第5章

朴素贝叶斯

贝叶斯推理提供了推理的一种概率手段，它为直接运用概率的学习算法提供了理论框架。本章讨论的朴素贝叶斯分类器便是基于贝叶斯定理与特征条件独立假设的分类算法。

5.1 朴素贝叶斯模型

5.1.1 贝叶斯公式

首先，我们通过一个简单例子来回顾在概率课程中学习过的贝叶斯公式。

某课程的老师对历年学生课堂出勤与考试成绩进行统计，发现：

（1）每年有 10%学生期末考试不及格。

（2）期末考试不及格的学生中，有 80%经常旷课。

（3）期末考试及格的学生中，只有 5%经常旷课。

现在有一名经常旷课的学生，考试前他想了解他在期末考试不及格的概率是多少。

将考试不及格记为事件F_1，考试及格记为事件F_2，根据（1）中的信息可知以下概率：

$$P(F_1) = 0.1$$
$$P(F_2) = 1 - P(F_1) = 0.9$$

再将经常旷课记为事件E。根据（2）、（3）中的信息可知以下条件概率：

$$P(E|F_1) = 0.8$$
$$P(E|F_2) = 0.05$$

因为已知该学生经常旷课，他考试不及格的概率即为 $P(F_1|E)$。根据条件概率公式，有：

$$P(F_1|E) = \frac{P(EF_1)}{P(E)}$$

其中，$P(E)$ 可根据全概率公式计算：

$$P(E) = P(EF_1 \cup EF_2) = P(EF_1) + P(EF_2)$$

再次根据条件概率公式，最终可得：

$$\begin{aligned} P(F_1|E) &= \frac{P(E|F_1)P(F_1)}{P(E|F_1)P(F_1) + P(E|F_2)P(F_2)} \\ &= \frac{0.8 \times 0.1}{0.8 \times 0.1 + 0.05 \times 0.9} \\ &= 0.64 \end{aligned}$$

这个喜欢逃课学生的考试不及格率为 64%。以上计算 $P(F_1|E)$ 使用的便是贝叶斯公式（空间仅由 F_1 和 F_2 构成）。

一般情况下，令 $F_1, F_2, \ldots, F_N$ 表示一组互不相容事件，在 E（新的证据）已发生的情况下，F_k 发生的概率为：

$$P(F_k|E) = \frac{P(E|F_k)P(F_k)}{\sum_{i=1}^{N} P(E|F_i)P(F_i)}$$

以上公式称为贝叶斯公式，其中：

- $P(F_k)$ 称为先验概率（Prior Probability）。
- $P(E|F_k)$ 称为类似然（Class Likelihood）。
- $P(E) = \sum_{i=1}^{N} P(E|F_i)P(F_i)$ 称为证据（Evidence）。

- $P(F_k \mid E)$ 称为后验概率（Posterior Probability）。

5.1.2 贝叶斯分类器

回到机器学习中的分类问题，我们来介绍一下贝叶斯分类器。

假设分类问题有 N 种类别 $c_1, c_2, \ldots, c_N$，对于一个实例 x 进行分类时，贝叶斯分类器先根据贝叶斯公式分别计算在 x 条件下属于各个类别的条件概率，即后验概率：

$$P(c_k \mid x) = \frac{P(x \mid c_k)P(c_k)}{P(x)} = \frac{P(x \mid c_k)P(c_k)}{\sum_{i=1}^{N} P(x \mid c_i)P(c_i)}$$

然后根据后验证概率最大化准则（期望风险最小化），将 x 归为有最大后验概率的类别。

贝叶斯分类器的假设函数可定义为：

$$h_\theta(x) = \arg\max_{c_k \in y} P(c_k \mid x)$$

由于各后验概率分母相同，假设函数也可定义为：

$$h_\theta(x) = \arg\max_{c_k \in y} P(x \mid c_k)P(c_k)$$

上式中，$P(c_k)$ 的计算较为简单，根据训练样本的类标记进行估计即可。下面讨论 $P(x \mid c_k)$ 的计算。

5.1.3 朴素贝叶斯分类器

通常来说 $P(x \mid c_k)$ 很难求得，因为 x 是一个包含多个特征的向量，$P(x \mid c_k)$ 是所有特征的联合概率，很难根据训练样本直接估计求得。朴素贝叶斯做了一个很强的“特征条件独立性假设”把问题简化，即假设 x 的各个特征之间相互独立，一个特征出现的概率不受其他特征的影响。

