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：正则化 | | | |
| 实验学时：2 | | 实验日期： 2019.10.18 | |
| 实验目的：  掌握线性回归和逻辑回归中的正则化，防止过拟合现象的发生。 | | | |
| 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 | | | |
| 实验步骤与内容：  线性回归   1. 读取数据并绘制散点图 2. 假定H函数为五次多项式     损失函数在L2正则化下为    3． 在不同lambda下利用公式求出最优的theta    然后对与lambda=0,1,10分别画出对应的函数图像    当不做正则化时会发生过拟合现象    当lambda为1时适合做预测    而当lambda为10时发生了欠拟合  逻辑回归   1. 读数据画散点图 2. 定义feature向量是训练数据每一项的单项式组合   对应的在L2正则下损失函数表达式为     1. 利用牛顿迭代法优化theta，其中梯度表达式为     比之前不做正则化时多了一个theta项  同时海森矩阵变为    多了一个对角矩阵   1. 求出theta后再绘制出决策边界     可以看出当lambda为0时发生了过拟合，右下角单独出现了一个区域。  当lambda为1时适合分类  当lambda为10时决策边界已经发生了偏移，为欠拟合。 | | | |
| 结论分析与体会：  在逻辑回归中，增加惩罚项可以看出迭代次数明显减少。  当lambda为0时迭代了13次，L2范式为7172.6662    而到了lambda为10时仅迭代了3次    同时对应的L2范式仅为0.9384  惩罚项的设置，明显地限制了theta的变化，因此可以避免抖动，避免过拟合的发生，但是当lambda设置不当时会导致欠拟合的发生，训练出的模型没有使用价值。因此，选择合适的参数，能够维持模型的鲁棒性。 | | | |

附录：程序源代码

# linear\_regularized.py

import numpy as np

def load\_data():

feature = []

with open("exp3/data/ex3Linx.dat") as f:

for each\_line in f.readlines():

feature\_tmp = []

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

for i in range(6):

feature\_tmp.append(float(data) \*\* i)

feature.append(feature\_tmp)

label = []

with open("exp3/data/ex3Liny.dat") as f:

for each\_line in f.readlines():

label\_tmp = []

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

label\_tmp.append(float(data))

label.append(label\_tmp)

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

def plt\_linear(feature, label, theta, lamb):

import matplotlib.pyplot as plt

plt.figure()

plt.title("%.4f" % lamb)

plt.scatter(feature[:, 1].tolist(), label.tolist(), marker='o')

x = np.arange(min(feature[:, 1]), max(feature[:, 1]), 0.01)

y = []

for i in x:

now = 0

for j in range(6):

now += theta[j, 0] \* (i \*\* j)

y.append(now)

plt.plot(x, y)

plt.show()

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

feature, label = load\_data()

m, n = np.shape(feature)

lamd = [0, 1, 10]

for lam in lamd:

E = np.eye(n)

E[0, 0] = 0

theta = (feature.T \* feature + lam \* E).I \* feature.T \* label

print("lambda=", lam)

print(theta.T)

plt\_linear(feature, label, theta, lam)

# logistic\_regularized.py

import numpy as np

import matplotlib.pyplot as plt

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

def load\_data():

feature = []

with open("exp3/data/ex3Logx.dat") as f:

for each\_line in f.readlines():

feature\_tmp = [1]

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

feature\_tmp.append(float(data))

feature.append(feature\_tmp)

label = []

with open("exp3/data/ex3Logy.dat") as f:

for each\_line in f.readlines():

label\_tmp = []

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

label\_tmp.append(float(data))

label.append(label\_tmp)

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

def map\_feature(feature1, feature2):

x = []

for i in range(7):

for j in range(i + 1):

x.append((feature1 \*\* (i - j)) \* (feature2 \*\* j))

return x

def cost(feature, label, theta, lamb):

ret = 0

m, n = np.shape(feature)

for i in range(m):

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

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

for i in range(1, n):

ret += lamb / 2 \* (theta[i, 0] \*\* 2)

return ret / m

def gradient(feature, label, theta, lamb):

m, n = np.shape(feature)

err = feature.T \* (sig(feature \* theta) - label)

for j in range(1, n):

err[j, 0] += lamb \* theta[j, 0]

return err / m

def hessian(feature, label, theta, lamb):

m, n = np.shape(feature)

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

for i in range(m):

h = sig(feature[i] \* theta)[0, 0]

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

E = np.eye(n)

E[0, 0] = 0

H += lamb \* E

return H / m

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

val = 0

iteration = 0

n = np.shape(feature)[1]

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

while True:

iteration += 1

H = hessian(feature, label, w, lamb)

dJ = gradient(feature, label, w, lamb)

w = w - H.I \* dJ

last, val = val, cost(feature, label, w, lamb)

print("iteration=", iteration, val)

if abs(last - val) <= epsilon:

break

return w

def plt\_logistic(feature, label, theta):

# divide pos and neg

x\_pos = []

x\_neg = []

y\_pos = []

y\_neg = []

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

if y[0] == 0:

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

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

else:

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

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

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

plt.scatter(x\_neg, y\_neg, marker='o')

# plot contour

u = np.linspace(-1, 1.5, 200)

v = np.linspace(-1, 1.5, 200)

z = np.zeros((len(u), len(v)))

for i in range(len(u)):

for j in range(len(v)):

z[j, i] = map\_feature(u[i], v[j]) \* theta

plt.contour(u, v, z, [0], colors='g')

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

feature, label = load\_data()

# all monomials

\_feature = []

for i in range(len(feature)):

\_feature.append(map\_feature(feature[i, 1], feature[i, 2]))

\_feature = np.mat(\_feature)

plt.figure()

# we calculate theta for each lambda

lamb = [0, 1, 10]

for i in range(len(lamb)):

theta = newton(\_feature, label, lamb[i])

print("theta=", theta.T)

print("norm=", np.linalg.norm(theta))

ax = plt.subplot(1, len(lamb), i + 1)

ax.set\_title("%.3f" % lamb[i])

plt\_logistic(feature, label, theta)

plt.show()