R implementation: Predict risk of Parkinson's disease based on clinical measurements

Sleiman Bassim, PhD

December 22, 2014

Loaded functions:

```
#source("/media/Data/Dropbox/humanR/01funcs.R")
rm(list=ls())
#setwd("/media/Data/Dropbox/humanR/PD/")
#setwd("~/Dropbox/humanR/PD/")
load("PD.Rdata", .GlobalEnv)
#lsos(pat="")
```

1 Data preprocessing

Load packages.

```
pkgs <- c('xlsx','caret','leaps','glmnet','lattice','latticeExtra')
lapply(pkgs, require, character.only = TRUE)
```

```
[[1]]
[1] FALSE

[[2]]
[1] TRUE

[[3]]
[1] TRUE

[[4]]
[1] TRUE

[[5]]
[1] TRUE

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

Read in the data. Clean some columns and create a sample testing set.

```
# read in the data
#raw.df <- read.xlsx("~/Dropbox/humanR/PD/Phenotypes.xlsx", sheetIndex = "Phenotypes")
#colnames(raw.df) <- paste("Ph", seq(1, ncol(raw.df)), sep="")
raw.df <- raw.df[-1,-c(1,97,98)]
dim(raw.df)
```

```
[1] 4011  107
```

↰ Description of column names can be found here.

```
# change the column name
# random sampling
sample.first <- sample(1:dim(raw.df)[1], 100)
```

Remove NAs. And if not, look for how many samples have complete records of clinical measurements. When all NAs are removed only 1 sample remains. So how to account for missing data? Either by removing missing data, imputation or through reduced-feature models Saar 2007

```
# if I remove all NAs how many samples remain?
```

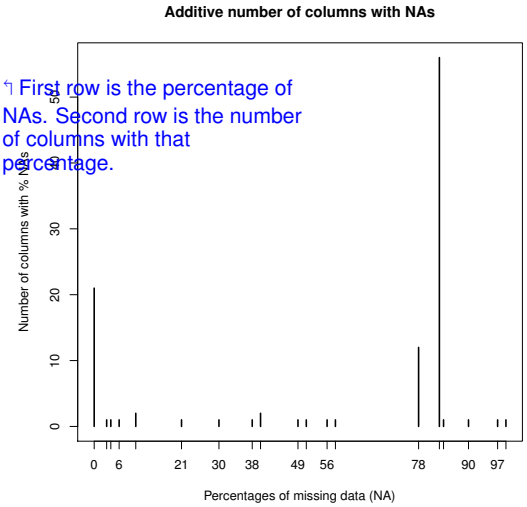
```
raw.df.na <- raw.df[complete.cases(raw.df), ]
dim(raw.df.na)
```

```
[1]   0 107
```

```
# sample the data for faster testing
df.sample <- raw.df[sample.first, ]
NN <- raw.df # df.sample or raw.df
```

↰ Change the number of samples. Choose from a small subset or the whole dataset

```
# count the number of NAs per column
Countcolnas <- function(df){
    m <- apply(is.na(df), 2, function(x) {
        y=mean(x)
        floor(y*100)})
table(m)
}
# number of NAs per column
plot(Countcolnas(NN),
     xlab = "Percentages of missing data (NA)",
     ylab= "Number of columns with % NAs",
     main = "Additive number of columns with NAs")
```



↰ First row is the percentage of NAs. Second row is the number of columns with that percentage.

Every column of the dataset contain missing information. Select the columns that contain below 5% missing data, remove the remaining.

```
# extract the names of the columns with 0% NAs
Extcolnames <- function(df, p){
    m <- apply(is.na(df), 2, function(x){
        y = mean(x)
        floor(y*100)})
    names(df[m < p])
}
p = 20
names.non.na <- Extcolnames(NN,p)
# extract 0% non NA subset
non.na.df <- NN[, names.non.na]
dim(non.na.df)
```

```
[1] 4011   26
```

↰ When using the random subset, more columns are chosen. In the raw dataset only 27 columns are below 5%

What is the maximum number of missing data for a *p* threshold of Sexpnrow(non.na.df) rows. Then again, is the missing data random or not.

