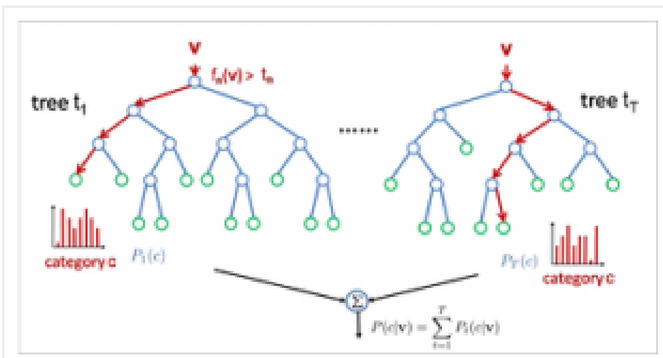
My Blog by Philippe Adjiman

Algorithms, Experiments, Coding, Mainly Geek Stuff

A Data Science Exploration From the Titanic in R

Posted on **September 12, 2013**



— Illustration of the (very hype) random forest learning method (click to see original website)

Kaggle offered this year a knowledge competition called "Titanic: Machine Learning from Disaster" exposing a popular "toy-yet-interesting" data set around the Titanic. The goal is to predict as accurately as possible the survival of the titanic's passengers based on their characteristics (age, sex, ticket fare etc…)

.

In that post, we'll use that data set in order to:

1. Illustrate through a comprehensive example a set of useful tools/packages to do some predictive modelling from the R statistical framework.

2. Take the opportunity of the example to illustrate the process and kind of tricks that it takes to improve/tune a predictive model.

The whole code creating all the plots/stats and models exposed in that post and also building an output reaching a score 0.79426 on the leaderboard can be found on github here or on Rpubs here (built with Knit HTML from R studio).

Preliminaries

First, download the test and training set from the data page of the competition (here is a zip of the two small files in case the page from kaggle is removed in the future).

Once you loaded the dataset into a data frame, you can do some data analysis/explorations. Even though that part is critical to start playing and feeling the data, I won't go into details because there already were blog posts written about that, in particular that one is a very nice R version of the getting started with excel data exploration tutorial on Kaggle's website.

However, i'll just illustrate a nice simple and effective way of observing one important aspect of the data: missing values.

The Amelia R package is a toolbox around missing values, in particular for performing imputation of the missing data. Getting a visual and global insight about missing data in the test and train set is as simple as that:

```
library(Amelia)
#... code for loading test and train data in a data frame
missmap(rawdata, main = "Missingness Map Train")
missmap(test, main = "Missingness Map Test")
```

— Missingness Maps (click for higher quality)

From those maps, you can immediately observe that only the ***age*** feature is badly suffering from missing data. Considering how small is the training set, you can hardly just ignore records having a missing age. We'll see later in the post what kind of strategy we can use to deal with that issue.

Building/Tuning models with Caret

The caret package is a kind of toolbox for homogenising the many existing R packages for classification and regression and also provide out of the box a standard way to perform common tasks like model parameters tuning and more. Also, the author (Max Khun) did an amazing job at documenting the package in the vignettes (here or here for a longer but older version) and on the package dedicated website.

Here is a snippet of code where i successively train a random forest and a gradient boosting machine (GBM) using the same train function from caret.

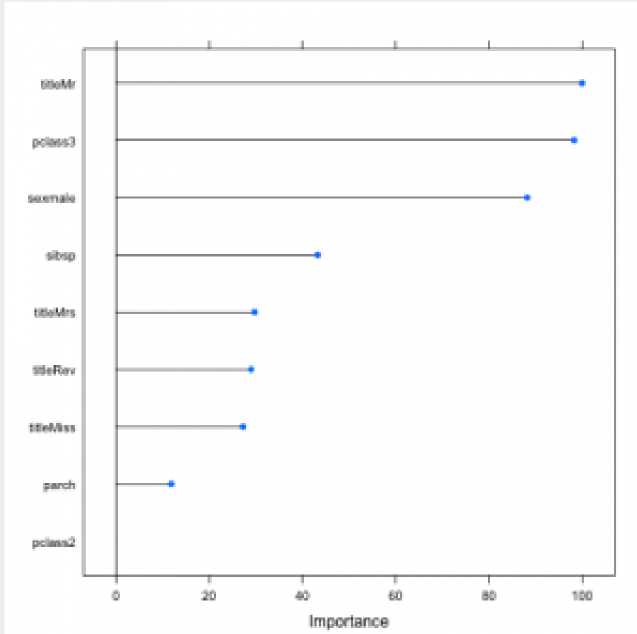
```
forest.model1 <- train(survived ~ pclass + sex + title + sibsp +pa
                              data.train,
                              importance=TRUE)

fitControl <- trainControl(## 10-fold CV
                         method = "repeatedcv",
                         number = 10,
```

```
                             ## repeated ten times
                             repeats = 10)

gbm.model2 <- train(survived ~ pclass + sex + title + sibsp +parcl
                    data.train,
                    distribution = "gaussian",
                    method = "gbm",
                    trControl = fitControl,
                    verbose = FALSE)
```