设 x 有 n 个特征，$x^{(j)}$ 为其第 j 个特征。根据条件独立假设则有：

$$P(x \mid c_k) = \prod_{j=1}^{n} P(x^{(j)} \mid c_k)$$

其中，$P(x^{(j)} \mid c_k)$ 同样可根据样本进行估计，这样 $P(x \mid c_k)$ 便可以求解了。

朴素贝叶斯分类器的假设函数为：

$$h_\theta(x) = \arg\max_{c_k \in y} P(c_k) \prod_{j=1}^{n} P(x^{(j)} \mid c_k)$$

在实际编程中请注意，计算 $P(c_k)\prod_{j=1}^{n} P(x^{(j)} \mid c_k)$ 时可能出现下溢，这是因为它是很多很小的数的乘积。解决该问题的方法是对乘积项取对数，从而把乘积项变为和项，对数函数是单调递增的，不影响分类结果。因此，实际编程时可把假设函数定义为：

$$h_\theta(x) = \arg\max_{c_k \in y} \left(\ln P(c_k) + \sum_{j=1}^{n} \ln P(x^{(j)} \mid c_k) \right)$$

虽然现实世界中的问题绝大多数都是不符合“特征条件独立性假设”的，处理某些问题时这种假设看似完全错误，但使用朴素贝叶斯分类器进行分类的准确率却非常高。

5.2 模型参数估计

5.2.1 极大似然估计

朴素贝叶斯分类器的假设函数确定后，接下来需要根据训练样本对模型参数使用极大似然法进行估计。模型参数包括所有的先验概率 $P(c_k)$ 以及所有的类似然 $P(x^{(j)} \mid c_k)$。估计时首先要针对具体问题假定数据服从某种分布。下面我们以一个例子进行讲解。

假设某课程的老师想根据学生的上课出勤情况、完成作业情况以及小测验成绩这 3 个特征来判断一名学生期末考试能否通过。现有数据集 D（上届某班学生的数据），如表 5-1 所示。

表5-1 数据集D（上届某班学生的数据）

学号	不旷课	完成作业	小测验及格	期末考试及格（标记值）
1	1	0	1	1
2	1	1	0	1
3	1	1	1	1
4	0	1	1	1

（续表）

学号	不旷课	完成作业	小测验及格	期末考试及格（标记值）
5	0	0	1	1
…	…	…	…	…
26	0	0	1	0
27	1	1	0	0
28	1	0	1	0
29	0	1	0	0
30	0	0	0	0

关于数据集 $\boldsymbol{D}$ 的信息如下：

（1）数据集大小 $m=30$。

（2）每个实例 x_i 特征数 $n=3$。

（3）每个特征都是 0 或 1 的布尔值。

（4）目标值 y_i 只有通过和不通过两种情况，类别数 $N=2$。

我们发现各个特征以及类标记的取值都是布尔值，因此可假定 $P(c_k)$和 $P(x^{(j)}\mid c_k)$ 服从伯努利分布。假如数据集中的类标记是“优、良、中、差”4 个等级，那么我们可假定 $P(c_k)$ 服从多项式分布；假如数据集中各特征是第 1~3 次小测验考试分数，那么我们可假定 $P(x^{(j)}\mid c_k)$ 服从高斯分布（即正态分布）。

按照$P(x^{(j)}\mid c_k)$分布划分，常用的朴素贝叶斯分类器分为伯努利模型、多项式模型、高斯模型等。为了使推导过程容易理解，我们以简单的伯努利模型来演示使用极大似然法估计模型参数的过程（注意，伯努利模型可以进行多元分类，即$P(c_k)$服从多项式分布，但为了推导简单，这里仅做二元分类，即$P(c_k)$服从伯努利分布）。

