A Glimpse into the State of R Packages

Code

Feature Extraction and Statistical Analysis of CRAN and Bioconductor Packages

[redacted]

Abstract

We present a statistical study of currently published R packages on CRAN and Bioconductor using Generalized Linear Models (GLMs), Classification and Regression Trees (CART), Random Forest (RF). and Principal Component Analysis (PCA). We found that of while there are no useful linear predictors of the package repository, linear regression and PCR analysis yielded useful insights into the relationship between extracted features and the package naming convention and exported symbols.

[redacted]

Introduction

This study extracts features by static analysis of R packages from CRAN and Bioconductor, and aims to uncover the relationship between the package features and their purposes.

If you are not familiar with R repositories, CRAN (https://cran.r-project.org/) is the primary repository for R packages, and Bioconductor (https://bioconductor.org/) is a separate repository primarily for bioinformatics packages. Since Bioconductor is managed by separate groups of people, over the time there has been a divergence in the code and moderation style of these two communities. This study aims to discover these differences using statistical analysis of code published on these two repositories. At the time of writing, there are 17908 packages on CRAN and 3178 on Bioconductor that we were able to fetch and parse as valid R package containing valid R code.

This project is inspired by this paper and this UseR 2017 presentation (https://www.youtube.com/watch?v=Pv5dfsHBBKE):

```
Bååth, R. The R Journal: The State of Naming Conventions in R. The R Journal 2012, 4 (2), 74–75. http
s://doi.org/10.32614/RJ-2012-018.
```

I selected this topic because:

- It is funny to do an R project about R itself
- This project will help me learn R in the following ways:
    - Learn more about R package structures

- Get high-level statistics of how package maintainers write R
- Practice writing native R extensions and interop with other non C-based languages like Go
- Understand R syntax better by writing a static analyzer

Data Acquisition

R Package List

We obtained a list of R packages on CRAN and Bioconductor using `BiocManager::repositories()` and `available.packages()`. We then assembled the URL of the source of each package and wrote the list to a CSV for use in the next step.

Hide

```
df.packages <- load.dataset("datasets/package.csv", read_csv, function(filename) {
  BiocManager::repositories() %>%
    map2_dfr(names(.), function(url, name) {
      available.packages(contrib.url(url), type = "source") %>%
        as_tibble() %>%
        mutate(RepositoryName = name)
    }) %>%
    select(Package, Version, Repository, RepositoryName) %>%
    group_by(Package) %>%
    arrange(RepositoryName != "CRAN", Version) %>%
    slice_tail() %>%
    mutate(SourceURL = sprintf("%s/%s_%s.tar.gz", Repository, Package, Version)) %>%
    write_csv(filename)
})
```

Master Dataset

We have 21086 packages to analyze. This is a very large number, and raw source code is very hard to statistically model, so we need to do some preliminary feature extraction to model every package. Here we needed some serious concurrency & heavy-lifting so we wrote this part in Go (https://go.dev/), which calls the R built-in tokenizer `getParseData(parse(file))` and then perform the static analysis in Go.

We extracted the following information from each package:

- Number of files in package by file extension
- Fields in package `DESCRIPTION` file:
  - `Package` : the name of the package
  - `Version`
  - `Title`
  - `Description`
  - `Depends` : dependencies that need to be directly attached
  - `Imports` : dependencies the package will import
  - `Suggests` : suggested dependencies
  - `License` : the license of the package
  - `biocViews` : only for Bioconductor packages, "tags" of the package
- Exports and Imports in package `NAMESPACE` file: (done by static analysis of calls to export functions, so we had to ignore `exportPattern()`, but luckily this is not recommended by R so only a small number of packages use this)
  - `Exports` : exported symbols
  - `Imports` : imported symbols
- R source files (only in the `${PACKAGE}/R/` folder, excluding tests, examples, etc):
  - Name of the file
  - Number of tokens in the file
  - All variable assignments:
    - Left side of the assignment
    - Type of Right hand side (function, symbol, function call, const literal, etc)

    - All function declarations:
        - The name of the variable the function is assigned to (if any)
        - The name of positional arguments
        - The name of keyword arguments and their default values
    - All function calls:
        - The name of the function
        - The name of positional arguments
        - The name of keyword arguments and their values
    - All library calls:
        - The method of library call (`::` selector, `require()`, `library()'`, `requireNamespace()`, etc)
        - The name of the library

▸ Show Go Code

Hide

```
system("go generate")
system('go build -o Project2_Go.so -buildmode="c-shared" .')
dyn.load("Project2_Go.so")
extract.packages <- function(urls, output = "", num.procs = 1L) {
  .Call("ExtractPackages", urls, output, as.integer(num.procs))
}
if (!file.exists("datasets/package_details.json")) {
  # takes about 15~60 minutes depending on network and your machine
  extract.packages(urls = df.packages$SourceURL, output = "datasets/package_details.json", num.procs =
16L)
}
```

```
## Starting to fetch 21086 packages with 16 threads...
## Fetched 21086/21086 (100.00%) ETA: 00h00m00s (R slave: Start: 16, Kill: 0, Err: 343, OK: 278287 )
```

Feature Collection

The extracted data is stored as a JSON stream as the master dataset and obviously we need to further extract features for actual analysis. So we wrote a second-stage feature extractor to extract more useful statistics.

This second-stage feature extractor is written purely in Go and computes the following:

- Number of words in package Title and Description
- % of assignments that use `=` instead of `<-`
- % naming convention of variables, exported symbols, and R source file themselves:
    - `period.separated`
    - `camelCase`
    - `snake_case`
    - `ALL_CAPS`
- Ratio of .Rd files to .R files
- Average number of tokens per file
- Number of R files
- Number of Exports
- Ratio of Imports to Depends
- Major Version Number
- Ratio of `.c` and `.cpp` files over `.R` files (C and C++ extensions)
- Ratio of `.f`, `.for` and `.f90` files over `.R` files (Fortran extensions)
- Ratio of `.java`, `.class` and `.jar` files over `.R` files (Java extensions)

After extraction a data frame is assembled in memory using the C interface of R and then passed into R for downstream wrangling and analysis.

Note that while this master feature collection is not tidy, it would be subset into tidy datasets for each part of the analysis

(see "Feature Selection" parts).

