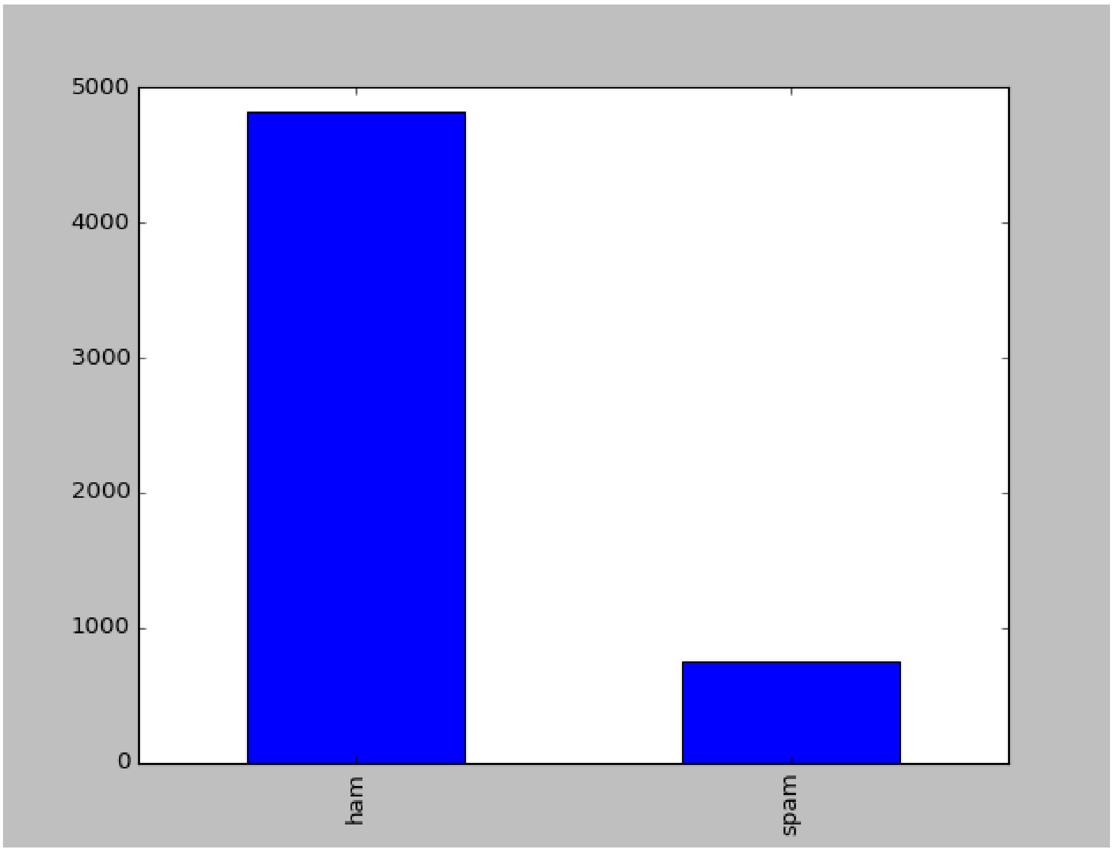
```python
# 案例：垃圾邮件预测

# 手动进行朴素贝叶斯分类

# 加载样本数据
df = pd.read_csv('sms.csv')

df
```

```
    label                                                msg
0     ham  Go until jurong point, crazy.. Available only ...
1     ham                      Ok lar... Joking wif u oni...
2    spam  Free entry in 2 a wkly comp to win FA Cup fina...
3     ham  U dun say so early hor... U c already then say...
...   ...                                                ...
5570  ham  The guy did some bitching but I acted like i'd...
5571  ham                         Rofl. Its true to its name

[5572 rows x 2 columns]
```

```python
# 看看两个类别的分布
df.label.value_counts()
```

```
ham     4825
spam     747
```

```python
%matplotlib

df.label.value_counts().plot(kind='bar')
```




```
# 空模型：对任何输入邮件都预测占多数的类别 ham
# 正确率：86.6%
df['label'].value_counts() / df.shape[0]
ham      0.865937
spam     0.134063
# 即：P(ham) = 0.865937, P(spam) = 0.134063
# 总选择概率大的类别

# 所以,如果利用输入特征 msg 进行预测,必须比空模型正确率更高

# 假设一个邮件内容是 send cash now，如何分类?
# 分别计算
# P(spam|send cash now) =
#     P(send cash now|spam)*P(spam)/P(send cash now)
# P(ham|send cash now) =
#     P(send cash now|ham)*P(ham)/P(send cash now)
# 然后比较大小
# 由于两者分母相同，只需计算分子然后比较大小
# P(send cash now|spam)*P(spam)
# P(send cash now|ham)*P(ham)

df.msg = df.msg.apply(lambda x:x.lower())
```

```
df[df.msg.str.contains('send cash now')].shape[0]
0    # 没有包含'send cash now'的邮件!

