Predictive Models in R Clustered by Tag Similarity

Max Kuhn

Pfizer Global R&D
Nonclinical Statistics
Groton, CT
max.kuhn@pfizer.com

March 17, 2014

Motivation

Theorem (No Free Lunch)

In the absence of any knowledge about the prediction problem, no model can be said to be uniformly better than any other

Given this, it makes sense to use a variety of different models to find one that best fits the data

R has many packages for predictive modeling (aka machine learning)(aka pattern recognition) . . .

Model Function Consistency

Since there are many modeling packages written by different people, there are some inconsistencies in how models are specified and predictions are made.

For example, many models have only one method of specifying the model (e.g. formula method only)

The table below shows the syntax to get probability estimates from several classification models:

obj Class	**Package**	`predict` **Function Syntax**
`lda`	`MASS`	`predict(obj)` (no options needed)
`glm`	`stats`	`predict(obj, type = "response")`
`gbm`	`gbm`	`predict(obj, type = "response", n.trees)`
`mda`	`mda`	`predict(obj, type = "posterior")`
`rpart`	`rpart`	`predict(obj, type = "prob")`
`Weka`	`RWeka`	`predict(obj, type = "probability")`
`LogitBoost`	`caTools`	`predict(obj, type = "raw", nIter)`

The caret Package

The caret package was developed to:

- create a unified interface for modeling and prediction (currently 147 different models)
- streamline model tuning using resampling
- provide a variety of “helper” functions and classes for day–to–day model building tasks
- increase computational efficiency using parallel processing

First commits within Pfizer: 6/2005

First version on CRAN: 10/2007

Website: http://caret.r-forge.r-project.org

JSS Paper: www.jstatsoft.org/v28/i05/paper

Book: *Applied Predictive Modeling* (AppliedPredictiveModeling.com)

Recent Package Updates

The last major update to the package added a new feature: a straightforward and well–documented method for adding custom models.

The new approach modularizes the model code in a faux database. Previously the code was embedded in the package's internal functions.

The entry for a model has a set of modules, such as `fit` (the code to train the model), `predict` etc.

For a custom model, the user can create these elements *de novo* or copy the functions from another model and modify them.

There are several examples on the packages web page.

Getting Model Code

```
> svm_code <- getModelInfo("svmRadial", regex = FALSE)[[1]]
> unlist(lapply(svm_code, class))

       label       library          type    parameters          grid
 "character"   "character"   "character"  "data.frame"    "function"
        loop           fit       predict          prob    predictors
      "NULL"    "function"    "function"    "function"    "function"
        tags        levels          sort
 "character"    "function"    "function"

> head(svm_code$prob, 5)


1 function (modelFit, newdata, submodels = NULL)
2 {
3     out <- try(predict(modelFit, newdata, type = "probabilities")
4         silent = TRUE)
5     if (class(out)[1] != "try-error") {
```

Describing Models

One question that I'm asked periodically is "how different are the models that are used by the package?"

To help answer this, I added another "module" that is a vector of one or more *tags* describing the model. On average, each model has 3 tags.

```
> all_mods <- getModelInfo()
> all_tags <- unlist(lapply(all_mods, function(x) x$tag))
> set.seed(1218)
> sample(unique(all_tags), 10)
```

```
 [1] "Prototype Models"        "Self-Organising Maps"
 [3] "Regularization"          "Logistic Regression"
 [5] "Cost Sensitive Learning" "Logic Regression"
 [7] "Kernel Method"           "Bagging"
 [9] "Discriminant Analysis"   "Support Vector Machines"
```

Describing Models

Now that we have these data, how can we quantify how similar the models are from one another?

We can get all the unique tags and create a binary matrix with tags in columns and models in rows (149×41).

```
> tag_list <- unique(all_tags)
> binary <- matrix(0, ncol = length(tag_list),
+                     nrow = length(all_mods))
> colnames(binary) <- tag_list
> rownames(binary) <- names(all_mods)
> for(i in seq(along = all_mods)) {
+   has_tag <- colnames(binary) %in% all_mods[[i]]$tags
+   binary[i, has_tag] <- 1
+ }
```

Tags → Similarity

```
> binary[1:5, 1:5]

         Tree-Based Model Ensemble Model Boosting
ada                     1              1        1
AMORE                   0              0        0
avNNet                  0              1        0
bag                     0              1        0
bagEarth                0              1        0
         Implicit Feature Selection Neural Network
ada                               1              0
AMORE                             0              1
avNNet                            0              1
bag                               0              0
bagEarth                          1              0
```

Describing Models

Now that we have these data, how can we quantify how similar the models are from one another?