```
floor(nrow(non.na.df) * (p*.01))
```

```
[1] 802
```

```
# if all NAs are removed
clean.df <- non.na.df[complete.cases(non.na.df),]
dim(clean.df)
```

```
[1] 3497   26
```

```
# then I'll be removing
dim(non.na.df)[1] - dim(clean.df)[1]
```

```
[1] 514
```

↰ 222 NAs might not be missing at random.

1.1 Create dummy variables

Preprocessing of the data to remove factors from the dataframe and convert them to numeric, remove near-zero variables, and correlated predictors. Then create K binary features for each categorical attribute. K is the number of levels for each attribute.

```
head(clean.df)
```

```
  Ph2 Ph3 Ph4 Ph5 Ph8 Ph9 Ph10 Ph11        Ph12           Ph13 Ph16
4   1   1   2  55   1   1     5    1      Norway Mixed European    2
5   1   1   2  68   1   1     5    1 Netherlands    Netherlands    1
6   1   1   2  67   1   1     5    9     Germany Mixed European    1
7   1   1   2  78   1   1     5    9      Norway         Norway    1
8   1   1   1  41   1   1     5    9      France        England    2
9   1   1   2  51   1   1     5    9     Unknown        Unknown    1
  Ph18 Ph20 Ph21 Ph23 Ph24 Ph25 Ph26 Ph28 Ph30 Ph31 Ph32    Ph33 Ph34
4    2    2    2    2    2    2    2    1   GG e3e4 H1H1 259_259    0
5    9    9    9    9    9    9    9    9   GG e3e4 H1H2 257_259    0
6    9    9    9    9    9    9    9    9   GG e2e3 H2H2 257_259    0
7    9    9    9    9    9    9    9    9   GG e3e3 H1H2 259_259    0
8    2    2    2    1    2    2    2    2   GG e3e4 H1H1 257_261    1
9    9    9    9    9    9    9    9    9   GG e4e4 H1H2 257_261    1
  Ph35 Ph39
4    1    1
5    1    1
6    1    1
7    1    1
8    1    1
9    1    1
```

```
numericals <- c(1:8,11:19,24:26) # for 20% NAs
#numericals <- c(1:8,11:21) # for 0% NAs
clean.df[, numericals] <- sapply(clean.df[, numericals], as.numeric)
dim(clean.df)
```

```
[1] 3497   26
```

```
summary(clean.df[, -numericals])
```

```
             Ph12                   Ph13                Ph30
 Unknown       :2141   Unknown       :2176   AA            :   1
 Mixed European: 310   Mixed European: 306   GA            :  21
 England       : 208   England       : 207   GG            :3475
 Germany       : 201   Germany       : 178   LRRK2_G2019S:   0
 Ireland       : 125   Ireland       : 136
 Italy         :  96   Italy         :  84
 (Other)       : 416   (Other)       : 410
          Ph31              Ph32           Ph33
 APOE_e2e3e4:   0   H1H1      :2200   259_259:1509
 e2e2       :  17   H1H2      :1161   257_259:1234
 e2e3       : 473   H2H2      : 136   259_261: 333
 e2e4       :  84   MAPT_H1H2:   0   257_257: 259
 e3e3       :2083                    257_261: 125
 e3e4       : 774                    261_261:  18
 e4e4       :  66                    (Other):  19
```

```
dummy.df <- dummyVars(Ph2~., data = clean.df)
# number of dummy columns
clean.df.dummy <- predict(dummy.df, newdata = clean.df)
dim(clean.df.dummy)
```

```
[1] 3497  148
```

1.2 Prepare discovery and replication sets

How many zero- and near-zero variant variables. The frequency ratio and the percent of unique variables are both used to decide which predictors to remove. First, a certain threshold is pre-selected. Values that falls outside of it are discarded.

