山东大学 计算机科学与技术 学院

机器学习（双语） 课程实验报告

|  |  |  |  |
| --- | --- | --- | --- |
| 学号：201705130113 | 姓名： 黄瑞哲 | | 班级：计科17.3 |
| 实验题目：Logistic回归 | | | |
| 实验学时：2 | | 实验日期： 2019.10.7 | |
| 实验目的：   1. 掌握Logistic回归 2. 掌握牛顿迭代法 3. 能够区分牛顿迭代法和梯度下降法的优劣 | | | |
| 硬件环境：  Intel Core i5-8300H @ 2.3GHz | | | |
| 软件环境：  Windows10 Pro 1903  Python 3.7  Visual Studio Code 1.38.1  Sublime Text 3  MinGW-w64 | | | |
| 实验步骤与内容：   1. 读取数据画出散点图 2. 利用梯度下降法求theta以及迭代过程中的损失函数值     并且绘制决策边界，对[20, 80]做预测。   1. 利用牛顿下降法求theta 2. 牛顿法原理      1. 求海森矩阵      1. 绘制决策边界并且对[20, 80]做预测 | | | |
| 结论分析与体会：  对于梯度下降，  当学习率为0.0009时    发现最终的损失函数只收敛到了0.58左右，而迭代了60000多次。  把学习率调整至0.0015后    损失函数收敛至0.5，迭代80000多次  将学习率调整至0.0025后    发现学习过程中损失函数不断震荡，最终收敛至0.4左右，但是迭代了500000次，相比于之前的效果更佳但是消耗的时间更多。  得到的theta值为  预测的概率为  在学习的过程中，由于需要迭代的次数过多，python不能在短时间内计算出答案，于是使用C++进行计算，计算时间在5s左右，最终把得到的结果在python中可视化。  使用牛顿迭代法后    发现只迭代了400多次就收敛到了0.4，迭代次数相比于梯度下降法大大减小，节省了不少的时间，得到的效果也很好。  得到的theta值为  预测的概率为 | | | |

附录：程序源代码

Python

import numpy as np

def sig(x): return 1 / (1 + np.exp(-x))

def J(feature, label, weights):

ret = 0

m = np.shape(feature)[0]

for i in range(m):

h = sig(feature[i] \* weights)

if h <= 0:

h = 0.0000001

elif h >= 1:

h = 0.9999999

ret += -label[i, 0] \* np.log(h) - (1 - label[i, 0]) \* np.log(1 - h)

return ret / m

def load\_data():

feature = []

label = []

with open('exp2/data/ex2x.dat') as f:

for each\_line in f.readlines():

feature\_temp = []

feature\_temp.append(1)

for data in each\_line.strip().split():

feature\_temp.append(float(data))

feature.append(feature\_temp)

with open('exp2/data/ex2y.dat') as f:

for each\_line in f.readlines():

label\_temp = []

for data in each\_line.strip().split():

label\_temp.append(float(data))

label.append(label\_temp)

return np.mat(feature), np.mat(label)

def load\_result(filename):

w = np.mat(np.zeros((3, 1)))

cost = []

with open(filename) as f:

alpha, w[0, 0], w[1, 0], w[2, 0], \_ = f.readline().split()

for each\_line in f.readlines():

cost.append(each\_line.strip())

return alpha, w, np.mat(cost).T

# 梯度下降 太慢

def fit(feature, label, alpha, epsilon=1e-7):

m, n = np.shape(feature)

# w = np.mat([-4.96253179, 0.07103084, 0.03521583]).T

w = np.mat(np.zeros((n, 1)))

val = 0

iteration = 0

cost = []

while True:

iteration += 1

h = sig(feature \* w)

err = label - h

w = w + alpha \* feature.T \* err / m

last = val

val = J(feature, label, w)

cost.append(val)

if abs(val - last) < epsilon:

break

return w, np.mat(cost).T, iteration

def plot\_fit(feature, label, \*\*kwrags):

x\_pos = []

y\_pos = []

x\_neg = []

y\_neg = []

for (x, y) in zip(feature, label):

if y[0] == 1:

x\_pos.append(x[0, 1])

y\_pos.append(x[0, 2])

else:

x\_neg.append(x[0, 1])

y\_neg.append(x[0, 2])

file = kwrags.get('filename', None)

weights = kwrags.get('weights', [])

cost = kwrags.get('cost', [])

alpha = kwrags.get('alpha', -1)

if file:

alpha, weights, cost = load\_result(file)

print(weights.T)

print(1 - predict(np.mat([1, 20, 80]), weights)) # probability of [20, 80] will not be admiited

import matplotlib.pyplot as plt

plt.figure(str(alpha) if alpha != -1 else "Newton")

plt.subplot(1, 2, 1)

plt.scatter(x\_pos, y\_pos, marker='+', label='Admitted')

plt.scatter(x\_neg, y\_neg, marker='o', label='Not Admitted')

x = np.arange(min(feature[:, 1]), max(feature[:, 1]), 0.1)

y = (-weights[0, 0] - weights[1, 0] \* x) / weights[2, 0]

plt.plot(x, y, 'y-', label='Decision Boundary')

plt.xlabel('x1')

plt.ylabel('x2')

plt.legend()

plt.subplot(1, 2, 2)

plt.plot(np.linspace(0, len(cost), len(cost)), cost)

plt.show()

def predict(data, w):

h = sig(data \* w)

