R implementation

Sleiman Bassim, PhD

November 7, 2014

Locally loaded functions:

```
source("/media/Data/Dropbox/R/01funcs.R")
#load("", .GlobalEnv)
lsos(pat="")
```

```
                      Type   Size PrettySize Rows Columns
bagging           function  23520      23 Kb   NA      NA
bagging.clas      function  24280    23.7 Kb   NA      NA
baggingTune       function  23856    23.3 Kb   NA      NA
circos.test       function  55408    54.1 Kb   NA      NA
Cols              function   6576     6.4 Kb   NA      NA
ensemble.mean     function  24232    23.7 Kb   NA      NA
locusRMR          function  11224      11 Kb   NA      NA
model.clas        function  20680    20.2 Kb   NA      NA
model.reg         function  18528    18.1 Kb   NA      NA
modelTune.clas    function  21016    20.5 Kb   NA      NA
modelTune.reg     function  18864    18.4 Kb   NA      NA
net2Bin           function  33240    32.5 Kb   NA      NA
normDataWithin    function  21816    21.3 Kb   NA      NA
olBarplot         function  31992    31.2 Kb   NA      NA
overLapper        function  86584    84.6 Kb   NA      NA
predict.regsubsets function 10208      10 Kb   NA      NA
range_correlation function  16472    16.1 Kb   NA      NA
regfit            function   9912     9.7 Kb   NA      NA
regfit.int        function  10360    10.1 Kb   NA      NA
rocplot           function   7768     7.6 Kb   NA      NA
set.cor           function  10712    10.5 Kb   NA      NA
set.var           function  12216    11.9 Kb   NA      NA
summarySE         function  30400    29.7 Kb   NA      NA
summarySEwithin   function  39616    38.7 Kb   NA      NA
vennPlot          function 562632   549.4 Kb   NA      NA
visualizeNet      function  30288    29.6 Kb   NA      NA
```

1 By-group aggregation

By-group aggregation uses the principle of split-and-conquer or **split-apply-combine** a term coined by Hadley Wickham. R jobs will run in parallel across several cores which reduces the calculation time of the row search.

```
pkgs <- c('rbenchmark', 'doParallel', 'foreach', 'Hmisc', 'dplyr', 'plyr')
```

```
lapply(pkgs, require, character.only = TRUE);
```

```
[[1]]
[1] TRUE

[[2]]
[1] TRUE

[[3]]
[1] TRUE

[[4]]
[1] TRUE

[[5]]
[1] TRUE

[[6]]
[1] TRUE
```

Get data from this site here. It is in *.xlsx form. Use `read.xlsx` from the package `xlsx` to read in the table into an R dataframe. Put it in a wrapped dataframe with `dplyr` and saved it in a .Rdata file.

```
load("LoF_Phase1_v3.Rdata", .GlobalEnv)
lsos(pattern="input.*")
```

```
               Type    Size PrettySize  Rows Columns
input.df data.frame 2432000     2.3 Mb 10027      20
```