```
y <- clean.df$Ph2
```

```
# before removing near-zero values
dim(clean.df.dummy)[2]
```

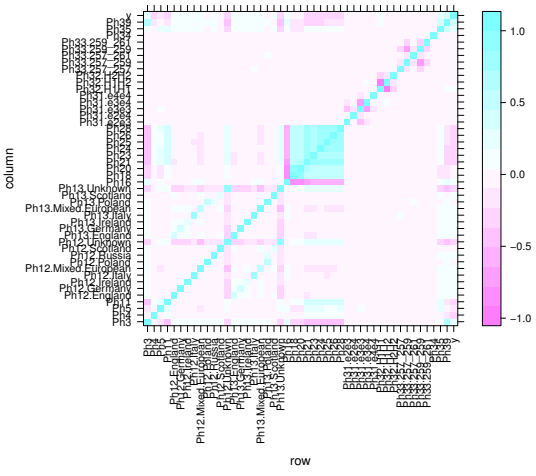
```
[1] 148
```

```
clean.df.dummy2 <- data.frame(clean.df.dummy, y = y)
nzp <- nearZeroVar(clean.df.dummy2, freqCut=99/1,
                   uniqueCut=10,
                   saveMetrics=FALSE)
clean.df.dummy <- clean.df.dummy2[, -nzp]
# after removing near-zero values
dim(clean.df.dummy)[2]
```

```
[1] 47
```

Identifying multi-collinearity.

```
Rm.collinearity <- function(d, cutoff){
    vrange <- range(apply(d, 2, var))
    corx <- cor(d)
    s1 <- summary(corx[upper.tri(corx)])
    num <- sum(abs(corx[upper.tri(corx)]) > cutoff)
    ncor <- findCorrelation(corx, cutoff = cutoff)
    corx <- cor(d[, -ncor])
    s2 <- summary(corx[upper.tri(corx)])
    le <- levelplot(cor(d),
                    aspect = "iso",
                    scales = list(x = list(rot = 90)),
                    colorkey = TRUE)
    listing = list(Correlation_Plot=le,
         Varriation_Range = vrange,
         Nb_of_variables_at_selected_cutoff = num,
         Correlation_Range_old = s1,
         Correlation_Range_new = s2,
         List_of_correlated_variables = ncor,
         New_non_collinear_matrix = d[, -ncor])
}
remaining <- Rm.collinearity(clean.df.dummy,.8)
remaining[[1]]
```

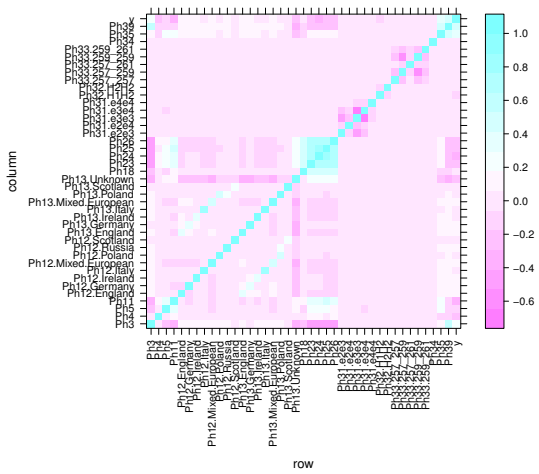


After removing collinear variables at a cutoff of .8/1.

```
remaining[2:5]
```

```
$Varriation_Range
[1]   0.0102 155.9504

$Nb_of_variables_at_selected_cutoff
[1] 7

$Correlation_Range_old
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 -0.918  -0.030  -0.004   0.003   0.019   0.971

$Correlation_Range_new
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 -0.647  -0.028  -0.005   0.001   0.017   0.797
```

```
levelplot(cor(remaining[[7]]),aspect = "iso",
                    scales = list(x = list(rot = 90)),
                    colorkey = TRUE)
```



Split the data into training and testing datasets.

