Decision Trees and Random Forests

Lab 4: Decision Tree and Random Forests

- Principles of a supervised learning process
- Decision Trees
- Random Forests

Please send comment to (version 1.0 – July 28, 2016):

Romain Billot *romain.billot@telecom-bretagne.eu*
Yannis Haralambous *yannis.haralambous@telecom-bretagne.eu*
Philippe Lenca *philippe.lenca@telecom-bretagne.eu*
Sorin Moga *sorin.moga@telecom-bretagne.eu*

1 Data presentation (as from Kaggle.com)

The sinking of the RMS Titanic is one of the most infamous shipwrecks in history. On April 15, 1912, during her maiden voyage, the Titanic sank after colliding with an iceberg, killing 1502 out of 2224 passengers and crew. This sensational tragedy shocked the international community and led to better safety regulations for ships.

One of the reasons that the shipwreck led to such loss of life was that there were not enough lifeboats for the passengers and crew. Although there was some element of luck involved in surviving the sinking, some groups of people were more likely to survive than others, such as women, children, and the upper-class.

In this challenge, we ask you to complete the analysis of what sorts of people were likely to survive. In particular, we ask you to apply the tools of machine learning to predict which passengers survived the tragedy.

–The Titanic dataset–
SPECIAL NOTES:

Slot (not variable)	Meaning
`transactionInfo`	Data frame with vectors of the same length as the number of transactions
`survival`	Survival (0 = No; 1 = Yes)
`pclass`	Passenger Class (1 = 1st; 2 = 2nd; 3 = 3rd)
`name`	Name
`sex`	Sex
`age`	Age
`sibsp`	Number of Siblings/Spouses Aboard
`parch`	Number of Parents/Children Aboard
`ticket`	Ticket Number
`fare`	Passenger Fare
`cabin`	Cabin
`embarked`	Port of Embarkation (C = Cherbourg; Q = Queenstown; S = Southampton)
`data`	Binary incidence matrix that indicates which item labels appear in every transaction

Table 1: Structure of the titanic dataset

- Pclass is a proxy for socio-economic status (SES): 1st Upper; 2nd Middle; 3rd Lower;

- Age is in Years; Fractional if Age less than One (1). If the Age is Estimated, it is in the form xx.5

With respect to the family relation variables (i.e. sibsp and parch) some relations were ignored. The following are the definitions used for sibsp and parch.

- Sibling: Brother, Sister, Stepbrother, or Stepsister of Passenger Aboard Titanic,
- Spouse: Husband or Wife of Passenger Aboard Titanic (Mistresses and Fiances Ignored),
- Parent: Mother or Father of Passenger Aboard Titanic,
- Child: Son, Daughter, Stepson, or Stepdaughter of Passenger Aboard Titanic.

Other family relatives excluded from this study include cousins, nephews/nieces, aunts/uncles, and in-laws. Some children travelled only with a nanny, therefore parch=0 for them. As well, some travelled with very close friends or neighbors in a village, however, the definitions do not support such relations.

Question 1

Load, understand and describe carefully the train.csv and test.csv files. Look at the number of people who survived. Create a new column in the test set that will contain your prediction that everyone dies

Solution 1

The train set contains 891 observations (passengers) with 12 variables while the test set is small with 418 passengers. The principle of a machine learning process is to learn a model on the train set and then to test on the test set, hence testing with a dataset which has not been used in the learning process. With respect to the proportions of survival, we see that 61% of the passengers died.

R code:

```
train <- read.csv("train.csv")
test <- read.csv("test.csv")
str(train)
table(train$Survived)
prop.table(table(train$Survived))
test$Survived <- rep(0, 418)
```

Question 2

Then we can analyze gender pattern. The disaster was famous for saving "women and children first", so let's take a look at the Sex and Age variables to see if any patterns are evident. We'll start with the gender of the passengers. After reloading the data into R, take a look at the summary of this variable. Build a cross table that shows the proportions of males and females crossed with the survival variable.

