朴素贝叶斯

贝叶斯(约1701-1761) Thomas Bayes，英国数学家。约1701年出生于伦敦，做过神甫。1742年成为英国皇家学会会员。1761年4月7日逝世。贝叶斯在数学方面主要研究概率论。他首先将归纳推理法用于概率论基础理论，并创立了贝叶斯统计理论，对于统计决策函数、统计推断、统计的估算等做出了贡献。他死后，理查德·普莱斯(Richard Price)于1763年将他的著作《机会问题的解法》(An essay towards solving a problem in the doctrine of chances)寄给了英国皇家学会，对于现代概率论和数理统计产生了重要的影响。

1. 贝叶斯公式

2. 贝叶斯算法原理

3. 高斯朴素贝叶斯

4. 垃圾邮件识别

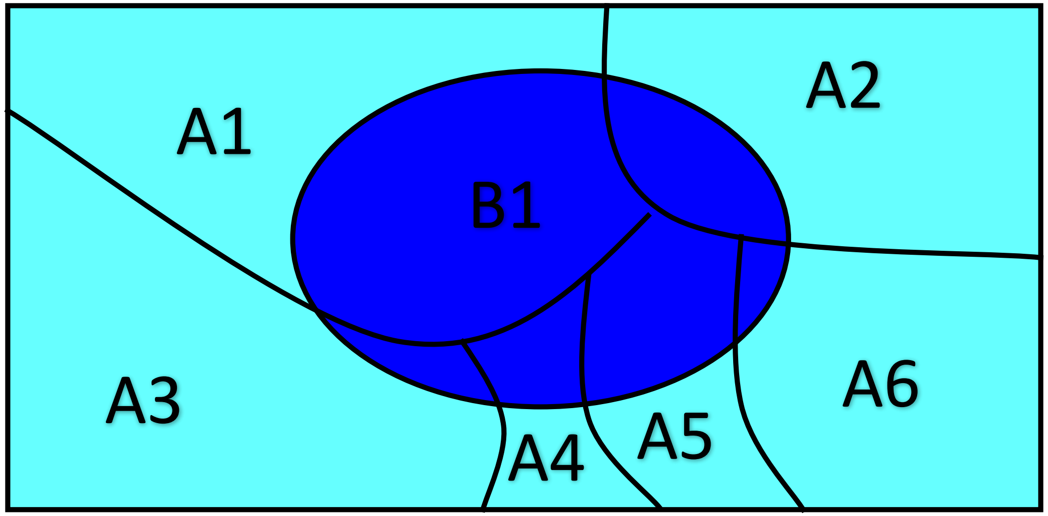
（贝叶斯公式） 设 $A_1, A_2, \cdots, A_n$ 为一个完备事件组，

$P(A_i) > 0$， $i = 1, \cdots, n$，对任一事件 B，若 $P(B) > 0$ ，有

$$P(A_k \mid B) = \frac{P(A_k B)}{P(B)}$$

$$= \frac{P(A_k)P(B \mid A_k)}{\sum_{i=1}^{n} P(A_i)P(B \mid A_i)}, \qquad (k = 1, 2, \cdots, n)$$

$$P(B)=\sum_{i=1}^{n} P(A_i)\,P(B\mid A_i)$$



$$P(A_k \mid B) = \frac{P(A_k B)}{P(B)} = \frac{P(A_k)P(B \mid A_k)}{\sum_{i=1}^{n} P(A_i)P(B \mid A_i)},$$

$\mathbf{P}(\boldsymbol{A_k})$先验概率

$\mathbf{P}(\boldsymbol{A_k}|\mathbf{B})\boldsymbol{A_k}$后验概率

$\mathbf{P}(\boldsymbol{B}|\boldsymbol{A_k})$似然函数

- 一所学校里面有 60% 的男生，40% 的女生。男生总是穿长裤，女生则一半穿长裤一半穿裙子。你在校园里面随机遇到一个穿长裤的人，他/她是男生的概率是多少？

2.贝叶斯算法原理

特征1-高	特征2-富	特征3-帅	见面
高	富	帅	见
高	富	锉	见
高	穷	锉	不见
矮	富	锉	不见
矮	穷	帅	见
矮	穷	锉	不见

矮	富	帅

见？不见？

朴素贝叶斯算法简介

- 在分类（classification）问题中，常常需要把一个事物分到某个类别。一个事物具有很多属性，把它的众多属性看做一个向量，即$x=(x_1,x_2,x_3,\ldots,x_n)$，用x这个向量来代表这个事物。
- 有类别集合 $y=(y_1,y_2,y_3,\ldots.y_n)$
- 分别计算 $p(y_1|x)$ $p(y_2|x)$ $p(y_3|x)\ldots.$ $p(y_n|x)$，如果$p(y_k|x)$

