1)[:,-1]
print(d2)                   print(t1)
```

Step 6 Split dataset into training & testing.

```
x_train, x_test, y_train, y_test = train(d2, t1, testsize=0.3, random_state=1)
```

Step 7. Build decision tree.

```
model = tre.DTC(criterion='entropy', random_state=100)
model.fit(x_train, y_train)
y_pred = model.pred(x_test)    print(y_pred)
```

Step 8. Test accuracy of model

```
from sklearn import metrics
print("Accuracy:", metrics.acc_score(y_test, y_pred))
```

Step 9 Visualize DT

```
import graphviz
import pydotplus
tre.plot_tree(model)
```

Sansar
Page No.:
Date

Experiment - 4.

Aim - Build ANN by Backpropagation algo & test using dataset.

→ What is ANN? It is a group of connected I/O units where each connectⁿ has a weight associated with its comp. program. It helps to build predictive model for large DB. It conducts image understanding, comp. speech, human learning etc.

→ What is backpropagation? It is essence of neural net training. It is method of fine tuning weights of neural net based on error rate obtained iteration. Proper tuning of weight allows to reduce error rates & to make model reliable by ↑↑ its generalization.

→ Working:-

i. Input x, arrive through pre connected path.
ii. Input is modified using real weights W. weights are usually randomly selected.
iii. Calculate o/p for every neuron from ip layer, to hidden layers, to o/p layer.
iv. Calculate Error, B = Actual o/p − Desired o/p.
v. Travel back from o/p layer to hidden layer to adjust weight such that error is ↓↓.

→ Implementation

Step 1 Import libraries.

```
import numpy as np.
```

Step 2 Initialize i/p & o/p.

```
inp=np.array([[2,5],[4,9],[3,8]),dtype=float)
aoutp=np.array([[76],[84],[98]),dtype=float)
inp=inp/np.amax(inp,axis=0)
aoutp=aoutp/100.
```

Step 3 Sigmoid fn

```
def sigmoid(x): return 1/(1+np.exp(-x))
```

Step 4 Derivate of sigmoid fn.

```
def D_sigmoid(x): return x*(1-x)
```

Step 5 Variable initialization.

```
epoch=2000 lr=0.7 iln=2 hln=3 on=2.
wh=np.random.uniform(size=(iln,hln))
bh=np.random.uniform(size=(1,hln))
wout=np.random.uniform(size=(hln,on))
bout=np.random.uniform(size=(1,on))
```

Step 6 F/w & B/w propagation

```
for i in range(epoch): hinp1=np.dot(inp,wh)
hinp=hinp1+bh hlayer_act=sigmoid(hinp)
outinp1=np.dot(hlayer_act,wout)
outinp=outinp1+bout output=sigmoid(outinp)
EO=aoutp-output
outgrad=D_sigmoid(output)
d_output=EO*outgrad.
EH=d_output.dot(wout.T)
hiddengrad=D_sigmoid(hlayer_act)
d_hiddenlayer=EH*hiddengrad.
```

```
wout += hlayer_act.T.dot(d_output) * lr
wh += inp.T.dot(d_hiddenlayer) * lr
```

Step 7 Print o/p.

```
print ("Input:\n" + str(inp))
print("Actual o/p:\n" + str(actoutp))
print("Predicted o/p:\n", output)
```

Sansar
Page No.:
Date

Experiment-5.

Aim - WAP for naive Bayesian classifier for sampling training data set stored as .csv file. Compute accuracy of classifier, considering few test data sets.

Step 1 - Create csv file using data set.

Step 2 - Import libraries.

```
import numpy as np
import pandas as pd.
from sklearn.model_selection import train_test_split
from sklearn import tree.
from sklearn.metrics import accuracy_score.
```

Step 3 - Read csv file. `d1=pd.read_csv("enp3.csv")`

step 4 - encode attributes in numerical values.

```
from sklearn import preprocessing.
label_encoder=preprocessing.labelencoder()

d1['outlook']=label_encoder.fit_transform(d1
['outlook'])
out_enc=d1['outlook'].unique()
print(out_enc)
d1['temp']=label_encoder.fit_transform(d1['temp'])
d1['temp'].unique()
d1['humidity']=label_encoder.fit_transform(d1
['humidity'])
```

```
d1['humidity'].unique()
d1['windy'] = label_encoder.fit_transform(d1['windy'])
d1['windy'].unique()
d1['play'] = label_encoder.fit_transform(d1['play'])
d1['play'].unique().
```

Step 5- Separate dataset into data & target.

