ps2_problem4

May 6, 2020

0.1 IDS/ACM/CS 158: Fundamentals of Statistical Learning

0.1.1 PS2, Problem 4: Linear Regression Analysis of the Prostate Cancer Data

Name: Li, Michael

Email address: mlli@caltech.edu

Notes: Please use python 3.6

You are required to properly comment and organize your code.

- Helper functions (add/remove part label according to the specific question requirements)

[1]:
```
import numpy as np
import numpy.matlib
import scipy.stats
import pandas as pd

def standardize_col(column):
    """
    column - an np array of values from a population

    returns the standardized column with mean 0 and std = 1
    """
    mean = np.mean(column)
    std = np.std(column)

    return (column - mean) / std

def find_beta(data):
    """
    data - a matrix where each row corresponds to the
           p predictors in the first p columns and
           the observed output y in the final column

    returns the OLS estimate of the regression parameter
    """
    x = data[:,:-1]
    y = data[:,-1]
```

```python
    # add bias term to training data
    bias = np.matlib.repmat(1, len(x), 1)
    x = np.concatenate((bias, x), axis=1)

    # calculate beta
    intermediate = np.matmul(x.transpose(), x)
    inverse_intermediate = np.linalg.inv(np.array(intermediate))
    pseudo_x = np.matmul(inverse_intermediate, x.transpose())

    return np.matmul(pseudo_x, y), inverse_intermediate

def predict(ols, data):
    """
    ols - ols estimate of the regression parameter
    data - a matrix where each row corresponds to the
           p predictors in the first p columns and
           the observed output y in the final column

    returns the predictions for the observations in data
    """
    x_with_bias_term = np.concatenate((np.matlib.repmat(1, len(data), 1), data[:
 ↪,:-1]), axis=1)
    return np.matmul(x_with_bias_term, ols)

def rss(data, preds):
    """
    data - a matrix where each row corresponds to the
           p predictors in the first p columns and
           the observed output y in the final column
    preds - the predictions for the observations in data

    returns the residual sum of squares for the values
    """
    return np.sum((data[:,-1] - preds)**2)

def find_sigma(data, preds):
    """
    data - a matrix where each row corresponds to the
           p predictors in the first p columns and
           the observed output y in the final column
    preds - the predictions for the observations in data

    returns sigma hat for the values
    """
    coef = 1 / (len(data) - len(data[0]))
    tot = rss(data, preds)
```

```
    return np.sqrt(coef * tot)

def l2_loss(data, preds):
    """
    data - a matrix where each row corresponds to the
           p predictors in the first p columns and
           the observed output y in the final column
    preds - the predictions for the observations in data

    returns the L2 loss of the values
    """
    return np.mean((data[:,-1] - preds)**2)
```

- Part A

[2]:
```
data = np.genfromtxt('prostate_cancer.csv', delimiter=',', skip_header=1)

standardized_data = data.copy()

for i in range(len(data[0])-2):
    standardized_data[:,i] = standardize_col(data[:,i])

# split the data into train and test
train_data = np.array([observation[:-1] for observation in standardized_data if␣
 ↪observation[-1] == 1])
test_data = np.array([observation[:-1] for observation in standardized_data if␣
 ↪observation[-1] == 0])

# find the OLS estimate and keep track of the inverse for calculations later
ols_full_model, full_model_inverse_intermediate = find_beta(train_data)
```

[3]:
```
ols_full_model
```

[3]:
```
array([ 2.46493292,  0.67601634,  0.26169361, -0.14073374,  0.20906052,
        0.30362332, -0.28700184, -0.02119493,  0.26557614])
```

- Part B

[4]:
```
full_model_training_preds = predict(ols_full_model, train_data)
full_model_sigma = find_sigma(train_data, full_model_training_preds)

# All the values of this part are summarized at bottom of the file in table
```

[5]:
```
z_scores = []

for i in range(len(ols_full_model)):
    z_scores.append(ols_full_model[i] / (full_model_sigma * np.
 ↪sqrt(full_model_inverse_intermediate[i][i])))
```

[6]:
```
z_scores
```

[6]:
```
[27.598203120218404,
 5.366290456150523,
 2.7507893898693854,
 -1.3959089818189607,
 2.055845625930907,
 2.4692551777938245,
 -1.8669126353948005,
 -0.14668120644372185,
 1.737839719569918]
```

[7]:
```
wald_test = []

for i in range(len(ols_full_model)):
    wald_test.append(2*scipy.stats.norm.cdf(-1 * np.abs(z_scores[i])))
```

[8]:
```
wald_test
```

[8]:
```
[1.1693547957255616e-167,
 8.037247566881759e-08,
 0.005945185311053233,
 0.16274190557571133,
 0.039797398582855026,
 0.013539463005511015,
 0.06191378907134302,
 0.8833836532246512,
 0.08223905974843178]
```

