Practical Data Science: Reducing High Dimensional Data in R

Manuel Amunategui - amunategui@gmail.com

Feature Selection - Variable Selection with GLMNET

So, what if you want to reduce you data's high dimensionality but still need to preserve you variables?

PCA variables won't do the trick as they're synthetic and made up of bits and pieces of other variables. A common solution to this conundrum is variable selection (https://en.wikipedia.org/wiki/Feature_selection). This is the process of taking a sample of your wide data set and attempt to find the most valuable ones.

We'll keep working with the wonderful library caret (http://topepo.github.io/caret/index.html) and use its great function **varImp**.

We're going to use two models: gbm (Generalized Boosted Models) (https://en.wikipedia.org/wiki/Gradient_boosting) and glmnet (Generalized Linear Models) (https://en.wikipedia.org/wiki/Generalized_linear_model). Approaching a new data set using different models is one way of getting a handle on your data. **gbm uses boosted trees while glmnet uses regression. (Note: gbm can handle NAs but glmnet cannot).**

We're also going to keep working with the same data set as in the earlier lectures Gisette (https://archive.ics.uci.edu/ml/datasets/Gisette).

```
library(RCurl) # download https data

urlfile <- 'https://archive.ics.uci.edu/ml/machine-learning-databases/gisette/GISETTE/gisette_train.data'
x <- getURL(urlfile, ssl.verifypeer = FALSE)
gisetteRaw <- read.table(textConnection(x), sep = '', header= FALSE, stringsAsFactors = FALSE)

urlfile <- "https://archive.ics.uci.edu/ml/machine-learning-databases/gisette/GISETTE/gisette_train.labels"
x <- getURL(urlfile, ssl.verifypeer = FALSE)
g_labels <- read.table(textConnection(x), sep = '', header = FALSE, stringsAsFactors = FALSE)
```

```
# build data set
gisette_df <- cbind(as.data.frame(sapply(gisetteRaw, as.numeric)), cluster=g_labels$V1)
```

We need to split the data into three sets, one for training, one for testing, and another for validating:

```
set.seed(1234)
split <- sample(nrow(gisette_df), floor(0.5*nrow(gisette_df)))
gisette_df_train_test <- gisette_df[split,]
gisette_df_validate <- gisette_df[-split,]

# split gisette_df_train_test data set into training and testing
set.seed(1234)
split <- sample(nrow(gisette_df_train_test), floor(0.5*nrow(gisette_df_train_tes
t)))
traindf <- gisette_df_train_test[split,]
testdf <- gisette_df_train_test[-split,]
```

GLMNET

OK, now let's change gears with a regression-based model - GLMNET - but still in the caret library as this will allow us to re-use most of our previous code (think about that - you have over 150 models supported in caret - type names(getModelInfo()) to see the full list of supported models).

Let's generalize our outcome name and predictor names going forward. This is a good habit and will make code re-use much easier:

```
library(caret)
outcome_name <- 'cluster'
predictors_names <- setdiff(names(traindf), outcome_name)
# caret requires a factor of non-numeric value
traindf$cluster <- ifelse(traindf$cluster == 1, "yes", "no")
traindf$cluster <- as.factor(traindf$cluster )

fitControl <- trainControl(method='cv', number=3, returnResamp='none',
summaryFunction = twoClassSummary, classProbs = TRUE)

glmnet_model <- train(x=traindf[,predictors_names], y=traindf[,outcome_name], meth
od='glmnet',  metric='roc', trControl=fitControl)
```

Warning - This is a big file so it is slow and may return a warning stating that maxit should be lager - we can ignore that here as its beyond the scope of this exercise. Here we print the glmnet_model object to get the best parameters:

```
print(glmnet_model)
```

```
## glmnet
##
## 1500 samples
## 5000 predictors
##    2 classes: 'no', 'yes'
##
## No pre-processing
## Resampling: Cross-Validated (3 fold)
## Summary of sample sizes: 1000, 1000, 1000
## Resampling results across tuning parameters:
##
##   alpha  ROC        Sens       Spec       ROC SD       Sens SD      Spec SD
##   0.10   0.9881283  0.9326883  0.9624318  0.002887481  0.004837466  0.008121952
##   0.55   0.9362884  0.8062726  0.9041314  0.012976952  0.035599776  0.025085403
##   1.00   0.8974664  0.8035291  0.8665631  0.005292832  0.026990746  0.019353298
##
## Tuning parameter 'lambda' was held constant at a value of 0.2136412
## ROC was used to select the optimal model using  the largest value.
## The final values used for the model were alpha = 0.1 and lambda = 0.2136412.
```

The final values used for the model were alpha = 0.1 and lambda = 0.2136412.

