```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Mon May  9 11:59:16 2022

@author: hillarywolff with Jason Winik
"""
import pandas as pd
import numpy as np
from numpy import mean
from numpy import absolute
from numpy import arange
import random

from sklearn.model_selection import train_test_split, cross_val_score,
LeaveOneOut, RepeatedKFold
from sklearn.linear_model import LogisticRegression,LinearRegression,
Lasso, LassoCV, Ridge, RidgeCV
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import PolynomialFeatures


import statsmodels.api as sm
import statsmodels.formula.api as smf

import matplotlib.pyplot as plt
import seaborn as sns

from itertools import combinations

# to ignore warnings
import warnings
warnings.filterwarnings("ignore", category=FutureWarning)

# 1. Chapter 5
# a. *question 6*
random.seed(5)
# 6. We continue to consider the use of a logistic regression model to
# predict the probability of default using income and balance on the
# Default data set. In particular, we will now compute estimates for
# the standard errors of the income and balance logistic regression
coefficients in two different ways: (1) using the bootstrap, and (2)
using
# the standard formula for computing the standard errors in the glm()
# function. Do not forget to set a random seed before beginning your
# analysis.
PATH = r"/Users/hillarywolff/Documents/GitHub/machine_learning/PS3/"
df = pd.read_csv(PATH + 'Default.csv')
# (a) Using the summary() and glm() functions, determine the estimated
standard
```

```
# errors for the coefficients associated with income and balance in a
multiple
# logistic regression model that uses both predictors.
df = df.drop('Unnamed: 0', axis=1)

df['default'] = np.where((df['default'].str.contains('Yes')), 1, 0)
df['student'] = np.where((df['student'].str.contains('Yes')), 1, 0)

X = df[['balance', 'income']]
X = sm.add_constant(X)
y = df['default']

results = sm.Logit(y, X).fit().summary()
print(results)

#                            Logit Regression Results
#
======================================================================
========
# Dep. Variable:                default   No. Observations:
10000
# Model:                          Logit   Df Residuals:
9997
# Method:                           MLE   Df Model:
2
# Date:                Mon, 09 May 2022   Pseudo R-squ.:
0.4594
# Time:                        16:05:26   Log-Likelihood:
-789.48
# converged:                       True   LL-Null:
-1460.3
# Covariance Type:            nonrobust   LLR p-value:
4.541e-292
#
======================================================================
========
#                  coef    std err          z      P>|z|      [0.025
0.975]
#
----------------------------------------------------------------------
--------
# const        -11.5405      0.435    -26.544      0.000     -12.393
-10.688
# balance        0.0056      0.000     24.835      0.000       0.005
0.006
# income      2.081e-05   4.99e-06      4.174      0.000     1.1e-05
3.06e-05
#
======================================================================
========
```

```
# std err for balance = 0.00, std err for income = 4.99e-6

# (b) Write a function, boot.fn(), that takes as input the Default
data
# set as well as an index of the observations, and that outputs
# the coefficient estimates for income and balance in the multiple
# logistic regression model.

def get_indices(data, num_samples):
    '''
    Gets a random subasmple (based on num_samples) of the
    indexes of the dataset (data)
    '''

    return np.random.choice(data.index, int(num_samples),
replace=True)

def boot_fn(data,index):
    '''
    Runs one logistic regression on only the indices specified
    on index that are found in data. It then returns the three
    coefficients associated with the regression.
    '''
    X = data[['balance','income']].loc[index]
    X = sm.add_constant(X)
    y = data['default'].loc[index]

    lr = sm.Logit(y,X).fit(disp=0)
    intercept = lr.params[0]
    coef_balance = lr.params[1]
    coef_income = lr.params[2]
    return [intercept,coef_balance,coef_income]