定义模型参数，首先是先验概率 $P(c_k)$：

$$\phi_y = P(y=1)$$

$$1-\phi_y = P(y=0)$$

对于样本 (x_i, y_i)，则有：

$$P(y_i) = \phi_y^{y_i} \cdot (1-\phi_y)^{1-y_i}$$

然后，定义某一特征 $x^{(j)}$ 对于类别 c_k 的条件概率 $P(x^{(j)}\mid c_k)$：

$$\phi_{j|y=1} = P(x^{(j)} = 1 | y = 1)$$
$$1 - \phi_{j|y=1} = P(x^{(j)} = 0 | y = 1)$$
$$\phi_{j|y=0} = P(x^{(j)} = 1 | y = 0)$$
$$1 - \phi_{j|y=0} = P(x^{(j)} = 0 | y = 0)$$

对于样本 (x_i, y_i)，则有：

$$P(x_i^{(j)} | y_i) = \left(\phi_{j|y=1}^{x_i^{(j)}} \cdot (1 - \phi_{j|y=1})^{1 - x_i^{(j)}} \right)^{y_i} \cdot \left(\phi_{j|y=0}^{x_i^{(j)}} \cdot (1 - \phi_{j|y=0})^{1 - x_i^{(j)}} \right)^{1 - y_i}$$

令所有参数构成的向量为 ϕ：

$$\phi = (\phi_y, \phi_{1|y=1}, \phi_{1|y=0}, \phi_{2|y=1}, \phi_{2|y=0} \cdots \phi_{n|y=1}, \phi_{n|y=0})$$

定义似然函数为：

$$L(\phi) = \prod_{i=1}^{m} P(x_i, y_i)$$

再根据样本独立同分布以及特征条件独立性假设，可得：

$$\begin{aligned} L(\phi) &= \prod_{i=1}^{m} P(x_i, y_i) \\ &= \prod_{i=1}^{m} P(y_i) P(x_i | y_i) \\ &= \prod_{i=1}^{m} \left(P(y_i) \prod_{j=1}^{n} P(x_i^{(j)} | y_i) \right) \end{aligned}$$

为方便求导，对似然函数 $L(\phi)$ 取对数，得到对数似然函数：

$$\begin{aligned} l(\phi) &= \ln L(\phi) \\ &= \sum_{i=1}^{m} \left(\ln P(y_i) + \sum_{j=1}^{n} \ln P(x_i^{(j)} | y_i) \right) \\ &= \sum_{i}^{m} \Bigg(y_i \ln \phi_y + (1 - y_i) \ln(1 - \phi_y) \\ &\qquad + y_i \sum_{j=1}^{n} \left(x_i^{(j)} \ln \phi_{j|y=1} + (1 - x_i^{(i)}) \ln(1 - \phi_{j|y=1}) \right) \\ &\qquad + (1 - y_i) \sum_{j=1}^{n} \left(x_i^{(j)} \ln \phi_{j|y=0} + (1 - x_i^{(j)}) \ln(1 - \phi_{j|y=0}) \right) \Bigg) \end{aligned}$$

极大似然估计的目标是找到使对数似然函数最大的参数值，可通过方程 $\nabla_{\phi} l(\phi)=0$ 求解。

先求ϕ_y的估计值，求偏导数：

$$\begin{aligned}\frac{\partial}{\partial \phi_y} l(\phi) &= \sum_{i=1}^{m}\left(\frac{y_i}{\phi_y}-\frac{1-y_i}{1-\phi_y}\right)\\ &= \sum_{i=1}^{m}\frac{y_i-\phi_y}{\phi_y(1-\phi_y)}\\ &= \frac{\sum_{i=1}^{m} y_i - m\phi_y}{\phi_y(1-\phi_y)}\end{aligned}$$

