```
//// wk3- linear regression for continuous vlue (correlatn) ///
//// Boston - lasso, ridge///

from sklearn.datasets import load_boston
boston = load_boston()
#see description of dataset
boston.DESCR
#boston.keys()
import pandas as pd
data=pd.DataFrame(boston.data, columns=boston.feature_names)
#see data in dataframe
data
#column of Median Value is usually the target - to be predicted by regression
data['MEDV']=pd.DataFrame(boston.target)
data
#find all correlation values, RM has highest with target MEDV, so select RM for training
pd.DataFrame(data.corr().round(2))
# x corresponds to train data
x=data['RM']
#y corresponds to labels
y=data['MEDV']
#import module for linear regression
from sklearn.linear_model import LinearRegression
  #import train_test_split module
from sklearn.model_selection import train_test_split
  linearRegressionClassifier = LinearRegression()
#convert x and y to pandas Dataframes
x=pd.DataFrame(x)
y=pd.DataFrame(y)
  #split the dataset using train_test_split function, pass train data, labels,and test data ratio
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.2)
#check number of rows and columns
print(x_train.shape)
#train the classifier by fitting to train data and corresponding labels
linearRegressionClassifier.fit(x_train, y_train)
import numpy as np
#import the mean squared error module
from sklearn.metrics import mean_squared_error
#determine the predicted values
y_pred=linearRegressionClassifier.predict(x_test)
y_pred.shape
#calculate root mean square error
np.sqrt(mean_squared_error(y_test, y_pred))
#determine R^2 scores (because regression)
linearRegressionClassifier.score(x_test,y_test)

0.2 working with Ridge
#L2 Regularization: It adds an L2 penalty which is equal to the square of the magnitude of coefficients.
#importing module to work with ridge regression model
from sklearn.linear_model import Ridge
#setting parameter alpha for new ridge model
ridge1=Ridge(alpha=1)
ridge1.fit(x_train,y_train)
  y_pred1=ridge1.predict(x_test)
np.sqrt(mean_squared_error(y_test, y_pred1))
```

```
ridge2=Ridge(alpha=100)
ridge2.fit(x_train,y_train)
y_pred2=ridge2.predict(x_test)
  np.sqrt(mean_squared_error(y_test, y_pred2))
ridge2.score(x_test,y_test)

1 # Working with Lasso
from sklearn.linear_model import Lasso
  Lasso1=Lasso(alpha=0.01)
  Lasso1.fit(x_train,y_train)
y_predL1=Lasso1.predict(x_test)
  np.sqrt(mean_squared_error(y_test, y_predL1))
  Lasso1.score(x_test,y_test)
[ ]: #working with ElasticNet - combines feature elimination from lasso and feature coefficient
reduction from Ridge
from sklearn.linear_model import ElasticNet
  en1=ElasticNet(alpha=0.1, l1_ratio=0.5)
  en1.fit(x_train,y_train)
  y_pred_en1=en1.predict(x_test)
  np.sqrt(mean_squared_error(y_test, y_pred_en1))
  en1.score(x_test,y_test)
```

```
*/// Boston - Desicion Tree Regression , Random forest ///* wk-5

from sklearn.datasets import load_boston
boston = load_boston()
import pandas as pd
data=pd.DataFrame(boston.data, columns=boston.feature_names)
data['MEDV']=pd.DataFrame(boston.target)
data
pd.DataFrame(data.corr().round(2))
x=data[['RM','ZN']]
x
y=data['MEDV']
y
#train test n split model
from sklearn.model_selection import train_test_split
x=pd.DataFrame(x) y=pd.DataFrame
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.2)
x_train
from sklearn.tree import DecisionTreeRegressor
dt1=DecisionTreeRegressor(max_depth=20)
  dt1.fit(x_train,y_train)
y_pred1=dt1.predict(x_test)
import numpy as np
  #import the mean squared error module
from sklearn.metrics import mean_squared_error
  #calculate root mean square error
np.sqrt(mean_squared_error(y_test, y_pred1))
from sklearn.ensemble import RandomForestRegressor
rf1=RandomForestRegressor()
  rf1.fit(x_train, y_train)
rf1.score(x_test,y_test)
y_pred2=rf1.predict(x_test)
```

```
  #calculate root mean square error
  np.sqrt(mean_squared_error(y_test, y_pred2))
# Decision Tree Classifier
from sklearn.tree import DecisionTreeClassifier
  from sklearn.datasets import load_digits
  digits = load_digits()
  x_train, x_test, y_train, y_test=train_test_split( digits.data,digits.target,test_size=0.25)