Sampling 1000 rows from the dataset is followed with a grouping and summarizing functions to show detail of the data and the levels of classification.

```
indices.df <- sample(1:nrow(input.df),1000)
```

```
#input.df <- input.df[complete.cases(input.df), ]
input.subset.df <- tbl_df(input.df[indices.df, ])
str(input.subset.df)
```

```
Classes 'tbl_df', 'tbl' and 'data.frame': 1000 obs. of  20 variables:
 $ chr             : Factor w/ 23 levels "1","10","11",..: 3 4 12 2 13 1 18 22 12 15 ...
 $ pos             : num  67395456 51723598 209204280 95097599 3838285 ...
 $ rsID            : Factor w/ 5383 levels ".","rs10009430",..: 481 1 1 889 1 1 1145 1 1 4247 ...
 $ ref_allele      : Factor w/ 145 levels "A","AAACC","AAC",..: 1 1 75 1 33 33 33 75 17 33 ...
 $ alt_allele      : Factor w/ 89 levels "A","AAC","AAG",..: 65 9 1 20 65 65 65 1 1 1 ...
 $ ancestral_allele: Factor w/ 125 levels "-",".","A","a",..: 3 3 67 3 29 29 29 67 16 30 ...
 $ effect          : Factor w/ 2 levels "full","partial": 2 1 2 2 2 2 2 1 1 2 ...
 $ type            : Factor w/ 3 levels "frameshift_indel",..: 3 1 2 3 3 3 2 3 1 3 ...
 $ an              : num  2184 2184 2184 2184 2184 ...
 $ ac              : num  56 327 1 4 1 2 2 1 15 3 ...
 $ gene            : Factor w/ 6433 levels "A1BG","A2M","A2ML1",..: 3914 1462 4342 3722 3444 2317 5211 1
 $ gene_id         : Factor w/ 6433 levels "ENSG00000000419.7",..: 3963 2530 1423 2449 616 3511 5115 507
 $ lof_trans       : num  1 1 1 4 3 1 1 1 1 2 ...
 $ all_trans       : num  2 1 5 7 4 2 3 1 1 7 ...
 $ failed_filters  : num  0 0 0 0 0 0 0 0 0 0 ...
 $ filters_failed  : Factor w/ 69 levels "0","all_alt",..: 1 1 1 1 1 1 1 1 1 1 ...
 $ splice_type     : Factor w/ 3 levels ".","acceptor",..: 1 1 3 1 1 1 3 1 1 1 ...
 $ canonical       : Factor w/ 3 levels ".","NO","YES": 1 1 3 1 1 1 3 1 1 1 ...
 $ other_canonical : Factor w/ 3 levels ".","NO","YES": 1 1 3 1 1 1 3 1 1 1 ...
 $ intron_length   : Factor w/ 2266 levels ".","1","100",..: 1 1 1523 1 1 1 1910 1 1 1 ...
```

```
unique(input.subset.df$effect)
```

```
[1] partial full
Levels: full partial
```

```
unique(input.subset.df$chr)
```

```
 [1] 11 12 2  10 20 1  5  9  22 17 X  7  18 8  16 14 13 19 3  15 6  4  21
Levels: 1 10 11 12 13 14 15 16 17 18 19 2 20 21 22 3 4 5 6 7 8 9 X
```

10 Extract subsets of the dataframe depending on the "effect" column.

```
input.summary <- function(x){
```

```
  y <- unique(x)
  sub = list(NULL)
  for(i in 1:length(y)){
    sub[[i]] <- subset(input.subset.df, x == y[i])
  }
  sub
}

testing <- input.summary(input.subset.df$effect)
testing[1]

[[1]]
Source: local data frame [451 x 20]

     chr       pos        rsID ref_allele alt_allele ancestral_allele  effect
5721  11  67395456 rs118032721          A          T                A partial
1538   2 209204280           .          G          A                G partial
5155  10  95097599 rs140386623          A          C                A partial
9330  20   3838285           .          C          T                C partial
887    1 212798487           .          C          T                C partial
3031   5 150711989 rs142258981          C          T                C partial
9760  22  36595608 rs192225524          C          A                c partial
8165  17  61988546 rs139915511          C          T                c partial
124    1  19810813           .          C         CT                C partial
5171  10  97959237  rs80337218          C          A                C partial
..   ...       ...         ...        ...        ...              ...     ...
Variables not shown: type (fctr), an (dbl), ac (dbl), gene (fctr), gene_id (fctr),
  lof_trans (dbl), all_trans (dbl), failed_filters (dbl), filters_failed (fctr),
  splice_type (fctr), canonical (fctr), other_canonical (fctr), intron_length (fctr)

testing[2]

[[1]]
Source: local data frame [549 x 20]

     chr       pos        rsID ref_allele alt_allele ancestral_allele effect
6192  12  51723598           .          A         AG                A   full
4668   9  43627213           .          G          A                G   full
1597   2 223559986           .         AG          A               AG   full
9978   X 118222555 rs147204920          G          A                G   full
9445  20  36869005  rs41282820          G          A                G   full
3840   7  37936527 rs142525551          C          T                C   full
4796   9 125273385           .          A         AT                A   full
8464  18  74090957           .         CG          C                C   full
4640   9  35376166           .          C          T                C   full
2990   5 140870416           .          C          T                C   full
..   ...       ...         ...        ...        ...              ...    ...
Variables not shown: type (fctr), an (dbl), ac (dbl), gene (fctr), gene_id (fctr),
  lof_trans (dbl), all_trans (dbl), failed_filters (dbl), filters_failed (fctr),
  splice_type (fctr), canonical (fctr), other_canonical (fctr), intron_length (fctr)
```