令以上偏导数为 0，解得：

$$\hat{\phi}_y = \frac{\sum_{i=1} y_i}{m} = \frac{|D_{y=1}|}{|D|}$$

式中 $|D|$ 为训练集容量，即样本总数；$|D_{y=1}|$ 为 $y=1$ 的样本的个数。估计值 $\hat{\phi}_y$ 在含义上很容易让人理解，即$y=1$的样本在总样本中所占的比例。

再来求 $\phi_{j|y=1}$ 的估计值，求偏导数：

$$\begin{aligned}\frac{\partial}{\partial \phi_{j|y=1}} l(\phi) &= \sum_{i=1}^{m} y_i\left(\frac{x_i^{(j)}}{\phi_{j|y=1}}-\frac{1-x_i^{(j)}}{1-\phi_{j|y=1}}\right)\\ &= \sum_{i=1}^{m}\frac{y_i(x_i^{(j)}-\phi_{j|y=1})}{\phi_{j|y=1}(1-\phi_{j|y=1})}\\ &= \frac{\sum_{i=1}^{m} y_i x_i^{(j)} - \phi_{j|y=1}\sum_{i=1}^{m} y_i}{\phi_{j|y=1}(1-\phi_{j|y=1})}\end{aligned}$$

令以上偏导数为 0，解得：

$$\hat{\phi}_{j|y=1} = \frac{\sum_{i=1}^{m} y_i x_i^{(j)}}{\sum_{i=1}^{m} y_i} = \frac{|D_{y=1,x^{(j)}=1}|}{|D_{y=1}|}$$

式中 $|D_{y=1}|$ 为 $y=1$ 的样本的个数；$|D_{y=1,x^{(j)}=1}|$ 为 $y=1$ 且 $x^{(j)}=1$ 的样本的个数。$\hat{\phi}_{j|y=1}$ 可描述为，在所有 $y=1$ 的样本中，第 j 个特征为 $x^{(j)}=1$ 的样本所占的比例。

使用同样的方法求得 $\hat{\phi}_{j|y=0}$ 的估计值为：

$$\hat{\phi}_{j|y=0}=\frac{\sum_{i=1}^{m}(1-y_i)x_i^{(j)}}{\sum_{i=1}^{m}(1-y_i)}=\frac{|D_{y=0,x^{(j)}=1}|}{|D_{y=0}|}$$

式中 $|D_{y=0}|$ 为 $y=0$ 的样本的个数；$|D_{y=0,x^{(j)}=1}|$ 为 $y=0$ 且 $x^{(j)}=1$ 的样本的个数。$\hat{\phi}_{j|y=0}$ 可描述为，在所有 $y=0$ 的样本中，第 j 个特征为 $x^{(j)}=1$ 的样本所占的比例。

根据以上推导，模型中的所有参数都可被估计出来了。使用极大似然法估计模型参数的例子就展示完毕了。

5.2.2 贝叶斯估计

使用极大似然估计可能会出现所估计的概率为 0 的情况，这会影响后验概率的计算结果，导致分类产生偏差，采用贝叶斯估计可以解决这个问题。

假设数据集 D 中有 N 个类别 $c_1,c_2,\cdots,c_N$，先验概率的贝叶斯估计为：

$$P_\lambda(c_k)=\frac{|D_{y=c_k}|+\lambda}{|D|+N\lambda}$$

再假设第 j 个特征 $x^{(j)}$ 的可取值为$a_{j1},a_{j2},\cdots,a_{jN_j}$，$N_j$为第 j 个特征可取值的个数。条件概率的贝叶斯估计为：

$$P_\lambda(x^{(j)}=a_{jl}|c_k)=\frac{|D_{y=c_k,x^{(j)}=a_{jl}}|+\lambda}{|D_{y=c_k}|+N_j\lambda}$$

以上两式中，$|D|$ 为训练集容量；$|D_{y=c_k}|$ 为 $y=c_k$ 的样本的个数；$|D_{y=c_k,x^{(j)}=a_{jl}}|$ 为 $y=1$ 且 $x^{(j)}=a_{jl}$ 的样本的个数。其中 $\lambda\geqslant 0$，当取 $\lambda=0$ 时，就是极大似然估计。在实际应用中常取 $\lambda=1$，称为拉普拉斯平滑（Laplace Smoothing）。