  $=\max\{p(y_1|x)\ p(y_2|x)\ p(y_3|x)\ldots.\ p(y_n|x)\}$, x就属于$y_k$类。

- 如何计算 $p(y_k|x)$

方法：运用贝叶斯公式 $p(y_k|x)=p(x|y_K)*p(y_k)/p(x)$

在之前已介绍$x=(x_1, x_2, x_3, \cdots, x_n)$，根据朴素贝叶斯假设

$x_1, x_2, x_3, \cdots, x_n$是相互独立的

则有 $p(x \mid y_k) = p(x_1, x_2, x_3, \cdots, x_n|y_k) = p(x_1 \mid y_k) *p(x_2 \mid y_k)$

$\ldots\ldots *p(x_n \mid y_k)$ (1)

特征1-高	特征2-富	特征3-帅	见面
高	富	帅	见
高	富	锉	见
高	穷	锉	不见
矮	富	锉	不见
矮	穷	帅	见
矮	穷	锉	不见

矮	富	帅

max(P(见|[矮， 富， 帅])， P(不见|[矮， 富， 帅]))

P(见|[矮， 富， 帅])=P（[矮， 富， 帅] | 见）P(见)/P(矮， 富， 帅)
=P（矮|见）*P（富|见）*P（帅|见）*P（见）/(P（矮）*P（富）*P（帅）)
=(1/3*2/3*2/3*1/2)/(1/2*1/2*1/3)
=

P(不见|[矮， 富， 帅])=P（[矮， 富， 帅] | 不见）P(不见)/P(矮， 富， 帅)
=P（矮|不见）*P（富|不见）*P（帅|不见）*P（不见）/(P（矮）*P（富）*P（帅）)
=(2/3*1/3*1/3*1/2)/(1/2*1/2*1/3)
=

特征1-高	特征2-富	特征3-帅	见面
高	富	帅	见
高	富	锉	见
高	穷	锉	不见
矮	富	锉	不见
高	穷	帅	见
矮	穷	锉	不见

矮	富	帅

max(P(见|[矮， 富， 帅])， P(不见|[矮， 富， 帅]))

P(见|[矮， 富， 帅])=P（[矮， 富， 帅] | 见）P(见)/P(矮， 富， 帅)
=P（矮|见）*P（富|见）*P（帅|见）*P（见）/(P（矮）*P（富）*P（帅）)
=(0/3*2/3*2/3*3/6)/P0
=0

P(不见|[矮， 富， 帅])=P（[矮， 富， 帅] | 不见）P(不见)/P(矮， 富， 帅)
=P（矮|不见）*P（富|不见）*P（帅|不见）*P（不见）/(P（矮）*P（富）*P（帅）)
=(2/3*1/3*1/3*1/2)/P0
=0.385

特征1-高	特征2-富	特征3-帅	见面
高	富	帅	见
高	富	锉	见
高	穷	锉	不见
矮	富	锉	不见
矮	穷	帅	见
矮	穷	锉	不见

矮	富	帅

$$P_{new} = \frac{N_x + 1}{N_M + k}$$

max(P(见|[矮，富，帅])，P(不见|[矮，富，帅]))

P(见|[矮，富，帅])=P（[矮，富，帅] | 见）P(见)/P0
=P（矮|见）*P（富|见）*P（帅|见）*P（见）
=(1/5*3/5*3/5*4/8)
=0.04
P(不见|[矮，富，帅])=P（[矮，富，帅] | 不见）P(不见)/P0
=P（矮|不见）*P（富|不见）*P（帅|不见）*P（不见）
=(3/5*2/5*2/5*4/8)
=0.05