Solution 2

We can see that the majority of females aboard survived, and a very low percentage of males did survive.

R code:

```
summary(train$Sex)
prop.table(table(train$Sex, train$Survived))
prop.table(table(train$Sex, train$Survived), 1)
```

Question 3

Now we are going to work on the age. Look at the age distribution with descriptive statistics. Then create a new variable `Child` which takes one is the passenger is below 18 years old, 0 otherwise. Add this variable to the train dataset. Then create a table with both gender and age to see the survival proportions for different subsets (help: use the *aggregate* function

Solution 3

We have first assigned a 0 to the `Child` variable to all passengers, beginning with the assumption that they were an adult, and then overwriting the value for passengers below the age of 18. If you click on the train object in the explorer, you will see that any passengers with an age of NA have been assigned a zero, this is because the NA will fail any boolean test. This is what we wanted though, since we had decided to use the average age, which was an adult. The *aggregate* command takes a formula with the target variable on the left hand side of the tilde symbol and the variables to subset over on the right. We then tell it which dataframe to look at with the data argument, and finally what function to apply to these subsets. Here it is quite technical since we want the proportions and hence we have to write and apply (through the FUN argument) a function that takes the subset vector as input and applies both the sum and length commands to it, and then does the division to give us a proportion.

R code:

```
summary(train$Age)
train$Child <- 0
train$Child[train$Age < 18] <- 1
aggregate(Survived ~ Child + Sex, data=train, FUN=function(x) {sum(x)/length(x)})
```

Question 4

Next we focus on the `fare` which is a continuous variable that needs to be reduced to something that can be easily tabulated. Let's bin the fares into less than 10 dollars, between 10 and 20, 20 to 30 and more than 30 and store it to a new variable. Use the *aggregate* function again in order to highlight some relevant crossed proportions.

Solution 4

While the majority of males, regardless of class or fare still don't do so well, we notice that most of the class 3 women who paid more than 20 dollars for their ticket actually also miss out on a lifeboat.
It's a little hard to imagine why someone in third class with an expensive ticket would be worse off in the accident, but perhaps those more expensive cabins were located close to the iceberg impact site, or further from exit stairs?

R code:

```
train$Fare2 <- '30+'
train$Fare2[train$Fare < 30 & train$Fare >= 20] <- '20-30'
train$Fare2[train$Fare < 20 & train$Fare >= 10] <- '10-20'
train$Fare2[train$Fare < 10] <- '<10'
aggregate(Survived ~ Fare2 + Pclass + Sex, data=train, FUN=function(x) {sum(x)/length(x)})
```

2 Decision Trees

In the first part of the lab, we have tried to find subsets of the passengers that were more, or less, likely to survive the disaster. To find more fine-grained subsets with predictive ability would require a lot of

time to adjust our bin sizes and look at the interaction of many different variables. In this section, we are going to illustrate, step by step, the implementation of a decision tree algorithm. Decision trees have a number of advantages. They are what's known as a glass-box model, after the model has found the patterns in the data you can see exactly what decisions will be made for unseen data that you want to predict. They are also intuitive and can be read by people with little experience in machine learning after a brief explanation. Finally, they are the basis for some of the most powerful and popular machine learning algorithms. Conceptually, the algorithm starts with all of the data at the root node (drawn at the top) and scans all of the variables for the best one to split on. The way it measures this is to make the split on the variable that results in the most pure nodes below it, i.e with either the most 1's or the most 0'9s in the resulting buckets.

Question 5

Install or/and load the *rpart*, *rattle*, *rpart.plot*, *RColorBrewer* packages. Draw a first decision tree for only the gender variable `Sex` by using the *rpart*, and *fancyRpartPlot* functions.

