实验三：基于贝叶斯算法的手机垃圾短信过滤

实验目的

研究一种可以过滤垃圾短信的分类算法，将会给移动电话供应商提供一种很有用的工具。因为朴素贝叶斯已经成功应用于垃圾邮件的过滤，所以它很可能也可以应用于垃圾短信的过滤，然而，相对于垃圾邮件来说，垃圾短信的自动过滤有着额外的挑战。

实验步骤

第 1 步——收集数据

数据来源： http://www.dt.fee.unicamp.br/~tiago/sms-spamcollection/

数据特征：该数据集包含短信的文本信息，而且带有表明该短信是否为垃圾短信的标签。垃圾短信标记为 spam，而非垃圾短信标记为 ham。

在下面的例子中有一些关于垃圾短信和非垃圾短信的例子。

下面是一些非垃圾短信的例子：

Better.Made up for Friday and stuffed myself like a pig yesterday.Now I feel bleh.But at least its not writhing pain kind of bleh.

If he started searching he will get job in few days.He have great potential and talent.

I got another job！The one at the hospital doing data analysis or something，starts on monday！Not sure when my thesis will got finished

下面是一些垃圾短信的例子：

Congratulations ur awarded 500of CD vouchers or 125gift guaranteed & Free entry 2100wkly draw txt MUSIC to 87066

December only！Had your mobile 11mths+？You are entitled to update to the latest colour camera mobile for Free！Call The Mobile Update Co FREE on 08002986906

Valentines Day Special！Win over£1000in our quiz and take your partner on the trip of a lifetime！Send GO to 83600now.150p/msg rcvd.

看到短信示例，注意垃圾短信的显著特点。

一方面，一个显著特点是 3 条垃圾短信中有 2 条短信使用了单词 free，但该单词没有出现在任何一条非垃圾短信中。

另一方面，与垃圾短信相比，有 2 条非垃圾短信引用了一周中具体的某一天，而垃圾短信中没有一条引用。

朴素贝叶斯分类器将利用词频中这种模式的优势来确定短信消息是更像垃圾短信还是非垃圾短信。尽管可以想象单词“free”可以出现在非垃圾短信中，但是一条合法的短信很有可能会根据上下文提供额外的单词信息。例如，一条非垃圾短信可能会这样陈述“are you free on Sunday？”；而一条垃圾短信可能会使用这样的短语“free ringtones”。朴素贝叶斯分类器将根据短信中所有单词提供的证据计算垃圾短信和非垃圾短信的概率。

第 2 步——探索和准备数据

构建分类器的第一步涉及原始数据的处理与分析，文本数据的准备具有挑战性，因为将词和句子转化成计算机能够理解的形式是很必要的。我们将把数据转化成一种称为词袋（bag-of-words）的表示方法，这种表示方法忽略了单词出现的顺序，只是简单地提供一个变量用来表示单词是否会出现。

首先使用 read.csv()函数导入上述 CSV 数据，将其保存到以 sms_raw 命名的数据文件中。

```
> sms_raw <- read.csv("sms_spam.csv", stringsAsFactors
 = FALSE)
```

使用数据结构探索函数 str()，可以看到 sms_raw 数据文件包含了 5558 条短信，每条短信都有两个特征：type 和 text。将 SMS 的特征 type 编码为 ham 或者 spam，而变量 text 存储整个 SMS 短信文本。

```
> str(sms_raw)
'data.frame': 5558 obs. of  2 variables:
 $ type: chr  "ham" "ham" "ham" "spam" ...
 $ text: chr  "Hope you are having a good week. Just ch
ecking in" "K..give back my thanks." "Am also doing in cbe
only. But have to pay." "complimentary 4 STAR Ibiza Holida
y or 拢10,000 cash needs your URGENT collection. 090663643
49 NOW from Landline"| __truncated__ ...
```

当前的变量 type 是一个字符串向量。由于它是一个分类变量，所以将其转换成一个因子将会更好，如下面的代码所示：

```
> sms_raw$type <- factor(sms_raw$type)
```

用函数 str()和 table()研究变量 type，可以看到该变量已经被很好

地重新编码为一个因子。此外，可以看到数据中有 747 条，（即大约 13%）短信被标记为 spam，其余的短信被标记为 ham。

```
> str(sms_raw$type)
 Factor w/ 2 levels "ham","spam": 1 1 1 2 2 1 1 1 2 1 ...
> table(sms_raw$type)
 ham spam
4811  747
```