We run the base predictions on the full data set:

```
# caret requires a factor of non-numeric value
testdf$cluster <- ifelse(testdf$cluster == 1, "yes", "no")
testdf$cluster <- as.factor(testdf$cluster )
predictions <- predict(object=glmnet_model, testdf[,setdiff(names(testdf), 'cluste
r')], type='raw')
head(predictions)
```

```
## [1] yes yes no  yes yes no
## Levels: no yes
```

```
print(postResample(pred=predictions, obs=as.factor(testdf$cluster)))
```

```
## Accuracy    Kappa
##   0.9486   0.8972
```

Accuracy is 94.86%

Now let's look at the variables using varImp (this may take a while as you are plotting 5000 vertical bars):

```
head(varImp(glmnet_model,scale=F)$importance,100)
```

```
##              Overall
##  V1     0.000000e+00
##  V2     0.000000e+00
##  V3     0.000000e+00
##  V4     0.000000e+00
##  V5     0.000000e+00
##  V6     0.000000e+00
##  V7     0.000000e+00
##  V8     0.000000e+00
##  V9     0.000000e+00
##  V10    0.000000e+00
##  V11    0.000000e+00
##  V12    0.000000e+00
##  V13    0.000000e+00
##  V14    0.000000e+00
##  V15    3.591286e-06
##  V16    0.000000e+00
##  V17    0.000000e+00
##  V18    0.000000e+00
...
##  V55    0.000000e+00
##  V56    0.000000e+00
##  V57    0.000000e+00
##  V58    0.000000e+00
##  V59    0.000000e+00
##  V60    0.000000e+00
##  V61   -9.692725e-05
##  V62    0.000000e+00
```

If you look closely, you see that GLMNET returns variable importance on a positive and negative scale! The positive variable coefficients are important to predict the outcome, while the negative one predict the none outcome. Those in the middle at zero are non-predictive. This is quite a handy feature if you need to explain which feature predicts what direction!

