Naive Bayes Classifier in R

朱新梅

R 中的包 e1071 提供了运用朴素贝叶斯分类器的教程：http://cran.r-project.org/web/packages/e1071/e1071.pdf

(一) 输入数据都是分类型数据

首先安装并载入这个包：

```
# install.packages('e1071', dependencies = TRUE)
library(e1071)
```

所使用的数据集是包 mlbench 中的 HouseVotes84 数据集。

```
data(HouseVotes84, package = "mlbench")
```

这个数据集是 1984 年美国国会的投票记录。共有 435 个观测值和 17 个分类型变量，其中的 Class 变量记录的是投票者是共和党 (republican) 还是民主党 (democrat)，剩下的 16 个分类变量记录的是这 435 个投票者对 16 个问题的投票结果，同意是 “y”，反对是 “n”，由于种种原因没有投票记录的话是缺失值 “NA”。可以用 structure 函数看一下这个数据集的结构：

```
str(HouseVotes84)
```

```
## 'data.frame':    435 obs. of  17 variables:
##  $ Class: Factor w/ 2 levels "democrat","republican": 2 2 1 1 1 1 1 2 2 1 ...
##  $ V1   : Factor w/ 2 levels "n","y": 1 1 NA 1 2 1 1 1 1 2 ...
##  $ V2   : Factor w/ 2 levels "n","y": 2 2 2 2 2 2 2 2 2 2 ...
##  $ V3   : Factor w/ 2 levels "n","y": 1 1 2 2 2 2 1 1 1 2 ...
##  $ V4   : Factor w/ 2 levels "n","y": 2 2 NA 1 1 1 2 2 2 1 ...
##  $ V5   : Factor w/ 2 levels "n","y": 2 2 2 NA 2 2 2 2 2 1 ...
##  $ V6   : Factor w/ 2 levels "n","y": 2 2 2 2 2 2 2 2 2 1 ...
##  $ V7   : Factor w/ 2 levels "n","y": 1 1 1 1 1 1 1 1 1 2 ...
##  $ V8   : Factor w/ 2 levels "n","y": 1 1 1 1 1 1 1 1 1 2 ...
##  $ V9   : Factor w/ 2 levels "n","y": 1 1 1 1 1 1 1 1 1 2 ...
##  $ V10  : Factor w/ 2 levels "n","y": 2 1 1 1 1 1 1 1 1 1 ...
##  $ V11  : Factor w/ 2 levels "n","y": NA 1 2 2 2 1 1 1 1 1 ...
##  $ V12  : Factor w/ 2 levels "n","y": 2 2 1 1 NA 1 1 1 2 1 ...
```

```
##  $ V13  : Factor w/ 2 levels "n","y": 2 2 2 2 2 2 NA 2 2 1 ...
##  $ V14  : Factor w/ 2 levels "n","y": 2 2 2 1 2 2 2 2 2 1 ...
##  $ V15  : Factor w/ 2 levels "n","y": 1 1 1 1 2 2 2 NA 1 NA ...
##  $ V16  : Factor w/ 2 levels "n","y": 2 NA 1 2 2 2 2 2 2 NA ...
```

我们的目标是根据对这 16 个项目的投票结果通过朴素贝叶斯分类器预测投票者是共和党还是民主党。首先抽取 60% 的数据作为训练集，剩下的作为测试集：

```
set.seed(2016)
n = nrow(HouseVotes84) # sample size
p = 0.6 # proportion of training set
idx = sample(n, n*p, replace = FALSE) # index set for training set
# training set
train = HouseVotes84[idx, ]
# test set
test = HouseVotes84[-idx, ]
```

然后用朴素贝叶斯分类器进行训练，需要用到的函数是 naiveBayes

```
model = naiveBayes(Class~., data = train)
```

最后用我们的模型进行预测：

```
result = predict(model, test[, -1], type = "class")
```

在 predict 函数中，type 变量可以设为 class 或 raw，前者得到的结果是预测的类别，后者得到的是各个类别的后验概率。默认值为 class。

看一下预测的准确率：

```
table(result, test$Class)
```

```
##
## result       democrat republican
##   democrat         89          5
##   republican       13         67
```

```
mean(result == test$Class) # accuracy
```

```
## [1] 0.8965517
```

