Classification Practical Work

GUIDELINES

For this practical work, the following guidelines apply:

1. The practical work must be completed in groups of two people.
2. After the completion of the exercise, a report should be written addressing all the questions raised in the following paragraphs. Additionally, the students should also provide the source code of the R script used to solve the practical work.
3. Both files, a PDF file with the report and a text file with .R extension with the script developed, should be sent electronically to the following e-mail address: atorre@fi.upm.es
4. The first page of the report must contain the names and IDs of the authors of the exercise. The same holds for the R script: a couple of comment lines should be added at the beginning of the file with that information.
5. The exercise must be solved using R, exclusively.
6. The deadline for this practical work is 2015/11/04, 11:55pm. Both files (the report and the source code) should have been sent to the aforementioned e-mail address before this time.
7. The students must refrain from copying the solution to the practical work. Any detected copy will imply that ALL the students involved in the copy will get a 0 in this practical work. If this behavior is repeated over time, such students might directly get a fail grade in the whole course.

Modeling the sinking of the RMS Titanic

According to Wikipedia:

“The sinking of the RMS Titanic occurred on the night of 14 April through to the morning of 15 April 1912 in the north Atlantic Ocean, four days into the ship's maiden voyage from Southampton to New York City. The largest passenger liner in service at the time, Titanic had an estimated 2,224 people on board when she struck an iceberg at around 23:40 (ship's time) on Sunday, 14 April 1912. Her sinking two hours and forty minutes later at 02:20 (05:18 GMT) on Monday, 15 April resulted in the deaths of more than 1,500 people, which made it one of the deadliest peacetime maritime disasters in history.”

Our task in this exercise will be two-fold: first, we will try to build an explicative model based on decision trees that predicts whether a passenger of the RMS Titanic survived or not; second, we will try to extend the previous model by using random forests to obtain a prediction as accurate as possible. Throughout this exercise we will learn how to build training and test sets from our data, how to run a parameter tuning to select the most promising values for the parameters of our algorithms and how to compare the performance of multiple classifiers. We will use several R packages that will be briefly described through this document.

Getting the data

The dataset can be downloaded from the course's web site. It is a ZIP file containing two other files: titanic.csv, which contains the dataset itself as a comma separated values file, and titanic_info.txt, which describes the meaning of the variables included in the dataset. For example, it explains that the class label, the variable to predict, is called *"Survived"* and informs whether a passenger died in the accident or not. The students should read this info file carefully so that they are able to interpret the models obtained in the subsequent parts of the exercise.

Cleaning the data and building train and test sets

The first step that should always be conducted when trying to learn a classification model is to clean the dataset. First, only the meaningful variables of the dataset should be kept (i.e. literals, strings, ids, etc. should be discarded, as they will not contribute to a classification model that is able to generalize to new instances). In this dataset there are several of these variables that should be removed. Second, those entries for which one of the variables is missing should be either imputed or discarded. As data imputation is out of the scope of this exercise, rows with missing values can be safely removed.

Note: In order to remove unnecessary variables, the [,] operator or the *subset()* function can be used. To filter out rows with missing values, the *na.omit()* function might be considered.

Question 1: Which are the variables that are discarded and why?

Once the dataset is clean, it is time to build separate train and test sets, so that we can validate the model obtained with the training set with a test set made up of completely unseen instances. Normally, these sets are made by sampling 70-80% of the instances into the train set and the remaining instances into the test set. This sampling can be manually done or, better, we can use some of the functions provided by the ***caret*** package (that we will have to install if it is not already installed):

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

This package provides a large set of functions for pre-processing, data splitting and model tuning that will make our life much easier when dealing with these problems.

For the data splitting issue, you might consider using the *createDataPartition()* function:

http://topepo.github.io/caret/training.html#example

Training a single decision tree

Once the dataset is clean, we can learn our first tree. For this purpose, we will use the *rpart()* function, which implements the CART algorithm. This function is provided by the ***rpart*** package (which we must install if it is not already installed) and, as most of the supervised learning functions, has the following prototype:

rpart(formula, data, params…)

To know how exactly a formula is written, check the documentation of the function.

The result of calling the *rpart()* function should be stored in a variable so that we can further analyze its results.

Question 2: Plot the tree with the *prp()* function (package ***rpart.plot*** that we must install if it is not already installed). Which are the most important variables according to the tree? Does the tree change if we rerun the *rpart()* function? Why?