dt2=DecisionTreeClassifier(criterion="entropy")
  dt2.fit(x_train, y_train)
# max_depth = longest path between root and leaf nodes
dt3=DecisionTreeClassifier(max_depth=30)
dt3.fit(x_train, y_train)
dt3.score(x_test,y_test)

================================================================================
==========================================

//// Digit dataset - Logistic Regression for Classification - target takes specific values only ///
[1]: from sklearn.datasets import load_digits
[2]: digits = load_digits()
[3]: print(digits.DESCR)
import matplotlib.pyplot as plt
[5]: #shows 2d data, each row contains multiple columns with meaning
digits.data
#dimension of actual data
digits.data.shape
[7]: d=digits.data[0:500]
[8]: d.shape
[9]: #target or label dimension
digits.target.shape
#taking only the first image pixel data
image = digits.data[0]
[15]: #image is represented as collection of 64 pixel values
print(image)
#determine class of the 64 pixels stored in image
digits.target[0]
[17]: import numpy as np
[18]: np.reshape(image,(8,8))
plt.imshow(np.reshape(image, (8,8)), cmap='BuGn')
from sklearn.model_selection import train_test_split
[21]: #split the dataset using train_test_split function, pass train data, labels and test data ratio
x_train, x_test, y_train, y_test = train_test_split(digits.data,digits.target,test_size=0.2)
[22]: from sklearn.linear_model import LogisticRegression
[23]: logReg1 = LogisticRegression(max_iter=10000)
[24]: logReg1.fit(x_train, y_train)
[25]: y_pred=logReg1.predict(x_test)
[26]: logReg1.score(x_test,y_test)

---

//// Digit daatset - Semi Supervised Learning - label propagation and spreading - wk8 ///

[1]: from sklearn.datasets import load_digits
[2]: digits=load_digits()
[3]: digits.data.shape
[4]: digits.target
```

```
[5]: import matplotlib.pyplot as plt
[6]: import numpy as np
[7]: # plot a value
[8]: from sklearn.model_selection import train_test_split
[9]: #split the dataset using train_test_split function, pass train data, labels,and test data ratio
x_train, x_test, y_train, y_test = train_test_split(digits.data,digits.target,test_size=0.2)
[10]: x_train.shape
[11]: x_test.shape
[12]: #target value
print(y_train)
#first get a list of indices of data
indexList=np.arange(len(x_train))
[14]: print(indexList)
[15]: #split training set to labelled and non-labelled
x_labelled=x_train[indexList[:300]]
[16]: #split training set to labelled and non-labelled
y_labelled=y_train[indexList[:300]]
[17]: nonLabelledIndices=indexList[300:]
[18]: #remove the labels for non labelled indices in training data
y_train_nonlabel = np.copy(y_train)
[19]: #setting -1 as non labelled
y_train_nonlabel[nonLabelledIndices] = -1
[20]: from sklearn.semi_supervised import LabelPropagation
[21]: lp=LabelPropagation(gamma=.30)
#gamma is measure of kernel func,
[22]: lp.fit(x_train, y_train_nonlabel)
[23]: lp.score(x_test,y_test)
[24]: #capture labels set by Label Propagataion on the set of non labelled data
y_labelled=lp.transduction_[nonLabelledIndices]
[25]: from sklearn.metrics import confusion_matrix
[26]: conf = confusion_matrix(y_train[nonLabelledIndices], y_labelled, labels=lp.classes_)
[27]: conf
from sklearn.semi_supervised import LabelSpreading
[29]: ls=LabelSpreading(gamma=0.3)
#gamma is measure of kernel func,
[30]: ls.fit(x_train, y_train_nonlabel)
[31]: ls.score(x_test,y_test)
[32]: #capture labels set by Label Propagataion on the set of non labelled data
y_labelled=lp.transduction_[nonLabelledIndices]
[33]: conf = confusion_matrix(y_train[nonLabelledIndices], y_labelled, labels=lp.classes_)
[34]: conf

================================================================================
================================================

/// wk-6 Kmeans blop ///

import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans
alldata=make_blobs(n_samples=200, centers=4, n_features=2, cluster_std=1.5, random_state=30
print(alldata)
data=alldata[0]
data
plt.scatter(data[:,0], data[:,1])
```

```
kmeans=KMeans(n_clusters=4)
y_pred=kmeans.fit_predict(data)
print(y_pred)
clusters=kmeans.cluster_centers_
clusters