# m = np.shape(h)[0]

# for i in range(m):

# h[i, 0] = 0.0 if h[i, 0] < 0.5 else 1.0

return h

def newton(feature, label, epsilon=1e-6):

m, n = np.shape(feature)

w = np.mat(np.zeros((n, 1)))

val = 0

iteration = 0

cost = []

while True:

iteration += 1

H = np.mat(np.zeros((n, n)))

for i in range(m):

h = sig(feature[i] \* w)

H += (h \* (1 - h))[0, 0] \* feature[i].T \* feature[i]

err = label - sig(feature \* w)

w = w + H.I \* feature.T \* err / m

last = val

val = J(feature, label, w)

cost.append(val[0, 0])

if abs(val - last) < epsilon:

break

return w, cost, iteration

if \_\_name\_\_ == '\_\_main\_\_':

feature, label = load\_data()

'''

exe1 梯度下降法 C++计算结果，Python可视化

'''

# # w, theta, iteration = fit(feature, label, 0.0012)

# result\_file = ["0.000900.txt", "0.001200.txt", "0.001500.txt", "0.001800.txt", "0.001900.txt", "0.002000.txt", "0.002500.txt"]

# for file in result\_file:

# plot\_fit(feature, label, filename="exp2/data/"+file)

'''

exe2 牛顿迭代

'''

w, cost, iteration = newton(feature, label)

plot\_fit(feature, label, weights=w, cost=cost)

C++

#include <algorithm>

#include <cassert>

#include <cmath>

#include <cstdio>

#include <cstring>

#include <iostream>

#include <map>

#include <set>

#include <string>

#include <vector>

#include <queue>

#include <fstream>

#define inf 0x3f3f3f3f

#define cases(t) for (int cas = 1; cas <= int(t); ++cas)

typedef long long ll;

typedef double db;

using namespace std;

#ifdef NO\_ONLINE\_JUDGE

#define LOG(args...) do { cout << #args << " -> "; err(args); } while (0)

void err() { cout << endl; }

template<typename T, typename... Args> void err(T a, Args... args) { cout << a << ' '; err(args...); }

#else

#define LOG(...)

#endif

const db eps = 1e-6;

db x[80][3];

db y[80];

db w[3], ww[3];

db sig(db x) {

return 1.0 / (1.0 + exp(-x));

}

db cal() {

db ret = 0;

for (int i = 0; i < 80; ++i) {

db h = 0;

for (int k = 0; k < 3; ++k) h += w[k] \* x[i][k];

h = sig(h);

ret += -y[i] \* log(h) - (1 - y[i]) \* log(1 - h);

}

return ret / 80;

}

vector<db> fit(db alpha) {

for (int i = 0; i < 3; ++i) w[i] = ww[i] = 0;

db val = 0, last;

vector<db> cost;

while (1) {

for (int j = 0; j < 3; ++j) {

db J = 0;

for (int i = 0; i < 80; ++i) {

db h = 0;

for (int k = 0; k < 3; ++k) h += ww[k] \* x[i][k];

J += (sig(h) - y[i]) \* x[i][j];

}

w[j] -= alpha \* J / 80;

}

for (int i = 0; i < 3; ++i) ww[i] = w[i];

last = val;

val = cal();

cost.push\_back(val);

// LOG(iteration, val - last);

if (fabs(val - last) <= eps) break;

}

return cost;

}

int main() {

ifstream fin("data/ex2x.dat");

for (int i = 0; i < 80; ++i) {

x[i][0] = 1;

fin >> x[i][1] >> x[i][2];

}

fin.close();

fin.open("data/ex2y.dat");

for (int i = 0; i < 80; ++i) {

fin >> y[i];

}

fin.close();

// for (int i = 0; i < 80; ++i) {

// cout << x[i][0] << ' ' << x[i][1] << ' ' << x[i][2] << ' ' << y[i] << endl;

// }

db alpha = 0.002;

vector<db> ret = fit(alpha);

// LOG(alpha, w[0], w[1], w[2]);

string name = "data/" + to\_string(alpha) + ".txt";

ofstream fout(name.c\_str());

fout << alpha << " " << w[0] << " " << w[1] << " " << w[2] << " " << cal() << endl;

for(auto v:ret) fout << v << endl;

return 0;

}

/\*

alpha, w[0], w[1], w[2], fit(alpha) -> 0.0004 -0.00924545 0.0459596 -0.0229827 535

alpha, w[0], w[1], w[2], fit(alpha) -> 0.0005 -0.0110848 0.0466746 -0.0233549 514

alpha, w[0], w[1], w[2], fit(alpha) -> 0.0006 -0.43249 0.0491012 -0.0185842 17245

alpha, w[0], w[1], w[2], fit(alpha) -> 0.0007 -1.10884 0.0519389 -0.0103293 39377

alpha, w[0], w[1], w[2], fit(alpha) -> 0.0008 -1.66343 0.0543808 -0.00361771 53419

alpha, w[0], w[1], w[2], fit(alpha) -> 0.0009 -2.1325 0.0565278 0.00201761 62666

alpha, w[0], w[1], w[2], fit(alpha) -> 0.001 -2.53826 0.0584452 0.00686168 68898

alpha, w[0], w[1], w[2], fit(alpha) -> 0.0012 -3.21363 0.0617594 0.0148612 76044

alpha, w[0], w[1], w[2], fit(alpha) -> 0.0015 -4.10757 0.066715 0.0258728 80821

alpha, w[0], w[1], w[2], fit(alpha) -> 0.0018 -8.70063 0.0939862 0.0776859 183547

\*/