Solution 5

The root node, at the top, shows first insights, 62% of passengers died, while 38% survived. The number above these proportions indicates the way that the node is voting (recall we decided at this top level that everyone would die, or be coded as zero) and the number below indicates the proportion of the population that resides in this node, or bucket (here at the top level it is everyone, 100%).
So far, so good. Now let's travel down the tree branches to the next nodes down the tree. If the passenger was a male, indicated by the boolean choice below the node, you move left, and if female, right. The survival proportions exactly match those we found with the proportion tables. If the passenger was male, only 19% survive, so the bucket votes that everyone here (65% of passengers) perish, while the female bucket votes in the opposite manner, most of them survive as we saw before. In fact, the above decision tree is an exact representation of the gender model. The final nodes at the bottom of the decision tree are known as terminal nodes, or sometimes as leaf nodes. After all the boolean choices have been made for a given passenger, they will end up in one of the leaf nodes, and the majority vote of all passengers in that bucket determine how we will predict for new passengers with unknown fates.

R code:

```
library(rpart)
library(rattle)
library(rpart.plot)
library(RColorBrewer)
fit <- rpart(Survived ~ Sex, data=train, method="class")
fancyRpartPlot(fit)
```

Question 6

Build a more complex tree with a refined model composed of all possible variables. For that purpose, just change the model formula in the *rpart* function.

Solution 6

The format of the *rpart* command works similarly to the *aggregate* function we used before. If you wanted to predict a continuous variable, such as age, you may use method="anova". But here, we just want a one or a zero, so method="class" is appropriate. The decisions that have been found go a lot deeper than what we saw last time when we looked for them manually. Decisions have been found for the SipSp variable, as well as the port of embarkation one that we didn't even look at. And on the male side, the kids younger than 6 years old have a better chance of survival, even if there weren't too many aboard. That resonates with the famous naval law we mentioned earlier.

R code:

```
fit <- rpart(Survived ~ Pclass + Sex + Age + SibSp + Parch + Fare + Embarked, data=train,
    method="class")
plot(fit)
text(fit)
fancyRpartPlot(fit)
```

Question 7

From the tree previously built, make a prediction for the survival status of each passenger in the test dataset. Put the results into a dataframe and write it into a csv file as if you were preparing a submission to a machine learning contest.

Solution 7

Here the easy trick is to use the *predict* function. Then, one can preparer a data frame with the results and use a write.csv to export it properly.

R code:

```
Prediction <- predict(fit, test, type = "class")
submit <- data.frame(PassengerId = test$PassengerId, Survived = Prediction)
write.csv(submit, file = "myfirstdtree.csv", row.names = FALSE)
```

3 Random Forests

Decision trees are faced with overfitting issues. To overcome these limitation, an ensemble method known as random forest is very popular in the machine learning community. The principe is to build a lot of different models, named weak classifiers, i.e an ensemble of simple decision trees, and let their outcomes be averaged or voted across the group. Before applying your first random forest algorithm, let's keep preparing the data with some feature engineering.