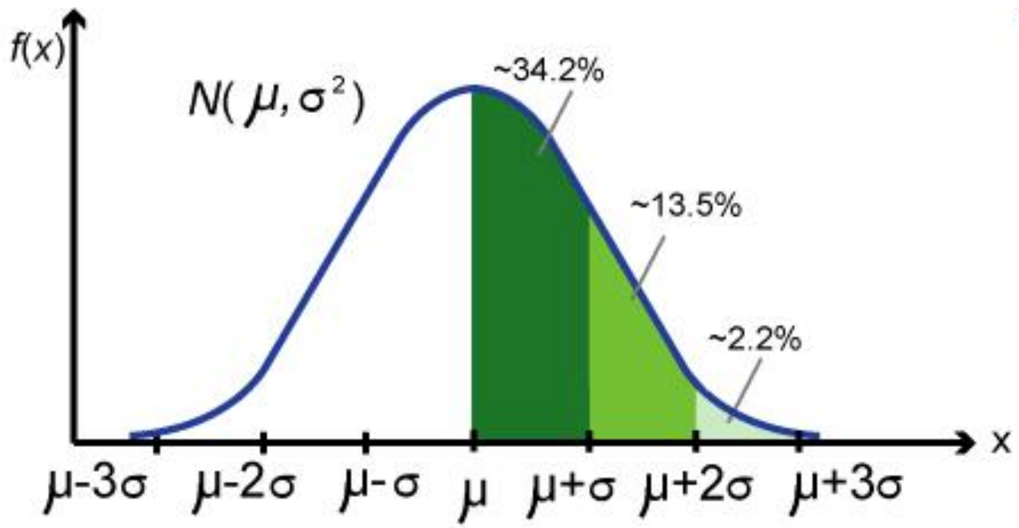
3.高斯朴素贝叶斯

特征1-高	特征2-富	特征3-帅	见面
180	50K	90	见
190	40K	30	见
175	5K	80	不见
160	20K	40	不见
170	6K	70	见
165	7K	59	不见

特征1-高	特征2-富	特征3-帅	见面
高	富	帅	见
高	富	锉	见
高	穷	帅	不见
矮	富	锉	不见
高	穷	帅	见
矮	穷	锉	不见

分箱法:将连续数据分段后成为离散数据的一种预处理办法。

>=170:高　　<170:矮
>=20K:富　　<20K:穷
>=60:帅　　<60:锉



$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

均值为μ:

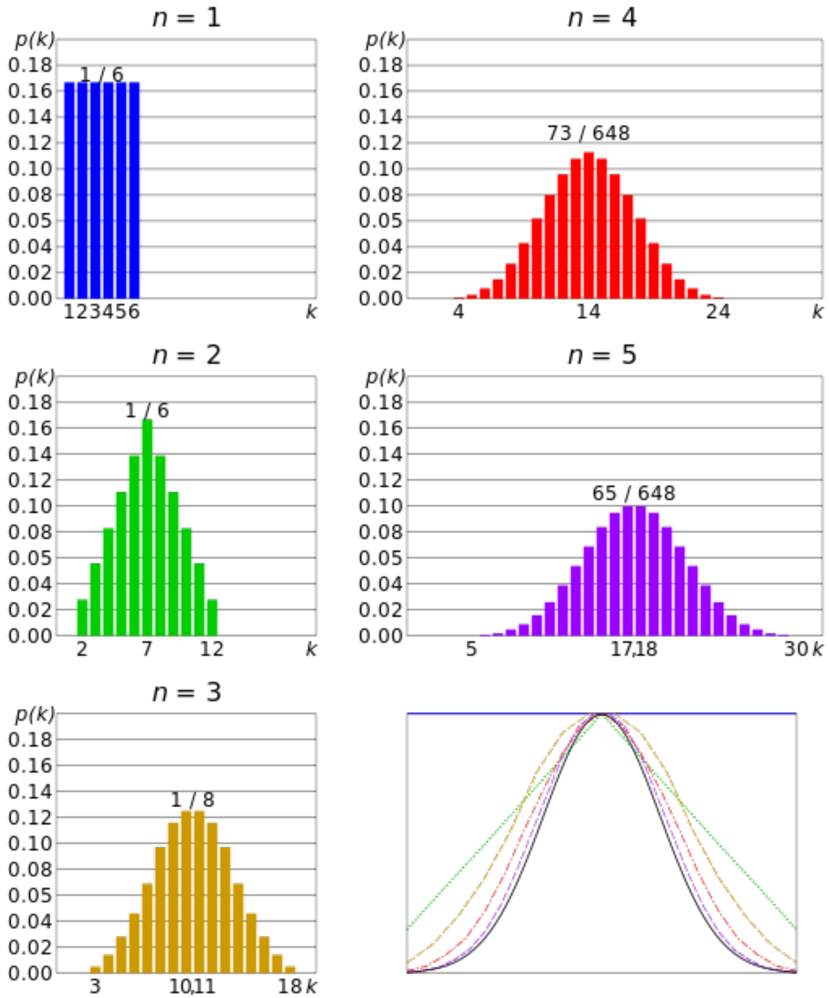
$$\overline{x} = \frac{x_1 + x_2 + \cdots + x_n}{n} = \frac{\sum_{i=1}^{n} x_i}{n}$$