2 Different grouping and summarizing Functions

There is several functions to be used to extract a specific amount of data. The `tapply, agreggate, ddply` functions:

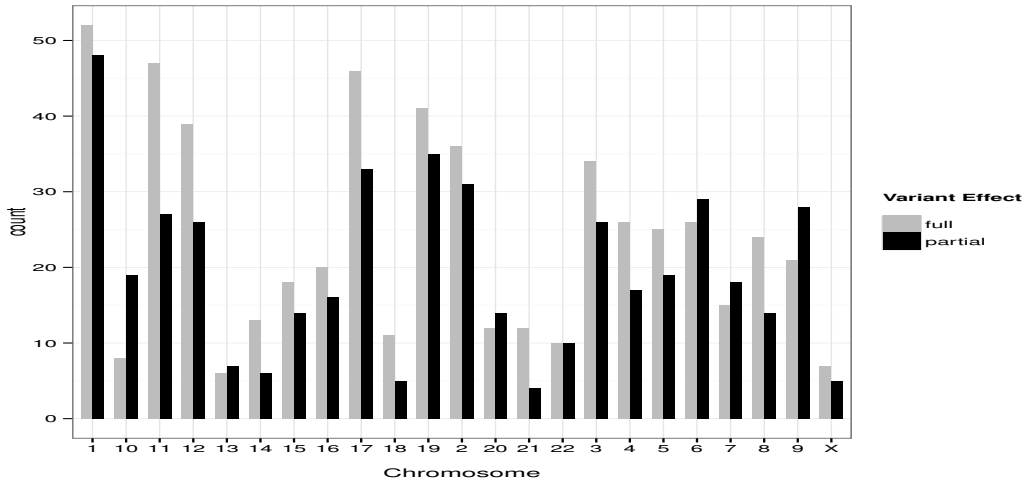
```
tapply(input.subset.df$pos, input.subset.df$effect, mean)

    full  partial
74975657 75445181
```

Extract and visualize a summary of the partial and full effects of variants relative to the nature of the chromosome.

```
require(plyr)
```

```
output.df <- ddply(
    input.subset.df,
    c("effect","chr"),
    function(x){
        c(count = nrow(x))
    })
## Plot
require(ggplot2)
ggplot(output.df) +
    geom_bar(aes(x = chr, y = count, fill = effect),
             stat = 'identity', position = 'dodge', width = .7) +
    scale_fill_manual("Variant Effect\n", values = c("grey", "black"),
                      labels = c('full', 'partial')) +
    labels(x = '\nChromosome', 'Count\n') +
    theme_bw()
```



Visualize a count summary relative to the nature of variants.

```
output.df.variants <- ddply(
```

```
    input.subset.df,
    c('effect','alt_allele'),
    function(x){
    c(count = nrow(x))
    })
head(output.df.variants)
```

```
  effect alt_allele count
1   full          A   227
2   full        ACT     1
3   full         AG     1
4   full         AT     2
5   full     ATAAGT     1
6   full          C    46
```