For binary variables, one similarity measure is the Jaccard measure. For models A and B:

$$J = \frac{N_{ab}}{N_{a\times} + N_{\times b} + N_{\times\times}}$$

We compute these pair–wise and get a 149×149 similarity matrix

Tags → Similarity

To do this, I used the proxy package to first compute the *distance* between two models and then convert that to a similarity:

```
> library(proxy)
>
> D <- dist(binary, method = "Jaccard")
> Dm <- as.matrix(D)
> sim <- 1 - Dm
> dim(sim)

[1] 149 149
```

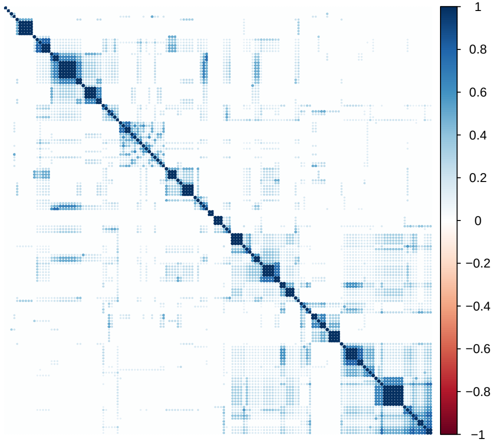
Tags → Similarity

```
> round(sim[1:5, 1:15], 2)

          ada AMORE avNNet bag bagEarth bagFDA bayesglm bdk
ada      1.00  0.00   0.14 0.2     0.33   0.33        0   0
AMORE    0.00  1.00   0.25 0.0     0.00   0.00        0   0
avNNet   0.14  0.25   1.00 0.5     0.33   0.33        0   0
bag      0.20  0.00   0.50 1.0     0.50   0.50        0   0
bagEarth 0.33  0.00   0.33 0.5     1.00   1.00        0   0
         blackboost Boruta brnn bstLs bstSm bstTree C5.0
ada            0.75   0.33 0.00  0.60  0.75    0.75 0.80
AMORE          0.00   0.00 0.33  0.00  0.00    0.00 0.00
avNNet         0.17   0.14 0.17  0.14  0.17    0.17 0.12
bag            0.25   0.20 0.00  0.20  0.25    0.25 0.17
bagEarth       0.17   0.14 0.00  0.33  0.40    0.17 0.29
```

How Similar are Models?: A Clustered Similarity Plot

```
> library(corrplot)
> corrplot(sim, order = "hclust", outline = FALSE,
+          addgrid.col = NA, tl.pos = "n")
```



Visualizing Similarity

One way of visualizing these relationships is to create a graph model (aka network diagram).

To do this, we could use a threshold for similarity between two models to label them as similar (i.e. connected) or not.

I used a threshold of 0.58 based on OEB technology.

```
> connected <- sim > .58
```

There are a few packages to make this visualization in R (`igraph`, `Rgraphviz`) but the results were a mess once you try to label each model (i.e. node).

I decided to use Mike Bostock's force directed graph code in D3.

Formatting the Graph

To use this, I needed to convert the matrix that I have to his format (in JSON).

First, the nodes need to be declared. For example:

```
{
  "nodes":[
    {"name":"node1","group":1},
    {"name":"node2","group":1},
    {"name":"node3","group":1},
    ]
}
```

The group is a field that can be used to color the nodes. I'm going to skip that step here but the real graph groups the model by type (e.g. classification, regression or both).

Formatting the Graph

```
> nodes <- paste('    {"name":"', rownames(sim),
+                '","group":1}', sep = "")
> nodes <- paste('  "nodes":[\n',
+                paste(nodes, collapse = ",\n"),
+                '\n  ],', sep = "")
```

For the first three models, this looks like:

```
  "nodes":[
    {"name":"ada","group":1},
    {"name":"AMORE","group":1},
    {"name":"avNNet","group":1}
  ],
```

Formatting the Graph

Now we need to define the connections. This should look like this:

```
"links":[
  {"source":1,"target":0,"value":1},
  {"source":2,"target":0,"value":1}
  ]
```

The numbers are the node index (e.g. node 0 is connected to nodes 1 and 2) and the value is the weight of the line.

Formatting the Graph

```
> links <- NULL
> for(i in 1:nrow(sim)){
+   for(j in i:ncol(sim)){
+     if(i != j) {
+       if(sim[i,j] > .58) {
+         tmp <- paste('    {"source":', i - 1,
+                      ',"target":', j - 1,
+                      ',"value":1}', sep = "")
+         links <- if(is.null(links)) tmp else c(links, tmp)
+         rm(tmp)
+       }
+     }
+   }
+ }
> cat(links[1:2], sep = "\n")

    {"source":0,"target":8,"value":1}
    {"source":0,"target":11,"value":1}
```

Formatting the Graph

```
> links <- paste('  "links":[\n',
+                paste(links, collapse = ",\n", sep = ""),
+                '\n  ]\n', sep = "")
> ## cat('{\n', nodes, links, '}', sep = "",
> ##     file = "model_tags.json")
```

Using D3

The rest is simple: put the json file with the D3 code from mbostock's page and you are all done!

Session Info

- R Under development (unstable) (2014-03-14 r65186), `x86_64-apple-darwin10.8.0`
- Base packages: base, datasets, graphics, grDevices, methods, stats, utils
- Other packages: caret 6.0-24, corrplot 0.73, ggplot2 0.9.3.1, knitr 1.5, lattice 0.20-27, proxy 0.4-12
- Loaded via a namespace (and not attached): car 2.0-19, codetools 0.2-8, colorspace 1.2-4, dichromat 2.0-0, digest 0.6.4, evaluate 0.5.1, foreach 1.4.1, formatR 0.10, grid 3.2.0, gtable 0.1.2, highr 0.3, iterators 1.0.6, labeling 0.2, MASS 7.3-30, munsell 0.4.2, nnet 7.3-7, plyr 1.8.1, proto 0.3-10, RColorBrewer 1.0-5, Rcpp 0.11.1, reshape2 1.2.2, scales 0.2.3, stringr 0.6.2, tools 3.2.0

This presentation was created with the `knitr` function at 17:36 on Monday, Mar 17, 2014.