根据上面的公式，可以推导出：

$$P_\lambda(c_k)>0$$

$$\sum_{k=1}^{N}P_\lambda(c_k)=1$$

$$P_\lambda(x^{(j)}=a_{jl}|c_k)>0$$

$$\sum_{l=1}^{S_j}P_\lambda(x^{(j)}=a_{jl}|c_k)=1$$

我们发现，使用贝叶斯估计得到的概率都是大于 0 的，并且 λ 任意取大于等于 0 的常数都不会破坏概率之和为 1 的概率律。

5.3 算法实现

下面我们来实现一个伯努利模型的朴素贝叶斯分类器，代码如下：

```
1.  import numpy as np
2.
3.  class BernoulliNavieBayes:
4.
5.      def __init__(self, alpha=1.):
6.          # 平滑系数， 默认为1(拉普拉斯平滑)
7.          self.alpha = alpha
8.
9.      def _class_prior_proba_log(self, y, classes):
10.         '''计算所有类别的先验概率 P(y=c_k)'''
11.
12.         # 统计各类别的样本数量
13.         c_count = np.count_nonzero(y == classes[:, None], axis=1)
14.         # 计算各类别的先验概率(平滑修正)
15.         p = (c_count + self.alpha) / (len(y) + len(classes) *
self.alpha)
16.
17.         return np.log(p)
18.
19.     def _conditional_proba_log(self, X, y, classes):
20.         '''计算所有条件概率 P(x^(j)|y=c_k)的对数'''
21.
22.         _, n = X.shape
23.         K = len(classes)
24.
25.         # P_log: 2 个条件概率的对数的矩阵
26.         # 矩阵 P_log[0]存储所有 log(P(x^(j)=0|y=c_k))
27.         # 矩阵 P_log[1]存储所有 log(P(x^(j)=1|y=c_k))
28.         P_log = np.empty((2, K, n))
29.
```

```
        # 迭代每一个类别 c_k
        for k, c in enumerate(classes):
            # 获取类别为 c_k 的实例
            X_c = X[y == c]
            # 统计各特征值为 1 的实例的数量
            count1 = np.count_nonzero(X_c, axis=0)
            # 计算条件概率 P(x^(j)=1|y=c_k)(平滑修正)
            p1 = (count1 + self.alpha) / (len(X_c) + 2 * self.alpha)
            # 将 log(P(x^(j)=0|y=c_k))和 log(P(x^(j)=1|y=c_k))存入矩阵
            P_log[0, k] = np.log(1 - p1)
            P_log[1, k] = np.log(p1)

        return P_log

    def train(self, X_train, y_train):
        '''训练模型'''

        # 获取所有类别
        self.classes = np.unique(y_train)
        # 计算并保存所有先验概率的对数
        self.pp_log = self._class_prior_proba_log(y_train,
self.classes)
        # 计算并保存所有条件概率的对数
        self.cp_log = self._conditional_proba_log(X_train, y_train,
self.classes)

    def _predict_one(self, x):
        '''根据贝叶斯公式对单个实例进行预测'''

        K = len(self.classes)
        p_log = np.empty(K)

        # 分别获取各特征值为 1 和 0 的索引
        idx1 = x == 1
        idx0 = ~idx1

        # 迭代每一个类别 c_k
        for k in range(K):
```

```
66.             # 计算后验概率 P(c_k|x) 分子部分的对数
67.             p_log[k]=self.pp_log[k]+np.sum(self.cp_log[0, k][idx0])\
68.                                    + np.sum(self.cp_log[1, k][idx1])
69.
70.         # 返回具有最大后验概率的类别
71.         return np.argmax(p_log)
72.
73.     def predict(self, X):
74.         '''预测'''
75.
76.         # 对 X 中每个实例调用 _predict_one 进行预测，收集结果并返回
77.         return np.apply_along_axis(self._predict_one, axis=1, arr=X)
```