```
# the treatment outcome
trainList <- sample(1:nrow(clean.df.dummy), .7*dim(clean.df.dummy)[1])
training <- clean.df.dummy[trainList,]
dim(training)
```

```
[1] 2447   47
```

```
#testing <- clean.df.dummy[-trainList,-dim(clean.df.dummy)[2]]
testing <- clean.df.dummy[-trainList,]
dim(testing)
```

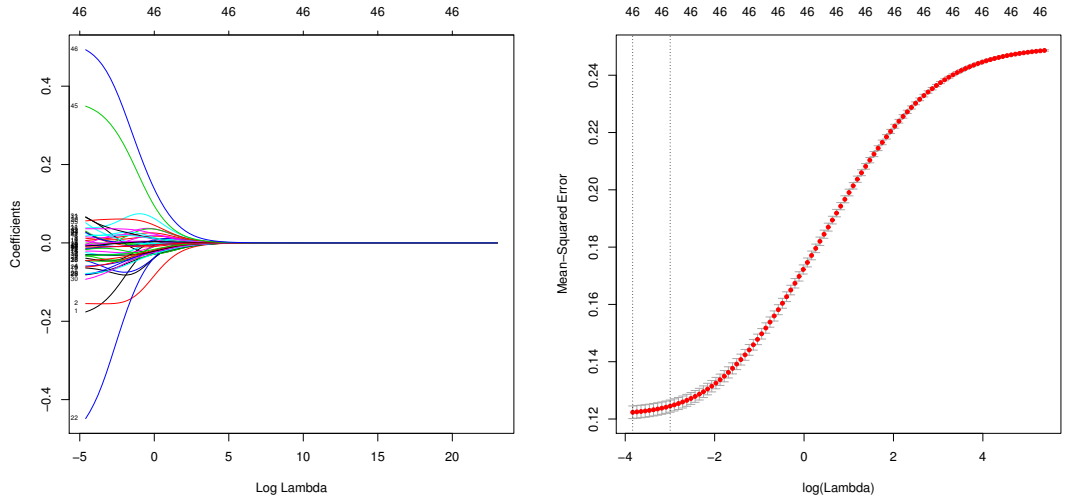
```
[1] 1050   47
```

2 Ranking of phenotypes

Best subset selection performs a ranking of variables (attributes) according to their R^2.

```
training.m <- as.matrix(training)
```

```
testing.m <- as.matrix(testing)
# fit a model on training set
set.seed(1234)
grid <- 10^seq(10,-2,length=100)
ridge.phenotypes <- glmnet(training.m[,-dim(training.m)[2]],
                           training.m[,dim(training.m)[2]],
                           alpha = 0,
                           lambda = grid,
                           family="gaussian")
plot(ridge.phenotypes, xvar="lambda", label=TRUE)
# get the best lambda using CV
cv.out <- cv.glmnet(training.m[,-dim(training.m)[2]],
                    training.m[,dim(training.m)[2]],
                    alpha=0,
                    family="gaussian",
                    nfolds=20)
plot(cv.out)
```



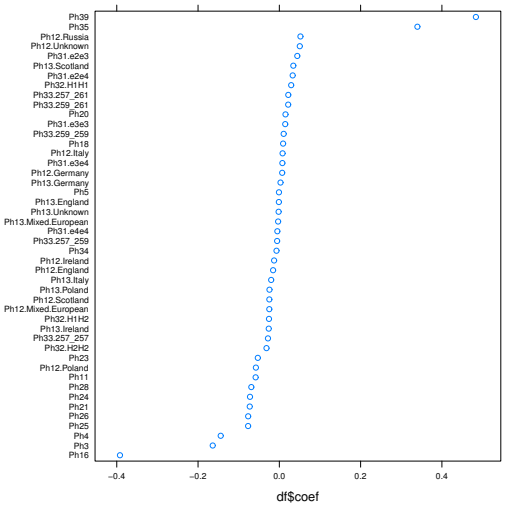
Predict using the testing set and the model with best lambda

```
bestlam <- cv.out$lambda.min;bestlam

[1] 0.0217

pred.phenotypes <- predict(ridge.phenotypes,
                           bestlam,
                           newx=testing.m[,-dim(training.m)[2]],
                           type="response")
mean((pred.phenotypes - testing.m[,dim(training.m)[2]])^2)

[1] 0.12

# predict on the whole dataset using the best lambda
ev.mat <- as.matrix(clean.df.dummy)
ev.reg <- glmnet(ev.mat[,-dim(training.m)[2]],
                 ev.mat[,dim(training.m)[2]],
                 alpha=0,
                 family="gaussian")
ev.pred <- predict(ev.reg, s=bestlam,type="coefficients")
```

Prepare data for plotting all these steps are required because the predict function creates an S4 object.