Hide

```
extract.features <- function(package.details.file, num.procs = 4L) {
  .Call("ExtractFeatures", package.details.file, num.procs)
}
df.features <- load.dataset("datasets/package_features.csv", read_csv, function(filename) {
  # this one is IO bound ... takes about 5 minutes
  df <- extract.features("datasets/package_details.json", num.procs = 4L)
  print(class(df))
  write_csv(df, filename)
})
df.features %<>%
  mutate(repo = as.factor(repo))
glimpse(df.features)
```

```
## Rows: 21,086
## Columns: 51
## $ package                              <chr> "abc.data", "a4Preproc", "abbrevi…
## $ repo                                 <fct> CRAN, Bioconductor, CRAN, CRAN, B…
## $ title.words                          <dbl> 8, 6, 3, 8, 6, 6, 6, 6, 9, 6, 6, …
## $ description.words                    <dbl> 11, 6, 21, 39, 6, 6, 6, 6, 28, 30…
## $ eq_assign.prop                       <dbl> NA, 0.00000000, 0.00000000, 0.000…
## $ var.naming..period_separated         <dbl> 0, 0, 0, 0, 0, 0, 0, 4, 51, 0, 1,…
## $ var.naming..camel_case               <dbl> 0, 4, 24, 0, 58, 63, 0, 22, 62, 4…
## $ var.naming..snake_case               <dbl> 0, 0, 1, 7, 0, 0, 0, 0, 0, 68, 32…
## $ var.naming..pascal_case              <dbl> 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 5, …
## $ var.naming..all_caps                 <dbl> 0, 0, 0, 0, 0, 3, 0, 0, 0, 0, 10,…
## $ var.naming..total                    <dbl> 0, 4, 25, 7, 66, 74, 1, 29, 116, …
## $ export.naming..period_separated      <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, …
## $ export.naming..camel_case            <dbl> 0, 1, 0, 0, 5, 3, 0, 3, 0, 0, 1, …
## $ export.naming..snake_case            <dbl> 0, 0, 1, 7, 0, 0, 0, 0, 0, 8, 4, …
## $ export.naming..pascal_case           <dbl> 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, …
## $ export.naming..all_caps              <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, …
## $ export.naming..total                 <dbl> 0, 1, 1, 7, 13, 9, 0, 6, 3, 10, 6…
## $ rfile.naming..period_separated       <dbl> 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, …
## $ rfile.naming..camel_case             <dbl> 0, 1, 0, 0, 1, 4, 1, 3, 0, 0, 1, …
## $ rfile.naming..snake_case             <dbl> 0, 0, 1, 7, 0, 0, 0, 0, 0, 8, 1, …
## $ rfile.naming..pascal_case            <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, …
## $ rfile.naming..all_caps               <dbl> 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, …
## $ rfile.naming..total                  <dbl> 0, 1, 1, 7, 2, 5, 1, 3, 2, 10, 2,…
## $ var.naming.prop..period_separated    <dbl> 0.000000000, 0.000000000, 0.00000…
## $ var.naming.prop..camel_case          <dbl> 0.00000000, 1.00000000, 0.9600000…
## $ var.naming.prop..snake_case          <dbl> 0.000000000, 0.000000000, 0.04000…
## $ var.naming.prop..pascal_case         <dbl> 0.000000000, 0.000000000, 0.00000…
## $ var.naming.prop..all_caps            <dbl> 0.00000000, 0.00000000, 0.0000000…
## $ export.naming.prop..period_separated <dbl> 0.00000000, 0.00000000, 0.0000000…
## $ export.naming.prop..camel_case       <dbl> 0.0000000, 1.0000000, 0.0000000, …
## $ export.naming.prop..snake_case       <dbl> 0.0000000, 0.0000000, 1.0000000, …
## $ export.naming.prop..pascal_case      <dbl> 0.00000000, 0.00000000, 0.0000000…
## $ export.naming.prop..all_caps         <dbl> 0, 0, 0, 0, 0, 0, NA, 0, 0, 0, 0,…
## $ rfile.naming.prop..period_separated  <dbl> 0.000, 0.000, 0.000, 0.000, 0.500…
## $ rfile.naming.prop..camel_case        <dbl> 0.0000000, 1.0000000, 0.0000000, …
## $ rfile.naming.prop..snake_case        <dbl> 0.00000000, 0.00000000, 1.0000000…
## $ rfile.naming.prop..pascal_case       <dbl> 0.00000000, 0.00000000, 0.0000000…
## $ rfile.naming.prop..all_caps          <dbl> 0.0, 0.0, 0.0, 0.0, 0.0, 0.2, 0.0…
## $ call.randomForest                    <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, …
## $ call.rpart                           <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, …
## $ avg_r_tokens                         <dbl> 0.00000, 331.00000, 590.00000, 95…
## $ export.num                           <dbl> 0, 1, 1, 7, 13, 9, 0, 6, 3, 10, 6…
## $ r_import_to_depend                   <dbl> 0.000, 0.000, NA, 3.000, 1.000, 0…
## $ version.major                        <dbl> 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, …
## $ native.c.prop                        <dbl> NA, 0.0000000, 0.0000000, 0.00000…
## $ native.f.prop                        <dbl> NA, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,…
## $ native.j.prop                        <dbl> NA, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,…
## $ ext.r                                <dbl> 0, 1, 3, 28, 3, 5, 3, 3, 2, 11, 7…
## $ ext.rd                               <dbl> 3, 1, 1, 8, 6, 4, 0, 4, 13, 13, 6…
## $ ext.rds                              <dbl> 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, …
## $ ext.rda                              <dbl> 3, 0, 0, 0, 0, 0, 1, 0, 2, 3, 0, …
```

Classification Universal Codebase

We wrote the following functions that provide a uniform interface for making classifications using `glm`, `rpart`, and `randomForest` models using either uniform or train/test split datasets and then assess the performance of the model. This

saves the trouble of rewriting the prediction, test and summarization part for each model.

Hide

```
partition.dataset <- function(df, train.prop = 0.8, seed = 1L) {
  set.seed(seed)
  train <- sample(seq_len(nrow(df)), size = floor(nrow(df) * train.prop))
  list(train = df[train, ], test = df[-train, ])
}
make.prediction <- function(outcome, predictors, method = "glm",
                            data = df.features,
                            data.train = data, data.test = data,
                            cutoff = 0.5, regression = FALSE) {
  ret <- list()

  form <- substitute(
    .y ~ .x,
    list(.y = substitute(outcome), .x = substitute(predictors))
  )
  if (method == "glm") {
    ret$glm <- glm(form, family = binomial, data = data.train)
    ret$pred <- predict(ret$glm, data.test, type = "response")
  } else if (method == "rpart") {
    ret$rpart <- rpart(form, data = data.train, model = TRUE)
    if (regression) {
      ret$pred <- predict(ret$rpart, data.test, type = "vector")
    } else {
      ret$pred <- predict(ret$rpart, data.test, type = "prob")[, 2]
    }
  } else if (method == "rf") {
    ret$rf <- randomForest(eval(form), data = data.train)
    ret$pred <- predict(ret$rf, data.test, type = "prob")[, 2]
  } else {
    stop("method not supported")
  }
  ret$pred.expect <- eval(substitute(outcome), data.test)
  if (!regression) {
    if (is.factor(ret$pred)) {
      ret$pred.factor <- ret$pred
    } else {
      ret$pred.binomial <- ifelse(ret$pred > cutoff, 1, 0)
      ret$pred.factor <- factor(ret$pred.binomial, levels = c(0, 1))
      attributes(ret$pred.factor) <- attributes(ret$pred.expect)
    }
    ret$confusion <- confusionMatrix(ret$pred.factor, ret$pred.expect)

    ret$roc <- pROC::roc(ret$pred.expect, ret$pred)
  }

  ret %>%
    set_class(prepend(
      class(.),
      c(ifelse(regression, "regression", "prediction"), method)
    ))
}
summary.prediction <- function(fitted.reg) {
  fitted.reg %$% c(confusion$overall,
    Sensitivity = confusion$table[1, 1] / sum(confusion$table[, 1]),
    Specificity = confusion$table[2, 2] / sum(confusion$table[, 2]),
    AUC = roc$auc
  )
}
```

Logistic Regression

For logistic regression we write a predictor of whether a package is published on CRAN or Bioconductor.