上述代码简要说明如下（详细内容参看代码注释）。

- __init__()方法：构造器，保存用户传入的超参数。
- _class_prior_proba_log()方法：计算所有类别的先验概率 $P(c_k)$ 的对数。
- _conditional_proba_log()方法：计算所有条件概率 $P(x^{(j)}=1|c_k)$ 和 $P(x^{(j)}=0|c_k)$的对数。
- train()方法：训练模型。该方法由 3 部分构成：
  - 获取并保存训练集中的所有类别。
  - 调用 _class_prior_proba_log()方法，计算并保存先验概率的对数，作为模型参数。
  - 调用 _conditional_proba_log()方法，计算并保存条件概率的对数，作为模型参数。
- _predict_one()方法：根据贝叶斯公式，使用已训练好的模型参数对单个实例进行预测。
- predict()方法：预测。对 $\boldsymbol{X}$ 中每个实例调用 _predict_one()方法进行预测，收集结果并返回。

5.4 项目实战

最后，我们来做一个贝叶斯分类器的实战项目：使用朴素贝叶斯分类器（伯努利模型）识别垃圾邮件，如表 5-2 所示。

表 5-2　垃圾邮件数据集（https://archive.ics.uci.edu/ml/datasets/spambase）

列号	列名	特征/类别	可取值
1	word_freq_make	特征	实数
2	word_freq_address	特征	实数
3	word_freq_all	特征	实数
4	word_freq_3d	特征	实数
5	word_freq_our	特征	实数
6	word_freq_over	特征	实数
…	…	…	…
46	word_freq_edu	特征	实数
47	word_freq_table	特征	实数
48	word_freq_conference	特征	实数
49	char_freq_;	特征	实数
…	…	…	…
58	Spam	类别	0, 1

数据集总共有 4601 条数据，其中的每一行包含 48 个不同单词出现的频数，6 个标点符号出现的频数，3 个关于连续大写字母字符串长度的特征，以及一个标识是否为垃圾邮件的类标记。处理文本分类问题时，可使用的朴素贝叶斯分类器通常有两种：伯努利模型或多项式模型。词频特征便是多项式模型所使用的，而我们之前实现的并且打算在该项目中使用的是伯努利模型，伯努利模型不关心词频，只关心单词是否出现。在这个项目中，我们只使用关于单词的前 48 个特征，并将它们转换成代表词是否出现的布尔值特征（0 或 1）。

读者可使用任意方式将数据集文件 spambase.data 下载到本地。这个文件所在的 URL 为：https://archive.ics.uci.edu/ml/machine-learning-databases/spambase/spambase.data。

5.4.1　准备数据

首先，调用 Numpy 的 genfromtxt 函数加载数据集：

```
1.  >>> import numpy as np
2.  >>> data = np.loadtxt('spambase.data', delimiter=',')
3.  >>> data
4.  array([[0.000e+00, 6.400e-01, 6.400e-01, ..., 6.100e+01, 2.780e+02,
5.          1.000e+00],
6.         [2.100e-01, 2.800e-01, 5.000e-01, ..., 1.010e+02, 1.028e+03,
```

```
7.          1.000e+00],
8.        [6.000e-02, 0.000e+00, 7.100e-01, ..., 4.850e+02, 2.259e+03,
9.         1.000e+00],
10.       ...,
11.       [3.000e-01, 0.000e+00, 3.000e-01, ..., 6.000e+00, 1.180e+02,
12.        0.000e+00],
13.       [9.600e-01, 0.000e+00, 0.000e+00, ..., 5.000e+00, 7.800e+01,
14.        0.000e+00],
15.       [0.000e+00, 0.000e+00, 6.500e-01, ..., 5.000e+00, 4.000e+01,
16.        0.000e+00]])
```

