```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Mon Apr  1 11:18:38 2019

@author: Sagar Khurana
Assignment # 2
BUS241f: Machine Learning and Data Analysis for Business and Finance
"""



"""
Q.1 Write a short Python program to explore some simulated classification data.
 First generate a vector of length nmc=100 of uniform random numbers.
 Then use these to generate y values of 0 or 1, where the x value is the probability of y = 1
"""
# Load files from introMLNB
import sys
import os
targetDirectory = os.path.abspath("C:/PythonCode/introML/")
sys.path.append(targetDirectory)

#matplotlib inline
from preamble import *

# import graphviz
from sklearn.model_selection import train_test_split

# Load helpers
# Will try to just load what I need on this
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
import matplotlib as mpl
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier

# These lines of code generate a feature and a class randomly with a monte-carlo
# The features are uniform(0,1)
# Class is a (0,1) integer where prob(1) = X[i]
nmc = 100

# X needs to be a matrix with nmc rows, and 1 column
X = np.random.uniform(size=(nmc,1))
Y = np.zeros(shape=nmc,dtype=int)+(np.random.uniform(size=nmc)<X[:,0])

# This is the code for splitting into test/train sets, note test_size argument
X_train, X_test, Y_train, Y_test = train_test_split(X,Y,test_size=0.5)



# 1.1 Write a k nearest neighbor predictor for your data, using k=3. Report the accuracy both for t

clf = KNeighborsClassifier(n_neighbors = 3)
```

```
clf.fit(X_train, Y_train)
print("Predictor: {}".format(clf.predict(X_test)))

##Checking the accuracy rate:
print("Accuracy score for train set: {}".format(clf.score(X_train, Y_train)))
print("Accuracy score for test set: {}".format(clf.score(X_test, Y_test)))
print("With k=3, the model is overfiting because the accuracy test on train set is merely higher th


# 1.2 Now repeat the figure we tried in class which showed forecast accuracy in training and testir
# Plot the accuracy for nearest neighbor predictors across the range of 1 to 50.
# Do this for both the training and testing data as we did in class.

training_accuracy = []
test_accuracy = []

# try n_neighbors from 1 to 51
neighbors_settings = range(1, 51)

for n_neighbors in neighbors_settings:
    clf = KNeighborsClassifier(n_neighbors=n_neighbors)
    clf.fit(X_train, Y_train)
    training_accuracy.append(clf.score(X_train, Y_train))
    test_accuracy.append(clf.score(X_test, Y_test))

plt.plot(neighbors_settings, training_accuracy, label="training accuracy")
plt.plot(neighbors_settings, test_accuracy, label="test accuracy")
plt.ylabel("Accuracy")
plt.xlabel("n_neighbors")
plt.legend()

print("Looking at the model complexity graph, it seems that k should be either 4 or 5 to give us th


"""
This problem explores linear regression along with ridge and lasso in a simple generated monte-carl
First, to get things working you will need to load these routines into python.
"""


# 2.1 Linea Regression

def genLinData(N,M,noise):
    sigNoise = np.sqrt(1./M)
    beta = np.random.normal(size=(M,1),loc=0.,scale=1.)
    beta[abs(beta)<1.0]=0.
    betaUsed= np.sum( beta != 0)
    X = np.random.normal(size=(N,M),loc=0,scale=sigNoise)
    eps = np.random.normal(size=(N,1),loc=0,scale=noise)
    y = X @ beta + eps
    return X,y, betaUsed

nmc = 100
rsquareVec = np.zeros(nmc)
rsquare_train = np.zeros(nmc)
X,y, betaUsed = genLinData(300,50,1)
for i in range(nmc):
```

```
    X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.5)
    reg = LinearRegression()
    reg.fit(X_train, y_train)
    rsquare_train[i] = reg.score(X_train, y_train)
    rsquareVec[i] = reg.score(X_test,y_test)

print("---TRAIN SET---")
print("Mean R^2: {:.2f}".format(np.mean(rsquare_train)))
print("STD R^2: {:.2f}".format(np.std(rsquare_train)))
print("---TEST SET---")
print("Mean of R^2: {:.2f}".format(np.mean(rsquareVec)))
print("STD of R^2: {:.2f}".format(np.std(rsquareVec)))
print("Due to smale sample size of (n=300) and high number of predictors (50), Mean R^2 between tra
print("Therefore, this model is clearly overfitting.")