plt.scatter(data[y_pred==0,0], data[y_pred==0,1], s=70, color='green')
plt.scatter(data[y_pred==1,0], data[y_pred==1,1], s=70, color='red')
plt.scatter(data[y_pred==2,0], data[y_pred==2,1], s=70, color='yellow')
plt.scatter(data[y_pred==3,0], data[y_pred==3,1], s=70, color='peru')
plt.scatter(clusters[0][0], clusters[0][1], marker='*', s=200, color='black')
plt.scatter(clusters[1][0], clusters[1][1], marker='*', color='black')
plt.scatter(clusters[2][0], clusters[2][1], marker='*', color='black')
plt.scatter(clusters[3][0], clusters[3][1], marker='*', color='black')
plt.scatter(clusters[0][0], clusters[0][1], marker='*', color='black')
plt.scatter(clusters[1][0], clusters[1][1], marker='*', color='black')
plt.scatter(clusters[2][0], clusters[2][1], marker='*', color='black')
plt.scatter(clusters[3][0], clusters[3][1], marker='*', color='black')

================================================================================
==========================================================

//// MLP wid Confusn matrix - Breast Cancer - wk7 ///

import pandas as pd
from sklearn.datasets import load_breast_cancer
[2]: cancer_dataset = load_breast_cancer()
[4]: from sklearn.model_selection import train_test_split
#split the dataset using train_test_split function, pass train data
X_train, X_test, y_train, y_test = train_test_split(
cancer_dataset.data,cancer_dataset.target,test_size=0.25)
[5]: #Importing MLPClassifier
from sklearn.neural_network import MLPClassifier
#Initializing the MLPClassifier
#solver is for weight optimization
#max_iter sets limit till convergence or this value
#activation is the activation function for the hidden layer
classifier = MLPClassifier(hidden_layer_sizes=(150,100,50), max_iter=300,
activation = 'relu',solver='adam',random_state=1)
classifier.fit(X_train, y_train)
[6]: y_pred = classifier.predict(X_test)
[7]: from sklearn.metrics import confusion_matrix
#Get the confusion matrix
cf_matrix = confusion_matrix(y_test, y_pred)
print(cf_matrix)
import numpy as np
cf_matrix=np.transpose(np.transpose(cf_matrix) / cf_matrix.astype(float).sum(axis=1))
[9]: cf_matrix

================================================================================
==========================================================

// set -5 -- qsn.1
// boston -    lasso - linear regression
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```
from sklearn.datasets import load_boston
from sklearn.linear_model import LinearRegression,Ridge,Lasso
from sklearn import metrics
from sklearn.model_selection import train_test_split
boston = load_boston()
print(boston.DESCR)
df = pd.DataFrame(boston.data)
df
df.columns = boston.feature_names
df
df['MEDV'] = pd.DataFrame(boston.target)
df
df.corr().round(3)
x = df[['RM','NOX']]
y = df['MEDV']
X_train,X_test,Y_train,Y_test=train_test_split(x,y,test_size = 0.3,random_state=2)
lm = LinearRegression()
lm.fit(X_train,Y_train)
Y_pred = lm.predict(X_test)
print("MSE:",metrics.mean_squared_error(Y_test,Y_pred))
print("RMSE:",np.sqrt(metrics.mean_squared_error(Y_test,Y_pred)))
print("R2 SCORE:",metrics.r2_score(Y_test,Y_pred))
#by ridge A
ridge1 = Ridge(alpha=1)
ridge1.fit(X_train,Y_train)
Y_pred1 = ridge1.predict(X_test)
print("RMSE:",np.sqrt(metrics.mean_squared_error(Y_test,Y_pred1)))
print("R2 SCORE:",metrics.r2_score(Y_test,Y_pred1))
# by ridge B
ridge2 = Ridge(alpha=11)
ridge2.fit(X_train,Y_train)
Y_pred2 = ridge2.predict(X_test)
print("RMSE:",np.sqrt(metrics.mean_squared_error(Y_test,Y_pred2)))
print("R2 SCORE:",metrics.r2_score(Y_test,Y_pred2))
# by lasso A
lasso1 = Lasso(alpha = 4)
lasso1.fit(X_train,Y_train)
Y_pred_l1 = lasso1.predict(X_test)
print("RMSE:",np.sqrt(metrics.mean_squared_error(Y_test,Y_pred_l1)))
print("R2 SCORE:",metrics.r2_score(Y_test,Y_pred_l1))
lasso2 = Lasso(alpha = 6)
lasso2.fit(X_train,Y_train)
Y_pred_l2 = lasso2.predict(X_test)

# by lasso B
lasso2 = Lasso(alpha = 6)
lasso2.fit(X_train,Y_train)
Y_pred_l2 = lasso2.predict(X_test)
print("RMSE:",np.sqrt(metrics.mean_squared_error(Y_test,Y_pred_l2)))
print("R2 SCORE:",metrics.r2_score(Y_test,Y_pred_l2))

.....

///Set5- qsn 2
/// Digitdataset - logistic regression classfctn - grey
```

