山东大学 计算机科学与技术 学院

机器学习 课程实验报告

|  |  |  |  |
| --- | --- | --- | --- |
| 学号：201900130059 | 姓名： 孙奇 | | 班级： 2019级1班 |
| 实验题目：Regularization | | | |
| 实验学时：2 | | 实验日期：2021/10/19 | |
| 实验目的：   1. 在线性回归和逻辑回归中运用正则化，观察正则化对于回归结果的影响； | | | |
| 硬件环境：  CPU: Intel i5-9300H  GPU: UHD630 | | | |
| 软件环境：  Python3.8  PyCharm CE | | | |
| 实验步骤与内容：   1. Linear Regression with Regularization： 2. 五阶多项式回归模型, hypothesis：      1. Loss function：      1. 在不使用正则化（即）的情况下，绘制训练数据的散点图和五阶多项式回归结果：   linear1-1   1. 训练数据只有7个，这种情况下模型回归过拟合情况严重，为了使模型泛化，更具普遍性，加入惩罚项进行正则化； 2. 加入惩罚项 ，在 的情况下观察模型回归效果：   linear1  当时，曲线过拟合，当比较小时，模型回归效果不错，同时也避免了过拟合，当比较大时，模型虽然避免了过拟合，但是回归效果很差，无法用来预测；   1. 查看对L2-Norm()大小的影响：   linear2linear3  发现作为惩罚项系数，在之后整个模型的复杂性变得很低，项对模型的影响力急剧缩小，同时，只要施加了惩罚项，整个模型中的影响力就被大幅缩小了；   1. Logistic Regression with Regularization： 2. Hypothesis function：      1. 为了使模型具有普适性，将x赋值为全部六阶单项式组成的向量，即      1. Loss function：      1. 使用牛顿法求解该逻辑回归问题：      1. 带惩罚项的Hessian矩阵为：      1. 在不使用正则化（即）的情况下，绘制训练数据的散点图和决策边界如图：      1. 加入惩罚项 ，在 的情况下观察模型回归效果：   logistic1  当时，模型过拟合，决策边界对于训练数据的划分过于具体；当比较小时，模型回归效果不错，同时泛化了决策边界，避免了过拟合，当比较大时，模型虽然避免了过拟合，泛化了决策边界，但是决策边界对于训练数据的划分偏差较大，回归效果差无法用来预测；   1. 查看对L2-Norm()大小的影响：   logistic2logistic3  发现作为惩罚项系数，在之后整个模型的复杂性变得很低，项对模型的影响力急剧缩小，同时，只要施加了惩罚项，整个模型中的影响力就被大幅缩小了； | | | |
| 结论分析与体会：   1. 训练数据较少时，正则化对于线性回归和逻辑回归都有很好的避免过拟合的效果； 2. 在正则化的过程中，如果超参数选取合适，能够很好地泛化模型，对于过拟合起到很好的抑制作用，但是如果取值不当，如果太小，起不到抑制过拟合的作用；如果太大，虽然避免了模型的过拟合，但是同时导致的大小对惩罚项的控制力度太大，使模型产生偏差，回归效果较差，因此正则化超参数的选取是重要的； | | | |

附录：程序源代码

1. **map\_feature.py：**

import numpy as np

def map\_feature(feat1, feat2):

degree = 6

out = np.ones(feat1.size)

for i in range(1, degree + 1):

for j in range(i + 1):

out = np.column\_stack( (out, (feat1 \*\* (i - j)) \* (feat2 \*\* j)) )

return out

1. **regularized\_linear\_regression.py：**

import numpy as np

import matplotlib.pyplot as plt

class RegularizedLinearRegression:

def \_\_init\_\_(self, x, y, lr=0.3, r\_lambda=0):

self.x = x

self.y = y

self.lr = lr

self.r\_lambda = r\_lambda

self.expand = lambda x, k: np.array([x \*\* i for i in range(2, k+1)]).T

def train(self):

# expand x

m = self.x.shape[0]

x\_t = np.column\_stack( (np.ones(m), self.x) )

self.x = np.concatenate( (x\_t, self.expand(self.x, 5)), axis=1 )

x\_t = self.x

# Matrix following lambda

n = x\_t.shape[1]

L = np.identity(n)