下面我们研究变量 text。

数据准备——处理和分析文本数据

短信就是由词、空格、数字和标点符号组成的文本字符串。处理这种类型的复杂数据需要大量的思考和工作，一方面需要考虑如何去除数字和标点符号，如何处理没有意义的单词，如 and、but 和 or 等，以及如何将句子分解成单个的单词。

R 社区中已经在文本挖掘添加包 tm 中提供了这些功能。

可以通过命令 install.packages("tm")安装 tm 文本挖掘添加包，并应用命令 library(tm)进行加载。

处理文本数据的第一步涉及创建一个语料库，即一个文本文件的集合。这里一个文本文件就是指一条短信，通过下面的命令建立一个包含训练数据中短信的语料库。

```
> sms_corpus <- Corpus(VectorSource(sms_raw$text))
```

这条命令使用了两个函数。首先，函数 Corpus()创建了一个 R 对

象来存储文本文档。这个函数通过一个参数来指定所加载的文本文档的格式。因为已经把短信读入 R 并存储在一个向量中，所以用函数 VectorSource()来指示函数 Corpus()使用向量 sms_train$text 的信息。函数 Corpus()将结果存储在一个名为 sms_corpus 的对象中。

如果用函数 print()输出刚刚创建的语料库，将会看到该语料库包含了训练数据中 5558 条短信的每一条短信。

```
> print(sms_corpus)
<<SimpleCorpus>>
Metadata:    corpus specific: 1, document level (indexed): 0
Content:    documents: 5558
```

如果要查看语料库的内容，可以使用函数 inspect()。将该函数与访问向量的方法结合在一起，可以查看具体的短信内容。下面的命令就是查看第一、二、三条短信的具体内容：

```
> inspect(sms_corpus[1:3])
<<SimpleCorpus>>
Metadata:    corpus specific: 1, document level (indexed): 0
Content:    documents: 3
[1] Hope you are having a good week. Just checking in
[2] K..give back my thanks.
[3] Am also doing in cbe only. But have to pay.
```

语料库现在包含 5558 条短信的原始文本内容。再将文本内容分解成单词之前，需要进行一些清理步骤以去除标点符号和可能会影响

结果的其他字符。例如，将把单词 hello!、HELLO……和 Hello 都作为单词 hello 的实例。

函数 tm_map()提供了一种用来转换 tm 语料库的方法。将使用一系列转换函数来清理的语料库，并将结果保存为一个叫做 corpus_clean 的新对象。

首先，我们将把所有短信的字母变成小写字母，并去除所有的数字：

```
> corpus_clean <- tm_map(sms_corpus, tolower)
> corpus_clean <- tm_map(corpus_clean, removeNumbers)
```

在分析文本数据时，一个常见的做法就是去除填充词，比如 to、and、but 和 or，这些词称为停用词。使用 tm 添加包中提供的函数 stopwords()，而不是自己定义一个停用词列表，这个函数包含了一组大量的停用词。如果想知道它包含的停用词，在命令行输入 stopwords()即可。通过使用函数 tm_map()来剔除数据中的停用词。

```
> corpus_clean <- tm_map(corpus_clean, removeWords, stopwords())
```

同样，也可以去除标点符号：

```
> corpus_clean <- tm_map(corpus_clean, removePunctuation)
```

既然已经去除了数字、停用词和标点符号，那么文本信息中这些字符曾经所在的地方就变成了空格。最后一步就是去除额外的空格，只在词与词之间留下一个空格。

```
> corpus_clean <- tm_map(corpus_clean, stripWhitespace)
```

下面代码显示了短信语料库中前 3 条短信在清理前后的对比。短信消息已经被限制得只剩下最有意义的词，标点符号和大小写都已经被清理。

```
> inspect(sms_corpus[1:3])
<<SimpleCorpus>>
Metadata:  corpus specific: 1, document level (indexed): 0
Content:  documents: 3
[1] Hope you are having a good week. Just checking in
[2] K..give back my thanks.
[3] Am also doing in cbe only. But have to pay.
> inspect(corpus_clean[1:3])
<<SimpleCorpus>>
Metadata:  corpus specific: 1, document level (indexed): 0
Content:  documents: 3

[1] hope good week just checking
[2] kgive back thanks
[3]  also cbe pay
```