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_digits
from sklearn import metrics
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
dataset = load_digits()
print(dataset.DESCR)
dataset.data.shape
dataset.target.shape
image = dataset.data[103]
print(image)
dataset.target[103]
np.reshape(image,(8,8))
plt.imshow(np.reshape(image,(8,8)),cmap = 'gray')
x = dataset.data
y = dataset.target
X_train,X_test,Y_train,Y_test = train_test_split(x,y,test_size = 0.25)
dtc = DecisionTreeClassifier(criterion = 'entropy',max_depth = 50)
dtc.fit(X_train,Y_train)
Y_pred = dtc.predict(X_test)
print("R2 SCORE:",metrics.r2_score(Y_test,Y_pred))
rfc = RandomForestClassifier()
rfc.fit(X_train,Y_train)
Y_pred1 = rfc.predict(X_test)
print("R2 SCORE:",metrics.r2_score(Y_test,Y_pred1))

.....

/// set 5 - qsn.3
/// digit dataset - Semi Supervised Learning - label propagation and spreading //

from sklearn.datasets import load_digits
digits=load_digits()
digits.data.shape
digits.target
import matplotlib.pyplot as plt
import numpy as np
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(digits.data,digits.target,test_size=0.30)
len(x_train)
x_test.shape
indexList=np.arange(len(x_train))
x_labelled=x_train[indexList[:320]]
y_labelled=y_train[indexList[:320]]
nonLabelledIndices=indexList[320:]
y_train_nonlabel = np.copy(y_train)
y_train_nonlabel[nonLabelledIndices] = -1
from sklearn.semi_supervised import LabelPropagation,LabelSpreading
#if said gamma then lp= (gamma=0.3)
lp=LabelPropagation()#if said gamma then lp=LabelPropagation(gamma=0.5)
lp.fit(x_train, y_train_nonlabel)
lp.score(x_test,y_test)
#r2 score
```

```
ls = LabelSpreading(gamma=0.3)
ls.fit(x_train, y_train_nonlabel)
ls.score(x_test,y_test)
y_labelled=ls.transduction_[nonLabelledIndices]
from sklearn.metrics import confusion_matrix
conf = confusion_matrix(y_train[nonLabelledIndices], y_labelled, labels=lp.classes_)
conf
```

.....

