Nobody Knows What It's Like To Be the Bad Man

The Development Process for the `caret` Package

Max Kuhn
Pfizer Global R&D
max.kuhn@pfizer.com

Zachary Deane–Mayer
Cognius
zach.mayer@gmail.com

Model Function Consistency

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

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

Generating Class Probabilities Using Different Packages

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

The caret Package

The caret package was developed to:

- create a unified interface for modeling and prediction (interfaces to 183 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://topepo.github.io/caret/

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

Model List: http://topepo.github.io/caret/bytag.html

Many computing sections in APM

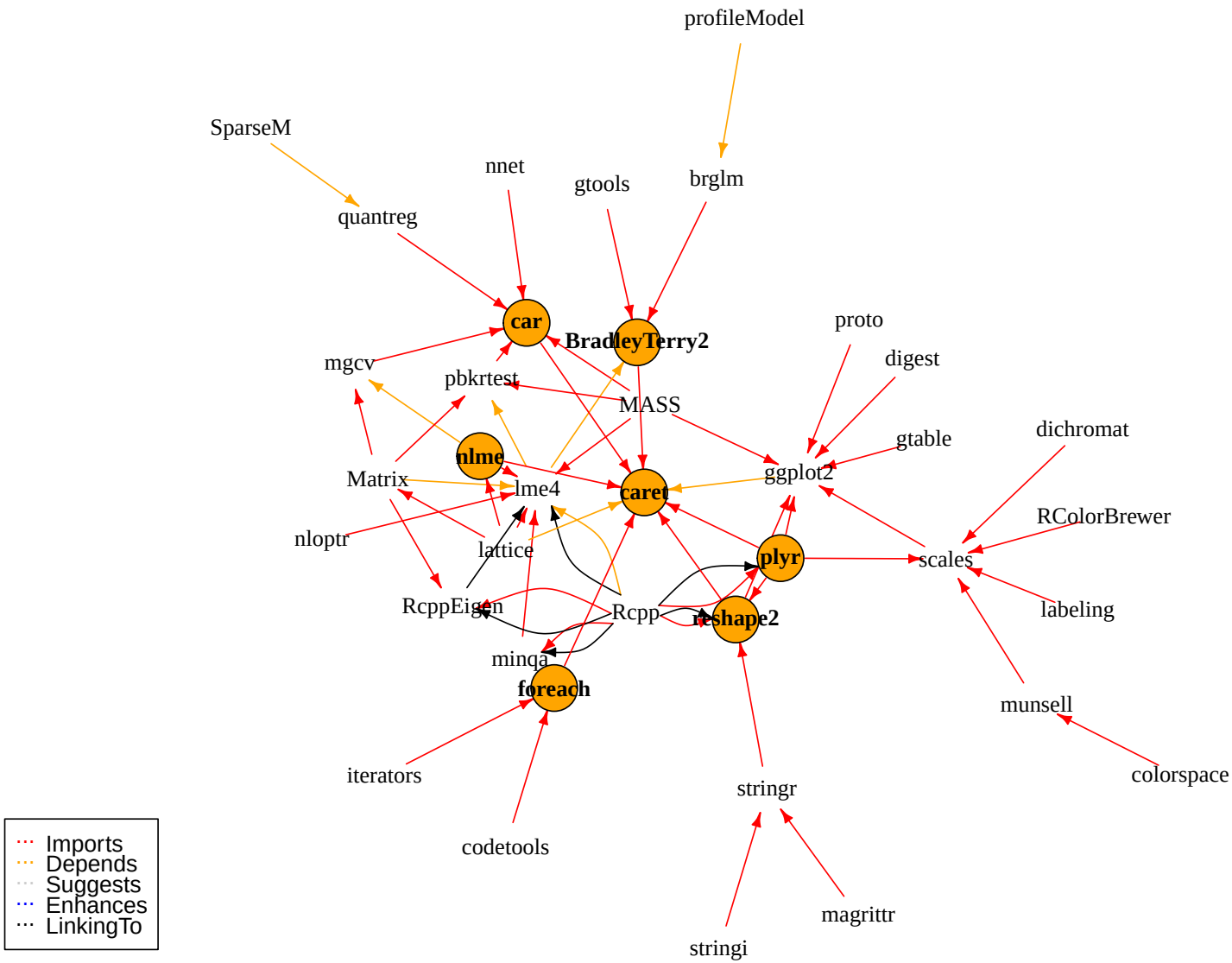
Package Dependencies

One thing that makes `caret` different from most other packages is that it uses code from an abnormally large number (> 80) of other packages.

