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：Naive Bayes | | | |
| 实验学时：2 | | 实验日期：2021/11/08 | |
| 实验目的：   1. 使用Naive Bayes在数据集上对结果进行推断； 2. 切分数据集，观察不同大小的training\_data在Naive Bayes模型下结果的变化； | | | |
| 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. Naive Bayes的多变量分类问题： 2. Training\_data：   截屏2021-11-01 下午1.57.32  每一个training\_data的自变量x是一个维向量，x的第j个feature 有v种不同的取值；  （2） 多分类问题的空间为：  截屏2021-11-01 下午1.59.59   1. 极大似然函数：   截屏2021-11-01 下午9.37.31   1. 分类问题可建模为：   截屏2021-11-01 下午9.38.40   1. 根据拉格朗日乘数法求的最优化结果：   截屏2021-11-01 下午1.40.06     1. 引入Laplace smoothing：      1. 本问题下Laplace smoothing后得到：   截屏2021-11-01 下午9.44.44   1. 训练&预测过程： 2. 在training\_data上建立Naive Bayes模型，计算出相应的和 3. 基于计算得到的和，在test\_data上面对y各种取值求解概率大小，保留最大概率的y作为对应x取值下的预测结果，将其与真实y值进行比较，得到预测准确率； 4. 基于不同的数据大小得到的预测结果如下：   Figure_1 | | | |
| 结论分析与体会：   1. Naive Bayes假设了x的各个feature是独立同分布的，因此才能够利用该模型进行预测，预测效果不错； 2. 数据集大小会影响预测效果，当数据集十分小时，模型不具有普遍性，因此预测的准确率会降低，当数据集足够大时（此问题中超过2000个数据），继续增大数据集大小，预测的准确度不会大幅提升，而是在轻微波动； | | | |

附录：程序源代码

import numpy as np

import matplotlib.pyplot as plt

class NaiveBayes:

def \_\_init\_\_(self, training\_data, test\_data):

self.training\_x = training\_data[:, :-1]

self.training\_y = training\_data[:, -1]

self.test\_x = test\_data[:, :-1]

self.test\_y = test\_data[:, -1]

self.py = np.zeros(5)

self.pjxy = np.zeros((8, 5, 5))

def get\_py\_with\_ls(self, count\_y, batch\_size, number\_of\_value\_y):

self.py = np.zeros(5)

for i in range(5):

self.py[i] = (count\_y[i] + 1) / (batch\_size + number\_of\_value\_y)

def get\_pjxy\_with\_ls(self, count\_jxy, count\_y, number\_of\_value\_x):

self.pjxy = np.zeros((8, 5, 5))

for j in range(8):

for x in range(5):

for y in range(5):

self.pjxy[j][x][y] += (count\_jxy[j][x][y] + 1) / (count\_y[y] + number\_of\_value\_x[x])

def max\_likelihood(self, x):

pred\_y = self.py.copy()

for y in range(5):

for j in range(8):

pred\_y[y] \*= self.pjxy[j][x[j]][y]

return np.argmax(pred\_y)

def train(self, training\_data):

self.training\_x = training\_data[:, :-1]

self.training\_y = training\_data[:, -1]

# count\_x[j][x]: 第j个feature, xj = x的个数

count\_x = np.zeros((8, 5))

# count\_y[y]: y\_label = y的个数

count\_y = np.zeros(5)

# count\_jxy[j][x][y]: 第j个feature，xj = x and y = y的个数

count\_jxy = np.zeros((8, 5, 5))

# 遍历所有training\_data, 记录每一行中count\_x和count\_y

for data\_x, data\_y in zip(self.training\_x, self.training\_y):

count\_y[data\_y] += 1

for j in range(8):

count\_x[j][data\_x[j]] += 1

count\_jxy[j][data\_x[j]][data\_y] += 1

number\_of\_value\_x = np.array([np.sum(count\_x[j] > 0) for j in range(8)])

number\_of\_value\_y = np.sum(count\_y > 0)

self.get\_py\_with\_ls(count\_y, training\_data.shape[0], number\_of\_value\_y)

self.get\_pjxy\_with\_ls(count\_jxy, count\_y, number\_of\_value\_x)

def predict(self, batch\_size):

right\_count = 0

m = self.test\_x.shape[0]

for data\_x, data\_y in zip(self.test\_x, self.test\_y):

pred\_y = self.max\_likelihood(data\_x)

if data\_y == pred\_y:

right\_count += 1

print(f'accuracy of batch size = {batch\_size}: {right\_count / m}')

return right\_count / m

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

# load data & initial model

training\_data = np.loadtxt("data4/training\_data.txt", dtype=int)

test\_data = np.loadtxt("data4/test\_data.txt", dtype=int)

nb = NaiveBayes(training\_data, test\_data)

batch\_size = np.arange(100, 1000, 100)

batch\_size = np.concatenate((batch\_size, np.arange(1000, 11000, 1000)))

test\_acc\_list = []

for size in batch\_size:

print(size)

np.random.shuffle(training\_data)

nb.train(training\_data[:size, :])

test\_acc\_list.append(nb.predict(size))

plt.figure(1)

plt.grid()

plt.ylim([0.5, 1])

plt.xlabel('Batch size')

plt.ylabel('Accuracy')

plt.plot(batch\_size, test\_acc\_list)

plt.show()