方差为σ:

$$s^2 = \frac{\sum_{i=1}^{n}(x-u)^2}{n-1}$$

- 中心极限定理

- 从总体中抽取样本容量为n的简单随机样本,当样本容量很大时,样本均值的抽样分布与正态概率分布近似。



特征1-高	特征2-富	特征3-帅	见面
180	50K	90	见
190	40K	30	见
175	5K	80	不见
160	20K	40	不见
170	6K	70	见
165	7K	59	不见

170	30K	70

以特征1为例：
见面的均值为 180，标准差为：8.24，所以P（170|见）概率为

$$P(Yi|Xi) = \frac{1}{\sqrt{2\pi}\delta} e^{-\frac{(x-\mu)^2}{2\delta^2}} = \frac{1}{\sqrt{2\pi}\delta} e^{-\frac{(x-\mu)^2}{2\delta^2}} = 0.028$$

同样我们可以计算出其余部分然后用贝叶斯定理计算即可。

高斯朴素贝叶斯

```
from sklearn.naive_bayes import GaussianNB
from sklearn import datasets
from sklearn.cross_validation import train_test_split
iris = datasets.load_iris()
X_train,X_test, y_train, y_test=
        train_test_split(iris.data,iris.target,test_size=0.4, random_state=0)
# iris.feature_names
clf = GaussianNB()
clf.fit(X_train, y_train)
print(clf.score(X_test,y_test))
```

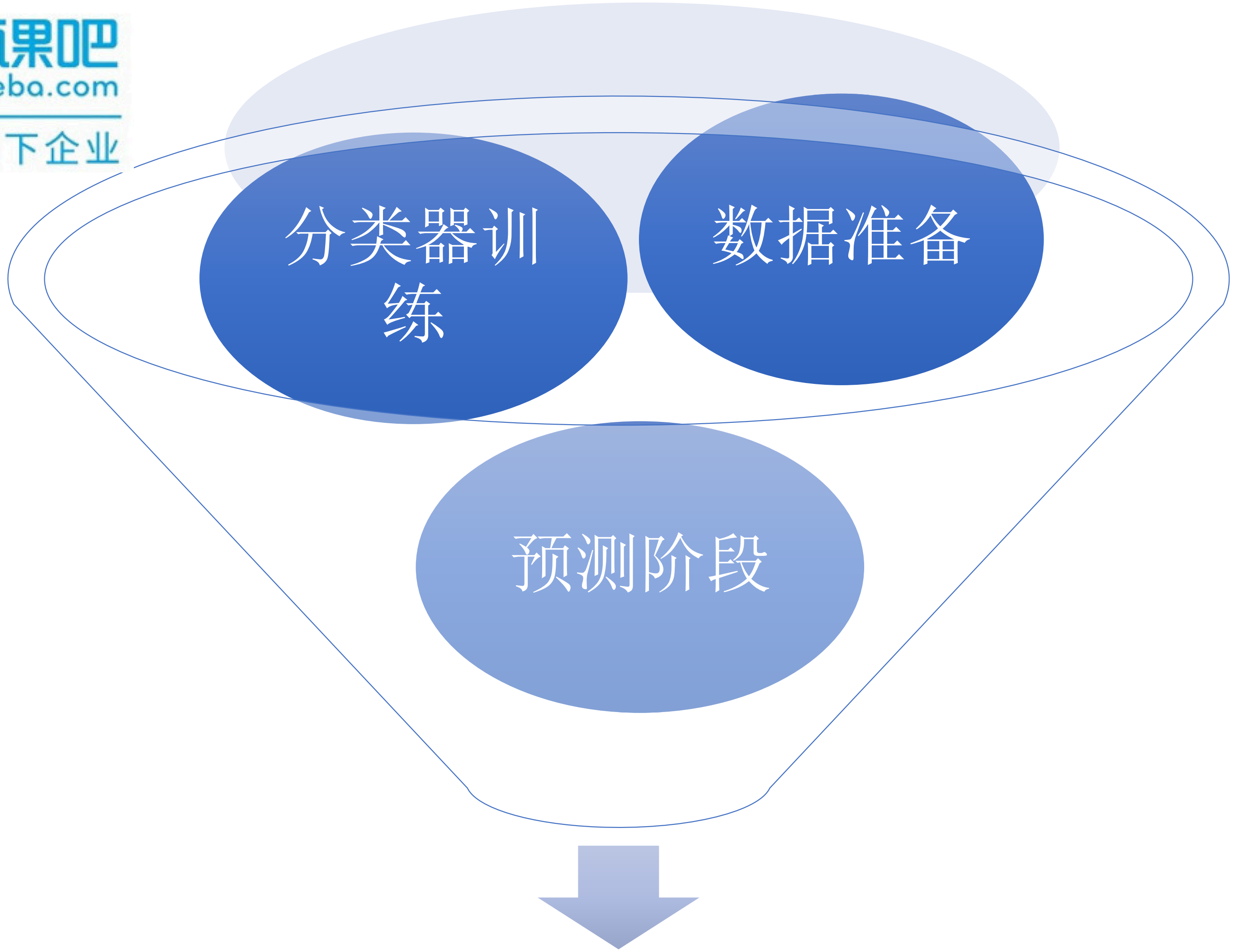
```
0.933333333333
```

```
from sklearn import neighbors
clf = neighbors.KNeighborsClassifier()
clf.fit(X_train, y_train)
print(clf.score(X_test,y_test))
```

```
/Library/Frameworks/Python.framework/Versions/3.4/lib/python3.4/site-packages/ipykernel/__ma
Warning: A column-vector y was passed when a 1d array was expected. Please change the shape
r example using ravel().
  app.launch_new_instance()
```

```
0.859283387622
```