We'll discuss later the features used in the formula but note the fitControl parameter which is passed in the call for training the GBM. This parameter allows to completely define the way the model parameters will be tuned. In that example, the model parameters of the GBM (namely *interaction.depth*, *n.trees* and *shrinkage*, see output below) were compared using a repeated 10-fold cross validation with accuracy being the metric for comparison, but everything is tuneable for that purpose (you can even pass a grid of specific values to compare for each model parameter).

```
712 samples
 13 predictors
  2 classes: 'yes', 'no'

No pre-processing
Resampling: Cross-Validation (10 fold, repeated 10 times)

Summary of sample sizes: 642, 640, 642, 641, 640, 640, ...

Resampling results across tuning parameters:

  interaction.depth  n.trees  Accuracy  Kappa  Accuracy SD  Kappa
  1                  50       0.8       0.565  0.0436       0.096
  1                  100      0.801     0.567  0.0436       0.096
  1                  150      0.801     0.568  0.0434       0.096
  2                  50       0.795     0.548  0.0426       0.097
  2                  100      0.801     0.559  0.0437       0.099
  2                  150      0.804     0.565  0.0435       0.1
  3                  50       0.805     0.568  0.0449       0.102
  3                  100      0.807     0.573  0.0464       0.106
  3                  150      0.809     0.576  0.0442       0.1

Tuning parameter 'shrinkage' was held constant at a value of 0.1
Accuracy was used to select the optimal model using  the largest 
The final values used for the model were interaction.depth = 3, n
```

Also, you can easily visualize variable importance (you need to specify importance=TRUE in the train function, as we did, for having it):



— Variable Importance (click for higher quality)

You can observe that the variable value with the most importance is the title Mr . The interesting part is that the feature "title" was not initially in the data set and was artificially created (we'll detail a bit more about it later in the post). But overall, caret offers a very nice framework for easy models comparison and tuning with proper/uniform built-in cross-validation routines.

One thing though that is so true and said in perfect way in this must-watch killer talk: "Don't get stuck in algorithm land! Focus on putting better data in the algorithm". We'll see an example illustrating that later in the post.

Pick the best threshold for your classifier using ROC curves

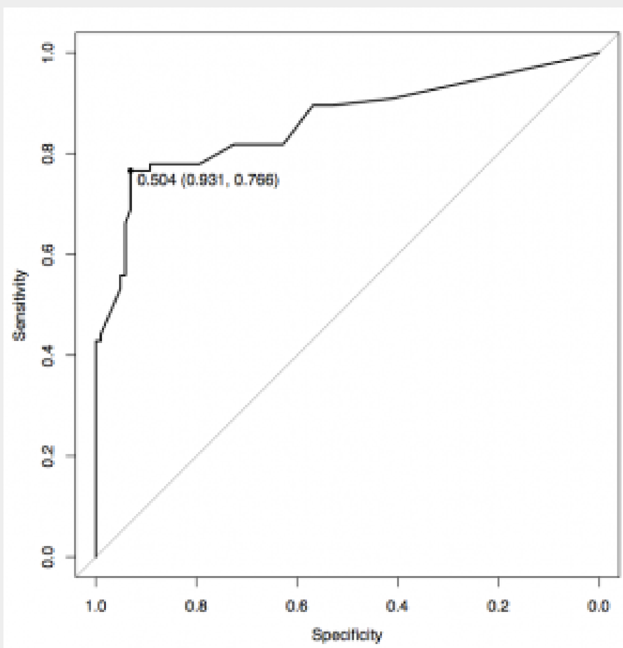
Most classifiers usually output the probability of an example belonging to a specific class (here 'survived' or 'died'). When the only matter is to optimise accuracy (as it is usually the case in competitions), it is useful to pick the optimal threshold/cutoff for assigning one class or the other.

