Lab

Decision tree options with C5.0, J48 and rpart

Aims

- To introduce decision tree algorithm options, e.g. winnowing, minimum number of cases, confidence factors , post-pruning and the complexity parameter.
- To introduce missing values handling by algorithms.
- To check which variables are important for a tree model.

Note: this lab does NOT deal with algorithm evaluation.

Other: remember to set the seed each time you run code so that your experiments are reproducible, e.g. to set the seed to 123 use `set.seed(123).`

C5.0Tree, J48 and rpart

In the previous lab, you saw how to use C5.0 (tree and rules options) J48 and rpart's part of the Caret package (http://caret.r-forge.r-project.org/) which offers a consistent way of using a wide variety of algorithms. For C5.0, in this lab we focus on `C5.0Tree`, but you may wish to try the options with the rules version of C5.0, i.e. `C5.0Rules` if you prefer that output. This lab also covers options for rpart

Packages

Load (install if needed) the following packages

- caret
- C50
- RWeka
- rattle
- partykit
- datasets

Using C5.0Tree within Caret

In the previous lab, you saw how to use C5.0Tree within caret.

Recall that to use the WeatherPlay dataset within caret we could use:

```
set.seed(123)
c5model <- train(play ~ .,
        data = WeatherPlay,
```

```
        method = "C5.0Tree",
        trControl = trainControl(method = "cv", number = 2))

summary(c5model$finalModel)
```

The output model looks very different to the one we have seen in the class as attributes have been converted to binary and a binary tree used.

There are several parameters which can be changed using C5.0Tree from within Caret, setting parameter `control` using function `C5.0Control()` with one or more of the following:

- **CF**, which is the pruning confidence factor. The smaller C is, the more C5.0 will prune the tree. By default, CF=0.25.
- **minCases** is the minimum number of instances in at least 2 of the partitions created by splitting the data at a node. The larger the value of this parameter, the more instances have to go through the 2 most popular branches from a node before branching is successful.
- **noGlobalPruning** – if set to true, no post-pruning of the tree will take place.
- **winnow** – if set to TRUE, it performs winnowing, i.e. feature selection.

You will see the use of the above control parameters in the remainder of this lab.

Download file labor.csv from CampusMoodle. Inspect the file.

To load it use

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

You can see the content of the loaded dataset as before, i.e. you can type

```
view(labor)
```

The class is *Class*.

Dealing with missing values within an algorithm (applicable to several algorithms)

Try to run this dataset. **You will find that it gives you an error because the dataset contains lots of missing values (NA).** To be able to use this dataset, missing values must be dealt with.

Exercises

1. Apply C5.0Tree to the labor dataset. **Expect an error** because of the very numerous missing values.
2. Update your program to include `na.action= na.pass` in the train function. This tells caret to pass the missing values as they are to C5.0Tree, and the algorithm will deal with them.

In the previous exercise, you may have used

```
control1 <- trainControl(method = "cv", number = 5)
set.seed(123)

c5Tree <- train(Class ~ .,
      data = labor,
      method = "C5.0Tree",
      na.action= na.pass,
      trControl = control1))
summary(c5Tree$finalModel)
print(c5Tree$results)
```

There are other settings which can be used with datasets containing missing values. One such setting is `na.action= na.omit` which disregards instances containing missing values. Try this setting with the labor dataset. **It will give you an error** because this action leaves virtually nothing in the dataset. In module CMM535 you will learn how to impute values.

The CF option - pruning confidence factor

CF controls the pruning confidence. The default is 0.25. The larger the value of CF, the less C5.0Tree will prune. Try increasing the pruning confidence, e.g.

```
set.seed(123)
c5Tree <- train(Class ~ .,
      data = labor,
      method = "C5.0Tree",
      na.action= na.pass,
      trControl = control1,
      control = C5.0Control(CF = 0.35 ))
summary(c5Tree$finalModel)
print(c5Tree$results)
```

Pay attention to the size of the tree and the attributes used to build it. You may not see any difference for this particular dataset. Try instead setting the pruning confidence factor to 0.15. You may see heavier pruning carried out.
Repeat the commands above, but this time set the pruning confidence to 0.1.
Repeat the above but this time set the confidence factor to 0.05. Are the results any good?

The noGlobalPruning option

With this option set to TRUE, no post-pruning is undertaken. Try the following:

```
set.seed(123)
c5Tree <- train(Class ~ .,
      data = labor,
      method = "C5.0Tree",
      na.action= na.pass,
      trControl = control1,
  control = C5.0Control(noGlobalPruning = TRUE))
summary(c5Tree$finalModel)
print(c5Tree$results)
```

Has the option made any difference to the results?

The minCases option – minimum number of instances

This option does not split nodes where the minimum number of instances (objects) per leaf is not at least the given number for 2 branches. By default it is 2.

When we run C5.0Tree with the WeatherPlay dataset with default parameters, we obtained a tree as follows:

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

We can see that this tree contains at least 4 instances in each branch (the first number appearing within brackets, with the 2nd number being the number of errors). What would happen if we allowed the algorithm to branch even if it resulted in branches with only one instance? Try

```
set.seed(123)
c5model <- train(play ~ .,
       data = WeatherPlay,
       method = "C5.0Tree",
       trControl = control1,
       control = C5.0Control(minCases=1))

summary(c5model$finalModel)
print(c5model$results)
```

You should see a larger tree that contains more levels. One of the branches contains only one instance.
Check whether the accuracy of the tree is better or worse. Do you think that the tree overfits the data?

