一.问题讨论

1.极大似然估计和贝叶斯估计的区别是什么？
讨论后的理解：极大似然估计是典型的频率学派观点，它的基本思想是：待估计参数是客观存在的，只是未知而已。
贝叶斯估计是典型的贝叶斯学派观点，它的基本思想是：待估计参数也是随机的，和一般随机变量没有本质区别，因此只能根据观测样本估计参数的分布。
在本次学习中，他们的主要区别是，当训练数据较少时，容易产生概率 0 的情况。贝叶斯估计为此加入了拉普拉斯平滑，可以说是进行了优化，使得每种属性至少出现一次。

2.贝叶斯网络是什么？
讨论后的理解：贝叶斯网络又称信度网络，是 Bayes 方法的扩展，是目前不确定知识表达和推理领域最有效的理论模型之一。一个贝叶斯网络是一个有向无环图，由代表变量结点及连接这些结点有向边构成。结点代表随机变量，结点间的有向边代表了结点间的互相关系(由父结点指向其子结点)，用条件概率进行表达关系强度，没有父结点的用先验概率进行信息表达。结点变量可以是任何问题的抽象，如：测试值，观测现象，意见征询等。适用于表达和分析不确定性和概率性的事件，应用于有条件地依赖多种控制因素的决策，可以从不完全、不精确或不确定的知识或信息中做出推理。

3.条件独立性假设是什么，它对于贝叶斯分类有什么意义？
讨论后的理解：不用条件独立性假设时，出现事件(X=x|Y=Ck)的概率很小（因为 X 的维度很高的情况下，可能在训练集中没有和 x 一模一样的向量），基本上是 0，所以很多甚至全部的 P(X=x|Y=Ck)都是 0，导致无法判断出现了(X=x)后，y 应该分为哪一类。而用条件独立性假设后，很多 P(X=x|Y=Ck)都不等于 0，这是就好判断 y 应该分为哪一类。
用条件独立性假设后，用极大似然估计来估计 P(Y=Ck)等相关参数时，还是可能出现 P(X=x|Y=Ck)=0 的情况，进而提出包含拉普拉斯平滑的贝叶斯估计，这才真正让所有 P(X=x|Y=Ck)都大于 0，从而好判断 y 的类别。
总的来说，使用条件独立性假设虽然有时会产生一定的误差，但大大减小了算法复杂度，使得运算比较简便，这也使贝叶斯分类成名的原因。

4.如何理解朴素贝叶斯算法中的特征条件独立？是指特征相互独立吗？
准确的说，**特征是条件独立的，而不是独立的**

二、读书计划

本周《统计学习方法》第四章
下周第五章

三、摘要或代码实现

```
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 Container

#  随机生成鸢尾花数据集

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']
    # 100 samples
    data = np.array(df.iloc[:100,:])
    # 4 features
    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,random_state=123)


#  定义贝叶斯各数据

class NaiveBayes:
    def __init__(self):
        self.model = None

    @staticmethod
    def mean(X):
        return sum(X) / float(len(X))

    def stdev(self, X):
        avg = self.mean(X)
        return np.sqrt(sum([np.power(x - avg, 2) for x in X]) / float(len(X)))

    def gaussian_probability(self, x, mean, stdev):
        exponent = np.exp(-(np.power(x - mean, 2) / (2 * np.power(stdev, 2))))

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

    # process train set
    def summarize(self, train_data):
        summaries = [(self.mean(i), self.stdev(i)) for i in zip(*train_data)]
        return summaries

    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):
            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)

            return probabilities


        # 返回标签

        def predict(self, X_test):
            label = sorted(
                self.calculate_probabilities(X_test).items(),
                key=lambda x: x[-1])[-1][0]
            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))

clf = NaiveBayes()
clf.fit(X_train,y_train)
print(clf.predict([4.4,    3.2,    1.3,    0.2]))
clf.score(X_test,y_test)
```