# Naive 假设:P({x1...xn}|C) = P(x1|C)*...*P(xn|C)

# P(send cash now|spam)
# = P(send|spam)*P(cash|spam)*P(now|spam)

# 样本中的全体 spam 邮件
spams = df[df.label == 'spam']
spam_total = float(spams.shape[0])

# 分别求 spam 条件下含 send,cash,now 的频率,并连乘起来
# 最后求出后验概率 P(spam|send cash now)
prod = 1
for word in ['send','cash','now']:
    spam_word = spams[spams.msg.str.contains(word)].shape[0]
    freq_word = spam_word / spam_total
    prod = prod * freq_word
    print word,freq_word
print "P(spam|send cash now) =",prod * 0.134063

send 0.0963855421687
cash 0.091030789826
now 0.279785809906
P(spam|send cash now) = 0.000329105259886

# P(send cash now|ham) * P(ham)
# = P(send|ham)*P(cash|ham)*P(now|ham)

# 样本中的全体 ham 邮件
hams = df[df.label == 'ham']
ham_total = float(hams.shape[0])

# 分别求 ham 条件下含 send,cash,now 的频率,并连乘起来
# 最后求出后验概率 P(ham|send cash now)
prod = 1
for word in ['send','cash','now']:
    ham_word = hams[hams.msg.str.contains(word)].shape[0]
    freq_word = ham_word / ham_total
    prod = prod * freq_word
    print word,freq_word
print "P(ham|send cash now) =",prod * 0.865937
```

```
send 0.0292227979275
cash 0.00269430051813
now 0.108808290155
P(ham|send cash now) = 7.41850018794e-06

# 两个概率值很小不是问题，因为我们忽略了贝叶斯公式的分母，重要的是前者比后者大很多：
0.000329105259886 / 7.41850018794e-06
44.362775702427705

# 因此我们预测邮件 send cash now 是垃圾邮件！

##############################################################
# 利用 sklearn 朴素贝叶斯分类

# 划分训练集和测试集
X_train,X_test,y_train,y_test =
    train_test_split(df.msg,df.label,random_state=1)

# 转换成文档词项矩阵
from sklearn.feature_extraction.text import CountVectorizer

vect = CountVectorizer()

train_dtm = vect.fit_transform(X_train)

train_dtm
<4179x7456 sparse matrix of type '<type 'numpy.int64'>'
        with 55209 stored elements in Compressed Sparse Row format>
# 7456 个单词

test_dtm =vect.transform(X_test)

test_dtm
<1393x7456 sparse matrix of type '<type 'numpy.int64'>'
        with 17604 stored elements in Compressed Sparse Row format>
# 测试集也按训练集的 dtm 进行转换

# 训练模型
from sklearn.naive_bayes import MultinomialNB

nb = MultinomialNB()
```

```
nb.fit(train_dtm,y_train)

# 对测试集作预测：为每个实例计算属于每个类别的概率，然后取最大概率的类别
y_predicted = nb.predict(test_dtm)

y_predicted
array(['ham', 'ham', 'ham', ..., 'ham', 'spam', 'ham'], dtype='|S4')


# 评估
from sklearn.metrics import accuracy_score,confusion_matrix

accuracy_score(y_test,y_predicted)
0.9885139985642498
# 大于空模型的 87%

confusion_matrix(y_test,y_predicted)
array([[1203,    5],
       [  11,  174]])

# 何为 postive/negative?按类别名称次序
nb.classes_
array(['ham', 'spam'], dtype='|S4')
# 所以 1203=真 ham，5=假 spam，11=假 ham，174 是真 spam
# 精确度
1.0*(1203+174)/(1203+174+5+11)
0.9885139985642498    # 与 accuracy_score()一样
```