Attribute selection - the winnow option

If winnow is set to true, attribute selection is performed. Try

```
set.seed(123)
c5model <- train(Class ~ .,
       data = labor,
       method = "C5.0Tree",
       na.action = na.pass,
       trControl = control1,
  control = C5.0Control(winnow = TRUE))

summary(c5model$finalModel)
print(c5model$results)
```

Attribute/variable importance

You can use functions predictors() and varImp() on your C5.0Tree models to check the variables used and the importance assigned to each variable.

For example, if your model is called `c5model`

```
predictors(c5model)
varImp(c5model)
```

Exercises

3. Experiment with the minCases parameter with the labor dataset. Compare the tree with the one you get with the default value for minCases with the one you get with 5 as the minimum number of objects. Repeat the experiment using 1 as the value. Is it always the case that a lower number for minCases results in a larger tree? Use the attribute functions to check the attributes considered and their importance. Pay attention to the size of the tree and the attributes used to build it.

4. Use C5.0Tree in the diabetes, the iris and the contact lenses datasets, using various parameter settings. Note that to read the contactLenses and the diabetes datasets you should include the option `stringsAsFactors= TRUE` in function `read.csv` parameter list. This is so that the resulting dataframe uses datatype factor for columns containing strings (instead of datatype character). For example

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

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

   You can obtain the iris dataset from the dataset package.

   Use the attribute functions to check the attributes considered and their importance. Pay attention to the size of the tree and the attributes used to build it.

Using J48 within Caret

In the previous lab, you saw how to use J48 within caret.

Recall that to use the WeatherPlay dataset within caret we could use:

```
set.seed(123)
j48Tree <- train(play ~ .,
        data = WeatherPlay,
        method = "J48",
        trControl = control1)

summary(j48Tree$finalModel)
print(j48Tree$results)
```

```
plot(j48Tree$finalModel)
```

The output model looks very different to the one we have seen in the class as attributes have been converted to binary and a binary tree used.

There are two parameters which can be changed using J48 from within Caret:

- **C**, which is the pruning confidence. The smaller C is, the more J48 will prune the tree. By default, C=0.25.
- **M** is the minimum number of objects. The larger the value of M, the more instances have to go through the 2 most popular branches from a node before branching is successful.

These options are explained in more detail below.

Here is an example of the use of these parameters within Caret with M=3 and C=0.25 (default value).

```
set.seed(123)   # ensure reproducibility of results

j48Tree2 <- train(play ~ .,
        data = WeatherPlay,
        method = "J48",
        trControl = control1,
        tuneGrid = expand.grid(M = 3, C = 0.25))

summary(j48Tree2$finalModel)
print(j48Tree2$results)

plot(j48Tree2$finalModel)
```

To run the WeatherPlay dataset with J48 with several values of M and one for C you can try

```
set.seed(123)   # ensure reproducibility of results

j48Tree3 <- train(play ~ .,
        data = WeatherPlay,
        method = "J48",
        trControl = control1,
        tuneGrid = expand.grid(M = c(2,3,4,5), C = 0.25))

summary(j48Tree3$finalModel)
print(j48Tree3$results)

plot(j48Tree3$finalModel)
```

To check the final value used for M type

```
j48Tree3
```

You will get output like the following

```
C4.5-like Trees

14 samples
 4 predictor
```

```
 2 classes: 'yes', 'no'

No pre-processing
Resampling: Cross-Validated (5 fold)
Summary of sample sizes: 12, 11, 11, 11, 11
Resampling results across tuning parameters:

  M  Accuracy   Kappa
  2  0.5666667  0.06
  3  0.6333333  0.18
  4  0.6333333  0.00
  5  0.6333333  0.00

Tuning parameter 'C' was held constant at a value of 0.25
Accuracy was used to select the optimal model using  the largest val
ue.
The final values used for the model were C = 0.25 and M = 3.
```

Note that the tree that you obtain will be the one for the best setting, i.e. M=3.

Exercises

5. Use J48 within Caret in the diabetes, the iris and the contact lenses datasets, using various parameter settings. Try to use several C and M parameters at the same time. Tip – you will need to use a vector with each option.
6. Try to use J48 within Caret with the labor dataset.As it contains far too many missing values, try to use the option of letting J48 handle the missing values. Can J48 do this?

Using rpart (CART) within caret

Algorithm rpart has one parameter that can be tuned, the complexity parameter **cp**. It indicates the minimum improvement required at each node which is based on the number of misclassifications, the number of splits and a penalty parameter (lambda). The default value is cp=0.01. Lower values lead to less pruning.
For example, if you apply rpart to the diabetes dataset with default values you may use

```
set.seed(123)
rPartMod <- train(class ~ .,
                  data = diabetes,
                  method = "rpart",
                  trControl = control1)
summary(rPartMod$finalModel)
print(rPartMod$results)
fancyRpartPlot(rPartMod$finalModel)
```

You can change the complexity parameter to check various limits 0.01, 0.005 and 0.001, as follows:

```
set.seed(123)
rPartMod <- train(class ~ .,
```

```
                data = diabetes,
                method = "rpart",
                tuneGrid = expand.grid(cp=c(0.01, 0.005, 0.001)),
                trControl = control1)
summary(rPartMod$finalModel)
print(rPartMod$results)
fancyRpartPlot(rPartMod$finalModel)
```

Exercise

7. Use rpart within Caret in the labor, the iris and the contact lenses datasets, using various parameter settings. Try to use several cp values parameters at the same time. For the labor dataset, try to use the option of letting rpart handle the missing values. Can rpart do this?