L[0][0] = 0

# solve theta

theta = np.dot( np.linalg.inv( np.dot(x\_t.T, x\_t) + self.r\_lambda \* L ), np.dot( x\_t.T, self.y ) )

return theta

def plot\_fit\_curve(self, theta):

x\_hat = np.arange(np.min(self.x[:, 1]), np.max(self.x[:, 1]), 0.01)

x = np.column\_stack( (np.ones(len(x\_hat)), x\_hat) )

x = np.concatenate( (x, self.expand(x\_hat, 5)), axis=1 )

y\_hat = np.dot(x, theta)

plt.plot(x\_hat, y\_hat, '--', label=f'Lambda={self.r\_lambda}')

if \_\_name\_\_ == "\_\_main\_\_":

x = np.loadtxt("data3/ex3Linx.dat")

y = np.loadtxt("data3/ex3Liny.dat")

plt.figure(1)

plt.plot(x, y, 'o', label='Training data')

lambdas = [0]

thetas = []

for r\_lambda in lambdas:

regLinear = RegularizedLinearRegression(x, y, r\_lambda=r\_lambda)

theta = regLinear.train()

thetas.append(theta)

print(theta)

regLinear.plot\_fit\_curve(theta)

plt.title("Regularized Linear Regression")

plt.legend()

plt.xlabel("x")

plt.ylabel("y")

# l2\_norm changes by lambda

l2\_norms = [np.linalg.norm(theta) for theta in thetas]

plt.figure(2)

plt.plot(lambdas, l2\_norms, 'o')

plt.title(r"L2\_Norm by $\lambda$")

plt.xlabel(r"$\lambda$")

plt.ylabel("L2-Norm")

# easy to observe

plt.figure(3)

plt.plot(lambdas[1:], l2\_norms[1:])

plt.title(r"L2\_Norm by $\lambda$")

plt.xlabel(r"$\lambda$")

plt.ylabel("L2-Norm")

plt.show()

1. **regularized\_logistic\_regression.py：**

import numpy as np

import matplotlib.pyplot as plt

from map\_feature import map\_feature

def Sigmoid(z):

return 1 / (1 + np.exp(-z))

class RegularizedLogisticRegression:

def \_\_init\_\_(self, x, y):

self.x = x

self.y = y

def Newton(self, r\_lambda=0):

m, n = self.x.shape

theta = np.zeros((n, 1))

# matrix L

L = np.identity(n)

L[0][0] = 0

# loop

loop\_max = 100

loop = 0

pre\_loss = 0

loss\_list = []

print(f'---------------------Lambda={r\_lambda}-------------------')

for i in range(loop\_max):

h = Sigmoid(np.dot(self.x, theta))

# loss

loss = -1/m \* np.sum( (self.y \* np.log(h) + (1 - self.y) \* np.log(1 - h)) ) + r\_lambda/(2 \* m) \* np.sum(theta[1:] \*\* 2)

print(f'Loss = {loss}')

loss\_list.append(loss)

if abs(loss - pre\_loss) < 1e-6:

break

pre\_loss = loss

# gradient: 28 \* 1

grad = 1/m \* np.dot(self.x.T, (h - self.y))

for i in range(1, n):

grad[i] += r\_lambda/m \* theta[i]

# update theta

# hessain: 28 \* 28

hessian = 0

for i in range(m):

hessian += h[i] \* (1 - h[i]) \* np.dot(self.x.T[:, i].reshape(-1, 1), self.x[i, :].reshape(1, -1))

hessian /= m

hessian += r\_lambda/m \* L

# theta: 28 \* 1

theta -= np.dot(np.linalg.inv(hessian), grad)

loop += 1

return theta

if \_\_name\_\_ == "\_\_main\_\_":

# load and scatter

x = np.loadtxt("data3/ex3Logx.dat", delimiter=',')

y = np.loadtxt("data3/ex3Logy.dat", delimiter=',').reshape(-1, 1)

pos = np.where(y == 1)

neg = np.where(y == 0)

# train

regLogistic = RegularizedLogisticRegression(map\_feature(x[:, 0], x[:, 1]), y)

lambdas = [0, 1, 3, 5, 7, 10]

thetas = []

# boundary

plt.figure(1)

rows = cols = int(np.sqrt(len(lambdas)))

if rows \* cols < len(lambdas):

cols += 1

for k in range(len(lambdas)):

plt.subplot(rows, cols, k + 1)

plt.scatter(x[pos, 0], x[pos, 1], marker='o')

plt.scatter(x[neg, 0], x[neg, 1], marker='+')

# solve theta

theta = regLogistic.Newton(r\_lambda=lambdas[k])

thetas.append(theta)

# plot boundary

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[i, j] = np.dot(map\_feature(u[i], v[j]), theta)

plt.contour(u, v, z.T, [0])

plt.title(f'$\lambda$={lambdas[k]}')

plt.xlabel('u')

plt.ylabel('v')

# lambda affects results

plt.figure(2)

l2\_norms = [np.linalg.norm(theta) for theta in thetas]

plt.plot(lambdas, l2\_norms, 'o')

plt.xlabel(r'$\lambda$')

plt.ylabel('L2-Norm')

# easy to observe

plt.figure(3)

plt.plot(lambdas[1:], l2\_norms[1:])

plt.xlabel(r'$\lambda$')

plt.ylabel('L2-Norm')

plt.show()