4.垃圾邮件识别


分类器训练
数据准备
预测阶段

- 如何将贝叶斯分类器应用到垃圾邮件识别中来。

在垃圾邮件识别过程中，假设我们有一个文档d{t1,t2,t3...}和一个固定的标签集合C={c1,c2}

比如：

需要|为|企业|开具|发票，请|联系|我。（垃圾邮件）

附件|是|我的|报表，如果|可行，请|批示。（非垃圾邮件）

计算条件概率

$P(X1 = t1, X2 = t2, \ldots, Xn = tn \mid yi)$ 和 $P(C = yi)$

对比后验概率

$P(C = yi \mid X1 = t1, X2 = t2, \ldots, Xn = tn), (i = 1, 2)$

哪个标签大就将其归为某个类别。

样本	Email内容	分词	标签
NO1	有偿开发票	['有偿', '开发票']	垃圾邮件
NO2	夜店酒吧	['夜店', '酒吧']	垃圾邮件
NO3	收到请回复	['收到', '请', '回复']	非垃圾邮件
NO4	请填写基本信息	['请', '填写', '基本', '信息']	非垃圾邮件
NO5	下午4点开会	['下午', '4', '点', '开会']	非垃圾邮件

样本出现14个集合为:
['下午', '酒吧', '开会', '开发票', '有偿', '回复', '4', '信息', '请', '基本', '填写', '收到', '夜店', '点']

样本	Email内容	分词	向量	标签
NO1	有偿开发票	['有偿', '开发票']	[0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]	1
NO2	夜店酒吧	['夜店', '酒吧']	[0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0]	1
NO3	收到请回复	['收到', '请', '回复']	[0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0]	0
NO4	请填写基本信息	['请', '填写', '基本', '信息']	[0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0]	0
NO5	下午4点开会	['下午', '4', '点', '开会']	[1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1]	0

将文本转化为了计算机可以处理的0，1变量，这种方法叫做词袋模型

标签	概率
1	3/7
0	4/7

计算标签概率

注意：已经做了拉普拉斯平滑

计算条件概率

(0+1)/(2+2)
2:垃圾邮件条数
2:标签的类别

	垃圾邮件中出现	P(x|1)	非垃圾邮件中出现	P（x|0)
下午	0	0.25	1	0.4
酒吧	1	0.5	0	0.2
开会	0	0.25	1	0.4
开发票	1	0.5	0	0.2
有偿	1	0.5	0	0.2
回复	0	0.25	1	0.4
4	0	0.25	1	0.4
信息	0	0.25	1	0.4
请	0	0.25	1	0.4
基本	0	0.25	1	0.4
填写	0	0.25	1	0.4
收到	0	0.25	1	0.4
夜店	1	0.5	0	0.2
点	0	0.25	1	0.4

- 请填写开发票信息 -> 请|填写|开发票|信息

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

- P(X|1)=log（1-p(x1|1)）*（1-p(x2|1)）*... p(x4|1) ...)