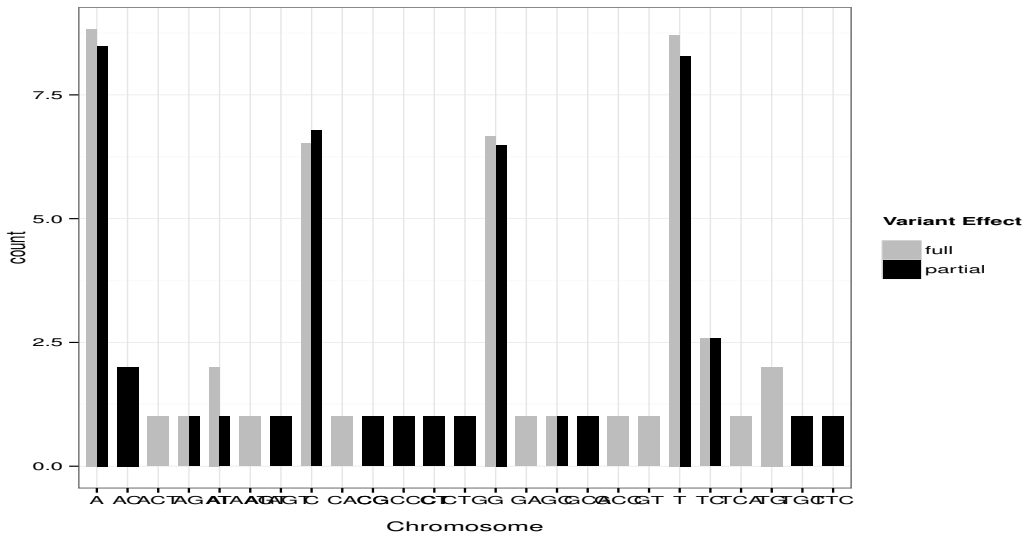
```
require(reshape)
output.df.reshape <- cast(output.df.variants, alt_allele~effect)
head(output.df.reshape)
```

```
  alt_allele full partial
1          A  227     179
2         AC   NA       2
3        ACT    1      NA
4         AG    1       1
5         AT    2       1
6     ATAAGT    1      NA
```

```
output.df.reshape[is.na(output.df.reshape)] <- 0

require(vegan)
output.df.stand <- decostand(output.df.variants$count, method="log")
output.df.log <- data.frame(effect = output.df.variants$effect,
                            pos= output.df.variants$alt_allele,
                            count= output.df.stand[,1])

ggplot(output.df.log) +
    geom_bar(aes(x = pos, y = count, fill = effect),
             stat = 'identity', position = 'dodge', width = .7) +
    scale_fill_manual("Variant Effect\n", values = c("grey", "black"),
                      labels = c('full', 'partial')) +
    labs(x = '\nChromosome', 'Count\n') +
    theme_bw()
```



Create a data frame from a list.

```
a.list <- list(x=output.df.reshape[,1], y=output.df.reshape[,2])
system.time(a.df <- tbl_df(do.call("rbind.fill",lapply(a.list, as.data.frame))))

   user  system elapsed
  0.000   0.000   0.002
```

```
registerDoParallel(cores = 2)
benchmark(replications=100, lapply(1:3, sqrt), foreach(i=1:3) %do% sqrt(i))

                          test replications elapsed relative user.self sys.self
2 foreach(i = 1:3) %do% sqrt(i)          100   0.467       NA     0.464        0
1             lapply(1:3, sqrt)          100   0.000       NA     0.000        0
  user.child sys.child
2          0         0
1          0         0
```

2.1 More dplyr verbs for data manipulation

Verbs in `dplyr` constitute the essential functions for data manipulation. Five exist and are basic dataframe manipulation functions. **Filter** to choose rows, **select** to choose from columns, **arrange** to order rows, **mutate** and **summarize** to make new columns.

```
input.subset.df %>%
    select(chr, pos, ref_allele, effect) %>%
    filter(ref_allele == "A") %>%
    group_by(chr, effect) %>%
    mutate(pos2 = rank(pos)) %>%
    filter(pos < 8.7*10^6 & pos > 8*10^6 ) %>%
    arrange(chr, effect, pos2)

Source: local data frame [0 x 5]
Groups: chr, effect
```

3 Other options of parallel computing

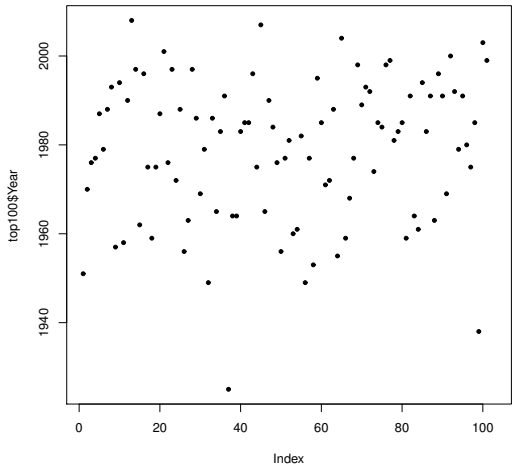
For MySQL try `dbApply()` to aggregate. And `sqldf` will create a temporary database. Or bigmemory.org to load data into memory, or use Hadoop.

