1. **自己提出的问题的理解：**

问题1：为什么说朴素贝叶斯是稳定的算法？

讨论后的理解：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2：既然朴素贝叶斯在各个方面都非常优秀，那么它有什么缺点呢？

讨论后的理解：朴素贝叶斯在理论上是非常优秀，但是在实际使用中，却并非如此，因为朴素贝叶斯的前提条件是属性需要相互独立，然而在实际应用中却是不成立的，朴素贝叶斯在实际使用中的效果往往是和属性的个数成反比的。不过也可以通过一些方法来改进这种情况，比如半朴素贝叶斯等。

1. **别人提出的问题的理解：**

问题1：Navie Bayes和Logistic回归区别是什么：

自己的理解：

朴素贝叶斯模型是生成模型，它通过已有数据生成一个模型，然后使用该模型来对数据进行分类，而Logistic模型是判别式模型，它使用数据训练得到参数，然后通过参数来预测数据的类别。相对来说，Logistic模型比朴素贝叶斯模型预测更加准确，因为它是直接预测，而朴素贝叶斯则是生成模型之后预测。同时，因为要生成模型，需要使用全部数据，所以朴素贝叶斯模型更加耗费时间和资源。

1. 问题2：贝叶斯和朴素贝叶斯的区别

自己的理解：

1）贝叶斯是一种数学理论，贝叶斯方法的特点是结合先验概率和后验概率，不只是使用先验概率，所以有相对比较高的准确率。和朴素贝叶斯算法来说，它是一种更大的概念，它包括了朴素贝叶斯算法。

2）朴素贝叶斯算法

朴素贝叶斯是一种分类算法，它使用贝叶斯理论，使用先验概率和条件概率来生成后验概率，生成模型来对数据类别进行预测，它是贝叶斯分类的一种具体方法。

1. 问题3：朴素贝叶斯的优点和缺点分别有哪些？

自己的理解：

优点：

（1）朴素贝叶斯效率相对其他分类算法来说效率比较高。

（2）对于多分类任务处理也比较有优势。

（3）算法简单，实现起来也非常容易。

朴素贝叶斯的主要缺点有：

（1）虽然理论上来说朴素贝叶斯的分类准确率很高，但是因为它的前提条件是需要各个属性是相互独立的，但是这个条件在现实中是不成立的，所以效果相对理论来说要差一些的，但是其实践中表现依然不错。

（2）决策效果依赖于假设。

（3）对输入数据的表达形式很敏感。

1. 读书计划

1、本周完成的内容章节：《统计学习方法》第3章

2、下周计划：《统计学习方法》第4章

1. 读书摘要及理解或伪代码的具体实现
2. **读书摘要：**

**4.1朴素贝叶斯的学习与分类**

（1）朴素贝叶斯法对条件概率分布作了条件独立性的假设。由于这是一个较强的假

设，朴素贝叶斯法也由此得名

（2）朴素贝叶斯实际学习到生成数据的机制，所以属于生成模型。

（3）朴素贝叶斯有一个先决条件即条件独立假设，这个条件假设各个属性都是独立分布的，这个假设使得朴素贝叶斯变得非常简单，但是因为实际中并不满足，所以会牺牲一定的准确性。

4.2.3贝叶斯估计

因为在使用极大似然估计的时候有可能会出现概率值为0的情况，为了解决这种问题，所以需要在分子和分母上同时加上一个常数，这种方法称为拉普拉斯平滑。

1. **代码实现**

class NaiveBayes(object):

    def \_\_init\_\_(self,lamda,train,labels):

        self.label\_map = {1:"水果",0:"电器"}

        self.lamda = lamda

        self.train = train

        self.labels = set(labels)

        #label\_prob\_pre : 每个类别的先验概率，即Pr(cj)

        labels = list(self.labels)

        label\_set = set(labels)

        self.label\_prob\_pre = {}

        for label in label\_set:

            self.label\_prob\_pre[label] = labels.count(label)/float(len(labels))

        #probs\_wc:Pr(w|c)

        word\_list = [word for sen in self.train for word in sen[0]]

        self.words\_set = set(word\_list)

        self.probs\_wc = {}

        for c in labels:

            for word in self.words\_set:

                self.probs\_wc[(word,c)]=self.get\_prob\_wc(word,c)

    def get\_prob\_wc(self,word,c):

        #求Pr(w|c)

        num\_word\_c = 0

        num\_c = 0

        v = len(self.words\_set)

        for i,data in enumerate(train):

            label = labels[i]

            if label == c:

                num\_c += len(data[0])

                num\_word\_c += data[0].count(word)

       # print(num\_c)

        return (num\_word\_c+self.lamda)/float(self.lamda\*v+num\_c)

    #def get\_multi\_wc(self):

    def classify(self,d):

        mul\_dc\_sum = 0 #分母上对每个c，求每个word的乘积

        for label in self.labels:

            mul\_dc\_tmp = 1

            for w in d:

                mul\_dc\_tmp \*=self.probs\_wc[(w,label)]

            mul\_dc\_sum +=mul\_dc\_tmp \* self.label\_prob\_pre[label]

        results = []

        for label in self.labels:

            mul\_dc = 1

            for w in d:

                mul\_dc \*= self.probs\_wc[(w,label)]

            results.append(self.label\_prob\_pre[label] \* mul\_dc)

        print(results)

        res\_c = np.argmax(results)

        return self.label\_map[res\_c]

nb = NaiveBayes(0.5,train,labels)

print(nb.classify("电器"))

print(nb.classify("香瓜"))