既然以想要的方式处理了数据，那么最后的步骤就是通过一个所谓的标记化过程将消息分解成由单个单词组成的组。一个记号就是一个文本字符串的单个元素，在这种情况下，本例中的记号就是单词。

tm 添加包提供了标记短信语料库的功能。函数

DocumentTerm-Matrix()将一个语料库作为输入，并创建一个称为稀疏矩阵的数据结构，其中矩阵的行表示文档（即短信），矩阵的列表示单词。矩阵中的每个单元存储一个数字，它代表由列标识的单词出现在由行所标识的文档中的次数。下面的截图显示了短信语料库的文档—单词（稀疏矩阵）矩阵的一小部分，而作为完整的稀疏矩阵则有5558 行和超过 7000 列。

A	B	C	D	E	F	G
	balloon	balls	bam	bambling	band	bandages
1	0	0	0	0	0	0
2	0	0	0	0	0	0
3	0	0	0	0	0	0
4	0	0	0	0	0	0
5	0	0	0	0	0	0

事实上，上表中的每一个格子中的 0 代表它们所在列的顶部所列出的单词没有出现在语料库的前 5 条短信中的任意一条中，这突出表明这个数据结构被称为稀疏矩阵的原因，即在该矩阵中，绝大多数元素是以 0 来填充的。尽管每个消息包含了一些单词，但是任意具体的单词出现在给定的一条消息中概率都是很小的。

给定 tm 语料库，创建一个稀疏矩阵要用到下述命令：

```
> sms_dtm <- DocumentTermMatrix(corpus_clean)
```

这条命令将语料库标记化，并返回一个名为 sms_dtm 的稀疏矩阵。从这里，就可以对包括词频在内的信息进行分析。

1.数据准备——建立训练数据集和测试数据集

由于已经为分析准备好了数据，所以现在需要将数据分成训练数据集和测试数据集，从而可以把垃圾短信分类器应用到之前没有学习过的数据上，并据此对分类器的性能进行评估。将数据分成两部分：75%的训练数据和 25%的测试数据。因为短信的排序是随机的，所以我们可以简单地取前 4168 条短信用于训练，剩下的 1390 条短信用于测试。

通过分解原始数据框开始：

```
> sms_raw_train <- sms_raw[1:4168, ]
> sms_raw_test  <- sms_raw[4170:5558, ]
```

然后输出文档—单词矩阵：

```
> sms_dtm_train <- sms_dtm[1:4168, ]
> sms_dtm_test  <- sms_dtm[4170:5558, ]
```

最后，得到语料库：

```
> sms_corpus_train <- corpus_clean[1:4168]
> sms_corpus_test  <- corpus_clean[4170:5558]
```

为了确认上述子集是一组完整的短信数据的代表，可以通过比较垃圾短信在训练数据和测试数据中所占的比例：

```
> prop.table(table(sms_raw_train$type))

      ham      spam

0.8646833 0.1353167

> prop.table(table(sms_raw_test$type))

      ham      spam

0.8682505 0.1317495
```

无论是训练数据集和测试数据集，它们都包含约 13%的垃圾短信，这表明垃圾短信被平均分配在这两个数据集中。

2.可视化文本数据——词云

词云是一种可视化地描绘单词出现在文本数据中频率的方式。词云是由随机分布在词云图中的单词构成的，经常出现在文本中的单词会以较大的字体呈现，而不太常见的单词将会以较小的字体呈现。最近，这种类型的图已经变得越来越流行，因为它提供了一种观察社交媒体网站上热门话题的方式。

wordcloud 添加包提供了一个简单的 R 函数来创建这种类型的图形，我们将应用这个函数使短信中单词类型可视化。比较垃圾短信和非垃圾短信的词云将有助于我们了解朴素贝叶斯短信过滤器是否有可能成功。如果还没有安装 wordcloud 添加包，你需要在 R 命令行输入命令 install.packages("wordcloud")来安装这个添加包，并输入 library(wordcloud)来加载它。

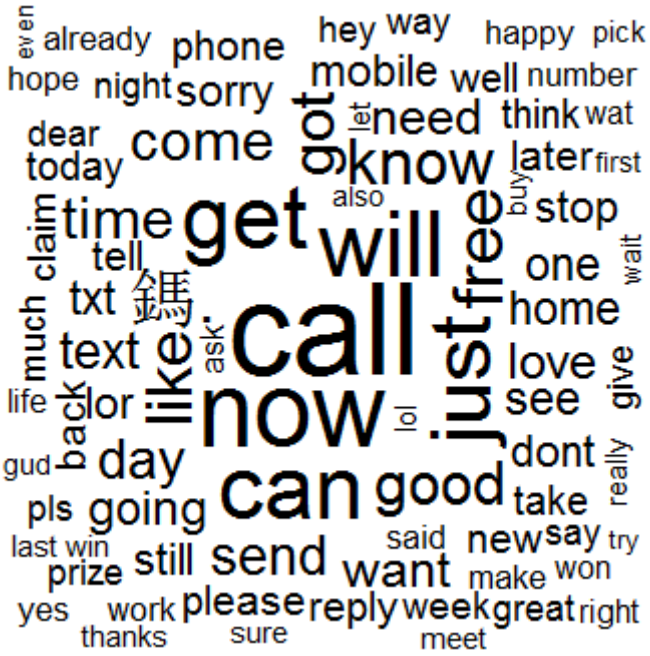
可以从 tm 语料库对象直接创建词云，命令如下所示：

```
> wordcloud(sms_corpus_train, min.freq = 40, random.order = FALSE)
```