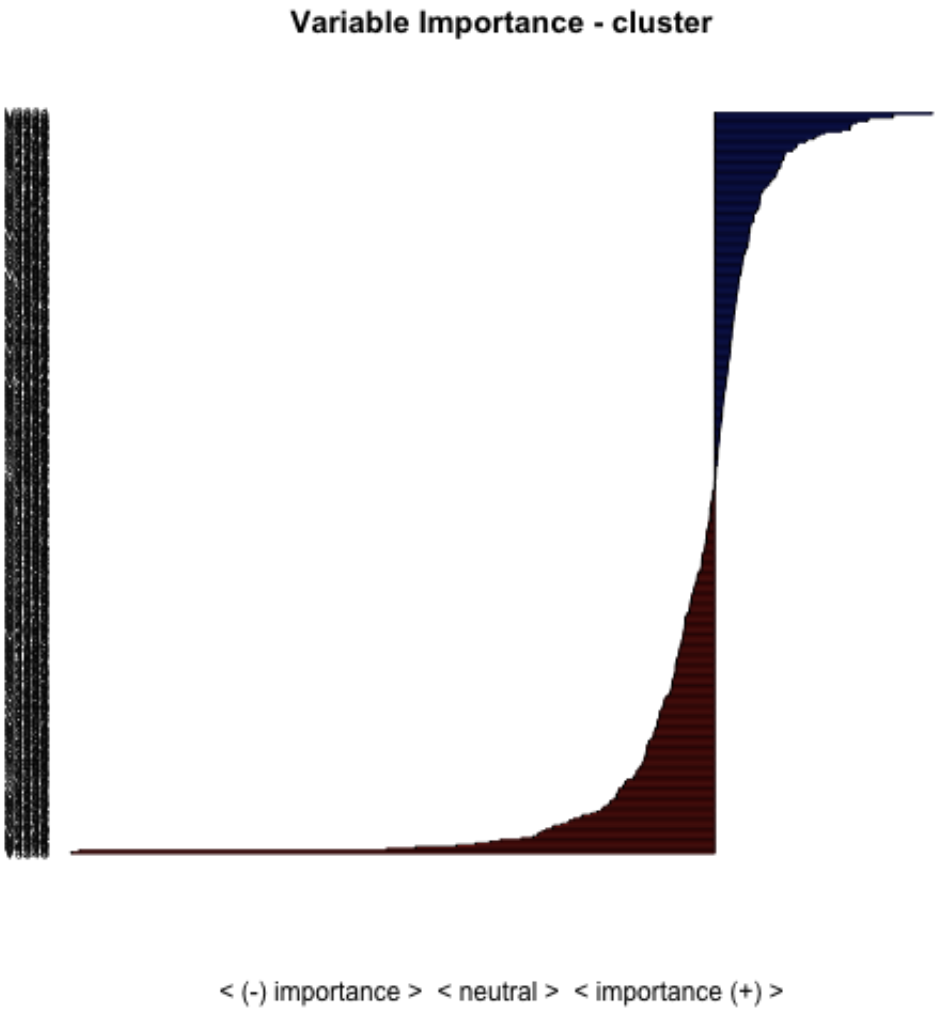
Let's see if we can plot this by removing intermediary values:

```
# display variable importance on a +/- scale
vimp <- varImp(glmnet_model, scale=F)
results <- data.frame(row.names(vimp$importance),vimp$importance$Overall)
results$VariableName <- rownames(vimp)
colnames(results) <- c('VariableName','Weight')
results <- results[order(results$Weight),]

# we do not want factors, just characters
results$VariableName <- as.character(results$VariableName)

par(mar=c(5,5,4,2)) # increase y-axis margin.

xx <- barplot(results$Weight, width = 0.85,
              main = paste("Variable Importance -",'cluster'), horiz = T,
              xlab = "< (-) importance >  < neutral >  < importance (+) >", axes =
FALSE,
              col = ifelse((results$Weight > 0), 'blue', 'red'))
axis(2, at=xx, labels=results$VariableName, tick=FALSE, las=2, line=-0.3, cex.axi
s=0.6)
```



