hw3_Appendix

October 20, 2020

[89]:

```
import csv
import numpy as np
import copy
import time
import math
from scipy import stats
```

[2]:

```
def split_train_label(data):
    train_x = []
    train_y = []
    for i in data:
        train_x.append(i[1:])
        train_y.append([i[0]])

    return train_x,train_y
```

[3]:

```
with open('titanic_data.csv','r') as file:
    temp = csv.reader(file)
    data = list(temp)

header = data[0]
data = data[1:]
for i in range(len(data)):
    row_len = len(data[0])
    for j in range(row_len):
        data[i][j] = float(data[i][j])

train_x, train_y = split_train_label(data)
```

[4]:

```
header
```

[4]:

```
['Survived',
 'Pclass',
 'Sex',
 'Age',
 'Siblings/Spouses Aboard',
 'Parents/Children Aboard',
 'Fare']
```

1 one hot encoding and normalization

[5]:

```
#normalization
for i in range(len(header[1:])):
    total = 0
    avg = 0
    if header[1:][i] not in ['Sex','Pclass']:
        for j in train_x:
            total += j[i]
        avg = total/len(train_x)
        for j in range(len(train_x)):
            train_x[j][i] = train_x[j][i]/avg
```

[7]:

```
## encoding class

total_data = []
for i in train_x:
    temp = []
    if i[0] == 1:
        temp.append(1)
    else:
        temp.append(0)
    if i[0] == 2:
        temp.append(1)
    else:
        temp.append(0)
    if i[0] == 3:
        temp.append(1)
    else:
        temp.append(0)
    temp = temp + i[1:]
    total_data.append(temp)
train_x = total_data
```

2 logistic regression

[66]:

```
# each x data is a row vector. y is a column vector. Need header to do␣
 ↪normalization

class logistic_titanic:
    def __init__(self, gradientRate= 3/4, max_iter = 1000, abstol = 1e-3,␣
 ↪add_intercept = True):
        self.max_iter =  max_iter
        self.abstol = abstol
        #self.reltol = reltol
        self.add_intercept = add_intercept
        self.gradientRate = gradientRate
        self.likelihoodScore = None

    def likelihood_score(self):
        likelihood = 0
        for i in range(len(self.training_y)):
            temp = 0
            x = np.array([self.training_x[i]]).T
            temp += self.training_y[i]*math.log(1/(1+ (math.exp((-np.dot(self.
↪theta.T,x))))))
            temp += (1-self.training_y[i])*math.log((1/(1+ (math.exp((np.
↪dot(self.theta.T,x)))))))
            likelihood += temp
        self.likelihoodScore = likelihood
        return likelihood
    def gradient(self):
        gradient = np.zeros((len(self.training_x[0]),1))
        for i in range(len(self.training_y)):
            x = np.array([self.training_x[i]]).T
            temp = self.training_y[i] - (1/(1+ (math.exp((-np.dot(self.theta.
↪T,x))))))
            gradient = gradient + temp*x

        return gradient

    def logistic_predict(self):
        answer = []
        for i in range(len(self.training_y)):
            temp = 0
            temp_x = np.array([self.training_x[i]]).T
            temp = 1/(1+math.exp(-np.dot(self.theta.T,temp_x)))
            if temp > 1/2:
                answer.append(1)
            else:
                answer.append(0)
        return answer

    def predict(self,x):##prediction one data
        temp_x = copy.deepcopy(x)
        if self. add_intercept == True:
            temp_x.append(1)
        temp_x = np.array([temp_x]).T

        temp = 1/(1+math.exp(-np.dot(self.theta.T,temp_x)))
        if temp > 1/2:
            return 1, temp
        else:
            return 0, temp

    def hessian(self):
        hessian = []
        temp = [0] * len(self.training_x[0])
        for i in range(len(self.training_x[0])):
            hessian.append(temp)

        hessian = np.array(hessian)
        for i in range(len(self.training_y)):
            temp = 0
            x = np.array([self.training_x[i]]).T
            temp = math.exp((-np.dot(self.theta.T,x)))/((1+math.exp((-np.
↪dot(self.theta.T,x))))**2)
            temp = temp * np.dot(x,x.T)
            hessian = hessian + temp

        return hessian
    def fit(self,x,y):
        ## deep copy data
        self.training_x = np.array(copy.deepcopy(x))
        self.training_y = np.array(copy.deepcopy(y))

        ## add intercept
        data_num = len(self.training_x)
        if self. add_intercept == True:
            temp = []
            for i in range(data_num):
                temp.append([1])
            self.training_x = np.append(self.training_x,temp,axis = 1)

        ## initialize theta
        theta = []

        for i in range(len(self.training_x[0])):
            theta.append([1])

        self.theta = np.array(theta)


        ## start training
        last_likelihood = float('-inf')
        parameter_rate = 1/20
        for i in range(self.max_iter):
            if i % 50 == 0:
                parameter_rate = parameter_rate*self.gradientRate
            current_likelihood = self.likelihood_score()
            if abs(current_likelihood - last_likelihood) <= self.abstol:
                break
            last_likelihood = current_likelihood
            gradient_val = self.gradient()
            self.theta = self.theta + parameter_rate*gradient_val

        return(self.theta)
```

