Lab

The Caret package

Aims

- To add/delete factor levels
- To introduce the main concepts in Caret (Classification and REgression Training) – an R package which offers a set of tools for building, amongst others, classification models.

Note: before you attempt this lab you should look at the data frames section of the lab in Week 1. The first section of the lab will teach you how to manage your data frames when factor values are added or deleted.

Adding and deleting factor values from data frames

In a previous lab you saw that data frames columns can be of different types. For data mining, nominal (categorical) values need to be of factors. Thus, you learnt that when constructing data frames it was important to set stringsAsFactors = T.

Columns (attributes / features) of type factor are categorical variables where the set of possible values is fixed. This fixed set of values is called the levels. For example, let's load the contactLenses dataset and check its contests.

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

We can see that there is a column called tearProductionRate. To check the allowed values, we can use the levels() function.

```
levels(contactLenses$tearProductionRate)
```

If we merge (row bind) two data frames, the levels for a column of type factor is the union of the 2 set of levels. For example, we can create a new dataset with just one row and value "very reduced" for tearProductionRate

```
newData <- data.frame( age="young", prescription ="myope",
astigmatism="no",
tearProductionRate="very reduced",
contactLenses="no",stringsAsFactors=T)
```

The set of levels for tearProductionRate for newData is

```
levels(newData$tearProductionRate)
```

If the two datasets are merged

```
contactLenses <- rbind(contactLenses, newData)
```

```
levels(contactLenses$tearProductionRate)
```

Assume that we want to change the value of our last tearProductionRate (the one in row 25 which currently has a value of "very reduced") to "normal".

```
contactLenses$tearProductionRate[25] <- "normal"
```

Check that the value has changed. Now all our values for tearProductionRate are "normal" or "reduced" but if we check the levels you will see that the value "very reduced" is still possible/allowed. If we wanted to limit the set of allowed values for tearProductionRate to the ones currently in the data frame we need to drop unused levels.

```
contactLenses$tearProductionRate <-
        droplevels(contactLenses$tearProductionRate)


levels(contactLenses$tearProductionRate)
```

Assume that we want to change the value of tear production rate for the last row (row 25) to "very reduced". As this value is no longer one of the allowable ones, it will give an error.

```
contactLenses$tearProductionRate[25] <- "very reduced"
```

Before we change the value to "very reduced" we need to add this value to the set of levels

```
levels(contactLenses$tearProductionRate) <-
        c(levels(contactLenses$tearProductionRate),"very reduced")
```

We can now change the value in the dataset to this new value

```
contactLenses$tearProductionRate[25] <- "very reduced"
```

About Caret

R offers a multitude of packages for classification and regression model building. Caret provides a wrapper – i.e. a common interface (syntax) to the various algorithms offered.

Caret offers facilities for:

- Pre-processing: a wide range of data preparation functions, including: dealing with missing values, centering and scaling data, attribute selection (removal of attributes), one-hot encoding.
- Data splitting: training and testing sets
- Model evaluation
- Variable selection

Documentation about Caret can be found at

http://topepo.github.io/caret/index.html

Packages

Load libraries caret and rattle, as we will use them later.

```
library(caret)
library(rattle)
```

To find out which datasets are available within the datasets package, type

```
library(help = "datasets")
```

There are other datasets available in other libraries, for example WeatherPlay is available in the partykit library. To see this dataset

```
data("WeatherPlay", package = "partykit")
WeatherPlay
```

Algorithms supported by Caret

For a list of algorithms (models) supported by Caret, go to

https://topepo.github.io/caret/available-models.html

Finding details about a dataset

To view a dataset in the dataset package type the name of the dataset, e.g.

```
Co2
```

To use a dataset from other libraries, the data to be used needs to be specified. For example, to use the WeatherPlay data in package partykit

```
data("WeatherPlay", package = "partykit")
```

The data can then be viewed and/or used. For example, to view it

```
View(WeatherPlay)
```

Note that this will open a new window pane (you can just close it).

To check its number of rows and columns

```
dim(WeatherPlay)
```

The first number is the number of rows (instances) and the second is the number of columns(attributes).

To find out about the attributes in a dataset use the *attributes* function, e.g.

```
attributes(WeatherPlay)
```

To get the number of instances in a dataset use the *nrow* function, e.g.

```
nrow(WeatherPlay)
```

Caret's train() function

The `train()` function is used to build a model (e.g. a classification tree). An important part of the train function is the train control, which is ofter defined earlier. For example we may put the definition of the train control in a variable called `myTrainControl`

```
myTrainControl <- trainControl(method = "cv", number = 2)
```

You will see the meaning of trainControl in week 4. For now, you are just going to use it.

You are going to use an algorithm called `C5.0Tree` on the WeatherPlay dataset. Using the `train` () function. Note: we will learn what the various components in the example below in later sessions).

```
set.seed (123)
```

```
c5model <- train(play ~ .,
      data = WeatherPlay,
      method = "C5.0Tree",
      trControl = myTrainControl)
```

`c5model` is the name of the variable where we want to store the results. The function `train` takes several parameters:

- The first argument says that we want to predict "play", i.e. "play" is the class. The next bit " ~ ." states that we should use the rest of the attributes for prediction (i.e. outlook, temperature, humidity and windy).
- On the second line, the dataset to be used is indicated, in this case `WeatherPlay`.
- The method specifies the algorithm to be used, in this case `C5.0Tree`, which is an implementation of an algorithm very similar to C5.0. Note that this algorithm binarises nominal attributes.
- The train control is related to training and evaluation of models. We will see this later in the course.

