Naive Bayes - An example using R language

The purpose of this code snippet is to demonstrate to the reader how a Naive Bayes (NB) classifier can be trained on a given dataset using the R language.

The dataset: In order to implement the above code, we use the well-known Iris dataset(included with R) for this purpose. It consists of four features (measurements), namely sepal length, sepal width, petal length, and petal width for 150 flowers. The dataset contains information about three types of iris plants: Setosa, Versicolor and Virginica.

Machine learning (ML) task: We will train a NB model to identify the plant type based on the four measurements of a given flower. So, our ML task in this problem would be a classification.

```r
data(iris) # Attach the Iris dataset to the R environment
mydata <- iris # Let's rename the dataset as mydata
dim(mydata) # check dimensions of mydata
```

```
## [1] 150   5
```

```r
sapply(mydata, class) # check the data types of each feature
```

```
## Sepal.Length  Sepal.Width Petal.Length  Petal.Width      Species
##    "numeric"    "numeric"    "numeric"    "numeric"     "factor"
```

```r
levels(mydata$Species) # check different levels (values) for each class
```

```
## [1] "setosa"     "versicolor" "virginica"
```

```r
head(mydata) # have a look at top data points in mydata
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

```r
summary(mydata) # a summary of class distributions
```

```
##   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  
##                 
##                 
## 
```

Creating training and validation datasets: We're going to construct a 80/20 partitioning for the training and validation sets. We use the createDataPartition function from the caret package for this purpose.

```
#install.packages("caret") #If the caret package is not installed on your system, uncomment this line t
library(caret) #Loading the library
```

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

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

```
tr_index <- createDataPartition(mydata$Species, p=0.80, list=FALSE) # List of 80% of the rows
trainSet <- mydata[tr_index,] # select 80% of the data for the trainSet
testSet <- mydata[-tr_index,] # Select the remaining 20% of data for testSet
```

Building a NB classifier: Now we will train our NB classifier using the above trainSet. For this purpose, we will utilize e1071 package in R. Note the priori probabilities when outputting the following lines of code. We get the same priori probabilities for all classes.

```
#install.packages("e1071") #If the e1071 package is not installed on your system, uncomment this line t
library(e1071)
NBclassfier=naiveBayes(Species~., data=trainSet) # Once you call this line, R fits the NB model using t
print(NBclassfier) # Check the newly fitted model to see if everything is OK.
```

```
##
## Naive Bayes Classifier for Discrete Predictors
##
## Call:
## naiveBayes.default(x = X, y = Y, laplace = laplace)
##
## A-priori probabilities:
## Y
##     setosa versicolor  virginica
##  0.3333333  0.3333333  0.3333333
##
## Conditional probabilities:
##             Sepal.Length
## Y              [,1]      [,2]
##   setosa     5.0050 0.3699965
##   versicolor 5.9625 0.4441572
##   virginica  6.6575 0.6872045
##
##             Sepal.Width
## Y              [,1]      [,2]
##   setosa     3.4075 0.3996072
##   versicolor 2.8025 0.2913166
##   virginica  2.9650 0.3453352
##
##             Petal.Length
## Y              [,1]      [,2]
##   setosa     1.4600 0.1676382
##   versicolor 4.2725 0.4101204
##   virginica  5.6250 0.5700877
##
##             Petal.Width
## Y              [,1]      [,2]
##   setosa     0.2450 0.1084861
##   versicolor 1.3375 0.1983102
##   virginica  2.0500 0.2726884
```

Make predictions: Now let's apply the above model to assign labels for test cases in testSet. Then we create the confusion matrix, a table that is often used to describe the performance of a classifier.

```
testPrediction=predict(NBclassfier, newdata=testSet, type="class") # Assign labels for each test case
confusionMatrix(testPrediction, testSet$Species) # Print confusion matrix
```

```
## Confusion Matrix and Statistics
##
##             Reference
## Prediction   setosa versicolor virginica
##   setosa         10          0         0
##   versicolor      0          9         1
##   virginica       0          1         9
##
## Overall Statistics
##
##                Accuracy : 0.9333
##                  95% CI : (0.7793, 0.9918)
##     No Information Rate : 0.3333
##     P-Value [Acc > NIR] : 8.747e-12
##
##                   Kappa : 0.9
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: setosa Class: versicolor Class: virginica
## Sensitivity                 1.0000            0.9000           0.9000
## Specificity                 1.0000            0.9500           0.9500
## Pos Pred Value              1.0000            0.9000           0.9000
## Neg Pred Value              1.0000            0.9500           0.9500
## Prevalence                  0.3333            0.3333           0.3333
## Detection Rate              0.3333            0.3000           0.3000
## Detection Prevalence        0.3333            0.3333           0.3333
## Balanced Accuracy           1.0000            0.9250           0.9250
```

We can see that the accuracy is 91.3%. It was a small test set, but whether or not that accuracy is sufficient depends on the problem context and is based on many other factors such as the cost of misclassification to the business.