- P(X|0)= log（1-p(x1|0)）*（1-p(x2|0)）*... p(x4|0) ...)

- 对比P(1|X)和P(0|X)

伯努利分布

```
X_train=pd.read_table('xtrain.txt',sep='\t').as_matrix()
X_test=pd.read_table('xtest.txt',sep='\t').as_matrix()
y_train=pd.read_table('ytrain.txt',sep='\t').as_matrix()
y_test=pd.read_table('ytest.txt',sep='\t').as_matrix()

from sklearn.naive_bayes import BernoulliNB
clf = BernoulliNB()
clf.fit(X_train, y_train)
print(clf.score(X_test,y_test))
```

```
0.889250814332
```

KNN

```
from sklearn import neighbors
clf = neighbors.KNeighborsClassifier()
clf.fit(X_train, y_train)
print(clf.score(X_test,y_test))
```

```
/Library/Frameworks/Python.framework/Versions/3
Warning: A column-vector y was passed when a 1c
r example using ravel().
  app.launch_new_instance()
```

```
0.859283387622
```

- 多项式分布

在多项式模型中， 设某文档d=(t1,t2,…,tk)，tk是该文档中出现过的单词，允许重复，则

先验概率P(c)= 类c下单词总数/整个训练样本的单词总数
类条件概率P(tk|c)=(类c下单词tk在各个文档中出现过的次数之和+1)/(类c下单词总数+|V|)

V是训练样本的单词表（即抽取单词，单词出现多次，只算一个），|V|则表示训练样本包含多少种单词。 P(tk|c)可以看作是单词tk在证明d属于类c上提供了多大的证据，而P(c)则可以认为是类别c在整体上占多大比例(有多大可能性)。

句子			类别
开发票	有偿	开发票	垃圾邮件
开发票	开发票	提成	垃圾邮件
需要	开发票		垃圾邮件
开发票	提供	纳税	非垃圾邮件

有偿	开发票	纳税	提供	需要	提成
1	2	0	0	0	0
0	2	0	0	0	1
0	1	0	0	1	0
0	1	1	1	0	0

P(1)=8/11

P(0)=3/11

$P(w0|1) = (1+1) / (8+6)$

$P(w1|1) = (5+1) / (8+6)$

$P(w2|1) = (0+1) / (8+6)$

$P(w3|1) = (0+1) / (8+6)$

$P(w4|1) = (1+1) / (8+6)$

$P(w5|1) = (1+1) / (8+6)$

$P(w0|0) = (0+1) / (3+6)$

$P(w1|0) = (1+1) / (3+6)$

$P(w2|0) = (1+1) / (3+6)$

$P(w2|0) = (1+1) / (3+6)$

$P(w4|0) = (0+1) / (3+6)$

$P(w5|0) = (0+1) / (3+6)$

多项式分布

```
from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import CountVectorizer

categories = ['comp.graphics','comp.os.ms-windows.misc'];
news = fetch_20newsgroups(subset='all',categories = categories)
X_train, X_test, y_train, y_test = train_test_split(news.data, news.target, test_size=0.3, random_state=33)
count_vec = CountVectorizer()
X_count_train = count_vec.fit_transform(X_train)
X_count_test = count_vec.transform(X_test)

from sklearn.naive_bayes import MultinomialNB
clf = MultinomialNB()
clf.fit(X_count_train, y_train)
print ( mnb_count.score(X_count_test, y_test))
```

```
0.680272108844
```

```
from sklearn import neighbors
clf = neighbors.KNeighborsClassifier()
clf.fit(X_count_train, y_train)
print(clf.score(X_count_test,y_test))
```

```
0.731292517007
```

数据预处理

条件概率计算

高斯朴素贝叶斯：（Gaussian Naive Bayes）

伯努利朴素贝叶斯（ Bernoulli Naive Bayes）

多项式朴素贝叶斯（Multinomial Naive Bayes）

后验概率计算