Feature Selection

We made a subset of possible dependent variables (11 total).

Hide

```
df.glm.features <- df.features %>%
  filter(ext.r > 0) %>%
  na.omit()
glm.depvars <- quote(title.words + description.words +
  eq_assign.prop +
  var.naming.prop..period_separated +
  var.naming.prop..camel_case +
  export.naming.prop..period_separated +
  export.naming.prop..camel_case +
  avg_r_tokens + version.major + native.c.prop + native.f.prop + native.j.prop)
```

Simple GLM Fit

we started with a simple GLM fit without cross-validation or train/test split, we expected there to be little over-fitting since the dataset is large (20k+) rows.

Hide

```
cran.prop <- summarize(df.glm.features, mean(repo == "CRAN")) %>% pull()
glm.repo.simple <- make.prediction(repo, eval(glm.depvars), data = df.glm.features)
summary(glm.repo.simple)
```

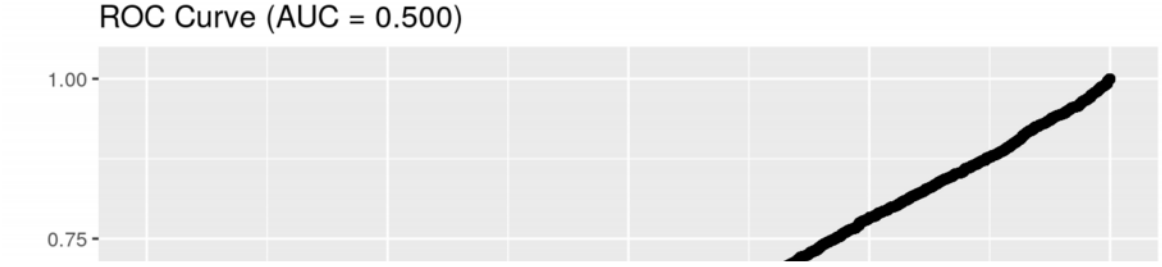
```
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##      0.8763345      0.0000000      0.8713410      0.8812013      0.8763345
## AccuracyPValue  McnemarPValue    Sensitivity    Specificity            AUC
##      0.5057574      0.0000000      0.0000000      1.0000000      0.4998199
```

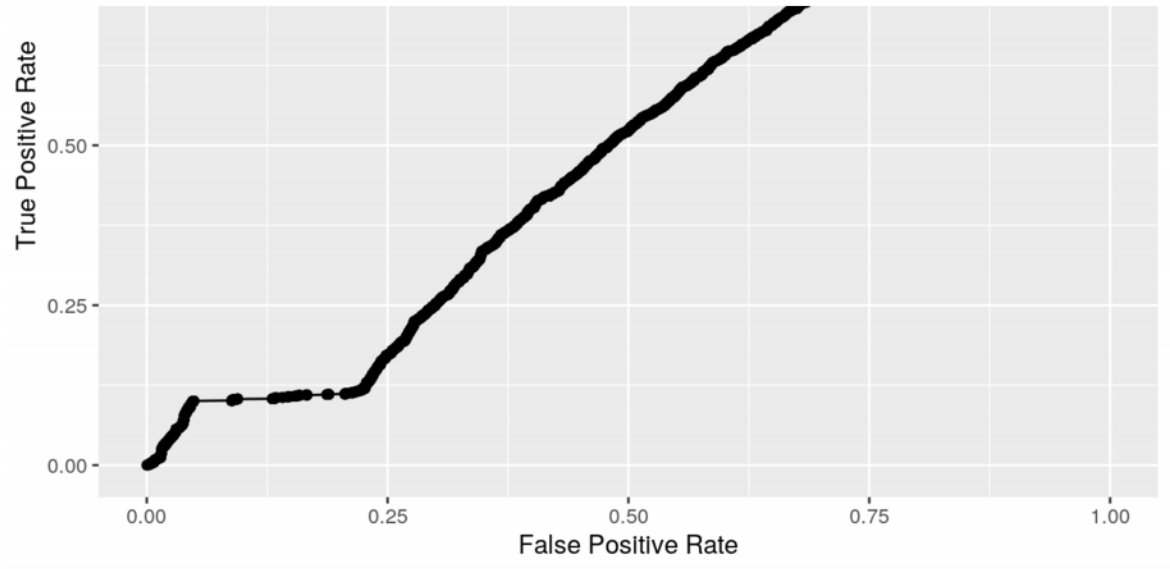
Unfortunately logistic regression did not find any useful predictors as we got a kappa value of zero, also it is noted that we got a Sensitivity of 0 and Specificity of 1 because the predictor just gave up and found guessing CRAN every single time is the best strategy.

And subsequently the AUC will be 0.5:

Hide

```
glm.repo.simple$roc %$% qplot(x = (1 - specificities), y = sensitivities, geom = "line") +
  geom_point() + labs(
    x = "False Positive Rate",
    y = "True Positive Rate",
    title = sprintf("ROC Curve (AUC = %0.3f)", glm.repo.simple$roc$auc)
  )
```





GLM with Train/Test Split

This time we split the dataset using 80/20 train/test split and repeated the process for 5 different random seeds. Given what has happened above there is no reason to expect any fitting happening at all.

Hide

```
map_dfr(1:5, function(seed) {
  df.tvsplit <- partition.dataset(df.glm.features, train.prop = 0.8, seed = seed)
  make.prediction(repo, eval(glm.depvars), data.train = df.tvsplit$train, data.test = df.tvsplit$test)
%>%
    summary() %>%
    c(seed = seed)
}) %>%
  select(seed, Accuracy, Sensitivity, Specificity, AUC) %>%
  knitr::kable(longtable = TRUE, caption = "Logistic Regression (TV Split) Performance")
```

Logistic Regression (TV Split) Performance

seed	Accuracy	Sensitivity	Specificity	AUC
1	0.8681477	0	1	0.5091625
2	0.8793999	0	1	0.5124358
3	0.8759377	0	1	0.5096286
4	0.8791114	0	1	0.4965994
5	0.8788229	0	1	0.5051848

Discussions

GLM does not seem to produce any useful predictors, it is likely that there is no monotonic relationship between any of the predictors and the outcome (CRAN/Bioconductor).

To make sure that this is not a mistake in the GLM setup we repeated the same process with dummy variables and the results confirmed that the data we have is not sufficient for GLM to produce useful models.