准确率达到 $(89+67)/(89+67+5+13)=90\%$。

(二) 输入数据都是连续型数据

我们使用的数据是 kernlab 包中的 spam 数据集。该数据集搜集了 4601 份电子邮件，并人工分类为垃圾邮件 (spam) 和非垃圾邮件 (non-spam)，记录在数据表最后一列 type 内。并记录了这些邮件中 57 个单词或符号的频率，作为 57 个特征。可以载入数据，并看一下数据的形式：

```
# install.packages("kernlab")
data(spam, package = "kernlab")
head(spam, 4)
```

```
##   make address  all num3d  our over remove internet order mail receive
## 1 0.00    0.64 0.64     0 0.32 0.00   0.00     0.00  0.00 0.00    0.00
## 2 0.21    0.28 0.50     0 0.14 0.28   0.21     0.07  0.00 0.94    0.21
## 3 0.06    0.00 0.71     0 1.23 0.19   0.19     0.12  0.64 0.25    0.38
## 4 0.00    0.00 0.00     0 0.63 0.00   0.31     0.63  0.31 0.63    0.31
##   will people report addresses free business email  you credit your font
## 1 0.64   0.00   0.00      0.00 0.32     0.00  1.29 1.93   0.00 0.96    0
## 2 0.79   0.65   0.21      0.14 0.14     0.07  0.28 3.47   0.00 1.59    0
## 3 0.45   0.12   0.00      1.75 0.06     0.06  1.03 1.36   0.32 0.51    0
## 4 0.31   0.31   0.00      0.00 0.31     0.00  0.00 3.18   0.00 0.31    0
##   num000 money hp hpl george num650 lab labs telnet num857 data num415
## 1   0.00  0.00  0   0      0      0   0    0      0      0    0      0
## 2   0.43  0.43  0   0      0      0   0    0      0      0    0      0
## 3   1.16  0.06  0   0      0      0   0    0      0      0    0      0
## 4   0.00  0.00  0   0      0      0   0    0      0      0    0      0
##   num85 technology num1999 parts pm direct cs meeting original project
## 1     0          0    0.00     0  0   0.00  0       0     0.00       0
## 2     0          0    0.07     0  0   0.00  0       0     0.00       0
## 3     0          0    0.00     0  0   0.06  0       0     0.12       0
## 4     0          0    0.00     0  0   0.00  0       0     0.00       0
##     re  edu table conference charSemicolon charRoundbracket
## 1 0.00 0.00     0          0          0.00            0.000
## 2 0.00 0.00     0          0          0.00            0.132
## 3 0.06 0.06     0          0          0.01            0.143
## 4 0.00 0.00     0          0          0.00            0.137
##   charSquarebracket charExclamation charDollar charHash capitalAve
## 1                 0           0.778      0.000    0.000      3.756
## 2                 0           0.372      0.180    0.048      5.114
## 3                 0           0.276      0.184    0.010      9.821
## 4                 0           0.137      0.000    0.000      3.537
```

```
##   capitalLong capitalTotal type
## 1          61          278 spam
## 2         101         1028 spam
## 3         485         2259 spam
## 4          40          191 spam
```

仍旧同之前一样，将数据分为训练集和测试集，然后用朴素贝叶斯模型进行训练：

```
set.seed(2016)
n = nrow(spam) # sample size
p = 0.6 # proportion of training set
idx = sample(n, n*p, replace = FALSE) # index set for training set
# training set
train = spam[idx, ]
# test set
test = spam[-idx, ]
model = naiveBayes(type~., data = train)
result = predict(model, test[, -ncol(spam)], type = "class")
```