[67]:

```
a =   logistic_titanic()
```

[68]:

```
start = time.time()
theta = a.fit(train_x, train_y)
end = time.time()
print('time = ', end - start)
```

```
time =  19.7120578289032
```

[69]:

```
current_hessian = a.hessian()

fisher = np.linalg.inv(current_hessian)/len(train_x)

x_my = np.array([[0,1,0,0,25,0,0,50,1]]).T
w_variance = np.dot(x_my.T,np.dot(fisher,x_my))
```

[102]:

```
prediction, odds = a.predict([0,1,0,0,25,0,0,50])
print(prediction)
```

```
0
```

[74]:

```
for i in range(9):
    print(fisher[i][i])
```

```
12205141649.06177
12205141649.061646
12205141649.061592
4.591772226672494e-05
6.0937098097135744e-05
3.886509845182897e-06
2.370755587569964e-06
6.992079378802705e-06
12205141649.06141
```

[93]:

```
for i in range(len(theta)):
    print("For feature", i+1, " :")
    score = (theta[i]**2)/fisher[i][i]
    print("    Chisquare score", (theta[i]**2)/fisher[i][i]   )
    test = stats.chi2.cdf(score, 1, loc=0, scale=1)
    if stats.chi2.cdf(score, 1, loc=0, scale=1) > 0.95:
        print("    P-value = ",test, "> 95%", "Therefore, feature" ,i+1 ,"is␣
↪significant.")
    else:
        print("    P-value = ",test, "< 95%", "Therefore, feature" ,i+1 ,"is␣
↪not significant.")
    print('----------------------------------------')
```

```
For feature 1  :
    Chisquare score [2.66124895e-10]
    P-value =  [1.3016158e-05] < 95% Therefore, feature 1 is not significant.
----------------------------------------
For feature 2  :
    Chisquare score [2.64299876e-11]
    P-value =  [4.1019328e-06] < 95% Therefore, feature 2 is not significant.
----------------------------------------
For feature 3  :
    Chisquare score [3.40895349e-11]
    P-value =  [4.65854826e-06] < 95% Therefore, feature 3 is not significant.
----------------------------------------
For feature 4  :
    Chisquare score [167026.22901181]
    P-value =  [1.] > 95% Therefore, feature 4 is significant.
----------------------------------------
For feature 5  :
    Chisquare score [30900.58985649]
    P-value =  [1.] > 95% Therefore, feature 5 is significant.
----------------------------------------
For feature 6  :
    Chisquare score [12149.26664875]
    P-value =  [1.] > 95% Therefore, feature 6 is significant.
----------------------------------------
For feature 7  :
    Chisquare score [707.10503867]
    P-value =  [1.] > 95% Therefore, feature 7 is significant.
----------------------------------------
For feature 8  :
    Chisquare score [911.18190862]
    P-value =  [1.] > 95% Therefore, feature 8 is significant.
----------------------------------------
For feature 9  :
    Chisquare score [6.1882233e-12]
    P-value =  [1.98482879e-06] < 95% Therefore, feature 9 is not significant.
----------------------------------------
```