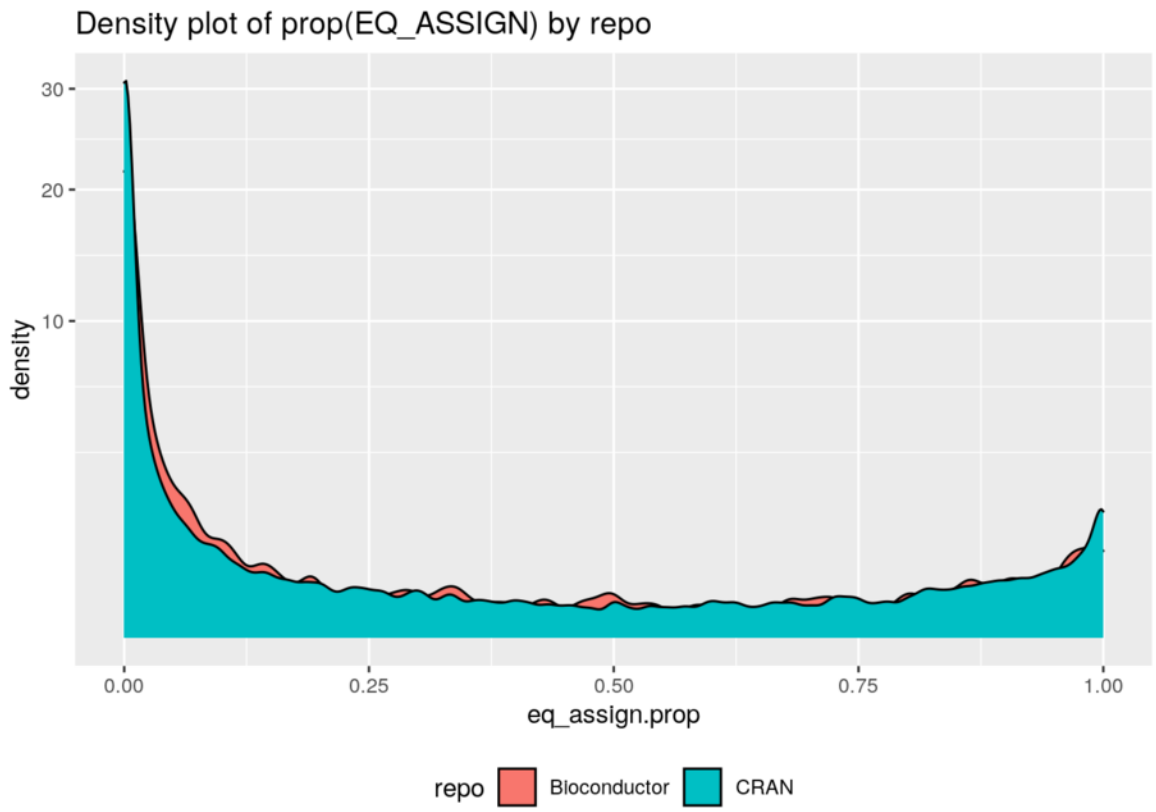
This is unexpected that at least I would expect fewer packages in Bioconductor to use EQ_ASSIGN (assignment with `=`) because it is clearly stated in their package guidelines that:

Use `<-` instead of `=` for assigning variables except in function arguments.

If we look at the distribution of the use of EQ_ASSIGN between the two groups, it looks like both repositories have a similar distribution pattern of most packages using LEFT_ASSIGN, with a small group of packages using EQ_ASSIGN:

Hide

```
df.glm.features %>%
  group_by(repo) %>%
  ggplot(aes(x = eq_assign.prop, fill = repo)) +
  geom_density() +
  scale_y_sqrt() +
  labs(title = "Density plot of prop(EQ_ASSIGN) by repo") +
  theme(legend.position = "bottom")
```



Tree-based Classifiers

Feature Selection

For CART classifiers we selected the following features:

Hide

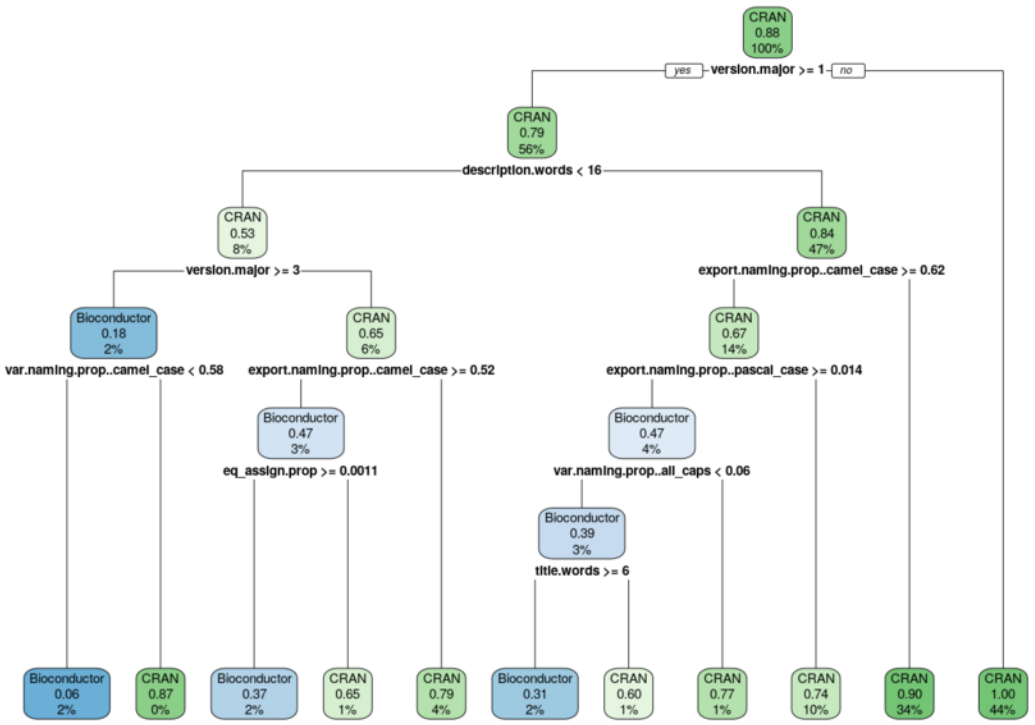
```
df.rpart.features <- df.features %>%
  select(
    matches("^.+naming.prop\\.\\.(period_separated|all_caps|.+_case)$"),
    eq_assign.prop,
    starts_with("native."),
    starts_with("call."),
    repo,
    version.major,
    export.num,
    title.words, description.words
  ) %>%
  na.omit()
```

Simple CART model

Going back to predicting the repository: this time we use a CART and a random forest model and hopefully it captures more intricate relationships between the predictors and the outcome.

Hide

```
df.rpart.repo <- make.prediction(repo, ., method = "rpart", data = df.rpart.features)
rpart.plot(df.rpart.repo$rpart)
```



Hide

```
summary(df.rpart.repo)
```

```
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##   9.102748e-01   4.509019e-01   9.060202e-01   9.143972e-01   8.808079e-01
## AccuracyPValue  McnemarPValue    Sensitivity    Specificity            AUC
##   2.341582e-37  6.425060e-166   3.671296e-01   9.837740e-01   8.729933e-01
```

As expected this time we get a much better Kappa at 45% and an AUC of 87%.