A refresher:

- `Depends`: required for package to function; loaded when `caret` is loaded
- `Imports`: required for package to function; not loaded
- `Suggests`: the package uses it sometimes; not loaded

“Simple”””” Example (38 Nodes)



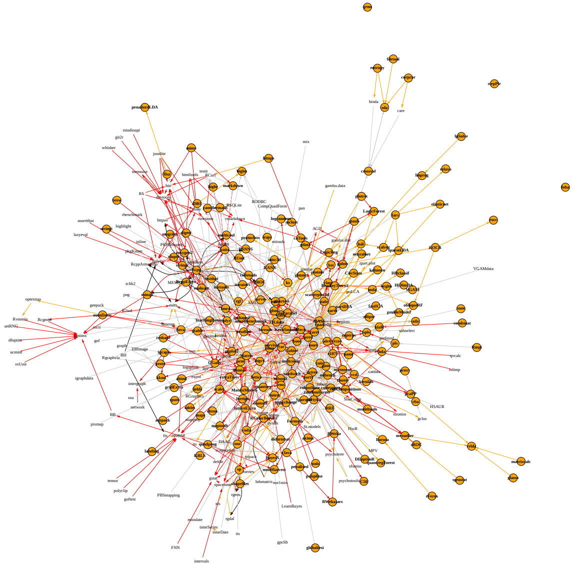
Package Dependencies

Originally, these were in the `Depends` field of the DESCRIPTION file which caused all of them to be loaded with `caret`.

For many years, they were moved to `Suggests`, which solved that issue.

However, their formal dependency in the DESCRIPTION file required CRAN to install hundreds of other packages to check `caret`. The maintainers were not pleased.

Package Dependencies in caret Version 5.17-07



Package Dependencies

This problem was somewhat alleviated at the end of 2013 when *custom methods* were expanded.

Although this functionality had already existed in the package for some time, it was refactored to be more user friendly.