# (c) Use the boot() function together with your boot.fn() function to
# estimate the standard errors of the logistic regression coefficients
# for income and balance.
def boot(data,func,R):
    intercept = []
    coeff_balance = []
    coeff_income = []
    for i in range(R):

        [inter,balance,income] =
func(data,get_indices(data,len(data)))
        intercept.append(float(inter))
        coeff_balance.append(balance)
```

```
        coeff_income.append(income)

    intercept_statistics =
{'estimated_value':np.mean(intercept),'std_error':np.std(intercept)}
    balance_statistics =
{'estimated_value':np.mean(coeff_balance),'std_error':np.std(coeff_bal
ance)}
    income_statistics =
{'estimated_value':np.mean(coeff_income),'std_error':np.std(coeff_inco
me)}
    return
{'intercept':intercept_statistics,'balance_statistices':balance_statis
tics,'income_statistics':income_statistics}


results = boot(df, boot_fn, 1000)
print('Intercept - ', results['intercept'])
print('Balance - ',results['balance_statistices'])
print('Income - ', results['income_statistics'])

# (d) Comment on the estimated standard errors obtained using the
# glm() function and using your bootstrap function.

# the standard errors from the Logit function were incredibly similar
to the
# standard errors in the bootstrapping function.

# std err balance: 0.0002, std err income: 4.87e-6

######################################################################
#######

# b. *question 8*

# Generate a simulated data set as follows:
#   > set.seed (1)
#   > x <- rnorm (100)
#   > y <- x - 2 * x^2 + rnorm (100)
# In this data set, what is n and what is p? Write out the model
# used to generate the data in equation form.

sim_df = pd.DataFrame()
N = 100
sim_df['x'] = np.random.normal(0, 1, N)
sim_df['y'] = sim_df['x']-2 * sim_df['x'].pow(2) +
np.random.normal(0,1,100)

# N = 100 and p = 2 which is found by looking at Y = X-2X^2+e
```

```
# (b) Create a scatterplot of X against Y . Comment on what you find.

sns.scatterplot(sim_df['x'], sim_df['y'])

# The data is quadratic which we know from the exponent, but a
majority of
# points are located between -1 and 1 and minics a normal distribution
which is
# expected since the simulated dataframe used a mean of 0 and stdev of
1 to
# generate the points.


# (c) Set a random seed, and then compute the LOOCV errors that
# result from fitting the following four models using least squares:
#   i. Y = β0 + β1X + ϵ
#   ii. Y = β0 + β1X + β2X2 + ϵ
#   iii. Y = β0 + β1X + β2X2 + β3X3 + ϵ
#   iv. Y = β0 + β1X + β2X2 + β3X3 + β4X4 + ϵ

# Note you may find it helpful to use the data.frame() function
# to create a single data set containing both X and Y .
random.seed(20)
sim_df['x2'] = np.power(sim_df['x'], 2)
sim_df['x3'] = np.power(sim_df['x'], 3)
sim_df['x4'] = np.power(sim_df['x'], 4)

X1 = sim_df[['x']]
X2 = sim_df[['x', 'x2']]
X3 = sim_df[['x', 'x2', 'x3']]
X4 = sim_df[['x', 'x2', 'x3', 'x4']]
y = sim_df['y']

cv = LeaveOneOut()
model = LinearRegression()

cols = [X1, X2, X3, X4]

def MSE_LOOCV(cols):
    mse_list = []
    for col in cols:
        scores = cross_val_score(model, col, y,
scoring='neg_mean_squared_error',
                                 cv=cv)
        mse_list.append(mean((absolute(scores))))

    return mse_list

sim_mse = MSE_LOOCV(cols)
print(sim_mse)
```

```
# [i: 4.757748722926266, ii: 1.6391419532039742, iii:
1.6672891086484705,
# iv: 1.6796562364607772]

# (d) Repeat (c) using another random seed, and report your results.
# Are your results the same as what you got in (c)? Why?

random.seed(54)
cols = [X1, X2, X3, X4]
new_mse = MSE_LOOCV(cols)
print(new_mse)

# [i: 4.757748722926266, ii: 1.6391419532039742, iii:
1.6672891086484705,
# iv: 1.6796562364607772]

# the results are the same because LOOCV uses N folds from the same
dataset, so
# any iterration of it will be the same.


# (e) Which of the models in (c) had the smallest LOOCV error? Is
# this what you expected? Explain your answer.

# the quadratic model had the smallest LOOCV error. this is expected
since we
# saw in our scatterplot that there was a quadratic relationship.


# (f) Comment on the statistical significance of the coefficient
estimates that
# results from fitting each of the models in (c) using least squares.
Do these
# results agree with the conclusions drawn based on the cross-
validation results?