To see the results, we can use the summary function on the final model produced (we will see more about what this means when we see evaluation).

```
summary(c5model$finalModel)
```

Note that `c5model$finalModel` means get the `finalModel` from the contents of variable `c5model`. You can see some statistics including number and percentage of errors.

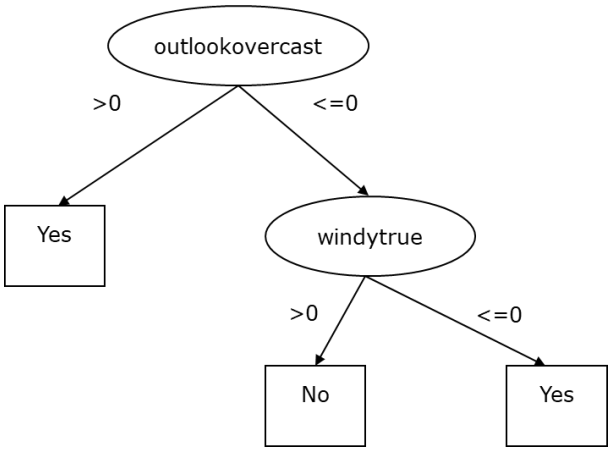
To see results, you can also use the print function on the results.

```
print(c5model$results)
```

The results show some statistics including % accuracy (percentage of correctly classified instances).

The classification tree is shown below and the corresponding graphical tree is immediately after. For classification trees, value TRUE is 1 (so ">0") and value FALSE is 0 (so "<= 0"). Note that this graphical tree is not part of the output:

```
outlookovercast > 0: yes (4)
outlookovercast <= 0:
:...windytrue <= 0: yes (6/2)
    windytrue > 0: no (4/1)
```



Also note that the algorithm has binarised attribute outlook. The first number in brackets indicates the number of instances going down that branch. If there is a second number after a "/" sign, it indicates the number of errors. For example (6/2) indicates 6 instances went down that branch and 2 of them were classified incorrectly.

A confusion matrix is also shown as follows:

```
=== Confusion Matrix ===

 a     b   <-- classified as
 8     1    (a): class yes
 2     3    (b): class no
```

The actual classes are represented in rows and the predicted classes in columns. "a" represents class yes and "b" represents class no. The main diagonal contains the number of correctly classified instances (8 yes and 3 no). The rest are

incorrectly classified. Note that the confusion matrix states where the errors are, in this case 2 instances of class no are classified as class yes and 1 instance of class yes is classified as class no.

Exercises 1

1. C5rules is similar to C5, only it outputs rules instead. Repeat the exercise above, but this time use C5rules. You will need to use `"C5.0Rules"` in the method field.

The segmentation dataset

In order to use the segmentation dataset, type

```
data(segmentationData)
```

You can now refer to this dataset by its name, i.e. `segmentationData`. Inspect this dataset, which contains information about images, including their class. This is a much bigger dataset in which an attribute ("Case") is used to specify if an instance is to be used in the training or in the testing set.

In order to use this dataset, it needs to be pre-processed a little.

Exercises 2

1. The segmentationData Cell attribute does not provide any useful information (just cell number), so it is best if it is not used for prediction. Remove it from the dataset by assigning `segmentationData$Cell` value `NULL`.

2. There is one dataset containing both the training and the testing sets. This is shown by the value of attribute `Case`. The dataset needs to be split into 2, the training set and the testing set. Split the dataset into a dataset `STrain` containing the training set and a dataset `STest` containing the testing set. Tip: to obtain `STrain` you could use

   ```
   STrain <- subset(segmentationData, Case == "Train")
   ```

3. These 2 datasets (`STrain` and `STest`) now contain a `Case` attribute which has no information (it is always train for the training set instances and test for the testing set instances, so it can be removed from both (train and test) datasets. Tip: this is similar to the first exercise in this section.

4. The dataset is ready to be used. To build a model using the training set and see the results

   ```
   control2 = trainControl(method="cv", number=5)
   set.seed(123)
   Smodel <- train(Class ~ .,
         data = STrain,
   ```

```
        method = "C5.0Tree",
        trControl = control2)

summary(Smodel$finalModel)
summary(Smodel$results)
```

How will this model perform with the unseen dataset?
To test the model on the `STest` datset use the `predict` function. You can then use the `confusionMatrix` function to check the results of this test.

```
TestRes <- predict(Smodel, newdata = STest, type="raw")
confusionMatrix(TestRes, STest$Class)
```

Note that the confusion matrix is on test results.

5. Download the following datasets from CampusMoodle and load them onto R. Use functions to inspect these datasets and find out about their attributes.

   - contact Lenses with class `contactLenses`
   - diabetes with class `class`

6. Apply C5.0Trees and C5.0Rules to these datasets and compare the results. Did you get any warning messages?
   Which algorithm is best?

7. Apply J48 to the above datasets and compare the results. J48 is a different implementation of a decision tree builder based on C4.5 (so similar to C5.0). Is it any better than the previous algorithms? Note that you can get a tree by using function `plot()`. For example, if your model is called `j48Model` you can use

   ```
   plot(j48Model$finalModel)
   ```

8. Apply CART to the *diabetes* dataset. CART is another algorithm for implementing decision trees which uses *Gini index* instead of *information gain*. To apply it, use method "`rpart`". Assuming you leave the results in a variable called `rPartModel` the resulting tree can be plotted using

   ```
   fancyRpartPlot(rPartModel$finalModel)
   ```