ps4_problem3

May 27, 2020

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

0.1.1 PS4, Problem 3: Logistic Regression Analysis of the Stock Market 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 pandas as pd
import scipy.stats

def logit(x, beta):
    """
    x - a random point p dimensional vector
    beta - an estimate for beta

    returns the logistic function for x using beta
    """
    return np.exp(np.matmul(x.transpose(), beta)) / (1 + np.exp(np.matmul(x.
 ↪transpose(), beta)))

def predict(data, beta):
    """
    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
    beta - coefficient estimates for logistic regression

    returns the probabilities of class 1 for each data point
    """
    x = data[:,:-1]
    bias = np.matlib.repmat(1, len(x), 1)
```

```python
    x = np.concatenate((bias, x), axis=1)

    return np.apply_along_axis(logit, 1, x, beta)

def logistic_regression(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 coefficient estimates for logistic regression using IRLS
    """
    x = data[:,:-1]
    y = data[:,-1]

    bias = np.matlib.repmat(1, len(x), 1)
    x = np.concatenate((bias, x), axis=1)

    # encode the data
    y = np.array([0 if item == -1 else item for item in y])
    tol = .000001

    # initialize beta to 0
    beta = np.array([0]*len(x[0]))
    w_k = None
    iters = 1 # just to prevent divide by 0 warning

    while True:
        p_k = np.apply_along_axis(logit, 1, x, beta)
        p_k_minus = 1 - p_k
        w_k = np.diag(np.multiply(p_k, p_k_minus))
        diff = y - p_k

        intermediate = np.matmul(np.matmul(x.transpose(), w_k), x)
        inverse_intermediate = np.linalg.inv(intermediate)
        pseudo_x = np.matmul(inverse_intermediate, x.transpose())
        beta_next = beta + np.matmul(pseudo_x, diff)

        # stopping condition if our two betas do not change enough
        if iters != 1 and np.linalg.norm(beta-beta_next, 2) / np.linalg.
↪norm(beta, 2) < tol:
            break
        else:
            beta = beta_next
            iters += 1

    return beta
```

```python
def sigma_squared(x, beta, j):
    """
    x - a matrix where each row corresponds to the
        p predictors
    beta - the coefficient estimates for logistic regression
    j - index of predictor

    returns the sigma_squared diagnol element
    """
    bias = np.matlib.repmat(1, len(x), 1)
    x = np.concatenate((bias, x), axis=1)

    p_k = np.apply_along_axis(logit, 1, x, beta)
    p_k_minus = 1 - p_k
    w_k = np.diag(np.multiply(p_k, p_k_minus))

    return np.linalg.inv(np.matmul(np.matmul(x.transpose(), w_k), x))[j][j]

def reduce_data(data, indices):
    """
    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
    indices - which indices to use from the data

    returns the reduced dataset containing only the predictors in indices
    """
    return np.append(data[:,indices], data[:,-1][...,None], 1)
```

- Part A

[2]:
```python
train_data = np.genfromtxt('stock_market_train.csv', delimiter=',',␣
↪skip_header=1)
test_data = np.genfromtxt('stock_market_test.csv', delimiter=',', skip_header=1)
```

[3]:
```python
x = train_data[:,:-1]
y = train_data[:,-1]

beta = logistic_regression(train_data)
z_scores = []

# calculate z_scores for each predictor
for i in range(len(beta)):
    b = beta.copy()
    b[i] = 0
```

```python
    sig = np.sqrt(sigma_squared(x, b, i))
    z_scores.append(beta[i] / sig)

z_scores
```

[3]:
```
[-0.4957056982540905,
 -1.2949980620766537,
 -1.3221554529148911,
 -0.1532395033236583,
 0.2928063364320254,
 1.0797555989798535,
 0.7440645920961294]
```

[4]:
```python
p_vals = []

# calculate p_vals using z_scores
for score in z_scores:
    p_vals.append(2*scipy.stats.norm.cdf(-1*np.abs(score)))

p_vals
```

[4]:
```
[0.6201020661861175,
 0.19532089800477326,
 0.18611639147942072,
 0.8782094063929503,
 0.7696701846565389,
 0.2802510280256544,
 0.45683739909704013]
```

[5]:
```python
params = ['1', 'Lag1', 'Lag2', 'Lag3', 'Lag4', 'Lag5', 'Volume']
pd.DataFrame(data={'OLS estimate': beta,
                   'z-score': z_scores,
                   'p val': p_vals},
             index=params)
```

[5]:

	OLS estimate	z-score	p val
1	-0.135275	-0.495706	0.620102
Lag1	-0.073533	-1.294998	0.195321
Lag2	-0.072720	-1.322155	0.186116
Lag3	-0.008628	-0.153240	0.878209
Lag4	0.016658	0.292806	0.769670
Lag5	0.057834	1.079756	0.280251
Volume	0.132874	0.744065	0.456837

[6]:
```python
test_preds = predict(test_data, beta)
test_preds = [1 if item >= .5 else -1 for item in test_preds]
average_err = sum(test_preds != test_data[:,-1]) / len(test_preds)
```

```
average_err
```

[6]: 0.496

- Part B

[7]:
```
most_significant_indexes = [0, 1]
train_data = reduce_data(train_data, most_significant_indexes)
new_beta = logistic_regression(train_data)
```

[8]:
```
test_data = reduce_data(test_data, most_significant_indexes)
test_preds = predict(test_data, new_beta)
test_preds = [1 if item >= .5 else -1 for item in test_preds]
average_err = sum(test_preds != test_data[:,-1]) / len(test_preds)
average_err
```

[8]: 0.472

[9]:
```
# find test errors for g_0 and g_1
n_10 = 0
n_11 = 0
n_00 = 0
n_01 = 0

for i in range(len(test_preds)):
    if test_data[:,-1][i] == 1:
        if test_preds[i] == 1:
            n_00 += 1
        else:
            n_10 += 1
    else:
        if test_preds[i] == 1:
            n_01 += 1
        else:
            n_11 += 1
```

[10]:
```
g_0_test_err = n_01 / (n_00 + n_01)
g_1_test_err = n_10 / (n_11 + n_10)
g_0_test_err, g_1_test_err
```

[10]: (0.4603174603174603, 0.5081967213114754)

From these errors we see that the model is wrong 46% of the time when it predicts the market will go up and 51% of the time when it predicts the market will go down. From this, we know the model is as good as guessing thus I would avoid trades using this model.