取 data 中前 48 列，并将词频数转换为 0 或 1 的布尔值，作为 X：

```
1.  >>> X = data[:, :48]
2.  >>> X
3.  array([[0.  , 0.64, 0.64, ..., 0.  , 0.  , 0.  ],
4.         [0.21, 0.28, 0.5 , ..., 0.  , 0.  , 0.  ],
5.         [0.06, 0.  , 0.71, ..., 0.06, 0.  , 0.  ],
6.         ...,
7.         [0.3 , 0.  , 0.3 , ..., 1.2 , 0.  , 0.  ],
8.         [0.96, 0.  , 0.  , ..., 0.32, 0.  , 0.  ],
9.         [0.  , 0.  , 0.65, ..., 0.65, 0.  , 0.  ]])
10. >>> X = np.where(X > 0, 1, 0) # 转换为布尔值
11. >>> X
12. array([[0, 1, 1, ..., 0, 0, 0],
13.        [1, 1, 1, ..., 0, 0, 0],
14.        [1, 0, 1, ..., 1, 0, 0],
15.        ...,
16.        [1, 0, 1, ..., 1, 0, 0],
17.        [1, 0, 0, ..., 1, 0, 0],
18.        [0, 0, 1, ..., 1, 0, 0]])
```

取 data 中的最后一列，并转换为整数类型（int），作为 y：

```
1.  >>> y = data[:, -1].astype(np.int)
2.  >>> y
3.  array([1, 1, 1, ..., 0, 0, 0])
```

至此，数据准备完毕。

5.4.2 模型训练与测试

BernoulliNavieBayes 只有一个超参数 alpha，通常我们使用默认值 1 即可（拉普拉斯平滑）。

创建模型：

```
1.  >>> from navie_bayes import BernoulliNavieBayes
2.  >>> clf = BernoulliNavieBayes()
```

然后，调用 sklearn 中的 train_test_split 函数将数据集切分为训练集和测试集（比例为 7:3）：

```
1.  >>> from sklearn.model_selection import train_test_split
2.  >>> X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3)
```

接下来，训练模型：

```
1.  >>> clf.train(X_train, y_train)
```

朴素贝叶斯分类器的训练是非常快速的，瞬间便完成了。

使用已训练好的模型对测试集进行预测，并调用 sklearn 中的 accuracy_score 函数计算预测的准确率：

```
1.  >>> from sklearn.metrics import accuracy_score
2.  >>> y_pred = clf.predict(X_test)
3.  >>> y_pred
4.  array([0, 0, 0, ..., 1, 0, 0])
5.  >>> accuracy = accuracy_score(y_test, y_pred)
6.  >>> accuracy
7.  0.8826937002172339
```

单次测试一下，预测的准确率为 88.27%。因为 BernoulliNavieBayes 只有一个对性能几乎没什么影响的超参数 alpha，所以我们无法通过调整超参数进行优化。

接下来以不同训练集/测试集比例训练多个模型，大量测试后比较各模型的性能：

```
1.  >>> def test(X, y, test_size, N):
2.  ...     acc = np.empty(N)
3.  ...     for i in range(N):
4.  ...         X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=test_size)
```

```
    clf = BernoulliNavieBayes()
    clf.train(X_train, y_train)
    y_pred = clf.predict(X_test)
    acc[i] = accuracy_score(y_test, y_pred)
return np.mean(acc)

>>> sizes = np.arange(0.3, 1, 0.1)
>>> sizes
array([0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9])
>>> [test(X, y, test_size, 100) for test_size in sizes]
[0.877950760318609 7, 0.875437262357414 5, 0.876106040851803 5,
0.876312930097790 5, 0.876473144986029 1, 0.874944308611790 3,
0.873931417531997 3]
```

以上代码依次使用了训练集比例 70%, 60%, …, 10%来训练模型并测试模型性能（每种比例测试 100 次取平均值），我们发现随着训练集的减小（测试集增大），模型性能仅有极微小的改变。这说明对于该分类问题，虽然朴素贝叶斯分类器预测准确率不是很高，但只需要使用很少的训练数据。

至此，我们这个识别垃圾邮件的项目就完成了。读者可以自行查阅相关资料实现多项式模型的朴素贝叶斯分类器，再对该项目进行实验。