If we look at this tree we could observe the following:

- First branch is if the version is less than 1 it is 100% CRAN, which is expected because Bioconductor has a very weird versioning guideline that major version 0 is reserved for unreleased packages.
- Second branch says packages with more than 16 words in their description are likely to be in CRAN, this may be because CRAN allows citations and other info to be included in the description field but Bioconductor only says "description should be at least 3 sentences."
- Looking at the deeper branches it seems like `camelCase` and `ALLCAPS` variable naming is correlated with CRAN packages.

Next, when we use random forest we got a 99.9% accuracy which is without doubt over-fitting.

Hide

```
df.rf.repo <- make.prediction(repo, ., method = "rf", data = df.rpart.features)
summary(df.rf.repo)
```

```
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##    0.999337821    0.996838736    0.998843593    0.999657797    0.880807858
## AccuracyPValue  McnemarPValue    Sensitivity    Specificity            AUC
##    0.000000000    0.001496164    0.994444444    1.000000000    0.999999391
```

Train/Test Split

If we repeat the process but this time using 80/20 train/test split we get the following results:

Hide

```
map_dfr(c("rpart", "rf"), function(method) {
  map_dfr(1:5, function(seed) {
    df.tvsplit <- partition.dataset(df.rpart.features, train.prop = 0.8, seed = seed)
    make.prediction(repo, ., method = method, data.train = df.tvsplit$train, data.test = df.tvsplit$tes
t) %>%
      summary() %>%
      c(seed = seed, Method = method)
  })
}) %>%
  select(Method, seed, Accuracy, Kappa, Sensitivity, Specificity, AUC) %>%
  knitr::kable(longtable = TRUE, caption = "CART/Random Forest (TV Split) Performance", digits = 4)
```

CART/Random Forest (TV Split) Performance

Method	seed	Accuracy	Kappa	Sensitivity	Specificity	AUC
rpart	1	0.910344827586207	0.426888622416591	0.351219512195122	0.981648522550544	0.876252323331942
rpart	2	0.907862068965517	0.440356803449334	0.375296912114014	0.977840199750312	0.863213589900985
rpart	3	0.910896551724138	0.40592976842994	0.330808080808081	0.982037782595231	0.862218890671972
rpart	4	0.900965517241379	0.441020134228188	0.38021978021978	0.975709779179811	0.869986133740077
rpart	5	0.900689655172414	0.41726503054335	0.352808988764045	0.977358490566038	0.870768143594092
rf	1	0.926620689655172	0.543997124710934	0.448780487804878	0.98755832037325	0.924580662291848
rf	2	0.921379310344828	0.507900504090886	0.406175771971496	0.989076154806492	0.917972931697611
rf	3	0.926344827586207	0.515885071878843	0.414141414141414	0.989160730876432	0.910414535569382
rf	4	0.913931034482759	0.502005627204318	0.408791208791209	0.986435331230284	0.923568135334697
rf	5	0.919172413793103	0.521628882648477	0.420224719101124	0.988993710691824	0.916318281393541

The Accuracy and AUC are slightly lower but comparable to the simple CART fit. Thus there is some over-fitting but not too significant. Random Forest still performed better than CART but this time the advantage is much smaller and more reasonable.

Cross-Validation

Hide

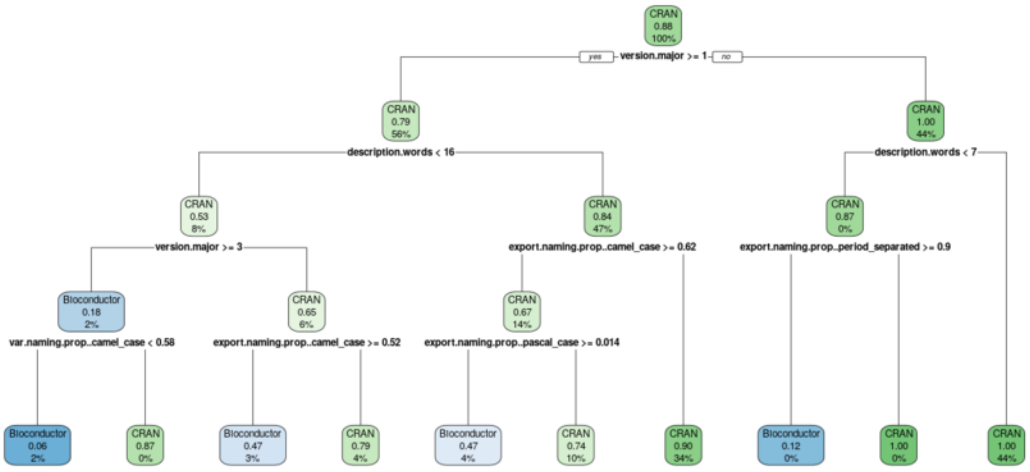
```
set.seed(12390)
tr.control <- trainControl(method = "cv", number = 20)
with.cluster(
  df.cv.rpart <- train(repo ~ .,
    method = "rpart", data = df.rpart.features,
    trControl = tr.control, control = rpart.control(xval = 10, minsplit = 2, maxdepth = 4),
    tuneGrid = data.frame(cp = seq(from = 0.001, to = 0.05, length = 20))
  ),
  nproc = 4
)
```

This time if we use a 20-fold cross-validation we get a tree that is comparable to the logic we saw in the simple CART model but with more splits:

- First branch is the same (version number)
- Second branch (description words) is split into three.
- There is an additional branching for major version >= 3 favoring Bioconductor.

Hide

```
rpart.plot(df.cv.rpart$finalModel)
```



Regression/Numeric Prediction

Feature Selection

We used the following features in the linear regression model:

Hide

```
df.lm.features <- df.features %>% select(
  export.num,
  avg_r_tokens,
  starts_with("native."),
  starts_with("ext."),
  title.words,
  description.words,
  starts_with("export.naming.prop.."),
)
```

Simple Linear Regression

We fit a linear regression model predicting the number of exported functions:

Hide

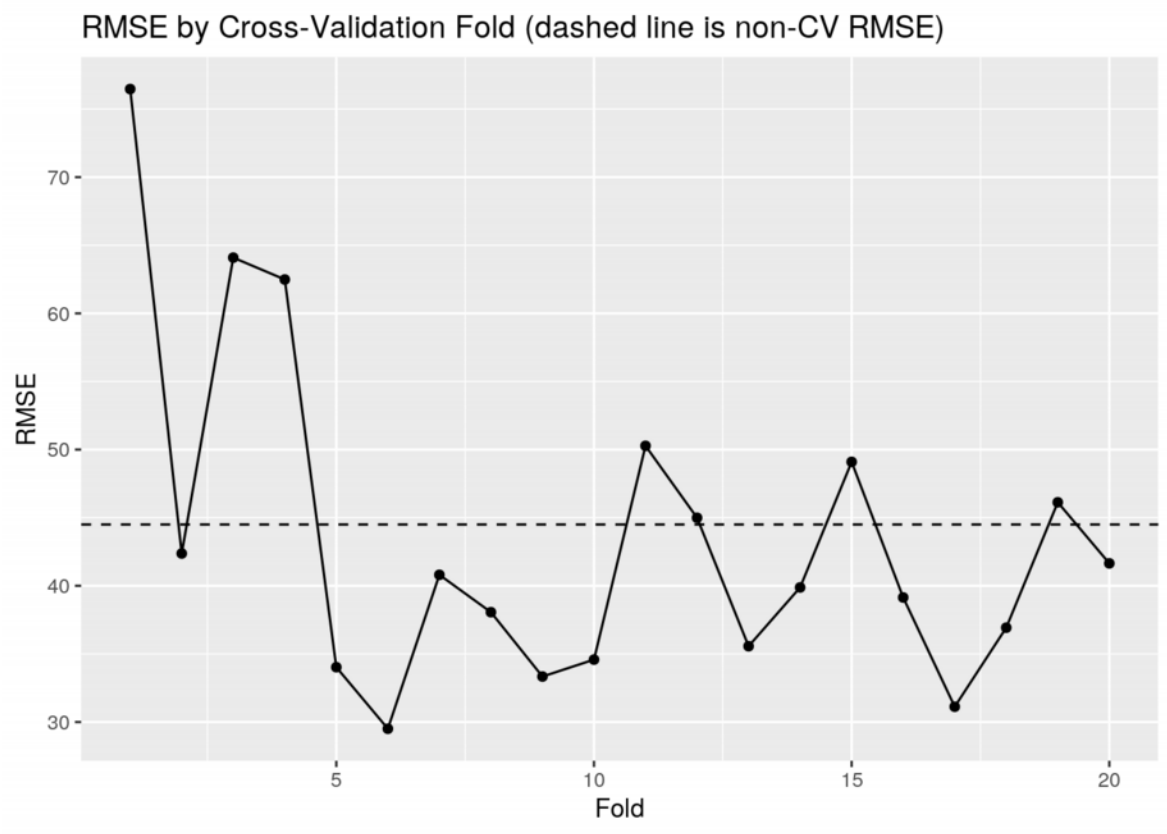
```
df.lm <- lm(export.num ~ ., data = df.lm.features)
df.lm$rmse <- sqrt(mean(df.lm$residuals^2))
summary(df.lm)
```

```
##
## Call:
## lm(formula = export.num ~ ., data = df.lm.features)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -835.77   -8.38   -2.65    2.53 1361.43
##
## Coefficients:
##                                      Estimate Std. Error t value Pr(>|t|)
## (Intercept)                          3.625e+00  1.525e+00   2.377   0.0175 *
## avg_r_tokens                         4.035e-04  8.935e-05   4.516 6.33e-06 ***
## native.c.prop                        1.189e-01  1.674e-01   0.710   0.4776
## native.f.prop                       -2.472e-02  6.155e-01  -0.040   0.9680
## native.j.prop                       -3.490e-02  3.063e-01  -0.114   0.9093
## ext.r                                5.462e-01  1.219e-02  44.808  < 2e-16 ***
## ext.rd                               8.462e-01  1.115e-02  75.881  < 2e-16 ***
## ext.rds                              4.120e-01  4.695e-02   8.776  < 2e-16 ***
## ext.rda                             -5.681e-01  3.603e-02 -15.769  < 2e-16 ***
## title.words                         -2.428e-01  1.208e-01  -2.010   0.0444 *
## description.words                    1.051e-02  8.032e-03   1.309   0.1907
## export.naming.prop..period_separated -5.145e-01  1.626e+00  -0.316   0.7517
## export.naming.prop..camel_case       -7.835e+00  1.431e+00  -5.473 4.48e-08 ***
## export.naming.prop..snake_case       -2.881e+00  1.442e+00  -1.999   0.0456 *
## export.naming.prop..pascal_case      -8.519e+00  2.075e+00  -4.105 4.05e-05 ***
## export.naming.prop..all_caps         -1.451e+00  3.553e+00  -0.408   0.6830
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 44.52 on 18298 degrees of freedom
##   (2772 observations deleted due to missingness)
## Multiple R-squared:  0.4717, Adjusted R-squared:  0.4713
## F-statistic:  1089 on 15 and 18298 DF,  p-value: < 2.2e-16
```

The RMSE is 44.5019886 and the R-squared is 0.4717366.

Hide

```
train_control <- trainControl(method = "cv", number = 20)
with.cluster(
  df.cv.lm <- train(export.num ~ ., data = df.lm.features, method = "lm", trControl = tr.control, na.ac
tion = na.omit),
  nproc = 4
)
df.cv.lm$resample %>%
  mutate(Fold = parse_number(Resample)) %>%
  ggplot(aes(x = Fold, y = RMSE)) +
  geom_point() +
  geom_line() +
  geom_hline(yintercept = df.lm$rmse, linetype = "dashed") +
  labs(x = "Fold", y = "RMSE", title = "RMSE by Cross-Validation Fold (dashed line is non-CV RMSE)")
```



Random Forest Comparison

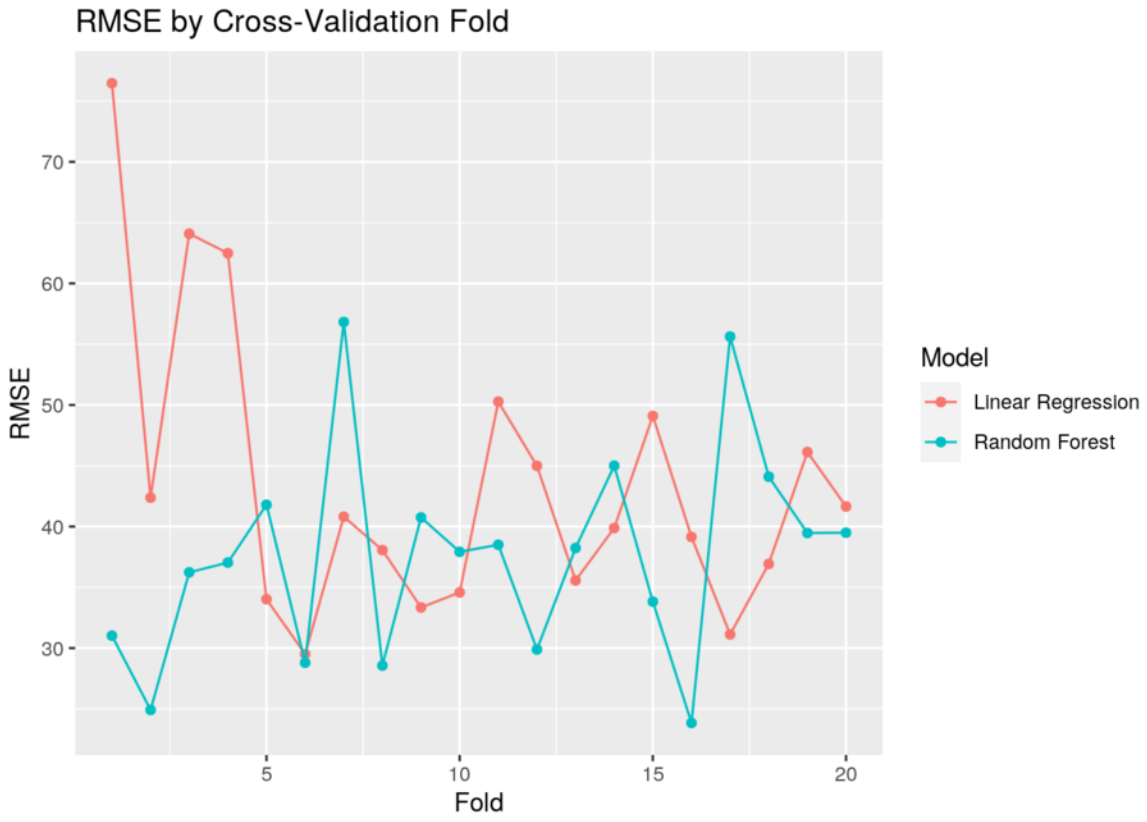
Hide

```
with.cluster(
  df.cv.rf <- train(export.num ~ .,
    data = df.lm.features,
    method = "rf", trControl = tr.control, na.action = na.omit
  ),
  nproc = 16
)
```