ROC curves can be used for that and also to assess the robustness of your model. If you've never heard about ROC curves, this article gives the basic intuition and that paper goes much more into details while still being crystal clear (i warmly recommend the later if you're interested in the subject). For a standalone very clear example in R, this post is what you need (the code below is inspired from it).

The pROC package allows to easily analyse and display ROC curves. Here, we're interested in the threshold corresponding to the top left corner of the curve maximising sensitivity and specificity.

```
#code inspired from http://mkseo.pe.kr/stats/?p=790
result.predicted.prob.model1 <- predict(forest.model1, data.test,
result.roc.model1 <-  roc(data.test$survived, result.predicted.pr
plot(result.roc.model1, print.thres="best", print.thres.best.meth

result.coords.model1 <- coords(  result.roc.model1, "best", best.
                               ret=c("threshold", "accuracy"))
result.coords.model1
```

Which will output both a graph:



— ROC curve (click for higher quality)

and high level information about the curve, e.g. :

```
Call:
roc.default(response = data.test$survived, predictor = result.pre

Data: result.predicted.prob.model1$yes in 78 controls (data.test$
```

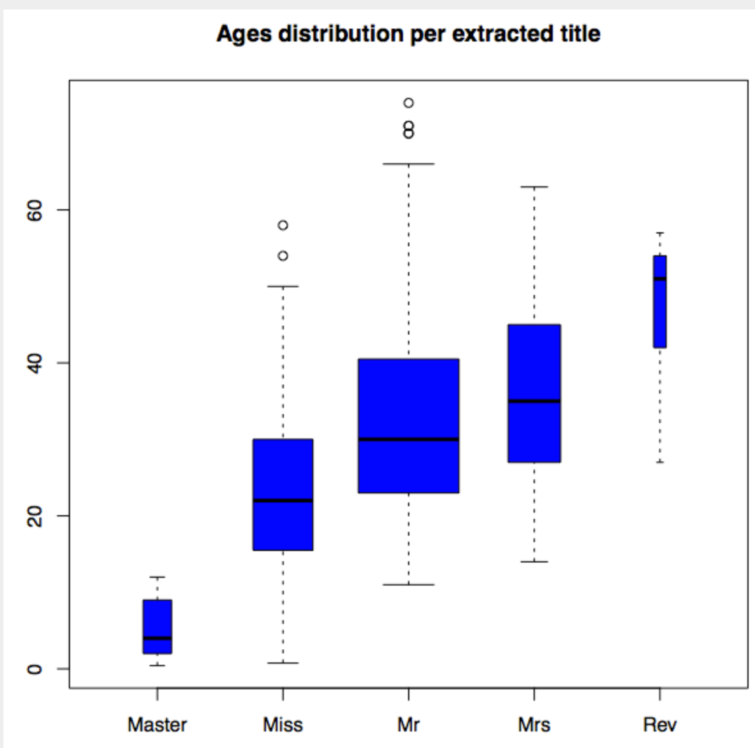
```
Area under the curve: 0.931
```

Note in particular the Area under the curve (a.k.a AUC) data point which is sometimes used to assess the robustness/quality of your model, although it has been questioned a lot and often criticised to not be a precise/useful classification performance measure (a small discussion around it can be found here). In other words, you're often better off relying on your K-fold cross validation measures to assess your out-of-sample performance (c.f. the previous section on caret).

Tweak and tricks

I've hinted earlier that the number of missing ages was too high and the training set too small to just ignore the records having a missing age. At least for me, any attempt to impute the missing ages (either in naive or more sophisticated ways) didn't lead to any significant accuracy improvement on the 10-fold cross validation test.