Now let us check the accuracy of the model. For this purpose, we will use the *confusionMatrix()* function from the caret package:

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

Question 3: Which are the performance values (accuracy, sensitivity, specificity, etc.) of the learned model on the testing subset?

Question 4: We have learned a decision tree model with the CART algorithm. Which are the values for the parameters of this algorithm (*minbucket, minsplit, complexity parameter*, *cost*, etc.)? (***CLUE***: check the *rpart()* function documentation…)

Question 5: Try different combinations of values for some of the parameters (decreasing *minsplit, minbucket, cp* and *cost* values, for example) and check the performance of each combination on the testing subset. How does this performance change? How do the obtained trees change? Is there any relationship between the parameters values and the shape of the trees?

Automatically tuning the parameters of a decision tree

Having to manually tune the parameters of an algorithm is not the type of task we should invest our time on. For this reason, there are some tools that automate this process. In the package caret there is a function called *train()* that makes most of the work for us. Check the documentation and examples of the function for details on how to use it (Basic Parameter Tuning section):

http://topepo.github.io/caret/training.html#tune

As you may have noticed, the *train()* function needs the method (algorithm) to be trained to be specified. In the following link there is a list of methods and their code names for the method argument of the *train()* function, as well as the list of parameters that may be tuned:

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

To make runs faster, configure the *trainControl()*function as in the provided example but **without** the repeats option, so that the 10-fold Cross Validation is only performed once.

Question 6: Which are the combinations of parameters values tested by the *train()* function? Are there any changes in the performance of the algorithm when different combinations of values are used (according to the results of the cross validation)?

Question 7: Which is the final combination of parameters values used? Which is the shape of the tree trained with this automatic tuning function? (**CLUE**: if the result of calling to the *train()* function is stored in an object called *rpartFit*, check the *rpartFit$finalModel* object).

Question 8: Plot the result of calling the *train()* function with the *plot()* function. What does this plot represent?

As you may have observed, the *train()* function automatically selects the combinations of parameters values to be tested. However, these default values are not always optimal and thus there is a function that allows us to define the combinations of parameters values that we want to explore. This function is called *expand.grid()* and you may find its documentation and examples in the following link (Alternate Tuning Grids section):

http://topepo.github.io/caret/training.html#grids

Rerun the *train()* function for the *rpart* algorithm and try the combinations for the following two parameters with these values (note that you should select the appropriate *rpart* method from the list of supported methods that allows you to tune these two parameters simultaneously):

Cost = c(1, 2, 3, 5, 10)
cp = c(0, 0.01, 0.02, 0.04, 0.07, 0.10)

Question 9: Answer Question 6 again but now with the results of this new run.

Question 10: Analogously to Question 9, respond to Question 7 with the results of this new run.

Question 11: Answer Question 8 again but now with the results of this new run.

Training a Random Forest (with default and custom parameters values)

The *train()* function allows us also to train random forests classifiers. If the ***randomForest*** package is not installed, we can do two things: install it manually or wait for the train function to ask for its installation (it detects packages needed but not available and installs them automatically). For this algorithm we will repeat the same approach as in the previous section: we will first train the model with the default combination of parameters values and then we will try a different subset of values for one of its most important parameters, the *mtry* parameter, which decides the size of the random subset of variables used for splitting at each node of each tree. Note that the call to the *train()* function will be almost identical for this algorithm: we need only to change the method used and pass an additional argument to the *train()* function with the number of trees that we want in our forest (*ntree=2000* in our case).

For the first part of the random forest exercise (tuning with default combinations of parameters values):

Question 12: Answer Question 6 again but now with the results of this new run.

Question 13: Which is the final combination of parameters values used?

Question 14: Answer Question 8 again but now with the results of this new run.

Question 15: Plot the importance of each variable for the model with function *VarImPlot()* from package caret. Which are the most relevant variables according to their mean decrease of the *Gini* index? Are these variables the ones selected when we built our decision trees?

For the second part of the random forest exercise (tuning with custom combinations of parameters values):

Use the following values for the *mtry* parameter:

mtry = c(2, 3, 4, 5, 6, 7, 8, 9)

Question 16: Answer Question 6 again but now with the results of this new run.

Question 17: Answer Question 13 again but now with the results of this new run.

Question 18: Answer Question 8 again but now with the results of this new run.

Question 19: Answer Question 15 again but now with the results of this new run.

Once we have trained all our models based on decision trees and random forests:

Question 20: Which is the difference in performance with regards to the testing subset between the best decision tree model and the best random forest model?