最后看一下准确率

```
table(result, test$type)
```

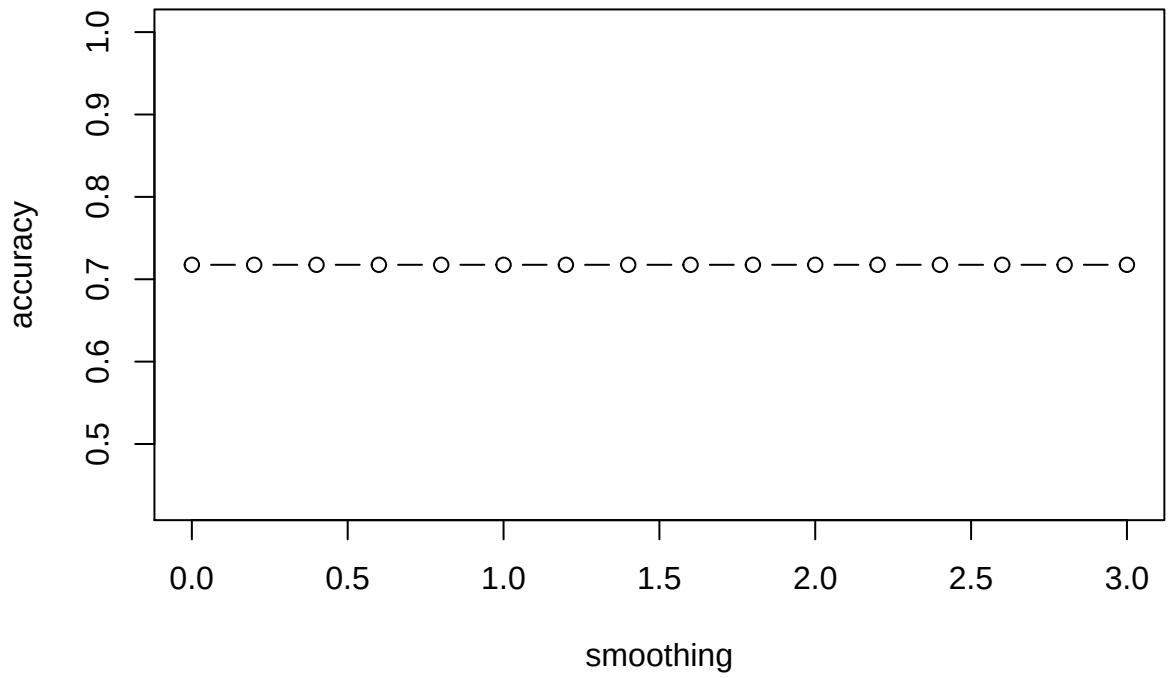
```
##
## result    nonspam spam
##   nonspam     639   49
##   spam        471  682
```

```
mean(result == test$type)
```

```
## [1] 0.7175448
```

准确率并不高，只有 72%。我们还可以看一下不同的 *smoothing* 参数对预测效果的影响

```
s = seq(from = 0, to = 3, by = 0.2)
accuracy = numeric(length(s))
for(i in seq_along(s)){
        model = naiveBayes(type~., data = train, laplace = s[i])
        result = predict(model, test[, -ncol(spam)], type = "class")
        accuracy[i] = mean(result == test$type)
}
plot(s, accuracy, type = "b", xlab = "smoothing")
```