result = smf.ols(formula="y ~ x+x2", data=sim_df).fit().summary()
print(result)

result = smf.ols(formula="y ~ x+x2+x3", data=sim_df).fit().summary()
print(result)

result = smf.ols(formula="y ~ x+x2+x3+x4",
data=sim_df).fit().summary()
print(result)

#                             OLS Regression Results
#
======================================================================
========
# Dep. Variable:                      y   R-squared:
```

```
0.687
# Model:                            OLS   Adj. R-squared:
0.674
# Method:                 Least Squares   F-statistic:
52.13
# Date:                Mon, 09 May 2022   Prob (F-statistic):
3.67e-23
# Time:                        17:38:51   Log-Likelihood:
-161.66
# No. Observations:                 100   AIC:
333.3
# Df Residuals:                      95   BIC:
346.3
# Df Model:                           4
# Covariance Type:            nonrobust
#
======================================================================
========
#                  coef    std err          t      P>|t|      [0.025
0.975]
#
----------------------------------------------------------------------
--------
# Intercept      0.0714      0.200      0.358      0.721      -0.325
0.468
# x              1.1071      0.279      3.966      0.000       0.553
1.661
# x2            -2.5017      0.372     -6.732      0.000      -3.239
-1.764
# x3            -0.0546      0.152     -0.359      0.721      -0.357
0.248
# x4             0.1792      0.116      1.548      0.125      -0.051
0.409
#
======================================================================
========
# Omnibus:                        3.559   Durbin-Watson:
1.999
# Prob(Omnibus):                  0.169   Jarque-Bera (JB):
2.687
# Skew:                           0.259   Prob(JB):
0.261
# Kurtosis:                       2.386   Cond. No.
14.0
#
======================================================================
========

# Notes:
# [1] Standard Errors assume that the covariance matrix of the errors
```

```
is correctly specified.

# the results from our OLS regression are in line with our LOOCV
results where
# our x and x2 models are significant while x3 and x4 are not.

######################################################################
########

# 2.
# a. *question 11*
# We will now try to predict per capita crime rate in the Boston data
# set.
df = pd.read_csv(PATH+'Boston.csv')

# (a) Try out some of the regression methods explored in this chapter,
# such as best subset selection, the lasso, ridge regression, and
# PCR. Present and discuss results for the approaches that you
# consider.

boston = df
# forward stepwise:


#Add constant to dataframe
boston['constant'] = 1
#specify target
Y = boston['CRIM']
#Variables to use in forward propagation
vars_left_add = boston.columns.tolist()
vars_left_add = [e for e in vars_left_add if e not in ('CRIM',
'constant')]
#Regression type
ols = LinearRegression()
#Starting variables (only constant)
current_vars = ['constant']

X = boston[current_vars]
benchmark_error = np.mean(-1*cross_val_score(ols, X, Y, cv = 5,
scoring = 'neg_mean_squared_error'))
print(' Initial run with only one var (constant term/only bias
weight):', current_vars)
print('      Benchmark error:', benchmark_error)
print('')

for iter in range(len(vars_left_add)):
    print('\033[1m'+ 'Iteration:', iter, '\033[0m')
    error_list = []
    for var in vars_left_add: #For each variable that we can add
        #Modify X according to current iteration
```

```
        X = boston[current_vars + [var]]
        #Perform 5-fold CV to get errors
        error = np.mean(-1*cross_val_score(ols, X, Y, cv = 5, scoring
= 'neg_mean_squared_error'))
        error_list.append(error)
        print(' Running model with:', current_vars + [var])
        print('      Error:', error)

    # Chose the smallest error
    min_error = min(error_list)
    chosen_col_index = error_list.index(min_error)

    # If our current smalles error is smaller than our previous error,