[9]:
```
t_test = []

for i in range(len(ols_full_model)):
    t_test.append(2*scipy.stats.t(len(train_data) - len(train_data[0])).cdf(-1␣
 ↪* np.abs(z_scores[i])))
```

[10]:
```
t_test
```

[10]:
```
[4.761696772938845e-35,
 1.4694149583757016e-06,
 0.007917894909336934,
 0.16806259017049052,
 0.044307842021366985,
 0.01650538687470883,
 0.06697084708906915,
 0.8838923143371643,
 0.0875462787480178]
```

[11]:
```
confidence_intervals = []

for i in range(len(ols_full_model)):
    factor = 2 * full_model_sigma * np.
↪sqrt(full_model_inverse_intermediate[i][i])
    interval = [round(ols_full_model[i]-factor, 2),␣
↪round(ols_full_model[i]+factor, 2)]
    confidence_intervals.append(interval)
```

[12]:
```
confidence_intervals
```

[12]:
```
[[2.29, 2.64],
 [0.42, 0.93],
 [0.07, 0.45],
 [-0.34, 0.06],
 [0.01, 0.41],
 [0.06, 0.55],
 [-0.59, 0.02],
 [-0.31, 0.27],
 [-0.04, 0.57]]
```

- Part C

[13]:
```
# find the indexes of the coefficients that are insignificant
insignificant_coefficients = (np.where(np.abs(z_scores) < 2)[0] & np.where(np.
↪array(wald_test) > .05)[0]) - 1

# reduce the dataset and find new OLS estimate and predictions
reduced_train_data = np.delete(train_data, insignificant_coefficients, 1)
reduced_test_data = np.delete(test_data, insignificant_coefficients, 1)
ols_reduced, _ = find_beta(reduced_train_data)
reduced_training_preds = predict(ols_reduced, reduced_train_data)

# calculate rss for both models
rss_h0 = rss(reduced_train_data, reduced_training_preds)
rss_h1 = rss(train_data, full_model_training_preds)
p = len(train_data[0])
p_reduced = len(reduced_train_data[0])

# calculate f and then find p value
f = ((rss_h0 - rss_h1) / (p - p_reduced)) / (rss_h1 / (len(train_data)- p))
f_test_p_val = 1 - scipy.stats.f(p-p_reduced, (len(train_data)- p)).cdf(f)
```

[14]:
```
f_test_p_val
```

[14]:
```
0.16933707265225229
```

- Part D

[15]:
```python
# Base model
b0 = np.mean(train_data[:,-1])
base_err = l2_loss(test_data, b0)

# full model
full_model_testing_preds = predict(ols_full_model, test_data)
full_err = l2_loss(test_data, full_model_testing_preds)

# reduced model
reduced_test_preds = predict(ols_reduced, reduced_test_data)
reduced_err = l2_loss(test_data, reduced_test_preds)

print("Base Model Average Test Error: {}".format(base_err))
print("Full Model Average Test Error: {}".format(full_err))
print("Reduced Model Average Test Error: {}".format(reduced_err))
```

```
Base Model Average Test Error: 1.0567332280603818
Full Model Average Test Error: 0.5212740055076003
Reduced Model Average Test Error: 0.45633212204016255
```

[16]:
```python
params = ['1', 'lcavol', 'lweight', 'age', 'lbph', 'svi', 'lcp', 'gleason',␣
 ↪'pgg45']
pd.DataFrame(data={'OLS estimate': ols_full_model,
                   'z-score': z_scores,
                   'wald test p val': wald_test,
                   't test p val': t_test,
                   '95% CI': confidence_intervals},
             index=['1', 'lcavol', 'lweight', 'age', 'lbph', 'svi', 'lcp',␣
 ↪'gleason', 'pgg45'])
```

[16]:

	OLS estimate	z-score	wald test p val	t test p val	95% CI
1	2.464933	27.598203	1.169355e-167	4.761697e-35	[2.29, 2.64]
lcavol	0.676016	5.366290	8.037248e-08	1.469415e-06	[0.42, 0.93]
lweight	0.261694	2.750789	5.945185e-03	7.917895e-03	[0.07, 0.45]
age	-0.140734	-1.395909	1.627419e-01	1.680626e-01	[-0.34, 0.06]
lbph	0.209061	2.055846	3.979740e-02	4.430784e-02	[0.01, 0.41]
svi	0.303623	2.469255	1.353946e-02	1.650539e-02	[0.06, 0.55]
lcp	-0.287002	-1.866913	6.191379e-02	6.697085e-02	[-0.59, 0.02]
gleason	-0.021195	-0.146681	8.833837e-01	8.838923e-01	[-0.31, 0.27]
pgg45	0.265576	1.737840	8.223906e-02	8.754628e-02	[-0.04, 0.57]