该命令将从 sms_corpus_train 语料库创建一个词云。由于设置了 random.order=FALSE，所以该词云将以非随机的顺序排列，而且出现频率越高的单词越靠近中心。如果没有设置 random.order，该词云将会以默认的随机方式排列。参数 min.freq 用来指定显示在词云中的单

词必须满足在语料库中出现的最小次数。通用规则是，开始时设置参数 min.freq 的值为语料库中文档总数目的 10%。这里，文档总数目的 10%大约为 40。因此，词云中的单词至少在 40 条短信中出现过。

该词云的结果如下图所示。



另一个有趣的可视化涉及垃圾短信和非垃圾短信词云的比较。由于没有对垃圾短信和非垃圾短信分别建立语料库，所以这时应该应用函数 wordcloud()一个非常有用的功能。给定原始文本，在建立语料库和显示词云之前，它会自动应用文本转换过程。

根据短信类型，通过 R 函数 subset()来获取其中一种类型的短信数据，即抽取 sms_raw_train 的一个子集。首先，我们将创建 type 等于 spam 的子集：

```
> spam <- subset(sms_raw_train, type == "spam")
```

其次，获取子集 ham：

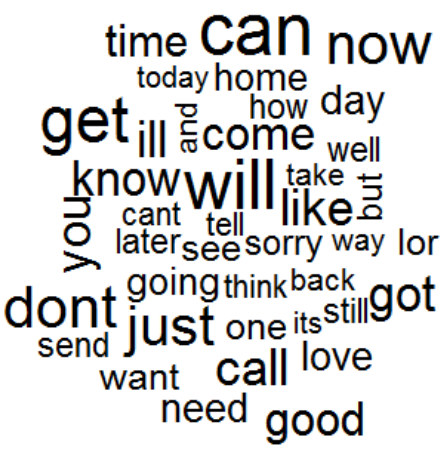
```
> ham <- subset(sms_raw_train, type == "ham")
```

现在，有两个短信数据框：spam（垃圾短信）和 ham（非垃圾短信），每一个都带有包含原始文本字符串的文本特征。创建词云就像之前一样简单。这一次，使用参数 max.words，它用来显示两个集合的任何一个集合中最常见的 40 个单词，而且参数 scale 允许调整词云中单词的最大字体和最小字体。可以自由调整这些参数直到合适。如下面的代码所示：

```
> wordcloud(spam$text, max.words = 40, scale = c(3, 0.5))

> wordcloud(ham$text, max.words = 40, scale = c(3, 0.5))
```

所得到的词云显示在下面的图形中。对于哪一幅图代表的是垃圾短信，哪一幅图代表的是非垃圾短信，你有直觉吗？



通过多次运行 wordcloud()函数来选择最令人满意的词云，从而达到演示的目的。

你是否已经猜到左边的图形就是垃圾短信的词云。垃圾短信包括 urgent、free、mobile、call、claim 和 stop 等词，而这些单词一次都没有出现在非垃圾短信中，相反非垃圾短信使用的单词有 can、sorry、

need 和 time 等。这些明显的差异表明朴素贝叶斯模型将会有一些强有力的关键词来对类别进行区分。

3.数据准备——为频繁出现的单词创建指示特征

数据准备过程中的最后一步就是把稀疏矩阵转换成可用于训练朴素贝叶斯分类器的数据结构。目前，该稀疏矩阵包含数量超过 7000 个的特征，即至少出现在一条短信中的每一个单词的特征。所有这些特征不可能都对分类发挥作用。为了减少特征的数量，剔除训练数据中出现在少于 5 条短信中或者少于记录总数的 0.1%的所有单词。