```
ranking <- as.matrix(ev.pred)
```

```
df <- data.frame(phenotypes=rownames(ranking), coef=ranking[,1])
df <- df[-1,]
df <- df[! df$coef == 0,]
df <- df[order(df[,2], decreasing=FALSE),]
# plot
#setEPS()
#postscript("ranking7.eps")
stripplot(factor(df$phenotypes,
                 levels=df$phenotypes,
                 order=T) ~ df$coef,
          df,scales=list(cex=0.7))
#dev.off()
```



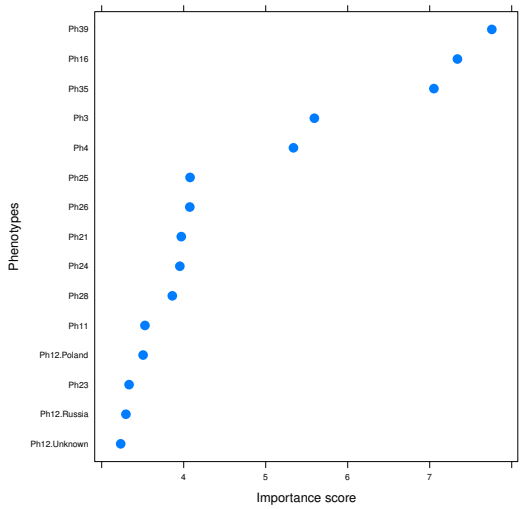
Extract the top ranking predictors after root square to remove negative values and to group the predictors by their highest impact coefficients. Higher impact predictors are the one with the biggest coefficient with a negative or positive sign.

```
Remove.low.coef <- function(dfx, p){
    # p must be from 1 to 100
    # dfx must contain first col of names
    # and second col as coefficients
    dfx[, 2] <- (df[, 2]*100)^2
    dfb <- dfx[ !dfx[,2] < p, ]
    dfb[, 1]
}
px = 20
sp <- Remove.low.coef(df,px)
```

Plot phenotypes by their importance score after logarithmic transformation of the square of their coefficients.

```
Importance.pheno <- function(dfx, p){
```

```
    # p must be from 1 to 100
    # dfx must contain first col of names
    # and second col as coefficients
    dfx[, 2] <- (df[, 2]*100)^2
    dfb <- dfx[ !dfx[,2] < p, ]
    dfb[, 2] <- log((dfb[, 2]))
    dfb[, 1:2]
    dfb[order(dfb[, 2], decreasing=FALSE), ]
}
imp <- Importance.pheno(df,px)
stripplot(factor(imp$phenotypes,
                 levels=imp$phenotypes,
                 order=T) ~ imp$coef,
          imp,scales=list(cex=0.7),
          xlab = "Importance score",
          ylab = "Phenotypes",
          pch = 16, cex= 1.5)
```



3 Testing for prediction accuracy

Write a function that sets the seed (randomly) for reproducibility, takes a model name and the hyperparameters in a list. This function runs 10-folds cross validation (CV) resampling and repeats it 10 times. There is a preprocessing step for centring and scaling numerical data. This function is for regression purposes. No categorical data allowed. It produces a Plot for RMSE estimation as function of the tuned (with CV) hyperparameters.

