Lab

Evaluation

Aim

To introduce evaluation methods.

Packages

You need at least the following packages

- caret
- mlbench
- partykit
- rattle

Experimentation

The `trainControl` function enables the setting of different evaluation methods for caret's `train` model building function. It will include stratification where required.

`trainControl` takes several arguments including:

- `method`: values include
  - "boot": bootstrap
  - "boot632": a second bootstrap method
  - "optimism_boot": a 3rd bootstrap method
  - "boot_all": a method which combines the 3 boostrap methods above.
  - "cv": cross validation
  - "repeatedcv": repeated cross validation
  - "LOOCV": leave one out
  - "LGOCV": leave group out – randomly leaves a group of instances out of the training set. This is the holdout method.
  - "oob": out of bag – [used, for example with algorithms based on randomisation]. Will not use this in this lab.
  - "none": fits one model to the entire training set.
- `number:` refers to either the number of folds (for k-fold cross validation) or number of resampling iterations (for bootstrap and leave group out). The default is 10 for "cv" and 25 for other methods.
- `repeats`: used for cross validation, it indicates the number of sets of cross validation to be executed. E.g. if `repeats` = 3 and `number` = 10, it would be 3 sets of 10-fold cross validations.
- `verboseIter`: if set to TRUE, run information is outputted.

- p: the training percentage for "LGOCV".

The trainControl can be set up within the train function using parameter trControl. For example

```
# defining the evaluation method:4-fold cross validation repeated 3 times.

ctrl <- trainControl(method = "repeatedcv",
               number = 4,
               repeats = 3,
               verboseIter=TRUE)

# ensuring reproducibility by setting the seed
set.seed(123)

# running C5.0Tree on the iris dataset.
# Note the use of trControl with the previously defined trainControl.
c5treemod <- train(Species ~ ., data = iris,
             method = "C5.0Tree",
             tuneLength = 12,
             trControl = ctrl)

# run values
c5treemod

# checking results
summary(c5treemod$finalModel)
```

The tuneLength parameter above tells caret to try 12 different settings (the settings will depend on the algorithm used as the parameters are algorithm dependent).

Note that setting the seed to the same value (the actual value does not matter) just before we run each of the algorithms on the data ensures the sampling is identical for all algorithms. It also ensures that the experiments can be reproduced, i.e. if the same experiments are run again, the same results are obtained.

Exercises

1. Run C5.0Tree with the iris dataset as above, i.e. 3 repeats of 4-fold cross validation.
2. Check the confusion matrix. Does it use rows or columns for actual class values?
3. Which class distinctions are most difficult?
4. How big is the classification tree? Number of nodes, leaves and levels.
5. The dataset has 4 attributes (excluding the class). State the 2 most important attributes for identifying irises.
6. What are the defining characteristics of iris setosa?

The `metric` parameter in `train`

Function train has a parameter called metric. This parameter indicates the metric used in order to select the best model (the results will give both metrics). For classification tasks, the metric can be set to either "Accuracy" or "Kappa". The default value is "Accuracy".

For example, try

```
ctrl <- trainControl(method = "repeatedcv",
                     number = 4,
                     repeats = 3,
                     verboseIter=TRUE)

# ensuring reproducibility by setting the seed
set.seed(123)

# running C5.0Tree on the iris dataset.
c5treemod <- train(Species ~ ., data = iris,
                   method = "C5.0Tree",
                   metric = "Accuracy",
                   trControl = ctrl)
# run values
c5treemod

# checking results
summary(c5treemod$finalModel)
```

Exercise

7. Run the C5.0Tree on the iris dataset (as above) but use Kappa as the performance metric.

Comparing performance on test sets

`confusionMatrix.train()` can be used to obtain a confusion matrix which estimates the results of the evaluation using the resampling procedure.

`norm="overall"` gives % figures over all repetitions
`norm="none"` gives figures over all runs (added)
`norm="average"` gives figures averaged over the number of samples.

Compare the following 3 confusion matrices:

```
# none - numbers obtained over independent runs added.
# It should add up number of repetitions times dataset size.
# If using 10-fold cross validation 3 times with the
# iris dataset (150 instances)
# sum(all numbers in confusionMatrix) = 150 * 3 = 450
confusionMatrix.train(c5treemod, norm="none")

# overall - figures are percentages of the results obtained over
# all the runs.
# If using 10-fold cross validation 3 times with the
# iris dataset (150 instances)
# sum(all numbers in confusionMatrix) = 100 [i.e. 100%]
confusionMatrix.train(c5treemod, norm ="overall")

#averaged over number of samples
confusionMatrix.train(c5treemod, norm="average")
```

<u>NOT to be used with leave one out, bootstrap or no repeats. Use print() or summary() instead</u>

Exercise

8. Compare the confusion matrices you obtained with the confusionMatrix.train function to the ones you obtained using the summary function on the final model. Do they use the same items in columns/rows?

