# Python实现naive bayes

**Training data：**

[1, 0, -1], [1, 1, -1], [1, 1, 1], [1, 0, 1],

[1, 0, -1], [2, 0, -1], [2, 1, -1], [2, 1, 1],  
[2, 2, 1], [2, 2, 1], [3, 2, 1], [3, 1, 1],  
[3, 1, 1], [3, 2, 1], [3, 2, -1]

其中前两个为特征值，后一个为类别

**Testing data：**

[2, 0]

**实验代码：**

**from** collections **import** Counter, defaultdict  
**import** numpy **as** np  
  
  
**class** NBayes:  
 **def** \_\_init\_\_(self, lambda\_):  
 self.lambda\_ = lambda\_ *# 贝叶斯估计方法参数lambda* self.p\_prior = {} *# 模型的先验概率, 注意这里的先验概率不是指预先人为设定的先验概率，而是需要估计的P(y=Ck)* self.p\_condition = {} *# 模型的条件概率* **def** fit(self, X\_data, y\_data):  
 N = y\_data.shape[0]  
 *# 后验期望估计P(y=Ck)的后验概率，设定先验概率为均匀分布* c\_y = Counter(y\_data)*#Counter类的目的是用来跟踪值出现的次数。它是一个无序的容器类型，以字典的键值对形式存储，其中元素作为key，其计数作为value。计数值可以是任意的Interger（包括0和负数）* K = len(c\_y)  
 **for** key, val **in** c\_y.items():  
 self.p\_prior[key] = (val + self.lambda\_) / (N + K \* self.lambda\_)  
 *# 后验期望估计P(Xd=a|y=Ck)的后验概率，同样先验概率为均匀分布* **for** d **in** range(X\_data.shape[1]): *# 对各个维度分别进行处理* Xd\_y = defaultdict(int)  
 vector = X\_data[:, d]  
 Sd = len(np.unique(vector))  
 **for** xd, y **in** zip(vector, y\_data): *# 这里Xd仅考虑出现在数据集D中的情况，故即使用极大似然估计叶没有概率为0的情况* Xd\_y[(xd, y)] += 1  
 **for** key, val **in** Xd\_y.items():  
 self.p\_condition[(d, key[0], key[1])] = (val + self.lambda\_) / (c\_y[key[1]] + Sd \* self.lambda\_)  
 **return  
  
 def** predict(self, X):  
 p\_post = defaultdict()  
 **for** y, py **in** self.p\_prior.items():  
 p\_joint = py *# 联合概率分布* **for** d, Xd **in** enumerate(X):  
 p\_joint \*= self.p\_condition[(d, Xd, y)] *# 条件独立性假设* p\_post[y] = p\_joint *# 分母P(X)相同，故直接存储联合概率分布即可* **return** max(p\_post, key=p\_post.get)  
  
  
**if** \_\_name\_\_ == **'\_\_main\_\_'**:  
 data = np.array([[1, 0, -1], [1, 1, -1], [1, 1, 1], [1, 0, 1],*#trainning data* [1, 0, -1], [2, 0, -1], [2, 1, -1], [2, 1, 1],  
 [2, 2, 1], [2, 2, 1], [3, 2, 1], [3, 1, 1],  
 [3, 1, 1], [3, 2, 1], [3, 2, -1]])  
 X\_data = data[:, :-1]  
 y\_data = data[:, -1]  
 clf = NBayes(lambda\_=1)  
 clf.fit(X\_data, y\_data)  
 print(clf.p\_prior, **'\n'**, clf.p\_condition)  
 print(clf.predict(np.array([2, 0])))*#testing data*