可以看出对这个数据集而言，*smoothing* 与否并没有太大的影响。

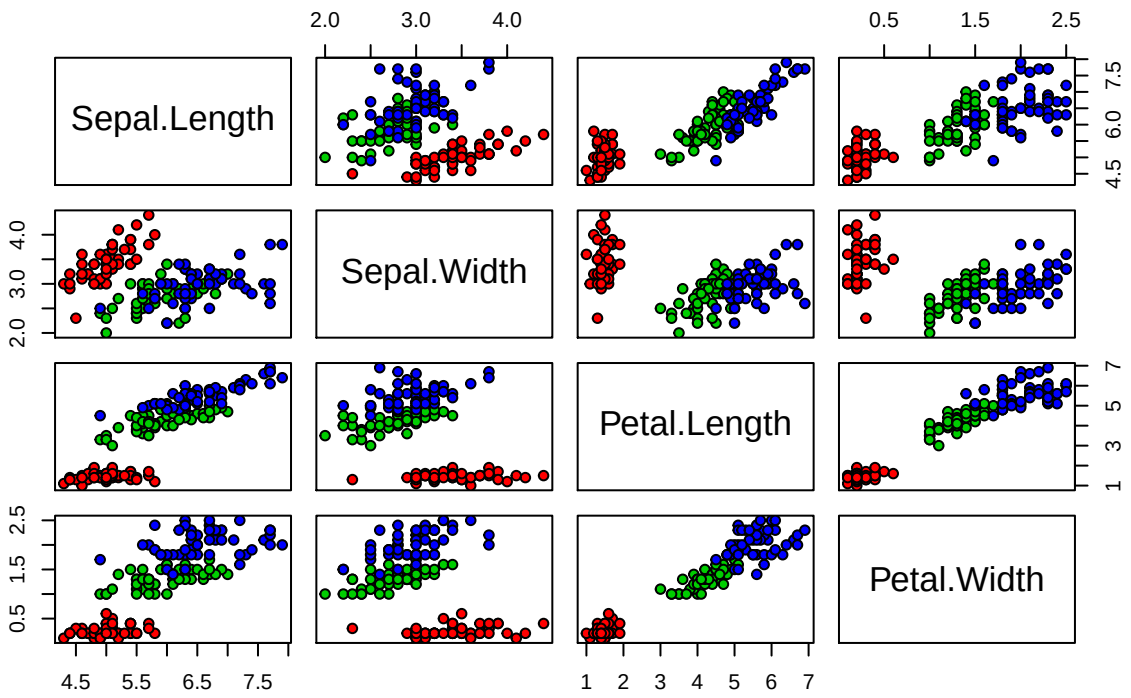
用 caret 包进行朴素贝叶斯分类

我们使用的是 R 自带的 iris 数据集。iris 给出了三种类型的鸢尾花 Iris setosa，versicolor 和 virginica 在四种特征变量上的差异。可以看一下这个数据集，并画出散点图：

```
summary(iris)
```

```
##   Sepal.Length    Sepal.Width     Petal.Length    Petal.Width
##  Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100
##  1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300
##  Median :5.800   Median :3.000   Median :4.350   Median :1.300
##  Mean   :5.843   Mean   :3.057   Mean   :3.758   Mean   :1.199
##  3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.:1.800
##  Max.   :7.900   Max.   :4.400   Max.   :6.900   Max.   :2.500
##        Species
##  setosa    :50
##  versicolor:50
##  virginica :50
##
##
##
```

```
pairs(iris[1:4],
      main = "Iris Data (red = setosa, green = versicolor, blue = virginica)",
      pch = 21, bg = c("red", "green3", "blue")[unclass(iris$Species)])
```



载入 caret 包，将数据分为训练集和测试集：

```
library(caret)
```

```
## Loading required package: lattice
```

```
## Loading required package: ggplot2
```

```
set.seed(2016)
idx = createDataPartition(y = iris$Species,
                          p = .6,
                          list = FALSE)
training = iris[idx, ]
test = iris[-idx, ]
```

然后用朴素贝叶斯模型进行训练，只需要将参数设为"nb" 即可。选用了 10 重交叉验证：

```
model = train(training[, -5], training[, 5], "nb",
              trControl = trainControl(method ="cv", number = 10))
```

```
## Loading required package: klaR
```

```
## Loading required package: MASS
```

用最终得到的模型在测试集上进行预测：

```
result = predict(model$finalModel, test[, -5])$class
table(result, test[, 5])
```

```
##
## result       setosa versicolor virginica
##   setosa         20          0         0
##   versicolor      0         19         1
##   virginica       0          1        19
```

```
mean(result == test[, 5])
```