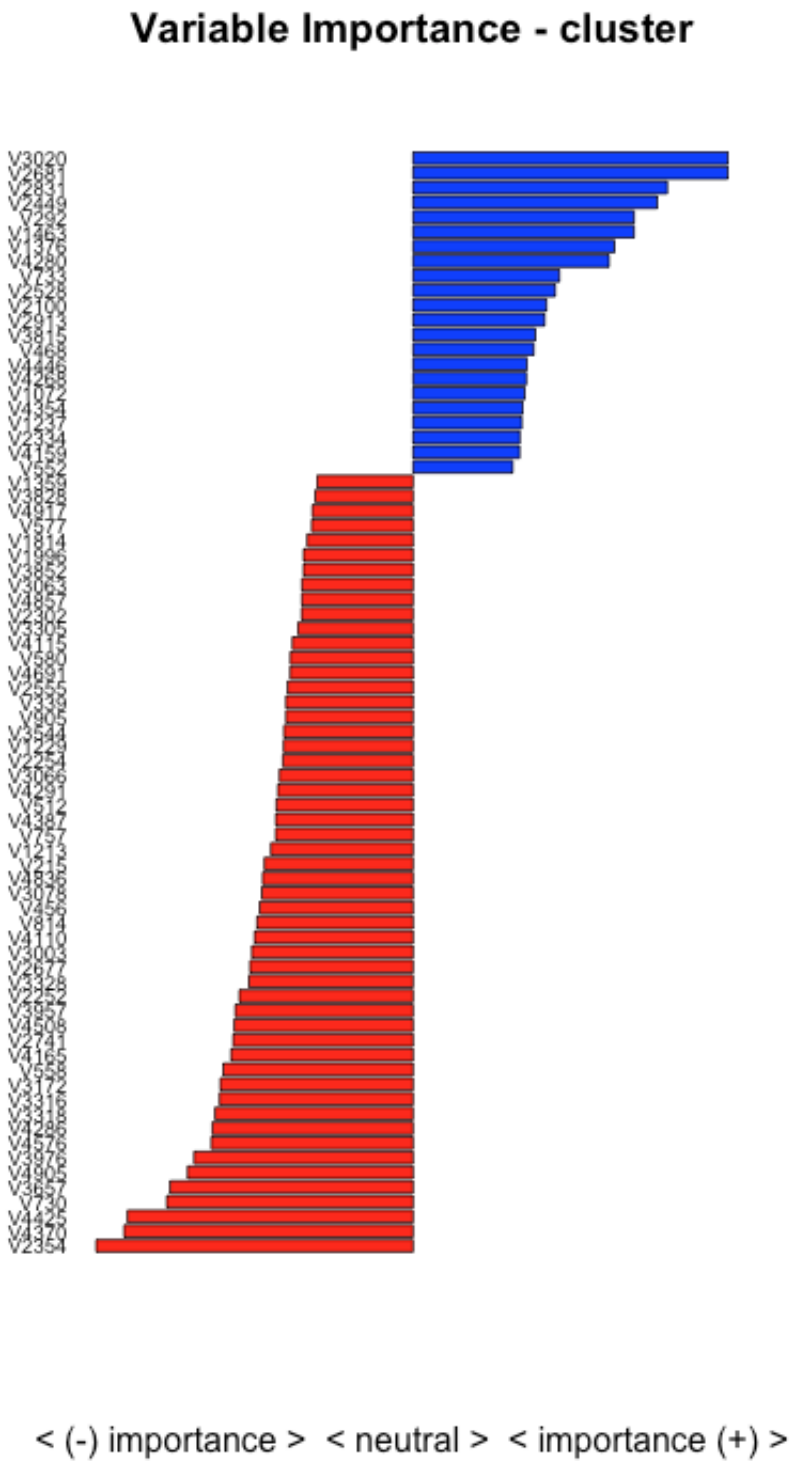
Still not ideal, let's remove more of the middle variables. Let's use the handy subset (http://www.inside-r.org/r-doc/base/subset) function:

```
# display variable importance on a +/- scale
vimp <- varImp(glmnet_model, scale=F)
results <- data.frame(row.names(vimp$importance),vimp$importance$Overall)
results$VariableName <- rownames(vimp)
colnames(results) <- c('VariableName','Weight')
results <- results[order(results$Weight),]
# remove all zero variables - non-predictive
results <- subset(results, results$Weight > 0.0001 | results$Weight < -0.0001 )


# we do not want factors, just characters
results$VariableName <- as.character(results$VariableName)

par(mar=c(5,5,4,2)) # increase y-axis margin.

xx <- barplot(results$Weight, width = 0.85,
              main = paste("Variable Importance -",'cluster'), horiz = T,
              xlab = "< (-) importance >  < neutral >  < importance (+) >", axes =
FALSE,
              col = ifelse((results$Weight > 0), 'blue', 'red'))
axis(2, at=xx, labels=results$VariableName, tick=FALSE, las=2, line=-0.3, cex.axi
s=0.6)
```



Now we see that V3020 is the most positive predictor while V2354, the most negative.

Let's try the above variable threshold that captures some positive and negative influencers:

```
traindf_truncated <- traindf[, c(results$VariableName, 'cluster')]
dim(traindf_truncated)
```

```
## [1] 1500   76
```

```
fitControl <- trainControl(method="none")

predictors_names <- setdiff(names(traindf_truncated), 'cluster')
glmnet_model <- train(traindf_truncated[,predictors_names], traindf[,outcome_nam
e], method='glmnet',   metric='roc', trControl=fitControl, tuneGrid = expand.gri
d(alpha=0.1, lambda=0.1))

predictions <- predict(object=glmnet_model, gisette_df_validate[,setdiff(names(tra
indf_truncated), 'cluster')], type='raw')

# caret requires a factor of non-numeric value
gisette_df_validate$cluster <- ifelse(gisette_df_validate$cluster == 1, "yes", "n
o")
gisette_df_validate$cluster <- as.factor(gisette_df_validate$cluster )
print(postResample(pred=predictions, obs=gisette_df_validate$cluster))
```

```
## Accuracy    Kappa
##   0.9500   0.8999
```