Data Reduction

# Need For Data Reduction

The advent of Big Data has meant an explosion in data, both in number of records and attributes. Large sized data often means worst performances in data analytics applications, especially in the phase of building statistical data models.

According to this [article](http://www.kdnuggets.com/2015/05/7-methods-data-dimensionality-reduction.html), there are seven data reduction techniques that can be applied to high-dimensional data including:

* **Missing Values Ratio**. Data columns with too many missing values are unlikely to carry much useful information. Thus data columns with number of missing values greater than a given threshold can be removed. The higher the threshold, the more aggressive the reduction.
* **Low Variance Filter**. Similarly to the previous technique, data columns with little changes in the data carry little information. Thus all data columns with variance lower than a given threshold are removed. A word of caution: variance is range dependent; therefore normalization is required before applying this technique.
* **High Correlation Filter**. Data columns with very similar trends are also likely to carry very similar information. In this case, only one of them will suffice to feed the machine learning model. Here we calculate the correlation coefficient between numerical columns and between nominal columns as the [Pearson’s Product Moment Coefficient](http://en.wikipedia.org/wiki/Pearson_product-moment_correlation_coefficient)and the [Pearson’s chi square value](http://en.wikipedia.org/wiki/Pearson%27s_chi-squared_test) respectively. Pairs of columns with correlation coefficient higher than a threshold are reduced to only one. A word of caution: correlation is scale sensitive; therefore column normalization is required for a meaningful correlation comparison.
* **Random Forests / Ensemble Trees**. Decision Tree Ensembles, also referred to as random forests, are useful for feature selection in addition to being effective classifiers.  One approach to dimensionality reduction is to generate a large and carefully constructed set of trees against a target attribute and then use each attribute’s usage statistics to find the most informative subset of features.  Specifically, we can generate a large set (2000) of very shallow trees (2 levels), with each tree being trained on a small fraction (3) of the total number of attributes. If an attribute is often selected as best split, it is most likely an informative feature to retain. A score calculated on the attribute usage statistics in the random forest tells us ‒ relative to the other attributes ‒ which are the most predictive attributes.
* **Principal Component Analysis (PCA)**. [Principal Component Analysis (PCA)](http://en.wikipedia.org/wiki/Principal_component_analysis) is a statistical procedure that orthogonally transforms the original n coordinates of a data set into a new set of n coordinates called principal components. As a result of the transformation, the first principal component has the largest possible [variance](http://en.wikipedia.org/wiki/Variance); each succeeding component has the highest possible variance under the constraint that it is [orthogonal](http://en.wikipedia.org/wiki/Orthogonal) to (i.e., uncorrelated with) the preceding components. Keeping only the first m < ncomponents reduces the data dimensionality while retaining most of the data information, i.e. the variation in the data. Notice that the PCA transformation is sensitive to the relative scaling of the original variables. Data column ranges need to be normalized before applying PCA. Also notice that the new coordinates (PCs) are not real system-produced variables anymore. Applying PCA to your data set loses its interpretability. If interpretability of the results is important for your analysis, PCA is not the transformation for your project.
* **Backward Feature Elimination**. In this technique, at a given iteration, the selected classification algorithm is trained on n input features. Then we remove one input feature at a time and train the same model on n-1 input features n times. The input feature whose removal has produced the smallest increase in the error rate is removed, leaving us with n-1 input features. The classification is then repeated using n-2 features, and so on. Each iteration k produces a model trained on n-k features and an error rate e(k).Selecting the maximum tolerable error rate, we define the smallest number of features necessary to reach that classification performance with the selected machine learning algorithm.
* **Forward Feature Construction**. This is the inverse process to the Backward Feature Elimination. We start with 1 feature only, progressively adding 1 feature at a time, i.e. the feature that produces the highest increase in performance. Both algorithms, Backward Feature Elimination and Forward Feature Construction, are quite time and computationally expensive. They are practically only applicable to a data set with an already relatively low number of input columns.