```
d2 = np.array(d1)[:,:-1]
t1 = np.array(d1)[:,-1]
print(d2)        print(t1)
```

Step 6- Split dataset into training & testing dataset.

```
x_train, x_test, y_train, y_test = train_test_split
(d2, t1, test_size = 0.3, random_state = 1)
```

Step 7- Naive Bayes Classifier

```
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()
gnb.fit(x_train, y_train)
y_pred = gnb.predict(x_test)
```

Step 8- Test accuracy of model

```
from sklearn import metrices
print("Accuracy:",
metrices.accuracy_score
(y_test, y_predict)).
```

Experiment-6.

Aim- assuming set of documents that need to be classified, use of naive Bayesian classifier model to perform this task. Built-in JAVA classes/API can be used to write a program. Calculate accuracy, precision & recall for datas

Implementation:-

Step1- Create csv file using given data set.

Step2- import libraries.
```
import numpy as np
import pandas as pd.
from sklearn.model_selection import train_test_split
```

Step3- Read csv file `d1 = pd.read_csv("exp6.csv")`

Step4- encode attributes in numerical values.
```
from sklearn import preprocessing
label_enc = preprocessing.LabelEncoder()
d1['Label'] = label_enc.fit_transform(d1['Label'])
L_enc = d1['Label'].unique()
print(L_enc).
```

Step5- separate data set into data & target.
```
d2 = np.array(d1)[:,:-1]    t1 = np.array(t1)[:,-1]
print(d2)    print(t1).
```

Step6- split dataset into training & testing dataset.

Roll No.

Sansar
Page No.:
Date

x_train, x_test, y_train, y_test = train_test_split (d2, t1)

Step7- O/p of count vectoriser is a sparse matrix.

```
from sklearn.feature_extraction.text import count vectorizer.
count_vect = countVectorizer()
x_train_trans = count_vect.fit_transform(x_train)
x_test_train = count_vect.transform(x_test)
print(count_vect.get_feature_names())
df = pd.DataFrame(x)train_trans.toarray(), column
= count_vect.get_feature_names())
print(df)                    print(x_train_trans).
```

Step8- Training NBC on training data.

```
from sklearn.naive_bayes import MultinomialNB
clf = MultinomialNB().fit(x_train_trans, y_train)
predicted = clf-predict(x_test_trans).
```

Step9- Printing accuracy metrics.

```
from sklearn import metrics    print('Accuracy metrics')
print('Accuracy of classifier is', metrics.accuracy_score
(y_test, predicted))
print ('confusion matrix')
print('metrics.confusion_matrix
(y_test, predicted))
print('Recall & Precision')
print(metrics.recall_score(y_test, predicted))
print(metrics.precision_score
(y_test, predicted))
```

Sansar
Page No.:
Date

experiment - 7.

aim- WAP to construct Bayesian nlw considering medical data. Use this model to demonstrate diagnosis of heart patients using std. Heart Disease Data set.

A BN falls under Prob. Graphical modelling technique i.e. used to compute uncertainties. Popularly known as Belief nlw, BN are used to model uncertainties by using DAG.

What is DAG? It is used to represent BN & like any other statistical graph, it contains set of nodes & links, where links denote relationship blw nodes.

DAG models uncertainty of event occuring based on CDP of each random variable. CPT is used to represent CPD of each variable in nlw.

Joint Probability - It is a measure of 2 or more events happening at same time, i.e. P(A,B,C). It can be represented as P(intersection 2 or more events)

Conditional Probability -

- If X & Y dependent $\Rightarrow P(X|Y) = P(X \text{ and } Y) / P(Y)$
- If X & Y independent $\Rightarrow P(X|Y) = P(X)$.

Program:-

Age: in years

Sex: (1 = male; 0 = female)

Cp: (chest pain type)

Trestbps: (resting BP (in mm Hg on admission to hospital)

Chol: (serum cholestrol in mg/dl)

Fbs: (fasting blood sugar > 120 mg/dl) (1 = true; 0 = false)

Restecg: (resting electrocardiographic results)

Thalach: (max heart rate achieved)

Exang: (exercise induced angina) (1 = yes; 0 = no)

Oldpeak: (ST depression induced by exercise relative to rest)

slope: (slope of peak exercise ST segment)

Ca: (no. of major vessels (0-3) colored by flourosopy)

Thal: (3 = normal; 6 = fixed defect; 7 = reversable defect)

Target: 1 or 0.

Sansar
Page No.:
Date

Experiment-8

Aim- Apply EM Algo to cluster set of data stored in .CSV file. Use same data set for clustering using k-means algo. Compare results of these two algos & comment on quality of clustering.