```
Train.model <- function(seeds, model, var, training, testing){
```

```
    # The outcome should be numerical
    # under a column "y"
    # the subset for training is called training
    set.seed(seeds)
    options(warn=-1) # stop warnings from going to stdout
    lapsed <- system.time(
    # training the model
    trained <- train(y~.,
      data=training,
                        method = model,
                        trControl=trainControl(method="repeatedcv",
          number=10, repeats=10),
                        tuneGrid=expand.grid(var),
                        preProc=c("center","scale"),
                        tuneLength=5))
    # testing the model
    tested <- predict(trained, newdata=testing)
    rmse <- sqrt((sum((testing$y-tested)^2))/nrow(testing))
    # ploting the RMSE
    # plotting R2
#    setEPS()
#    postscript(paste("R2",rmse,sep="_",".eps"))
    plot(var[[1]],trained$results$Rsquared,
         ylab="Computed Rsquared",
         xlab=c(model," parameter"),
         pch=16, type="b",
         main=list(c("R2 with parameter tuning"),
                   cex = .7))
    # plotting the computational time
    plot(lapsed[[3]],rmse,
         main=list(c("Elapsed timexMinimum RMSE for",
             length(var[[1]]),"base parameters"),
                   cex = .7),
         xlab=list(c("Time (s) for Dsc:",
             nrow(training),"and Rep:",
             nrow(testing),"samples")),
         ylab="RMSE")
    # plotting the minimum RMSE
    Sys.sleep(1) # wait for snapshot
    plot(trained, type=c("b"), pch=16,
         main="Error estimation with parameter tuning")
}
```

Restructuring the data, so they are read by the custom function.

```
y <- training[, dim(training)[2]]
training <- data.frame(training[, sp],y=y)
y <- testing[, dim(testing)[2]]
testing <- data.frame(testing[, sp],y=y)
```

3.1 Random Forest

Lets try building a decision tree with a random forest model because the correlation between variables might just be linear after all.

3.2 Support Vector Machines

Support vector machine with a radial basis function kernel. Testing for prediction accuracy on non labelled data.

```
#Train.model(123,model="svmRadial",var=list(C=(1:3)/10,sigma=(1:3)/10),training,testing)
```

Lets try a linear svm because the correlation between variables might just be linear after all. In a sense where missing data are non-randomly missing. Testing for prediction accuracy on non labelled data.

```
#Train.model(123,model="svmLinear",
#            var=list(C=seq(.1,2,.5)),training,testing)
```

4 System Information

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

```
###save(list=ls(pattern=".*|.*"),file="PD.Rdata")
sessionInfo()

R version 3.1.2 (2014-10-31)
Platform: x86_64-unknown-linux-gnu (64-bit)

locale:
 [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
 [3] LC_TIME=en_US.UTF-8        LC_COLLATE=en_US.UTF-8
 [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
 [7] LC_PAPER=en_US.UTF-8       LC_NAME=C
 [9] LC_ADDRESS=C               LC_TELEPHONE=C
[11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C

attached base packages:
[1] stats     graphics  grDevices utils     datasets  methods
[7] base

other attached packages:
[1] latticeExtra_0.6-26 RColorBrewer_1.1-2  glmnet_1.9-8
[4] Matrix_1.1-4        leaps_2.9           caret_6.0-37
[7] ggplot2_1.0.0       lattice_0.20-29     knitr_1.8.6

loaded via a namespace (and not attached):
 [1] BradleyTerry2_1.0-5 brglm_0.5-9         car_2.0-22
 [4] codetools_0.2-9     colorspace_1.2-4    digest_0.6.6
 [7] evaluate_0.5.5      foreach_1.4.2       formatR_1.0
[10] grid_3.1.2          gtable_0.1.2        gtools_3.4.1
[13] highr_0.4           iterators_1.0.7     lme4_1.1-7
[16] MASS_7.3-35         minqa_1.2.4         munsell_0.4.2
[19] nlme_3.1-118        nloptr_1.0.4        nnet_7.3-8
[22] plyr_1.8.1          proto_0.3-10        Rcpp_0.11.3
[25] reshape2_1.4.1      scales_0.2.4        splines_3.1.2
[28] stringr_0.6.2       tools_3.1.2
```