# 2.2 Regression with sample size 100,000

X,y, betaUsed = genLinData(100000,50,1)
for i in range(nmc):
    X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.5)
    reg = LinearRegression()
    reg.fit(X_train, y_train)
    rsquare_train[i] = reg.score(X_train, y_train)
    rsquareVec[i] = reg.score(X_test,y_test)
print("---TRAIN SET---")
print("Mean R^2: {:.2f}".format(np.mean(rsquare_train)))
print("STD R^2: {:.2f}".format(np.std(rsquare_train)))
print("---TEST SET---")
print("Mean of R^2: {:.2f}".format(np.mean(rsquareVec)))
print("STD of R^2: {:.2f}".format(np.std(rsquareVec)))
print("Yes, the Mean R^2 of train set and test set are very close. The reason is that we increased



# 2.3 Monte-Carlo(iterations 100)

scoreVec = np.zeros(100)
score_train = np.zeros(100)
for i in range(100):
    X, y, betaUsed = genLinData(300, 50, 1.0)
    X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.5)
    reg = LinearRegression()
    reg.fit(X_train, y_train)
    score_train[i] = reg.score(X_train, y_train)
    scoreVec[i] = reg.score(X_test, y_test)

print("---TRAIN SET---")
print("Mean R^2: {:.2f}".format(np.mean(score_train)))
print("STD R^2: {:.2f}".format(np.std(score_train)))
print("---TEST SET---")
print("Mean of R^2: {:.2f}".format(np.mean(scoreVec)))
print("STD of R^2: {:.2f}".format(np.std(scoreVec)))
print("Unlike section 2.1, this model creates new data set everytime it iterates, because the Mote-
print("Due to smale sample size of (n=300) and high number of predictors (50), Mean R^2 between tra
print("Therefore, this model is clearly overfitting.")
```

```
# 2.4 Ridge Regression

X, y, betaUsed = genLinData(300, 50, 1.0)
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.5)

N = range(1, 26)
for i in N:
    my_ridge = Ridge(alpha = i)
    my_ridge.fit(X_train, y_train)
    print("\nTraining Set Score: {:.2f}".format(my_ridge.score(X_train, y_train)))
    print("Test Set Score: {:.2f}".format(my_ridge.score(X_test, y_test)))

# Plot the r-squared values for both training and test sets across all values of alpha
    mglearn.plots.plot_ridge_n_samples()



# 2.5 Lasso Regression

m = np.arange(0.01, 0.04, 0.0005)
for j in m:
    my_lasso = Lasso(alpha = j)
    my_lasso.fit(X_train, y_train)
    print("\nTraining set score: {:.2f}".format(my_lasso.score(X_train, y_train)))
    print("Test set score: {:.2f}".format(my_lasso.score(X_test, y_test)))

# Plot the number of coefficients estimated by Lasso as a franction of true non=zero coefficients
    print("Number of features used: {}".format(np.sum(my_lasso.coef_ != 0)))

# Plot the training and test sets for all values of alpha
    plt.plot(my_lasso.coef_, 'o', label = "Lasso alpha range")
    plt.plot(my_ridge.coef_, 'v', label = "Ridge alpha range")
    plt.legend(ncol= 2, loc= (0, 1.05))
    plt.ylim(-0.80, 0.80)
    plt.xlabel("Coefficient Index")
    plt.ylabel("Coefficient Magnitude")
```