1. （必填）自己提出的问题的理解（罗列全部）：
2. 提出的问题1：多项式模型和伯努利模型在实现上有什么区别，如何评价优劣

讨论后的理解：伯努利模型并不考虑词在文档中出现的次数，只考虑出不出现，在这个意义上相当于假设词是等权重的。多项式模型考虑此在文档中的出现次数。二者的计算粒度不一样，多项式模型以单词为粒度，伯努利模型以文件为粒度。当训练集文本较短时，我们更倾向于使用贝努利模型。而文本较长时，我们更倾向于多项式模型，因为，在一篇文档中的高频词，会使该词的似然概率值相对较大。这两种方法是用来处理离散数据的，对于连续型数据采用高斯模型。

提出的问题2：如何解决朴素贝叶斯模型常出现的数据下溢出问题

讨论后的理解：对概率取对数

1. （必填）别人提出的问题的理解（选择几个问题罗列，并给出理解）：
2. 问题3：为什么说朴素贝叶斯是稳定的算法？

自己的理解：因为朴素贝叶斯是基于古典数学理论，使用已经获得的数据的先验概率，通过先验概率来计算出后验概率，以此对一条数据进行预测，得到需要的与该数据所对应的类别，因为先验概率是固定的，计算的方法和路径的都是固定的，因此每次计算所得的后验概率都是相同的，所以这是一种稳定的方法。

问题4：拉普拉斯修正是否有可能改变原先的预测结果？

自己的理解： 首先拉普拉斯修正可以有效的避免因训练样本不足而导致概率估值为零的问题，并且在训练集变大的时候，修正所引入的先验的影响也会逐渐变得可忽略，使得估值趋向于实际概率值

并且因为拉普拉斯引入的分母n是类别c的个数，使得修正较为平滑，所以拉普拉斯修正是比较优秀的。

问题5：朴素贝叶斯中每个属性对于类别决策都有相同的影响，不符合实际i情况，怎么解决这个问题？

自己的理解：可以对各特征属性赋予一个权值，通过属性的重要性来确定权重

三、（必填）读书计划

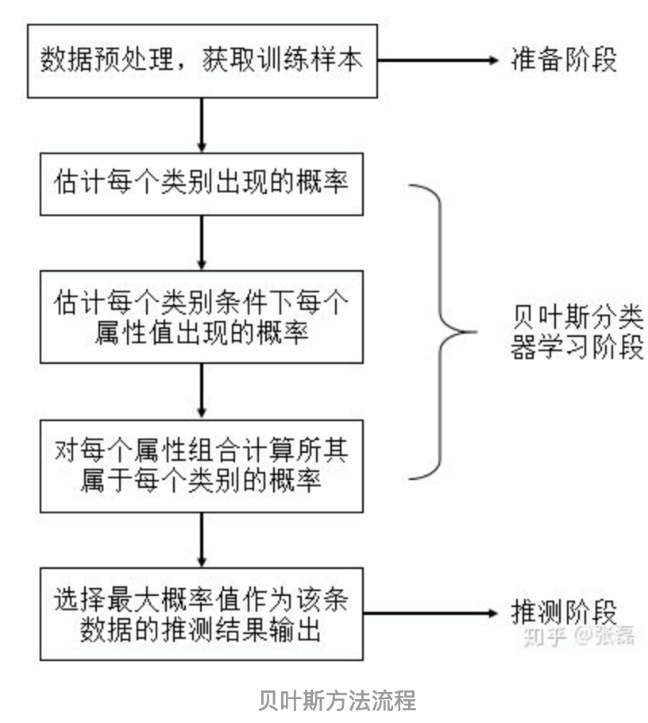
1、本周完成的内容章节：李航书第四章

2、下周计划：李航书第五章

四、（选做）读书摘要及理解或伪代码的具体实现（读书摘要、伪代码的具体实现代码等可以写到这个部分）

1、读书摘要及理解（选做）

#### 算法流程



实际应用方式：

- 若任务对预测速度要求较高，则对给定的训练集，可将朴素贝叶斯分类器涉及的所有概率估值事先计算好存储起来，这样在进行预测时只需要 “查表” 即可进行判别；

- 若任务数据更替频繁，则可采用 “懒惰学习” (lazy learning) 方式，先不进行任何训练，待收到预测请求时再根据当前数据集进行概率估值；

- 若数据不断增加，则可在现有估值的基础上，仅对新增样本的属性值所涉及的概率估值进行计数修正即可实现增量学习。

2.代码实现

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.datasets import load\_iris

from sklearn.model\_selection import train\_test\_split

from collections import Counter

import math

# data

def create\_data():

iris = load\_iris()

df = pd.DataFrame(iris.data, columns=iris.feature\_names)

df['label'] = iris.target

df.columns = ['sepal length', 'sepal width', 'petal length', 'petal width', 'label']

data = np.array(df.iloc[:100, :])

# print(data)

return data[:,:-1], data[:,-1]

X, y = create\_data()

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3)

class NaiveBayes:

def \_\_init\_\_(self):

self.model = None

# 数学期望

@staticmethod #这里定义为成员函数也是可以的

def mean(X):

return sum(X) / float(len(X))

# 标准差（方差）

def stdev(self, X):

avg = self.mean(X)

return math.sqrt(sum([pow(x-avg, 2) for x in X]) / float(len(X))) #这里不用for x in X也可以

# 概率密度函数

def gaussian\_probability(self, x, mean, stdev):

exponent = math.exp(-(math.pow(x-mean,2)/(2\*math.pow(stdev,2))))

return (1 / (math.sqrt(2\*math.pi) \* stdev)) \* exponent

# 处理X\_train

def summarize(self, train\_data):

summaries = [(self.mean(i), self.stdev(i)) for i in zip(\*train\_data)] #注意这里的zip(\*train\_data)的用法

return summaries

# 分类别求出数学期望和标准差

def fit(self, X, y):

labels = list(set(y))

data = {label:[] for label in labels}

for f, label in zip(X, y):

data[label].append(f)

self.model = {label: self.summarize(value) for label, value in data.items()}

return 'gaussianNB train done!'

# 计算概率

def calculate\_probabilities(self, input\_data):

# summaries:{0.0: [(5.0, 0.37),(3.42, 0.40)], 1.0: [(5.8, 0.449),(2.7, 0.27)]}

# input\_data:[1.1, 2.2]

probabilities = {}

for label, value in self.model.items():

probabilities[label] = 1

for i in range(len(value)):

mean, stdev = value[i]

probabilities[label] \*= self.gaussian\_probability(input\_data[i], mean, stdev) #probability[label]=1???

return probabilities

# 类别

def predict(self, X\_test):

# {0.0: 2.9680340789325763e-27, 1.0: 3.5749783019849535e-26}

label = sorted(self.calculate\_probabilities(X\_test).items(), key=lambda x: x[-1])[-1][0] #sorted之后是一个list[tuple, tuple]

return label

def score(self, X\_test, y\_test):

right = 0

for X, y in zip(X\_test, y\_test):

label = self.predict(X)

if label == y:

right += 1

return right / float(len(X\_test))

model = NaiveBayes()

model.fit(X\_train, y\_train)

print(model.predict([4.4, 3.2, 1.3, 0.2]))

print(model.score(X\_test, y\_test))