If we plot the RMSE over folds we can see that although at first Linear Regression seem to perform considerably worse but after 5 folds the there is no longer any difference.

Hide

```
rbind(
  df.cv.lm$resample %>%
    mutate(Fold = parse_number(Resample), Model = "Linear Regression"),
  df.cv.rf$resample %>%
    mutate(Fold = parse_number(Resample), Model = "Random Forest")
) %>%
  ggplot(aes(x = Fold, y = RMSE, color = Model)) +
  geom_point() +
  geom_line() +
  labs(x = "Fold", y = "RMSE", title = "RMSE by Cross-Validation Fold")
```



Unsupervised Learning

Feature Selection

We used the following features in the PCA model:

Hide

```
df.pca.features <- df.features %>% select(
  matches("^.+naming\\.\\.(period_separated|all_caps|.+_case)$"),
  starts_with("ext."),
  version.major,
  avg_r_tokens,
  export.num,
  title.words, description.words
)
```

Hide

```
df.pca <- prcomp(df.pca.features, center = TRUE, scale = TRUE)
df.pca.pcs <- data.frame(df.pca$x) %>% mutate(repo = df.features$repo)
df.pca.loadings <- df.pca$rotation %>%
  as_tibble(rownames = NA) %>%
  rownames_to_column("var")
```

PC Loadings

It seems like the first principal component is just how much R code there is.

Hide

```
df.pca.loadings %>%
  arrange(desc(abs(PC1))) %>%
  slice_head(n = 10) %>%
  select(var, PC1) %>%
  knitr::kable(longtable = TRUE, caption = "Top 10 Loadings for PC1")
```

Top 10 Loadings for PC1

var	PC1
export.num	0.3575670
var.naming..camel_case	0.3437358
ext.r	0.3293555
ext.rd	0.3244833
rfile.naming..camel_case	0.2796546
export.naming..camel_case	0.2784278
export.naming..period_separated	0.2451391
var.naming..all_caps	0.2262946
var.naming..period_separated	0.2172627
var.naming..pascal_case	0.2131519

The second principal component seems to be whether the package uses snake_case naming convention, which corroborates with the UseR talk mentioned in the introduction that snake_case is the new 1st place naming convention following the rise of Tidyverse.

Hide

```
df.pca.loadings %>%
  arrange(desc(abs(PC2))) %>%
  slice_head(n = 10) %>%
  select(var, PC2) %>%
  knitr::kable(longtable = TRUE, caption = "Top 10 Loadings for PC2")
```

Top 10 Loadings for PC2

var	PC2
export.naming..snake_case	0.4923848
rfile.naming..snake_case	0.3838601
var.naming..snake_case	0.3775792

var	PC2
var.naming..pascal_case	-0.3241324
var.naming..all_caps	-0.3069365
var.naming..period_separated	-0.2165705
var.naming..camel_case	-0.1870310
export.num	0.1727584
avg_r_tokens	-0.1679454
ext.r	0.1669241

The third principal component seems to be whether the package uses PascalCase naming convention.

Hide

```
df.pca.loadings %>%
  arrange(desc(abs(PC3))) %>%
  slice_head(n = 5) %>%
  select(var, PC3) %>%
  knitr::kable(longtable = TRUE, caption = "Top 5 Loadings for PC3")
```

Top 5 Loadings for PC3

var	PC3
rfile.naming..pascal_case	0.5665335
export.naming..pascal_case	0.5587470
rfile.naming..period_separated	-0.2628653
var.naming..period_separated	-0.2429566
avg_r_tokens	-0.1953932