than we add a variable
    # if not, we stop our model
    if min_error<benchmark_error:
        print('          *** Variable selected:',
vars_left_add[chosen_col_index])
        print('          *** Min error selected:', min_error)
        print('          *** Chose the variable that generated the min
error + was lower than previous error')
        print('')
        # Add the variable that produced the smallest error to
current_vars
        current_vars.append( vars_left_add[chosen_col_index] )
        del vars_left_add[chosen_col_index] #delete chosen variable
from vars_left_add
        benchmark_error = min_error # Update benchmark_error
    else:
        print('          \033[4m*** No variable was selected',
'\033[0m')
        print('          *** Previous error rate (',
benchmark_error,') is lower than smallest error rate of this iteration
(', min_error ,')')
        print('          *** Break')
        break

print('')
print('Variables chosen for our model', current_vars)

# Variables chosen for our model ['constant', 'RAD', 'LSTAT', 'ZN']
with error rate of 44.46
result = smf.ols(formula="Y ~ constant+RAD+LSTAT+ZN",
data=boston).fit().summary()


#                              OLS Regression Results
#
======================================================================
========
```

```
# Dep. Variable:                      Y   R-squared:
0.418
# Model:                            OLS   Adj. R-squared:
0.415
# Method:                 Least Squares   F-statistic:
120.3
# Date:                Mon, 09 May 2022   Prob (F-statistic):
1.00e-58
# Time:                        19:57:43   Log-Likelihood:
-1669.0
# No. Observations:                 506   AIC:
3346.
# Df Residuals:                     502   BIC:
3363.
# Df Model:                           3
# Covariance Type:            nonrobust
#
======================================================================
========
#                  coef    std err          t      P>|t|      [0.025
0.975]
#
----------------------------------------------------------------------
--------
# Intercept     -2.4701      0.362     -6.815      0.000      -3.182
-1.758
# constant      -2.4701      0.362     -6.815      0.000      -3.182
-1.758
# RAD            0.5281      0.039     13.578      0.000       0.452
0.605
# LSTAT          0.2574      0.049      5.203      0.000       0.160
0.355
# ZN             0.0205      0.014      1.476      0.140      -0.007
0.048
#
======================================================================
========
# Omnibus:                      676.740   Durbin-Watson:
1.459
# Prob(Omnibus):                  0.000   Jarque-Bera (JB):
88649.095
# Skew:                           6.798   Prob(JB):
0.00
# Kurtosis:                      66.402   Cond. No.
1.85e+16
#
======================================================================
========

np.random.seed(5)
```

```
X = boston[current_vars]
Y = boston['CRIM']
ols = LinearRegression()
np.mean(-1*cross_val_score(ols, X, Y, cv = 5,scoring =
'neg_mean_squared_error'))
# MSE = 44.46



# backward stepwise
vars_left_to_drop = boston.columns.tolist()
vars_left_to_drop = [e for e in vars_left_to_drop if e not in ('CRIM',
'constant')]
#Regression type
ols = LinearRegression()
#Starting variables (only constant)
current_vars = ['constant'] + vars_left_to_drop

X = boston[current_vars]
benchmark_error = np.mean(-1*cross_val_score(ols, X, Y, cv = 5,
scoring = 'neg_mean_squared_error'))
print(' Initial run with all vars:', current_vars)
print('      Benchmark error:', benchmark_error)
print('')

for iter in range(len(vars_left_to_drop)):
    print('\033[1m'+ 'Iteration:', iter, '\033[0m')
    error_list = []
    for var in vars_left_to_drop: #For each variable that we can add
        #Modify X according to current iteration
        vars_to_be_used = ['constant'] + [i for i in vars_left_to_drop
if i != var]
        X = boston[['constant'] + [i for i in vars_left_to_drop if i !