查找频繁出现的单词需要使用 tm 添加包中的 findFreqTerms()函数，该函数输入一个文档-单词矩阵，并返回一个字符向量，该向量包含出现次数不少于指定次数的单词。举个例子，下面的命令将显示一个字符向量，该向量中的单词在矩阵 sms_dtm_train 中至少出现 5 次：

```
> sms_dict <- findFreqTerms(sms_dtm_train, 5)
```

为了限制训练矩阵和测试矩阵只包含前面词典中的单词，可以使用下面的命令：

```
> sms_train <- DocumentTermMatrix(sms_corpus_train,list(dictionary = sms_dict))
> sms_test <- DocumentTermMatrix(sms_corpus_test, list(dictionary = sms_dict))
```

现在，训练数据和测试数据包含了大约 1200 个特征，只对应于

至少出现在 5 条短信中的单词。

朴素贝叶斯分类器通常是训练具有明确特征的数据，这就带来了一个问题，因为稀疏矩阵中的元素表示一个单词出现在一条消息中的次数。于是，需要将其改变成因子变量，根据单词是否出现，简单地表示成 yes 或者 no。

下面的代码定义了一个函数 convert_counts()，它将计数转换成因子：

```
> convert_counts <- function(x) {
x <- ifelse(x > 0, 1, 0)
    x <- factor(x, levels = c(0, 1), labels = c(""No"", ""Yes""))
    return(x)
  }
```

其中，第一行用来定义函数，语句 ifelse（x>0，1，0）将改变 x 中的值，如果该值大于 0，则它会被 1 代替，否则，它仍然为 0。命令 factor 简单地将值 0 和 1 转化为用 no 和 yes 所标记的因子。最后，返回变换后的向量 x。

现在，只需要将 convert_counts 应用于稀疏矩阵的每一列。这里使用 apply()函数。

函数 apply()允许一个函数作用于一个矩阵的每一行或者每一列，它使用参数 MARGIN 来指定作用的对象是矩阵的行或者列。这里处理的是矩阵的列，所以令 MARGIN=2（MARGIN=1 表示的是行）。用来转换训练矩阵和测试矩阵的完整命令如下所示：

```
> sms_train <- apply(sms_train, MARGIN = 2, convert_counts)
> sms_test  <- apply(sms_test, MARGIN = 2, convert_counts)
```

结果将是两个矩阵，每个矩阵都带有因子类型的列，用 Yes 和 No 来表示每一列的单词是否出现在构成行的信息中。

第 3 步——基于数据训练模型

因为已经将原始短信转换成了可以用一个统计模型代表的形式，所以此时是应用朴素贝叶斯算法的时候了。该算法将根据单词的存在与否来估计一条给定的短信是垃圾短信的概率。

采用 e1017 添加包中的朴素贝叶斯算法实现。在继续之前，需要使用命令 install.packages("e1071")和 library(e1071)来准备好这个包。

与前面章节中用于分类的 KNN 算法不同，训练一个朴素贝叶斯分类器和使用朴素贝叶斯进行分类是发生在不同的阶段的。尽管如此，分类相当简单，如下表所示。

朴素贝叶斯分类语法
应用 e1071 添加包中的函数 naiveBayes()

创建分类器：
`m <- naiveBayes(train, class, laplace = 0)`
• train：数据框或者包含训练数据的矩阵
• class：包含训练数据每一行的分类的一个因子向量
• laplace：控制拉普拉斯估计的一个数值（默认为 0）
该函数返回一个朴素贝叶斯模型对象，该对象能够用于预测。
进行预测：
`p <- predict(m, test, type = "class")`
• m：由函数 naiveBayes() 训练的一个模型
• test：数据框或者包含测试数据的矩阵，包含与用来建立分类器的训练数据相同的特征
• type：值为“class”或者“raw”，标识预测是最可能的类别值或者原始的预测概率
该函数将返回一个向量，根据参数 type 的值，该向量含有预测的类别值或者原始预测的概率值。
例子：
`sms_classifier <- naiveBayes(sms_train, sms_type)`
`sms_predictions <- predict(sms_classifier, sms_test)`

为了基于 sms_train 矩阵建立模型，使用如下的命令：

```
> sms_classifier <- naiveBayes(sms_train, sms_raw_train$type)
```

sms_classifier 变量现在包含一个可以用于预测的 naiveBayes 分类器对象。

第 4 步——评估模型的性能

为了评估短信分类器，需要基于测试数据中未知的短信来检验分类器的预测值。回想一下，未知的短信特征存储在一个名为 sms_test 的矩阵中，而分类标签 spam 和 ham 存储在 sms_raw_test 数据框中一个名为 type 的向量中。把已经训练过的分类器命名为 sms_classifier，用它来产生预测值，并将预测值与真实值相比较。