```
from sklearn.cluster import KMeans
from sklearn import preprocessing
from sklearn.mixture import GaussianMixture
from sklearn.datasets import load_iris
import sklearn.metrics as sm
import pandas as pd
import numpy as np.
import matplotlib.pyplot as plt.
dataset = load_iris()
x = pd.DataFrame(dataset.data)
x.column = ['sepal_length', 'sepal_width', 'Petal_length',
            'Petal_width']
y = pd.DataFrame(dataset.target)
y.column = ['Target']
plt.fig(figsize = (4, 7))
colormap = np.array(['red', 'lime', 'black'])
plt.subplot(1, 3, 1)
plt.scatter(x.Petal_length, x.Petal_width, c=colormap
        [y.Targets], s = 40)
plt.title('Real')
plt.suplot(), 3, 2)
model = KMeans(n_clusters = 3)
```

```
model.fit(x)
predY = np.choose(model.labels_,[0,1,2].astype(np.int64)
plt.scatter(x.Petal_length, x.petal_width, c=colormap
[predY], s=40)
plt.title('K Means')
scaler = preprocessing.standard scaler()
scaler.fit(x)
xsa = scalar.transform(x)
xs = pd.DataFrame(xsa, columns=X.columns)
gmm = Gaussian Mixture(n_components =3)
gmm.fit(xs)
y_cluster_gmm = gmm.predict(xs)
plt.subplot(1,3,3)
plt.scatter(x.Petal_length, x.Petal_width, c=colormap
[y_cluster_gmm], s=40)
plt.title('GMM Classification')
```

Experiment - 9

Aim - WAP for KNN algo to classify iris data set. Print both correct & wrong predictions.

Introduction - KNN is a type of supervised ML algo which can be used for both classification as well as regression predictive prob. It is used for predictive problems in industry.

- Lazy learning algo
- Non-parametric learning algo.

Working -

Step 1 - For algo, we need dataset. We load training as well as test data.

Step 2 - Next, we need to choose value of k i.e. nearest data points. K can be any integer.

Step 3 - Calculate distance.

- Sort in ascending order.
- Top K rows from sorted array.
- Assign class to test point based on most frequent
- class of these rows.

Step 4 - End.

Implementation -

Step 1 - Import necessary python packages.

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

Step 2 - Download iris dataset from web link.

```
path = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data".
```

Step 3 - Assign column names to dataset.

```
headernames = ['sepal-length', 'sepal-width', 'petal-length',
               'petal-width', 'class']
```

Step 4 - Read dataset

```
dataset = pd.read_csv(path, names = headernames)
dataset.head()
```

Step 5 - Data preprocessing.

```
x = dataset.iloc[:, :-1].values.
y = dataset.iloc[:, 4].values.
```

Step 6 - Divide data: 60% - training & 40% - testing.

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y,
test_size = 0.40)
```

Step 7 - Data scaling

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()    scaler.fit(x_train)
x_train = scaler.transform(x_train)
x_test = scaler.transform(x_test)
```

Step 8 - Train model.