The print function can be used with leave one out as follows:

```
# Using leave one out

ctrl <- trainControl(method = "LOOCV")
set.seed(123)
c5weathermod <- train(play ~ ., data = WeatherPlay,
             method = "C5.0Tree",
             trControl = ctrl)
print(c5weathermod)
```

Exercise

9. Compare the performance of the following algorithms:

   - C5.0Tree
   - C5.0Rules
   - rpart
   - rpart2
   - ctree (an algorithm in library mlbench)
   - ctree2 (also in mlbench)

   Note: you may wish to set the verboseIter option to FALSE.
   Note2: in a previous lab you learnt how to print trees produced by algorithm rpart using function `fancyRpartPlot()` (you will need library rattle). For some trees (not the ones produced by C5.0Tree or rpart/rpart2), you can use function `plot()` to print the "`finalModel`". E.g. if the model is put in variable `myModel`

   ```
   plot(myModel$finalModel)
   ```

   In order to compare them, you will need to use an evaluation method (e.g. cross validation, bootstrap, leave one out, holdout) for each of the datasets you test. For holdout, use `method = "LOGCV"` with `p` = percentage of instances on training set). Use each of the following datasets:

   a. iris
   b. contact Lenses
   c. Glass [In mlbench package. Class is "Type". To use run `data(Glass)`]
   d. LetterRecognition [In mlbench package. Class is "lettr". To use call `data(LetterRecognition)`]

Note that the comparisons above do NOT take into account statistical significance.

OPTIONAL: that there are ways to do this that minimise the lines of R that you utilise. You can use a "for" loop with a "list" to first list all the models and then loop over them. You need to be careful about what you display, but it can be a quick way to decide upon the best model.

Measuring statistical significance

We would like to compare several algorithms' results for statistical significance.

It is important to ***set the seed*** to the same specific value just before we run each of the algorithms on the data so the sampling is identical for all algorithms. It also ensures that the experiments can be reproduced, i.e. if the same experiments are run again, the same results are obtained.

```
# load the diabetes dataset
data(PimaIndiansDiabetes)

# prepare training method.

ctrl <- trainControl(method="repeatedcv", number=10, repeats=3)


#use several algorithms: CART, C5.0Rules, C5.0Tree.

# CART

# setting the seed makes the experiment reproducible.

set.seed(123)

mod.cart <- train(diabetes~., data=PimaIndiansDiabetes, method="rpart",
trControl=ctrl)

# C5.0 Rules

set.seed(123)

mod.c5rules <- train(diabetes~., data=PimaIndiansDiabetes,
method="C5.0Rules", trControl=ctrl)

# C5.0 Trees

set.seed(123)

mod.c5t <- train(diabetes~., data=PimaIndiansDiabetes, method="C5.0Tree",
trControl=ctrl)
```

Collecting results for comparison: Once the models for all the algorithms have been obtained, the results can be collected in a list.

```
# collect resamples

results <- resamples(list(CART=mod.cart, c5Rules= mod.c5rules, c5tree=
mod.c5t))

# show accuracy and kappa details
```

```
results
```

```
summary(results)
```

Note that "CART", "c5Rules" and "c5tree" are the names chosen for the models. These names appear when the results are shown. Other names such as "CART-algorithm", "C5-rules" and "C5-tree" could have been used.

Comparison of results: Now, a plot can be obtained which shows the models' accuracy and kappa statistic as well as the error bars for a given ***confidence level*** (e.g. 0.95).

```
scales <- list(x=list(relation="free"), y=list(relation= "free"))
dotplot(results, scales=scales, conf.level = 0.95)
```

If intervals do not overlap, the difference in performance is statistically significant and one algorithm's performance can be said to be better than another one. If intervals overlap, the algorithms' performance cannot be deemed to be different.

Other options include plotting boxplots with

```
bwplot(results)
```

and getting a scatter plot matrix showing accuracies for each of the runs, for each algorithm.

```
splom(results)
```

Exercises

10. Write R code to compare the performance of C5.0Tree, C5.0Rules and rpart and ctree with a confidence level of 99%. Are the intervals bigger or smaller than the ones you obtained earlier for the first 3 algorithms?

11. Find the confidence level at which at least 2 of the algorithms above can be said to have a different performance in terms of % accuracy.