# Lab

# Aim

To learn about Naïve Bayes classification.

To use a separate training test so that predictions for problems can be extracted.

**Before you start**

Load the following packages (download them if needed):

- naivebayes
- caret
- mlbench
- partykit

**Running Naïve Bayes within the caret package**

The naïve Bayes classifier can be called within caret's train function using "naïve_bayes" as the method.

For example, load the contact lenses dataset onto variable contactLenses, ensuring that stringsAsFactors is set to TRUE.

```
contactLenses <- read.csv("contactLenses.csv", header=T,
                 stringsAsFactors = TRUE)
```

Set up the train control to 2 repeats or 5-fold cross validation. Also setting the seed.

```
ctrl <- trainControl(method="repeatedcv", number=5, repeats=2)

set.seed(123)
```

Run the following experiment which applies Naïve Bayes to the contact lenses dataset.

```
NBmodCL <- train(contactLenses ~., contactLenses,
             method = "naive_bayes", na.action = na.omit,
             trControl = ctrl)
```

Checking the accuracy and where the errors are.

```
confusionMatrix(NBmodCL)
```

Checking the results.

```
print(NBmodCL)
```

Looking at the model (table of probabilities). Note that numeric values are labelled as following a Gaussian distribution.

```
NBmodCL$finalModel
```

You will see that the dataset has been converted to numeric by converting nominal attributes into a set of attributes (dummy variables) with binary values.

To run Naïve Bayes on nominal values, you can use the algorithm outside caret (see below).

**Running Naïve Bayes outside caret**

To run Naïve Bayes outside caret, we separate the dataset into training and testing sets. We will use 2/3 for training and 1/3 for testing

```
set.seed(123)
inTrain <- createDataPartition(y = contactLenses$contactLenses,
                               p = .67, list = FALSE)
trainingCL <- contactLenses[inTrain,]
testingCL <- contactLenses[-inTrain,]
```

We are going to predict the class for the test data, so it is not needed as input to the predict function. However, we need to know the class to be able to ascertain if the predictions are correct, so we create a new dataset without the class.

```
testingCLnoClass <- testingCL
testingCLnoClass$contactLenses <- NULL
```

We obtain the model calling function `naive_bayes()`. Note that `laplace` has to be set to a numeric value (1) as otherwise it will fail because some probabilities are zero.

```
nbCL <- naive_bayes(contactLenses ~ ., trainingCL, laplace=1)
```

To check the model properties and to obtain the table of probabilities.

```
summary(nbCL)
nbCL
```

Testing the model on the testing instances. Note that `type="prob"` means that the probabilities for each instance for each class will be outputted.

```
predict(nbCL,testingCLnoClass,type="prob")
```

This will give you probabilities of each instance belonging to each of the classes. The class with the highest probability will be assigned to the instance as follows (highest in bold for each instance).

| | hard | none | soft |
|---|---|---|---|

```
[1,] 0.05968874 0.2001236 0.7401877
[2,] 0.38492100 0.3764127 0.2386663
[3,] 0.01335292 0.7879429 0.1987042
[4,] 0.36868041 0.2884249 0.3428947
[5,] 0.01180889 0.8710394 0.1171517
[6,] 0.38492100 0.3764127 0.2386663
```

We can also predict the class on the testing set by using `type="class"` – thus we get the predicted class for each instance in the test set.

```
resCL <- predict(nbCL,testingCLnoClass,type="class")
resCL
```

The predicted (in `resCL`) and actual classes (in `testingCL$contactLenses`) can be compared using the `confusionMatrix()` function.

```
# confusion matrix with predicted class vs. real class
confusionMatrix(resCL, testingCL$contactLenses)
```

You may want to obtain the final model for the contactLenses dataset using the whole dataset, not just the training data. Compare the 2 models.

```
nbw <- naive_bayes(contactLenses ~ ., contactLenses, laplace=1)
summary(nbw)
nbw
```

**Exercises**

1. Apply Naïve Bayes within caret to the same training and testing sets as you applied in the above section using Naïve Bayes outside caret. Use the following settings

   ```
   tunegrid= expand.grid(laplace=1, usekernel=FALSE,
                         adjust=FALSE)
   NBmodW <- train(contactLenses ~., data=trainingCL,
         method = "naive_bayes",
         trControl=trainControl(method="none"),
         na.action = na.omit, tuneGrid=tuneGrid
         )
   ```

   Use the predict function within caret to test

   ```
   res <- predict(NBmodW, newdata=testingCL, type="raw")
   confusionMatrix(res, testingCL$contactLenses)
   ```

   You will find you get different results. This is because within caret the algorithm uses dummy variables to convert nominal attributes into binary ones (one-hot encoding).

Repeat the above exercise but this time use the following tuneGrid so that more options are tested.

```
tunegrid = expand.grid(usekernel = c(TRUE, FALSE),
                       laplace = c(0,0.5,1),
                       adjust = c(0.75, 1, 1.25,1.5))
```

2. One-hot encode the training and testing sets (dummyVars). Apply Naïve Bayes outside caret to the training set and check whether the probabilities in the model are now the same as the ones you obtained using caret. Test the model in the test set and compare the results to the ones you got in exercise 1.

3. Apply Naïve Bayes to other datasets that you have seen in the labs including the pimaIndiansDiabetes (in mlbench package) and the weatherPlay dataset (in the partykit package).

4. Compare the performance of this algorithm against the performance of other algorithms which you are familiar with. Ensure your comparison makes reference to the statistical significance of the difference in results.