```
from sklearn.neighbours import KNC.
classifier = KNC(n_neighbors = 8)
classifier.fit(x_train, y_train)
```

Step 9 - Make prediction

```
y_pred = classifier.predict(x_test)
```

Step 10 - Print result

```
from sklearn.metrics import classification_report,
confusion_matrix, accuracy_score.
result = confusion_matrix(y_test, y_pred)
print("confusion matrix:")
print(result)
result1 = classification_report(y_test, y_pred)
print("Classification Report:")
print(result1)
result 2 = accuracy_score(y_test, y_pred)
print("Accuracy:", result2).
```

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

TP - True positive
TN - True (-)ve
FP - False (+)ve
FN - False (-)ve

$$Recall = \frac{TP}{TP + FN}$$

$$Precision = \frac{TP}{TP + FP}$$

$$F\text{-measure} = \frac{2 * Recall * Precision}{Recall + Precision}.$$

Sansar
Page No.:
Date

Experiment - 10

Aim - Implement non parametric locally weighted Regression algo in order to fit data points.

Introduction - LR is supervised learning algo used for computing linear relationship b/w i/p → X & o/p → Y.

As evident from image, algo cannot be used for making predictions.

Locally weighted LR is non-parametric algo, model does not learn fixed set of parameters as is done in ordinary LR.

Modified cost fn, $J(\theta) = \sum_{i=1}^{m} w^{x(i)} (\theta^T w^{x(i)} - y^i)^2$

Typical choice of, $w^{x(i)} = \exp\left(\frac{-(x^i - x)^2}{2\tau^2}\right)$

Steps :-

i° Compute θ to minimize cost:

$$J(\theta) = \sum_{i=1}^{m} w^{x(i)} (\theta^T w^{x(i)} - y^i)^2$$

ii° Predict o/p for query point x

iii Return: $\theta^T x$.

Locally weighted Regression - Model based methods, such as neural n/w & mix of Gaussian, use data to build parametrized model. In contrast, 'memory-based' are NP approaches that explicitly retain training data & use it each time prediction needs to be made. LWR performs regression around pt of interest using only training e are local to that pt.

Implementation of LWR:-

Step 1 - Import necessary python packages.

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np1
import numpy.linalg as np
from scipy.stats.stats import pearsonr.
```

step 2 - Write kernel fn

```
def KER(point, xmat, k):
m, n = np1.shape(xmat)
weights = np1.mat(np1.eye((m)))
for j in range(m):      diff = point - x[j]
weights[j,j] = np1.exp(diff * diff.T/(-2.0 * k**2))
return weights
```

Step 3 - Write local weight fn

```
def LW(point, xmat, ymat, k):
weight = KER(point, xmat, k)
W = (x.T*(weight *x)).I*(x.T*(weight * ymat.T))
return W.
```

Step 4 - Write LWR fn

```
def LWR(xmat, ymat, k):
m,n = np1.shape(xmat)
ypred = np1.zeros(m)
for i in range(m):
ypred[i] = xmat[i] * LW(xmat[i], xmat, ymat, k)
return ypred
```

Step 5 - Download Tips dataset

```
path = "https://raw.githubusercontent.com/pandas-dev/
pandas/master/doc/data/tips.csv"
```

Step 6- assign columns.

```
headernames=['total_bill','tip','sex','smoker','day','time','size']
```

Step 7- Read dataset to pandas dataframe.

```
dataset=pd.read_csv(path,names=headernames)
```

Step 8- Data preprocessing

```
bill=np1.array(dataset['total_bill'])
tip=np1.array(dataset['tip'])
bill1=bill[1:255]        tip1=tip[1:255]
bill2=bill1.astype(np1.float)
tip2=tip1.astype(np1.float)
```

Step 9- Main fn.

```
mbill=np1.mat(bill2)    mtip=np1.mat(tip2)
m=np1.shape(mbill)[1]   one=np1.mat(np1.ones(m))
x=np1.hstack((one.T,mbill.T))
ypred=LWR(x,mtip,2)     sortIndex=x[:,1].argsort(0)
xsort=x[sortIndex][:,0]
```

Step 10- Plot graph

```
fig=plt.figure()        ax=fig.add_subplot(1,1,1)
ax.scatter(bill2,tip2,color='green')
ax.plot(xsort[:,1],ypred[sortIndex],color='red',
linewidth=5)
plt.xlabel("Total bill")
plt.ylabel("Tip")
plt.show();
```

Step 11- Plot graph b/w actual & predicted tip value.

```
import matplotlib.pyplot as plt.
z1=np1.arange(244)
```

```
ax.scatter(bill2, tip2, color='green')
plt.plot(z1, tip2, label="Actual Tip value')
plt.plot(z1, ypred, label='Predicted Tip
                          value')
plt.xlabel('Steps')
plt.ylabel('Tip')
plt.legend()
plt.show()
```