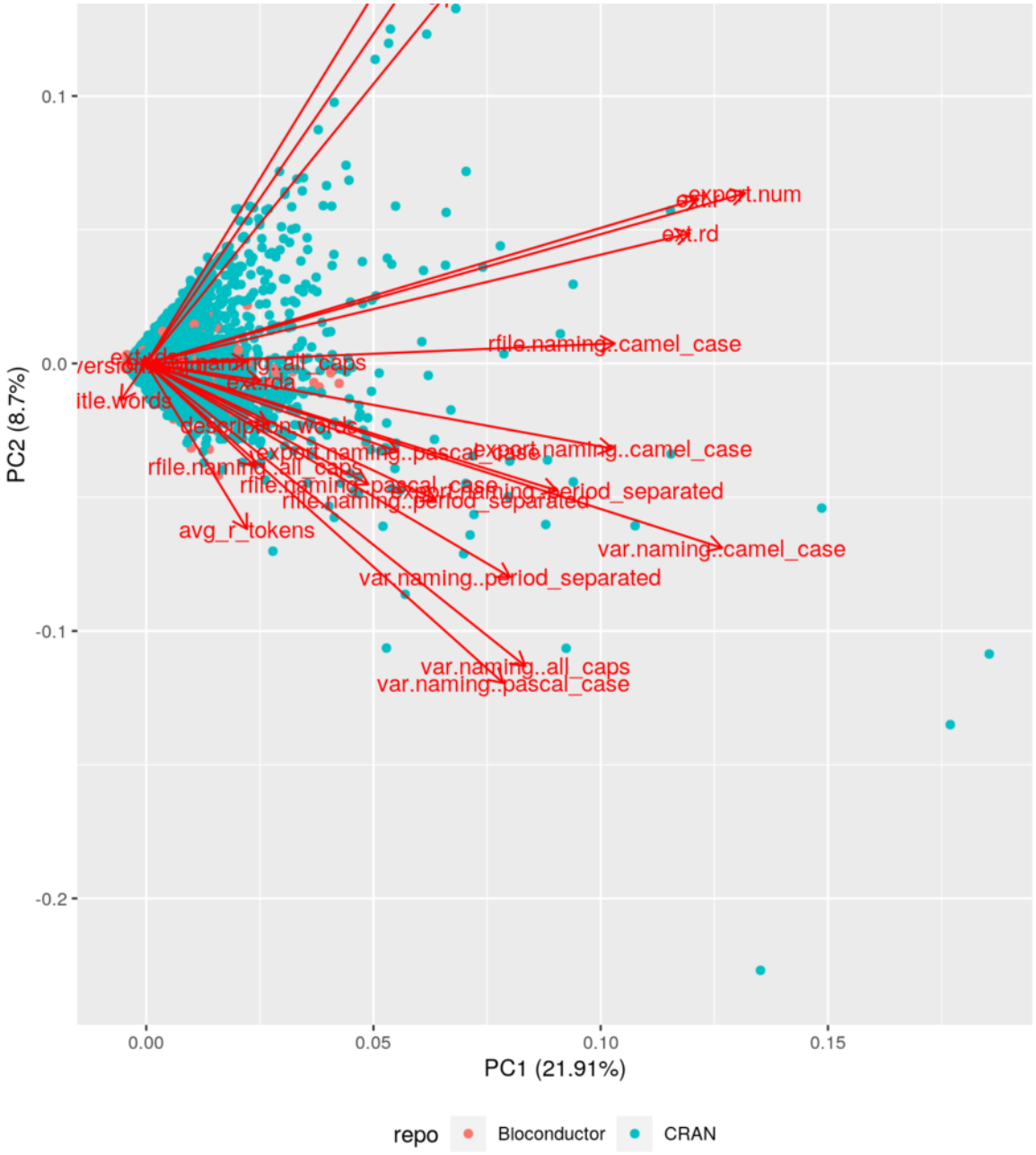
PC Plots

If we plot PC1 and PC2 we can see that PC1 just seems to be the amount of code so almost all variables have a positive loading, and it seems like on this axis CRAN packages are more spread out meaning there are more variance in amount of code in CRAN packages compared to Bioconductor.

Looking at PC2 the top three variables matches the tables above (snake_case export naming, R file naming and variable naming).

Hide

```
autoplot(df.pca,
  loadings = TRUE, loadings.label = TRUE,
  data = df.features, colour = "repo", opacity = 0.7
) +
  theme(legend.position = "bottom")
```

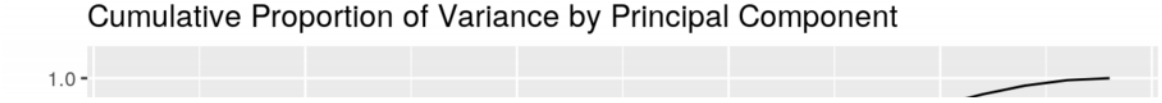


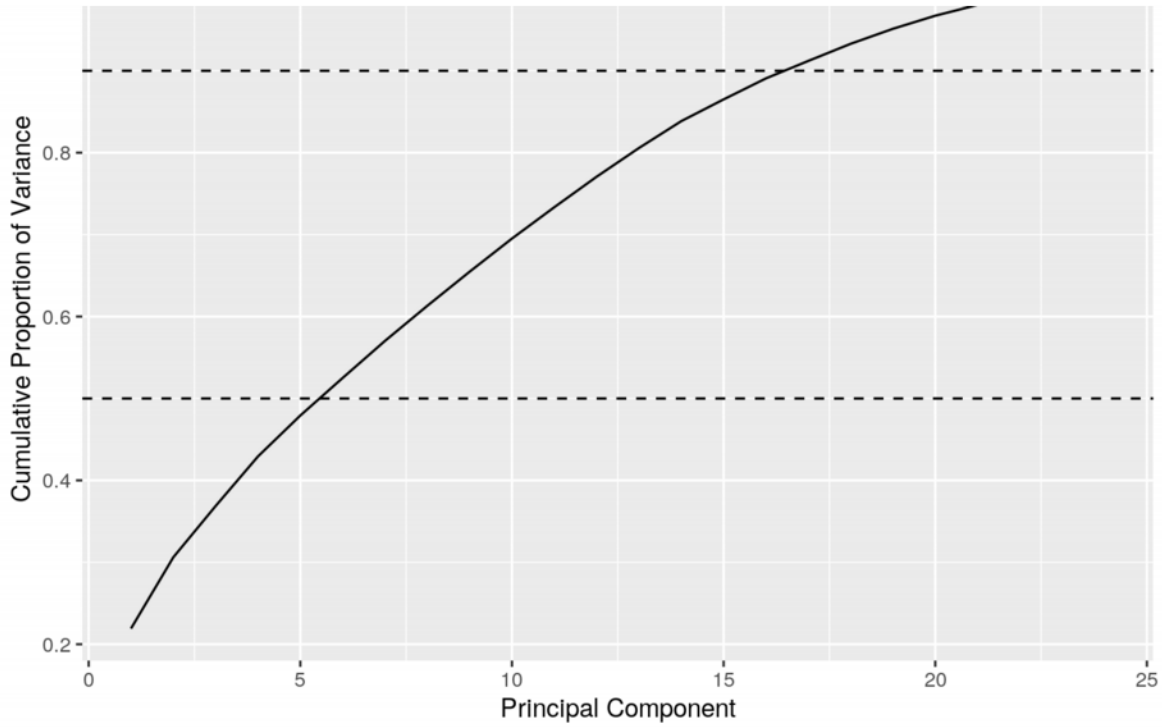
Explained Variance

Plotting the cumulative proportion of variance by principal component we can see that around 50% of the variance can be explained by the first 5 PCs, and 90% of the variance can be explained by the first 16 PCs.

Hide

```
df.pca$sdev^2 %>%
  prop.table() %>%
  cumsum() %>%
  qplot(geom = "line", x = seq_along(.)) +
  geom_hline(yintercept = 0.5, linetype = "dashed") +
  geom_hline(yintercept = 0.90, linetype = "dashed") +
  labs(x = "Principal Component", y = "Cumulative Proportion of Variance", title = "Cumulative Proporti
on of Variance by Principal Component")
```





Concluding Remarks

Classification Tasks

To conclude, we have demonstrated that while there are no obvious monotonic differences between packages published on CRAN and Bioconductor within the feature collection that are useful in building a GLM model, there are intricate differences that could be picked up by Tree-based classifiers such as CART and Random Forest, which are more capable of generating complex classification logic.

R community

We saw that while Bioconductor has more stringent guidelines for package development, the actual proportion of packages that conform to those guidelines are comparable to CRAN. This corroborates with one of the talking points in *The current state of naming conventions in R* that a lot of the differences in coding style are due to historical reasons and that even base R itself do not have a uniform coding/naming style.

PCA Analysis

From the PCA analysis we observed that the first distinguishing differences between the packages is whether their functions and variables conform to snake_case, which supports that claim in *The current state of naming conventions in R* that snake_case is the new 1st place trending convention in R and that could explain why apart from PC1 which is the size of the package, snake_case naming is the major contributor to PC2. As most new packages conform to the tidyverse snake_case style but older packages are not, creating large variance that can be picked up by PCA.