= var]]
        #Perform 5-fold CV to get errors
        error = np.mean(-1*cross_val_score(ols, X, Y, cv = 5, scoring
= 'neg_mean_squared_error'))
        error_list.append(error)
        print(' Running model with:', vars_to_be_used)
        print('      Error:', error)

    # Chose the smallest error
    min_error = min(error_list)
    chosen_col_index = error_list.index(min_error)

    # If our current smallest error is smaller than our previous
error, than we drop the variable associated with it
    # if not, we keep our model
```

```
    if min_error<benchmark_error:
        print('          *** Will drop:',
vars_left_to_drop[chosen_col_index])
        print('          *** Min error selected:', min_error)
        print('          *** Chose the variable that generated the min
error + was lower than previous error')
        print('')
        # Add the variable that produced the smallest error to
current_vars
        current_vars = vars_to_be_used
        del vars_left_to_drop[chosen_col_index] #delete chosen
variable from vars_left_to_drop
        benchmark_error = min_error # Update benchmark_error
    else:
        print('          \033[4m*** No variable was selected',
'\033[0m')
        print('          *** Previous error rate (',
benchmark_error,') is lower than smallest error rate of this iteration
(', min_error ,')')
        print('          *** Break')
        break

print('')
print('Variables chosen for our model', current_vars)

# Variables chosen for our model ['constant', 'ZN', 'INDUS', 'DIS',
'RAD', 'TAX', 'PTRATIO', 'LSTAT'] with error 44.57
result = smf.ols(formula="Y ~
constant+RAD+LSTAT+ZN+INDUS+DIS+PTRATIO+TAX",
data=boston).fit().summary()

#                              OLS Regression Results
#
======================================================================
========
# Dep. Variable:                      Y   R-squared:
0.425
# Model:                            OLS   Adj. R-squared:
0.417
# Method:                 Least Squares   F-statistic:
52.64
# Date:                Mon, 09 May 2022   Prob (F-statistic):
4.56e-56
# Time:                        19:59:32   Log-Likelihood:
-1666.0
# No. Observations:                 506   AIC:
3348.
# Df Residuals:                     498   BIC:
3382.
# Df Model:                           7
```

```
# Covariance Type:            nonrobust
#
==============================================================================
========
#                  coef    std err          t      P>|t|      [0.025
0.975]
#
------------------------------------------------------------------------------
--------
# Intercept     -0.9394      1.510     -0.622      0.534      -3.907
2.028
# constant      -0.9394      1.510     -0.622      0.534      -3.907
2.028
# RAD            0.5326      0.085      6.232      0.000       0.365
0.700
# LSTAT          0.2641      0.053      4.939      0.000       0.159
0.369
# ZN             0.0372      0.019      2.005      0.045       0.001
0.074
# INDUS         -0.1007      0.081     -1.250      0.212      -0.259
0.058
# DIS           -0.5403      0.236     -2.289      0.022      -1.004
-0.077
# PTRATIO        0.0025      0.167      0.015      0.988      -0.327
0.332
# TAX           -0.0006      0.005     -0.121      0.904      -0.011
0.009
#
==============================================================================
========
# Omnibus:                      678.269   Durbin-Watson:
1.484
# Prob(Omnibus):                  0.000   Jarque-Bera (JB):
89641.511
# Skew:                           6.823   Prob(JB):
0.00
# Kurtosis:                      66.762   Cond. No.
4.57e+20
#
==============================================================================
========

np.random.seed(5)

X = boston[current_vars]
Y = boston['CRIM']
ols = LinearRegression()
np.mean(-1*cross_val_score(ols, X, Y, cv = 5,scoring =
'neg_mean_squared_error'))
# MSE= 45
```

```
# (b) Propose a model (or set of models) that seem to perform well on
# this data set, and justify your answer. Make sure that you are
evaluating
# model performance using validation set error, crossvalidation, or
some other
# reasonable alternative, as opposed to using training error.

# our forward stepwise method produced the lowest error which was
44.46 with
# variables ['constant', 'RAD', 'LSTAT', 'ZN']


# (c) Does your chosen model involve all of the features in the data
# set? Why or why not?

# no, it only involves those most pertinant variables that are
relevant for the
# analysis. The process of stepwise goes through all possible
combinations
# of variables to find the lowest error rate and therefore the best
predictors
```