用函数 predict()进行预测，并将这些预测值存储在一个名为 sms_test_pred 的向量中：

```
> sms_test_pred <- predict(sms_classifier, sms_test)
```

为了比较预测值和真实值，使用 gmodels 添加包中的函数 CrossTable()，在之前已经使用过这个函数。这一次，将增加一些额外的参数来消除不必要的元素的比例，并使用参数 dnn（维度名称）来重新标记行和列，如下面的代码所示：

```
> library(gmodels)
> CrossTable(sms_test_pred, sms_raw_test$type,
prop.chisq = FALSE, prop.t = FALSE,
      dnn = c('predicted', 'actual'))
```

这就产生了下面的表格：

	actual		
predicted	ham	spam	Row Total
ham	1203	32	1235
	0.974	0.026	0.888
	0.997	0.175	
spam	4	151	155
	0.026	0.974	0.112
	0.003	0.825	
Column Total	1207	183	1390
	0.868	0.132	

从表中可以看出，1206 条非垃圾短信中有 4 条短信被错误地归为垃圾短信，比例为 0.3%，而 183 条垃圾短信中有 32 条短信被错误地归为非垃圾短信，比例为 17.5%。考虑到在这个案例中，几乎没有做什么工作，具有这种水平的表现是相当好的。另外，本案例的研究也说明了为什么说朴素贝叶斯算法是用于文本分类的一种标准算法，而且朴素贝叶斯方法可以直接拿来使用，执行的效果也是出奇地好。

另一方面，被错误地归为垃圾短信的 4 条短信可能会为过滤算法的部署带来显著的问题。如果过滤器导致某人错过一条重要的约会或者紧急情况的短信，那么会很快放弃这种产品，因此需要研究这些错误分类的短信，看看是哪里出了问题。

第 5 步——提升模型的性能

在训练模型时，并没有设置一个值来用于拉普拉斯估计。毫无疑问，这样就允许在分类的过程中单词可能出现在 0 条垃圾短信或者 0 条非垃圾短信中。比如，虽然单词“ringtone”只出现在训练数据的垃圾短信中，但这并不意味着每一个含有单词“ringtone”的短信就

应该被归为垃圾短信。

我像之前一样建立朴素贝叶斯模型，但这一次我们设置 laplace=1：

```
> sms_classifier2 <- naiveBayes(sms_train, sms_raw_train$type,
      laplace = 1)
```

接下来，进行预测：

```
> sms_test_pred2 <- predict(sms_classifier2, sms_test)
```

最后，使用交叉表来比较预测的分类和真实的分类情况：

```
> CrossTable(sms_test_pred2, sms_raw_test$type,
prop.chisq = FALSE, prop.t = FALSE, prop.r = FALSE,
      dnn = c('predicted', 'actual'))
```

该结果如下表所示。

```
Total Observations in Table:  1390


             | actual
   predicted |        ham |       spam | Row Total |
-------------|------------|------------|-----------|
         ham |       1203 |         31 |      1234 |
             |      0.997 |      0.169 |           |
-------------|------------|------------|-----------|
        spam |          4 |        152 |       156 |
             |      0.003 |      0.831 |           |
-------------|------------|------------|-----------|
Column Total |       1207 |        183 |      1390 |
             |      0.868 |      0.132 |           |
-------------|------------|------------|-----------|
```

将错误地归为非垃圾短信的垃圾短信数量由 32 减少到 31。虽然这看上去是一个很小的改进，但必须意识到如果对垃圾短信的过滤过于激进，那么重要信息被遗漏的可能性是很大的。

实验结论

朴素贝叶斯进行分类算法构建了概率表用来估计新案例属于不同类别的似然。概率是通过一个称为贝叶斯定理的公式来计算的，它表明相关事件是如何相关的。尽管贝叶斯公式的计算是很复杂的过程，但是应用一个事件相互独立的“简单”假设后，就得到一个可用于巨大数据集的简化贝叶斯算法。

朴素贝叶斯分类器通常用于文本分类。为了说明其有效性，采用朴素贝叶斯进行了一个关于过滤垃圾短信的分类任务。需要专门的 R 添加包用于准备需要分析的文本数据、预处理文本以及文本的可视化。最终，该模型能够将大约 98%的短信正确地分成垃圾短信和非垃圾短信。