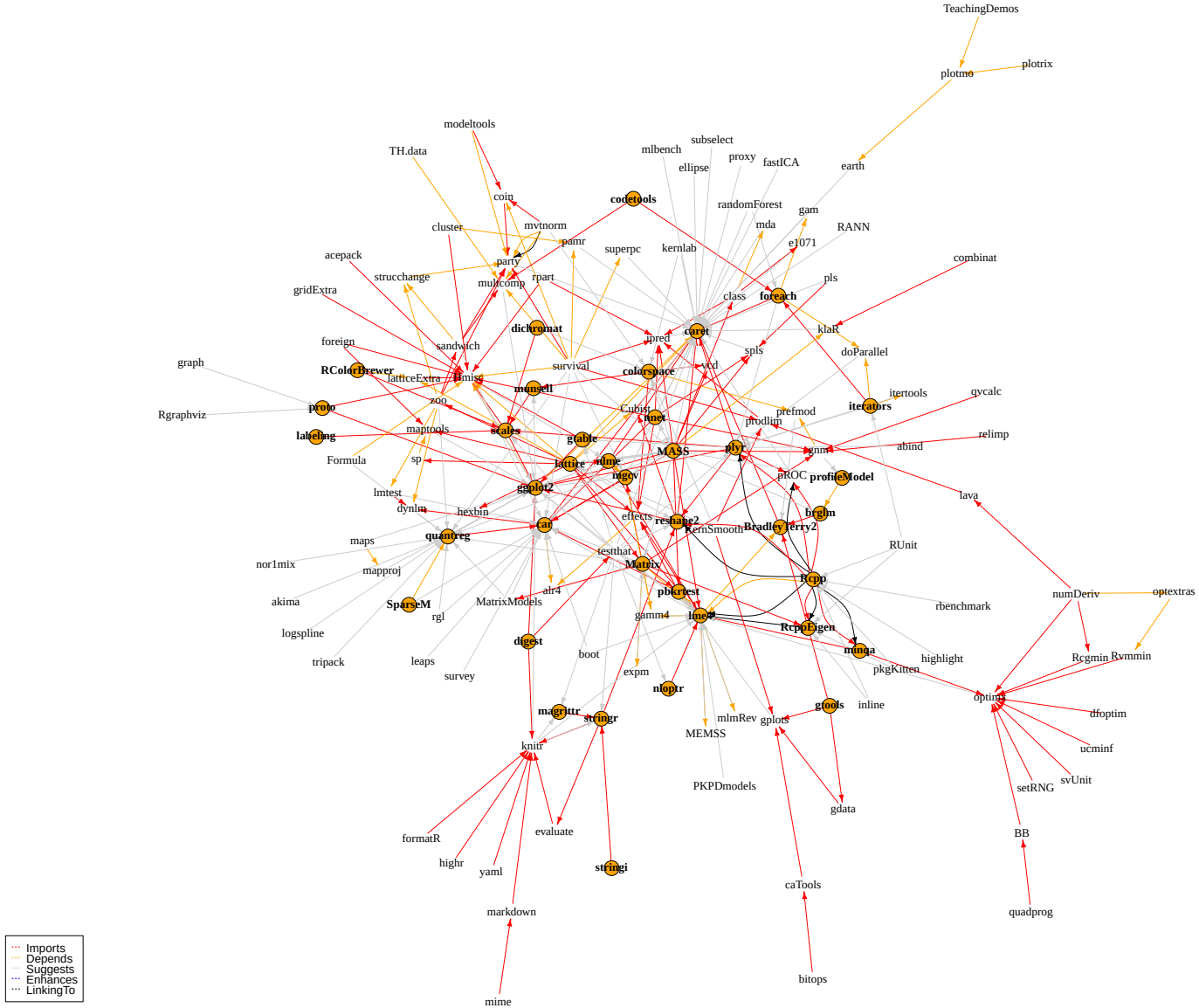
In the process, much of the modeling code was moved out of `caret`'s R files and into R objects, eliminating the formal dependencies.

Right now, the *total* number of dependencies is much smaller (2 `Depends`, 7 `Imports`, and 25 `Suggests`).

This still affects testing though (described later). Also:

```
1 package is needed for this model and is not installed.  (gbm).
Would you like to try to install it now?
```

38 Dependencies for caret Version 6.0-47



The Basic Release Process

1. create a few dynamic man pages
2. use `R CMD check --as-cran` to ensure passing CRAN tests and unit tests
3. update all packages (and R)
4. run regression tests and evaluate results
5. send to CRAN
6. repeat
7. repeat
8. install passed `caret` version
9. generate HTML documentation and sync github io branch
10. profit!

Toolbox

- RStudio projects: a clean, self contained package development environment
  - No more `setwd('/path/to/some/folder')` in scripts
  - Keep track of project-wide standards, e.g. code formatting
  - An RStudio project was the first thing we added after moving the `caret` repository to github
- `devtools`: automate boring tasks
- `testthat`: automated unit testing
- `roxygen2`: combine source code with documentation
- github: source control

devtools

`devtools::install` builds package and installs it locally

`devtools::check`:

1. Builds documentation
2. Runs unit tests
3. Builds tarball
4. Runs `R CMD CHECK`

`devtools::release` builds package and submits it to CRAN

`devtools::install_github` enables non-CRAN code distribution or distribution of private packages

`devtools::use_travis` enables automated unit testing through travis-CI and test coverage reports through coveralls

Testing

Testing for the package occurs in a few different ways:

- units tests via `testthat` and travis-CI
- regression tests for consistency

Automated unit testing via `testthat`

- `testthat::use_testthat`
- Unit tests prevent new features from breaking old code
- All functions should have associated tests
- Run during `R CMD check --as-cran`
- Can specify that certain tests be skipped on CRAN

`caret` is slowly adding more `testthat` tests

github + travis + coveralls

Travis and coveralls are tools for automated unit testing

- Travis reports test failutes and CRAN errors/warnings
- Coveralls reports % of code covered by unit tests
- Both automatically comment on the PR itself

Contributor submits code via a pull request

- Travis notifies them of test failures
- Coveralls notifies them to write tests for new functions
- Automated feedback on code quality allows rapid iteration

Code review once unit tests and R CMD CHECK pass

- github supports line-by-line comments
- Usually several more iterations here

Regression Testing

Prior to CRAN release (or whenever required), a comprehensive set of regression tests can be conducted.

All modeling packages are updated to their current CRAN versions.

For each model accessed by `train`, `rfe`, and/or `sbf`, a set of test cases are computed with the production version of `caret` and the devel version.

First, test cases are evaluated to make sure that nothing has been broken by updated versions of the consistuent packages.

Diffs of the model results are computed to assess any differences in `caret` versions.

This process takes approximately 3hrs to complete using `make -j 12` on a Mac Pro.

Regression Testing

```
$ R CMD BATCH move_files.R
$ cd ~/tmp/2015_04_19_09__6.0-41/
$ make -j 12 -i
 2015-04-19 09:13:44: Starting ada
 2015-04-19 09:13:44: Starting AdaBag
 2015-04-19 09:13:44: Starting AdaBoost.M1
 2015-04-19 09:13:44: Starting ANFIS
                    :
 make: [FH.GBML.RData] Error 1 (ignored)
                    :
 2015-04-19 12:03:52: Finished WM
 2015-04-19 12:04:48: Finished xyf
```

Documentation

`caret` originally contained four package vignettes with in–depth descriptions of functionality with examples.

However, this added time to `R CMD check` and was a general pain for CRAN.

Efforts to make the vignettes more computationally efficient (e.g. reducing the number of examples, resamples, etc.) diminished the effectiveness of the documentation.

Documentation

The documentation was moved out of the package and to the github IO page.

These pages are built using `knitr` whenever a new version is sent to CRAN. Some advantages are:

- longer and more relevant examples are available
- update schedule is under my control
- dynamic documentation (e.g. D3 network graphs, JS tables)
- better formatting

It currently takes about 4hr to create these (using parallel processing when possible).

Backup Slides

roxygen2

Simplified package documentation

Automates many parts of the documentation process

- Special comment block above each function
- Name, description, arguments, etc.
- Code and documentation are in the same source file

A must have for new packages but hard to convert existing packages

- caret has 92 .Rd files
- I'm not in a hurry to re-write them all in roxygen2 format

Required "Optimizations"

For example, there is one check that produces a large number of false positive warnings. For example:

```
> bwplot.diff.resamples <- function (x, data, metric = x$metric, ...)  {
+     ## some code
+     plotData <- subset(plotData, Metric %in% metric)
+     ## more code
+ }
```

will trigger a warning that "`bwplot.diff.resamples: no visible binding for global variable 'Metric'`".

The "solution" is to have a file that is sourced first in the package (e.g. `aaa.R`) with the line

```
> Metric <- NULL
```

Judging the Severity of Problems

It's hard to tell which warnings should be ignored and which should not. There is also the issue of inconsistencies related to who is "on duty" when you submit your package.

It is hard to believe that someone took the time for this:

Description Field: `"My awesome R package"`
`R CMD check:` `"Malformed Description field: should contain one or more complete sentences."`

Description Field: `"This package is awesome"`
`R CMD check:` `"The Description field should not start with the package name, 'This package' or similar."`

Description Field: `"Blah blah blah."`
`R CMD check:` **PASS**