We will investigate two of the techniques above, first with a guided tutorial then followed by an exercise. The two techniques involved are **Low Variance Filter** and **Principal Component Analysis.**

The 3 common ways of dealing with **high-dimensionality data** (i.e. having too many variables) are:

1. Get more computing muscle (like RStudio on an [Amazon Web Services EC2](http://amunategui.github.io/EC2-RStudioServer/) instance),
2. Prune your data set using [feature selection](http://en.wikipedia.org/wiki/Feature_selection) (measure variables effectiveness and keeps only the best - built-in feature selection - [see fscaret](http://amunategui.github.io/fscaret-Walkthrough/)),
3. Use **feature reduction** (also refereed as [feature extraction](http://en.wikipedia.org/wiki/Dimensionality_reduction)) to create new variables made of bits and pieces of the original variables, which is what we will do.

## 1.PCA and Near Zero Variance Walkthrough

### SETUP

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)

print(dim(gisetteRaw))

# this takes a while…

Q1: What is the content of this dataset?

Q2 : How many columns does the gisetteRaw data frame posses?

We next load the caret package and use the nzv() function to weed out the columns which has less than 0.1% variance

#load caret, and use the nzv() function to weed out columns or attributes which are too similar

library(caret)

nzv <- nearZeroVar(gisetteRaw, saveMetrics = TRUE)

print(paste('Range:',range(nzv$percentUnique)))

#This will return a data frame with the degree of zero variance for each feature

print(head(nzv))

print(paste('Column count before cutoff:',ncol(gisetteRaw)))

dim(nzv[nzv$percentUnique > 0.1,])

#This removes features with less than 0.1% variance

gisette\_nzv <- gisetteRaw[c(rownames(nzv[nzv$percentUnique > 0.1,])) ]

gisette\_nzv

print(paste('Column count after cutoff:',ncol(gisette\_nzv)))

Q3 : How many columns are left after the ones with less than 0.1 variance are removed?

Let’s consider how well this reduced data set perform without **PCA.**

dfEvaluate <- cbind(as.data.frame(sapply(gisette\_nzv, as.numeric)), cluster=g\_labels$V1)

#cross-validation function using the zxgboost model

EvaluateAUC <- function(dfEvaluate) {

require(xgboost)

require(Metrics)

CVs <- 5

cvDivider <- floor(nrow(dfEvaluate) / (CVs+1))

indexCount <- 1

outcomeName <- c('cluster')

predictors <- names(dfEvaluate)[!names(dfEvaluate) %in% outcomeName]

lsErr <- c()

lsAUC <- c()

for (cv in seq(1:CVs)) {

print(paste('cv',cv))

dataTestIndex <- c((cv \* cvDivider):(cv \* cvDivider + cvDivider))

dataTest <- dfEvaluate[dataTestIndex,]

dataTrain <- dfEvaluate[-dataTestIndex,]

bst <- xgboost(data = as.matrix(dataTrain[,predictors]),

label = dataTrain[,outcomeName],

max.depth=6, eta = 1, verbose=0,

nround=5, nthread=4,

objective = "reg:linear")

predictions <- predict(bst, as.matrix(dataTest[,predictors]), outputmargin=TRUE)

err <- rmse(dataTest[,outcomeName], predictions**)**

auc <- auc(dataTest[,outcomeName],predictions)

lsErr <- c(lsErr, err)

lsAUC <- c(lsAUC, auc)

gc()

}

print(paste('Mean Error:',mean(lsErr)))

print(paste('Mean AUC:',mean(lsAUC)))

}

#Run evaluation

#install the Metrics andxgboost packages

EvaluateAUC(dfEvaluate)

The result is 0.967 for AUC. How can we improve this?

Let’s run **PCA** on this

pmatrix <- scale(gisette\_nzv) #notice the scaling here!

princ <- prcomp(pmatrix)

Choose the first PCA component

nComp <- 1

dfComponents <- predict(princ, newdata=pmatrix)[,1:nComp]

dfEvaluate <- cbind(as.data.frame(dfComponents),

cluster=g\_labels$V1)

EvaluateAUC(dfEvaluate)

AUC is worst!

Q4: Repeat nComp in steps of 1…end at nComp <- 10. What is your final conclusion (discuss in terms of data reduction and number of columns)

## 2. Repeat on crimtab

Rerun the steps above, this time using the crimtab data (data(crimtab))