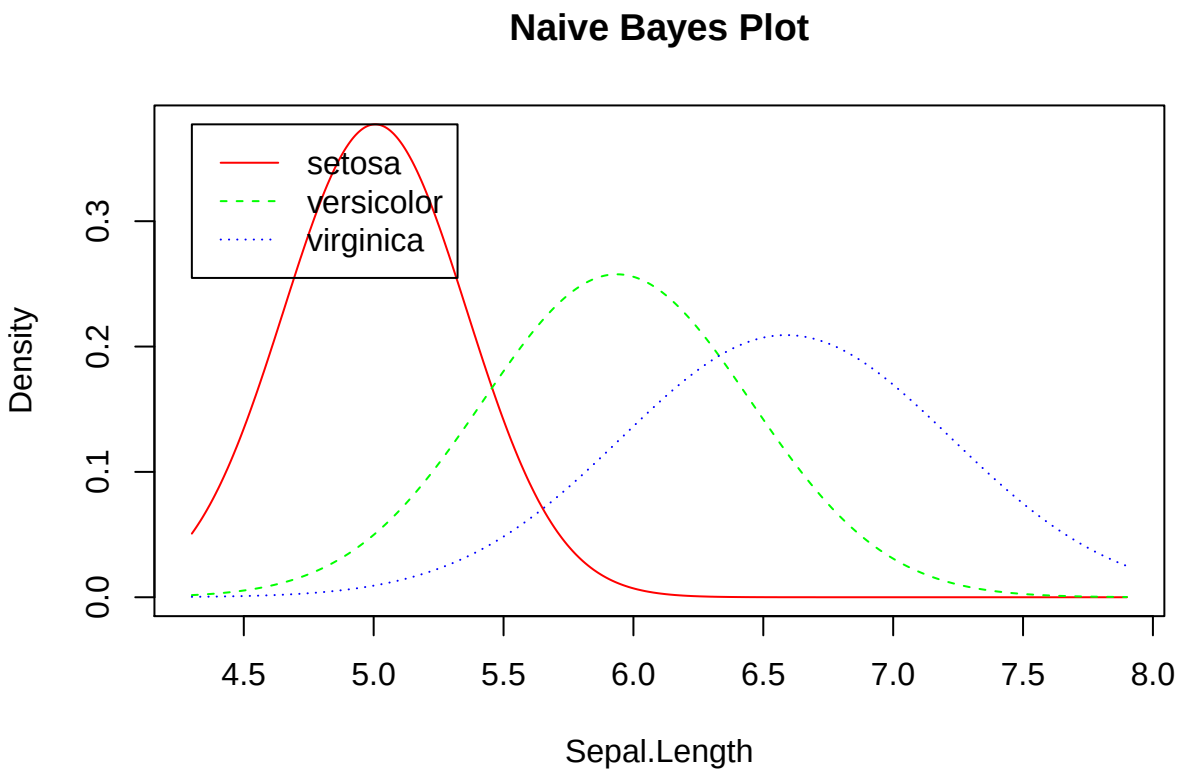
```
## [1] 0.9666667
```

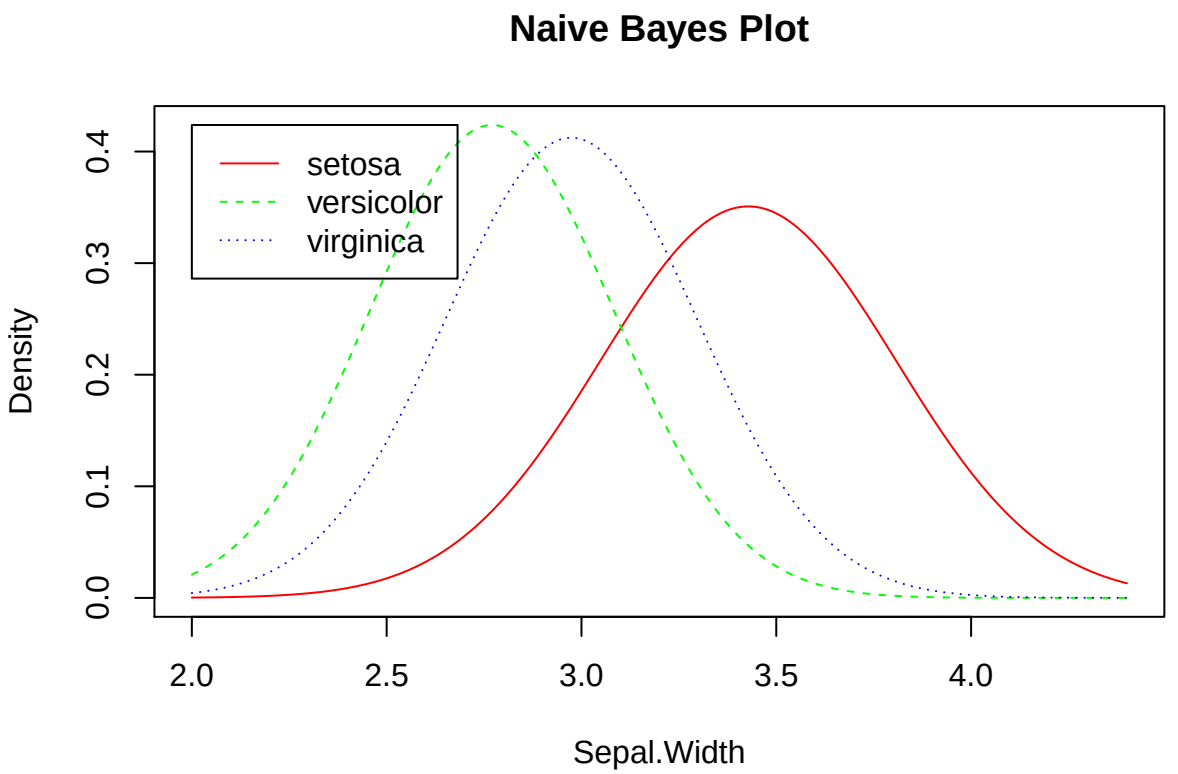
准确率达到 97%。

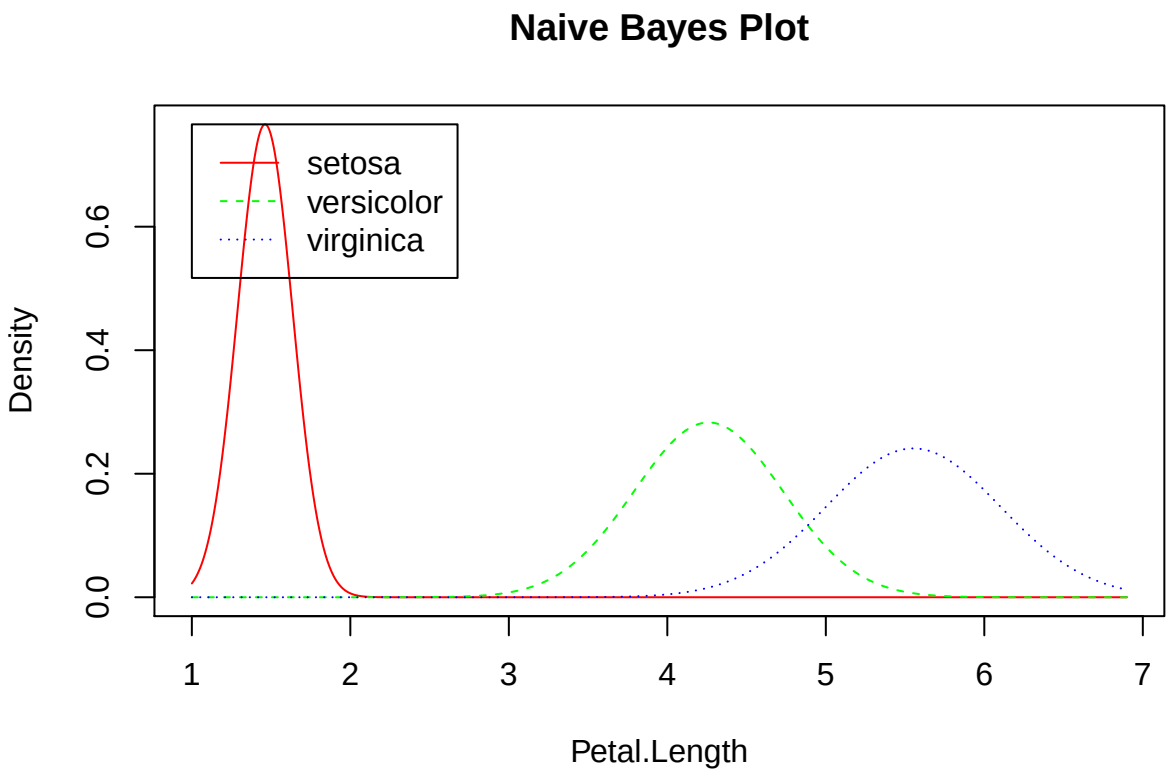
用 klaR 包进行朴素贝叶斯分类

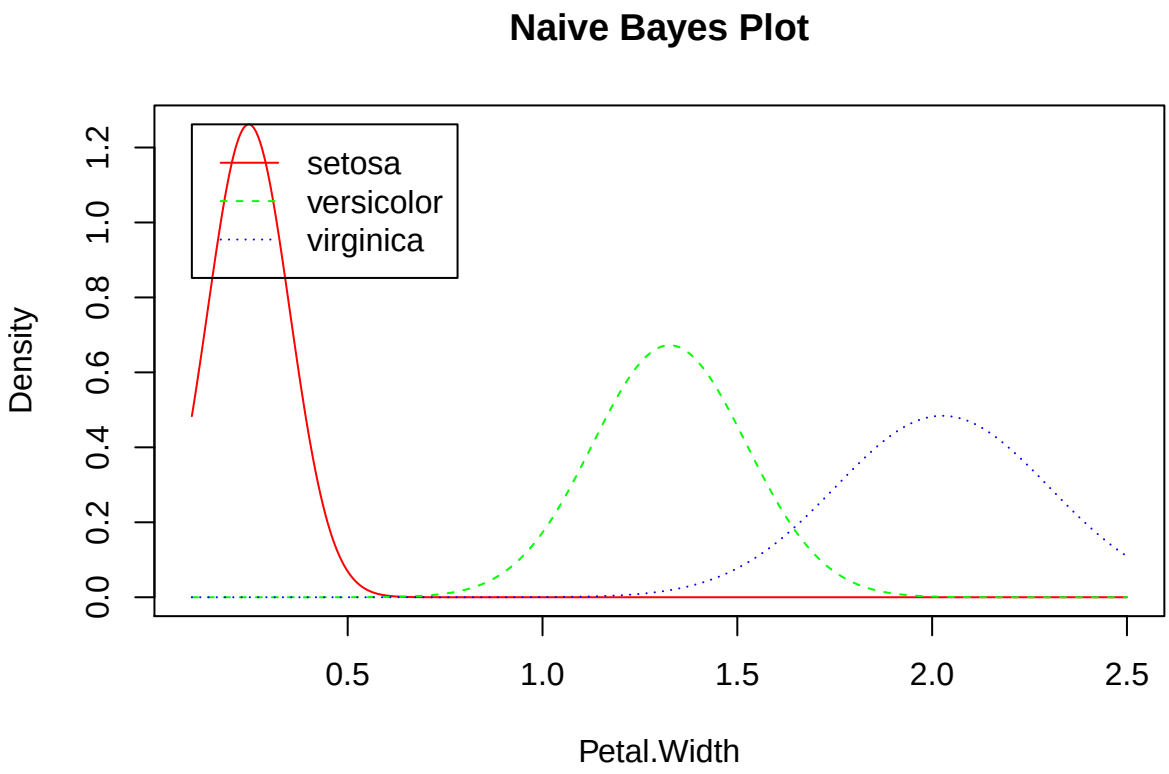
步骤都是类似的，另外可以画出高斯朴素贝叶斯分类后，不同类别的鸢尾花对应的不同的高斯分布的密度曲线。

```
plot(NaiveBayes(iris$Species ~ ., data = iris))
```


Naive Bayes Plot
setosa
versicolor
virginica
Density
0.0
0.1
0.2
0.3
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0
Sepal.Length


Naive Bayes Plot
setosa
versicolor
virginica
Density
0.0
0.1
0.2
0.3
0.4
2.0
2.5
3.0
3.5
4.0
Sepal.Width


Naive Bayes Plot
setosa
versicolor
virginica
Density
0.0
0.2
0.4
0.6
1
2
3
4
5
6
7
Petal.Length


Naive Bayes Plot
setosa
versicolor
virginica
Density
0.0
0.2
0.4
0.6
0.8
1.0
1.2
0.5
1.0
1.5
2.0
2.5
Petal.Width