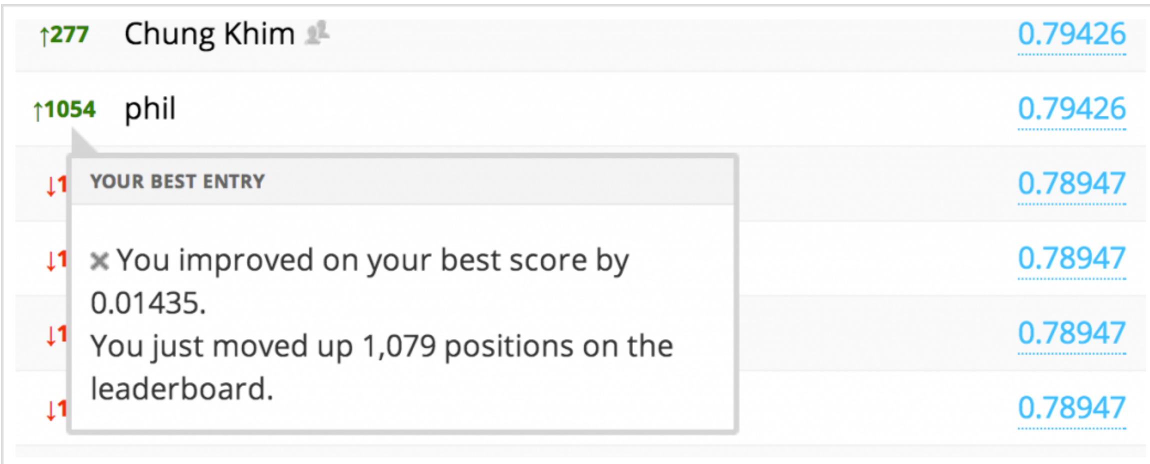
Turns out that extracting the title (i.e. Mr or Mrs. etc…) in the Name attribute of the data set did lead to an improvement (from the competition's forums, i saw that few people used that feature as well). Let's have a look at the age distributions per extracted title in the training set (some rare occurrences of titles were aggregated into larger titles, e.g. "Capt", "Col", "Major","Sir", "Don","Dr" were mapped to "Mr"):



— Age distributions per Title (click for higher quality)

This somehow matches the intuition (though I didn't know that in apparently old/traditional english, "Master" denotes a young/unmarried man). And it also makes sense intuitively that Title is a good proxy for the too many missing ages, allowing for totally ignoring the age feature and thus keep all the data in the training set, without introducing any potential noise with an imputation method.

When i've plugged in this new Title feature into the random forest, i saw an improvement from 0.785 to 0.801 on my 10-fold cross validation out-of-sample accuracy estimation, and it was reflected in my submission on the public leaderboard where i jumped to the top 5% best submissions at that time.



Note that an improvement on your cross validation is not always reflected on the leaderboard, sometimes even the opposite (c.f "Lesson One" from this very cool blog post by @rouli, highly recommended). Note also that this particular competition lasts 1 year and was just for learning purpose, so there are thousands and thousands of participants, including not few people who obviously spent useless time to extract the answers from publicly available lists (e.g. here or here) to get a near perfect score (though you could use them to know you near real final score on the private leaderboard if you can't wait the end of the competition, but still kind of pointless). Finally, more things can be done to try improve the accuracy even more, an obvious one being to combine multiple models together (majority vote is often used in binary/multi-class settings) but we won't cover that in this post.

Conclusion

We explored on a comprehensive example how R can be used to build and tune quickly robust predictive models which are significantly outperforming the baseline. Of course, it is somehow a toy example but it was interesting enough to explore some important aspects needed when building predictive models. For much bigger data sets (both in terms of training set size and/or number of features in the data) you might need to introduce different/additional technical and theoretical tools that we might explore in future posts.

Also, note that a competition settings might be very different than a real production settings.

Not only talking about why Netflix never implemented the model that won the $1M challenge, but also the whole infrastructure that you'd need to build in order to do big data science at scale on many different problems (Scala is quickly becoming a trend around that, check those killer slides and talk by my friend @BigDataSc from LinkedIn and @ccservers from eBay for more on that).

I'll conclude by citing again this awesome sentence from this must-watch talk by @nmkridler : "Don't get stuck in algorithm land! Focus on putting better data in the algorithm". I really think that this is what data science is all about.

References / Useful Links

- Full code of the plots/models exposed in that post: on github and Rpubs
- Kaggle: Getting Started with Excel — In R. Very nice R conversion of kaggle's initial explorative analysis of the data set.
- An introduction to ROC analysis. Crystal clear primer if you want to know more around ROC.
- Data Agnosticism: Feature Engineering Without Domain Expertise. Must watch talk if you're a Kaggler (by @nmkridler).
- Five Lessons from Kaggle's Event Recommendation Engine Challenge. Same comment as above (by @rouli).

This entry was posted in **data science**, **machine learning**, **R**, **tutorial** and tagged **data science**, **machine learning**, **R** by **padjiman**. Bookmark the **permalink [http://www.philippeadjiman.com/blog/2013/09/12/a-data-science-exploration-from-the-titanic-in-r/]** .