4 Top 100 papers cited in Nature

Download data from onedrive in xlsx form.

```
#require(xlsx)
#top100 <- read.xlsx("~/Downloads/WebofSciencetop100.xlsx", sheetIndex = "Sheet1")
load("WebofScienceTop100.Rdata", .GlobalEnv)
top100 <- tbl_df(top100)
plot(top100$Year,top100$Rnk, pch = 20)
abline(lm(top100$Rank~top100$Year), col = "red");

Warning in model.response(mf, "numeric"):  using type = "numeric" with a factor
response will be ignored
Warning in Ops.factor(y, z$residuals):  - not meaningful for factors
```

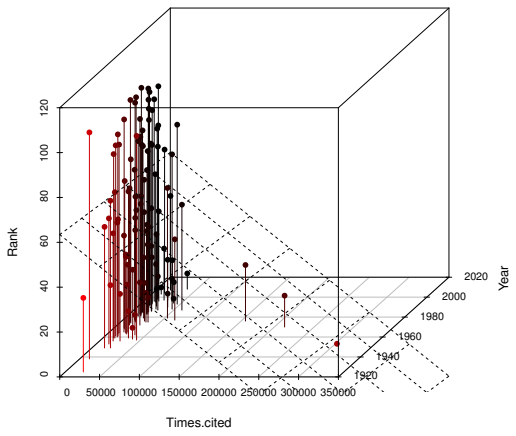


Plot in 3d. `rgl` provides an interactive 3d box.

```
require(scatterplot3d)
scat <- with(top100,
             scatterplot3d(Times.cited,
                           Year, Rank, pch=16,
                           highlight.3d=TRUE,
                           type="h"
                           )
             )
fit <- with(top100, lm(Rank~Times.cited+Year))
```

```
Warning in model.response(mf, "numeric"):  using type = "numeric" with a factor
response will be ignored
Warning in Ops.factor(y, z$residuals):  - not meaningful for factors
```

```
scat$plane3d(fit)
```



5 System Information

The version number of R and packages loaded for generating the vignette were:

```
R version 3.1.1 (2014-07-10)
Platform: x86_64-pc-linux-gnu (64-bit)

locale:
 [1] LC_CTYPE=en_CA.UTF-8       LC_NUMERIC=C               LC_TIME=en_CA.UTF-8
 [4] LC_COLLATE=en_CA.UTF-8     LC_MONETARY=en_CA.UTF-8    LC_MESSAGES=en_CA.UTF-8
 [7] LC_PAPER=en_CA.UTF-8       LC_NAME=C                  LC_ADDRESS=C
[10] LC_TELEPHONE=C             LC_MEASUREMENT=en_CA.UTF-8 LC_IDENTIFICATION=C

attached base packages:
 [1] splines   grid      parallel  stats     graphics  grDevices utils     datasets
 [9] methods   base

other attached packages:
 [1] scatterplot3d_0.3-35 vegan_2.0-10         permute_0.8-3
 [4] reshape_0.8.5        ggplot2_1.0.0        plyr_1.8.1
 [7] dplyr_0.2            Hmisc_3.14-5         Formula_1.1-2
[10] survival_2.37-7      lattice_0.20-29      doParallel_1.0.8
[13] iterators_1.0.7      foreach_1.4.2        rbenchmark_1.0.0
[16] knitr_1.7

loaded via a namespace (and not attached):
 [1] acepack_1.3-3.3     assertthat_0.1      cluster_1.15.3      codetools_0.2-9
 [5] colorspace_1.2-4    compiler_3.1.1      digest_0.6.4        evaluate_0.5.5
 [9] foreign_0.8-61      formatR_1.0         gtable_0.1.2        highr_0.3
[13] labeling_0.3        latticeExtra_0.6-26 magrittr_1.0.1      MASS_7.3-35
[17] munsell_0.4.2       nnet_7.3-8          proto_0.3-10        RColorBrewer_1.0-5
[21] Rcpp_0.11.3         reshape2_1.4        rpart_4.1-8         scales_0.2.4
[25] stringr_0.6.2       tools_3.1.1
```