Question 8

Execute and understand the following code that makes some variable engineering in order to improve data quality and sense and improve the results. Execute BLOCK by BLOCK and spend some time to understand each BLOCK.

```
### BLOCK 1
test$Survived <- NA
combi <- rbind(train, test)
combi$Name <- as.character(combi$Name)

# BLOCK 2
combi$Title <- sapply(combi$Name, FUN=function(x) {strsplit(x, split='[,.]')[[1]][2]})
combi$Title <- sub(' ', '', combi$Title)
combi$Title[combi$Title %in% c('Mme', 'Mlle')] <- 'Mlle'
combi$Title[combi$Title %in% c('Capt', 'Don', 'Major', 'Sir')] <- 'Sir'
combi$Title[combi$Title %in% c('Dona', 'Lady', 'the Countess', 'Jonkheer')] <- 'Lady'
combi$Title <- factor(combi$Title)

# BLOCK 3
combi$FamilySize <- combi$SibSp + combi$Parch + 1

# BLOCK 4
combi$Surname <- sapply(combi$Name, FUN=function(x) {strsplit(x, split='[,.]')[[1]][1]})
combi$FamilyID <- paste(as.character(combi$FamilySize), combi$Surname, sep="")
combi$FamilyID[combi$FamilySize <= 2] <- 'Small'
```

```
famIDs <- data.frame(table(combi$FamilyID))
famIDs <- famIDs[famIDs$Freq <= 2,]
combi$FamilyID[combi$FamilyID %in% famIDs$Var1] <- 'Small'
combi$FamilyID <- factor(combi$FamilyID)

# BLOCK5
summary(combi$Age)
Agefit <- rpart(Age ~ Pclass + Sex + SibSp + Parch + Fare + Embarked + Title + FamilySize,
              data=combi[!is.na(combi$Age),], method="anova")
combi$Age[is.na(combi$Age)] <- predict(Agefit, combi[is.na(combi$Age),])

# BLOCK6
summary(combi)
summary(combi$Embarked)
which(combi$Embarked == '')
combi$Embarked[c(62,830)] = "S"
combi$Embarked <- factor(combi$Embarked)
summary(combi$Fare)
which(is.na(combi$Fare))
combi$Fare[1044] <- median(combi$Fare, na.rm=TRUE)

# BLOCK7
combi$FamilyID2 <- combi$FamilyID
combi$FamilyID2 <- as.character(combi$FamilyID2)
combi$FamilyID2[combi$FamilySize <= 3] <- 'Small'
combi$FamilyID2 <- factor(combi$FamilyID2)

# BLOCK8
train <- combi[1:891,]
test <- combi[892:1309,]
```

Solution 8

Solution is here:
http://trevorstephens.com/kaggle-titanic-tutorial/r-part-4-feature-engineering/

Question 9

After all this data preparation, you are ready to build your first random forest model with the *randomForest* function. Try to explain the same target (converted to a factor) with the variables `Pclass`, `Sex`, `Age`, `SibSp`, `Parch`, `Fare`, `Embarked`, `Title`, `FamilySize`, `FamilyID2`. Plot the importance of variables with the *varImpPlot* function.

Solution 9

Instead of specifying method="class" as with rpart, we force the model to predict our classification by temporarily changing our target variable to a factor with only two levels using as.factor(). The importance=TRUE argument allows us to inspect variable importance as we'll see, and the ntree argument specifies how many trees we want to grow. If you were working with a larger dataset you may want to reduce the number of trees, at least for initial exploration, or restrict the complexity of each tree using nodesize as well as reduce the number of rows sampled with sampsize. You can also override the default number of variables to choose from with mtry, but the default is the square root of the total number available and that should work just fine. Since we only have a small dataset to play with, we can grow a large number of trees and not worry too much about their complexity, it will still run pretty fast

R code:

```
set.seed(415)
```

```
fit <- randomForest(as.factor(Survived) ~ Pclass + Sex + Age + SibSp + Parch + Fare +
    Embarked + Title + FamilySize + FamilyID2,data=train, importance=TRUE, ntree=2000)
varImpPlot(fit)
Prediction <- predict(fit, test)
submit <- data.frame(PassengerId = test$PassengerId, Survived = Prediction)
write.csv(submit, file = "firstforest.csv", row.names = FALSE)
```

Question 10

You may finish with another ensemble method, a forest of conditional inference trees. Use the *cforest* function from the *party* package. Use the variable `FamilyID` instead of `FamilyID2` as this function is able to handle more categories that the previous one.

Solution 10

Now we have to specify the number of trees inside a more complicated command, as arguments are passed to cforest differently. We also have to manually set the number of variables to sample at each node as the default of 5 is pretty high for our dataset.

R code:

```
set.seed(415)
fit <- cforest(as.factor(Survived) ~ Pclass + Sex + Age + SibSp + Parch + Fare + Embarked +
    Title + FamilySize + FamilyID, data = train, controls=cforest_unbiased(ntree=2000, mtry=3))
```