```
/// set-4 Q-1
/// Boston - decision Tree
  import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_boston
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn import metrics
from sklearn.model_selection import train_test_split
boston = load_boston()
print(boston.DESCR)
df = pd.DataFrame(boston.data)
df
df.columns = boston.feature_names
df
df['MEDV'] = boston.target
df
x = df[['PTRATIO','NOX']]
y = df['MEDV']
X_train,X_test,Y_train,Y_test = train_test_split(x,y,test_size=0.25,random_state=3)
dt = DecisionTreeRegressor(max_depth=50)
dt.fit(X_train,Y_train)
Y_pred = dt.predict(X_test)
print("RMSE:",np.sqrt(metrics.mean_squared_error(Y_test,Y_pred)))
print("R2 SCORE:",metrics.r2_score(Y_test,Y_pred))
rf = RandomForestRegressor()
rf.fit(X_train,Y_train)
Y_pred1 = rf.predict(X_test)
print("RMSE:",np.sqrt(metrics.mean_squared_error(Y_test,Y_pred1)))
print("R2 SCORE:",metrics.r2_score(Y_test,Y_pred1))
```

```
================================================================================
============================================================
```

```
/// wk-2 Use pandas to create a student marksheet with Student ID, Name, and marks obtained in
6 subjects ///

import pandas as pd
In [2]: # store column headers in a list
col_names = ['Student ID', 'Name', 'Marks1', 'Marks2', 'Marks3', 'Marks4', 'Marks5', 'Marks6']
#add header row to the dataframe
df = pd.DataFrame(columns = col_names)
In [3]: #view the dataframe
df
In [4]: # adding student details to the dataframe with help of dictionary and append()
```

```
stud1 = {'Student ID':2, 'Name':'E', 'Marks1':45, 'Marks2':67, 'Marks3':53, 'Marks4':99, 'Marks5':74, 'Marks6':82}
df=df.append(stud1, ignore_index=True)
In [5]: df
In [6]: # adding student details to the dataframe with help of another dataframe and concat()
stud1 = pd.DataFrame({'Student ID':[5], 'Name':['D'], 'Marks1':[32], 'Marks2':[55], 'Marks3':[64], 'Marks4':[63], 'Marks5
':[71], 'Marks6':[77]})
df=pd.concat([df, stud1], axis=0, ignore_index=True)
df
In [8]: # adding student details to the dataframe with loc[] attribute using row index
df.loc[2] = [4,'A',39,51,76,80,78,91]
In [9]: df
In [10]: # adding student details to the first row of dataframe with loc[] attribute as -1
df.loc[-1] = [8,'V',65,67,72,75,66,81]
In [11]: df
#rectifying indices
df.index = df.index + 1
df = df.sort_index()
In [13]: df
In [14]: # 1b Use pandas to get the name and ID of the student with the highest percentage of marks
sum=df['Marks1']+df['Marks2']+df['Marks3']+df['Marks4']+df['Marks5']+df['Marks6']
print(sum)
In [15]: per=sum/6
print(per)
df["per"]=per
print(df)
In [17]: #print the Student ID and Name of the row where the percentage is the highest among all the percentage values
print(df[['Student ID','Name']][df.per == df.per.max()])
In [18]: # 1c. Use pandas to identify the student who got lowest marks in more than 2 subjects.
In [22]: # 1d. Write the dataframe to a csv file
df.to_csv("file.csv")
```