# Multiple Linear Regression using OLS

# Dataset: Concrete

# import the libraries

import pandas as pd

import numpy as np

import math

import pylab

# uses Ordinary Least Squares (OLS) method

import statsmodels.api as sm

# in case of error:

# replace "cross\_validation" with "model\_selection"

from sklearn.cross\_validation import train\_test\_split

import scipy.stats as stats

import seaborn as sns

# cross validation

from sklearn.model\_selection import KFold

# Feature selection

from sklearn.feature\_selection import f\_regression as fs

# </import libraries>

# read the input file

# --------------------

path="F:/aegis/4 ml/dataset/supervised/regression/concrete/concrete.csv"

conc = pd.read\_csv(path)

conc.head()

type(conc)

# count of Rows and Columns

# -------------------------

conc.shape

# describe the dataset (R,C)

# --------------------------

conc.dtypes

# get the record count

# --------------------

conc.count()[1]

# view the dataset

# --------------------

conc.head(5)

# to rename the columns of a dataset

# -----------------------------------

conc = conc.rename(columns={'cementcomp':'ccomp',

'superplastisizer':'super',

'coraseaggr':'caggr',

'finraggr':'faggr'})

# summarize the dataset

# clearer view. removed the 1st row as it contains same info (total records)

# ------------------------------------------------------------

desc = conc.describe()

desc = desc.drop(desc.index[0])

desc

# check for NULLS, blanks and zeroes

# -------------------------------

cols = list(conc.columns)

type(cols)

cols.remove("CCS")

print(cols)

for c in cols:

if (len(conc[c][conc[c].isnull()])) > 0:

print("WARNING: Column '{}' has NULL values".format(c))

if (len(conc[c][conc[c] == 0])) > 0:

print("WARNING: Column '{}' has value = 0".format(c))

conc.head()

# check for outliers in dataset

# -------------------------------

conc.boxplot(column='ccomp',vert=False)

conc.boxplot(column='slag',vert=False)

conc.boxplot(column='age',vert=False)

# to find the correlation among variables (Multicollinearity)

# ----------------------------------------------------------

conc.corr()

cor = conc.iloc[:,0:8].corr()

print(cor)

# correlation using visualization

# -------------------------------

# cor --> defined above as the correlation amongst the x-variables

sns.heatmap(cor, xticklabels=cor.columns, yticklabels=cor.columns)

# split the dataset into train and test

# --------------------------------------

train, test = train\_test\_split(conc, test\_size = 0.3)

print(train.shape)

print(test.shape)

# split the train and test into X and Y variables

# ------------------------------------------------

train\_x = train.iloc[:,0:8]; train\_y = train.iloc[:,8]

test\_x = test.iloc[:,0:8]; test\_y = test.iloc[:,8]

print(train\_x.shape)

print(train\_y.shape)

print(test\_x.shape)

print(test\_y.shape)

train\_x.head()

train\_y.head()

train.head()

# ensure that the X variables are all numeric for regression

# ----------------------------------------------------------

train.dtypes

# gives a nice R-like summary()

# To add the constant term A (Y = A + B1X1 + B2X2 + ... + BnXn)

# Xn = ccomp,slag,flyash.....

# ----------------------------------------------------------

train\_x = sm.add\_constant(train\_x)

test\_x = sm.add\_constant(test\_x)

# build the LR model

lm1 = sm.OLS(train\_y, train\_x).fit()

# interpret the result

# =====================

lm1.summary()

# coefficients

lm1.params

### validating the assumptions

# ----------------------------

# function -> getresiduals()

def getresiduals(lm,train\_x,train\_y):

predicted = lm.predict(train\_x)

actual = train\_y

residual = actual-predicted

return(residual)

# 1) Residual mean is 0

# ----------------------------

residuals = getresiduals(lm1,train\_x,train\_y)

print(residuals.mean())

# 2) Residuals have constant variance

# ------------------------------------

y = lm1.predict(train\_x)

sns.set(style="whitegrid")

sns.residplot(residuals,y,lowess=True,color="g")

# 3) Residuals are normally distributed

# --------------------------------------

stats.probplot(residuals,dist="norm",plot=pylab)

pylab.show()

# 4) rows > columns

# ------------------

conc.shape

# cross-validation

# ----------------

# number of folds

kf = KFold(n\_splits=5)

kf.get\_n\_splits(train\_x)

print(kf)

fold = 1

# split the training further into train and test

for train\_index, test\_index in kf.split(train\_x):

# print("Train={0}, Test={1}".format(train\_index,test\_index))

cv\_train\_x = train\_x.iloc[train\_index,]

cv\_train\_y = train\_y.iloc[train\_index,]

cv\_test\_x = train\_x.iloc[test\_index,]

cv\_test\_y = train\_y.iloc[test\_index,]

# build the model on the CV training data and predict on CV testing data

cv\_lm = sm.OLS(cv\_train\_y, cv\_train\_x).fit()

cv\_pdct = cv\_lm.predict(cv\_test\_x)

# mean square error

cv\_mse = np.mean((cv\_pdct - cv\_test\_y)\*\*2)

print("Iteration = {0}, MSE = {1}, RMSE = {2}".format(fold,cv\_mse,math.sqrt(cv\_mse)))

fold+=1

# feature selection - technique 1

X=train\_x.iloc[:,1:9]

features = fs(X,train\_y,center=True)

features[0]

list(features[0])

df\_features = pd.DataFrame({"columns":train\_x.columns[1:9],

"score":features[0],

"p-val":features[1]

})

print(df\_features)

# sort on columns

df\_features.sort\_values(['score'],ascending=False)

# predict

# -----------------

pdct1 = lm1.predict(test\_x)

print(pdct1)

# mean square error

# -----------------

mse = np.mean((pdct1 - test\_y)\*\*2)

print("MSE = {0}, RMSE = {1}".format(mse,math.sqrt(mse)))

# store the actual and predicted values in a dataframe for comparison

actual = list(test\_y.head(50))

predicted = np.round(np.array(list(pdct1.head(50))),2)

print(predicted)

df\_results = pd.DataFrame({'actual':actual, 'predicted':predicted})

print(df\_results)