Practical Data Science: Reducing High Dimensional Data in R

To get started, we need a data set with a lot of columns. We're going to borrow a data set from `NIPS (Neural Information Processing Systems)` for a completed 2013 competition. The meaning of the data is immaterial for our needs though has to do with differentiating between two handwritten digits. Let's download our data from the UC Irvine Machine Learning Repository (https://archive.ics.uci.edu/ml/datasets/Gisette) (**warning:** this is a large file).

We use the RCurl (https://cran.r-project.org/web/packages/RCurl/index.html) library to download the data files. We do the same for the labels:

```
library(RCurl) # download https data
```

```
## Loading required package: bitops
```

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

print(dim(gisetteRaw))
```

```
## [1] 6000 5000
```

The **gisetteRaw** data frame has **5000** columns, that's big and that's the kind of size we're looking for. It also has one outcome variable, 'cluster'. Before we can start the **PCA** transformation process, we need to remove the extreme near-zero variance as it won't help us much, risks crashing the script, and we'll slow us down. We load the caret (http://topepo.github.io/caret/index.html) package and call `nearZeroVar` function with `saveMetrics` parameter set to **true**. This will return a data frame with the percentage of zero variance for each feature:

```
# SMALLER DATA SET:
# truncate data set if you're having trouble running prcomp but note the scores won't be
 the same as in the walkthrough, a few percentage points lower:
# gisetteRaw <-gisetteRaw[1:2000,]
# g_labels <-data.frame('V1'=g_labels[1:2000,] )

library(caret)
```

```
## Loading required package: lattice
```

```
## Loading required package: ggplot2
```

```
nzv <- nearZeroVar(gisetteRaw, saveMetrics = TRUE)
print(paste('Range:',range(nzv$percentUnique)))
```

```
## [1] "Range: 0.0166666666666667" "Range: 8.6"
```

```
print(head(nzv))
```

```
##     freqRatio percentUnique zeroVar  nzv
## V1   48.25234     5.2166667   FALSE TRUE
## V2 1180.80000     1.3666667   FALSE TRUE
## V3   41.31579     6.1500000   FALSE TRUE
## V4 5991.00000     0.1666667   FALSE TRUE
## V5  980.00000     1.5333333   FALSE TRUE
## V6  140.00000     3.5166667   FALSE TRUE
```

We remove features with less than 0.1% variance:

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

```
## [1] "Column count before cutoff: 5000"
```

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

```
## [1] 4639    4
```

```
gisette_nzv <- gisetteRaw[c(rownames(nzv[nzv$percentUnique > 0.1,])) ]
print(paste('Column count after cutoff:',ncol(gisette_nzv)))
```

```
## [1] "Column count after cutoff: 4639"
```

The data is cleaned up and ready to go. Let's see how well it performs without any **PCA** transformation. We bind the labels (response/outcome variables) to the set:

```
gisette_df <- cbind(as.data.frame(sapply(gisette_nzv, as.numeric)),
              cluster=g_labels$V1)
```

We're going to use GBM (Generalized Boosted Models) (https://cran.r-project.org/web/packages/gbm/index.html). GBM uses boosted trees. It is also one of the models supported by the great library caret (http://topepo.github.io/caret/index.html).

If you're interested in learning more about GBM or the Caret package, check my walk through Modeling 101 - Predicting Binary Outcomes with R, gbm, glmnet, and {caret} (http://amunategui.github.io/binary-outcome-modeling/).

To evaluate the data, we split the data set into two parts, one for training and the other for evaluating. If you wanted a more accurate evaluation, I would recommend cross validating the data using multiple splits (see link in previous paragraph).

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

In this case, we are keeping things simple, we set the model's parameters - trees, shrinkage, and interaction depth - to 50, 3, 0.1 respectively. And run the `caret` train (http://www.inside-r.org/packages/cran/caret/docs/train) and predict methods:

```
traindf$cluster <- as.factor(traindf$cluster )
fitControl <- trainControl(method="none")
model <- train(cluster~., data=traindf,
               tuneGrid = expand.grid(n.trees = 50, interaction.depth = 3, shrinkage =
0.1, n.minobsinnode=10),
               trControl=fitControl, method="gbm", metric='roc')
```

```
## Loading required package: gbm
```

```
## Loading required package: survival
```

```
##
## Attaching package: 'survival'
```

```
## The following object is masked from 'package:caret':
##
##     cluster
```

```
## Loading required package: splines
```

```
## Loading required package: parallel
```

```
## Loaded gbm 2.1.3
```

```
## Loading required package: plyr
```

```
## Warning in gbm.fit(x = structure(c(0, 0, 0, 0, 0, 0, 0, 0, 999, 0, 0, 0, :
## variable 3915: V4230 has no variation.
```

```
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.2662             nan     0.1000    0.0597
##      2        1.1667             nan     0.1000    0.0489
##      3        1.0824             nan     0.1000    0.0418
##      4        1.0074             nan     0.1000    0.0369
##      5        0.9434             nan     0.1000    0.0304
##      6        0.8863             nan     0.1000    0.0277
##      7        0.8377             nan     0.1000    0.0233
##      8        0.7945             nan     0.1000    0.0200
##      9        0.7553             nan     0.1000    0.0197
##     10        0.7229             nan     0.1000    0.0153
##     20        0.5086             nan     0.1000    0.0075
##     40        0.3360             nan     0.1000    0.0023
##     50        0.2935             nan     0.1000    0.0003
```

```
testdf$cluster <- as.factor(testdf$cluster )
predictions <- predict(object=model, testdf[,setdiff(names(testdf), 'cluster')], type='r
aw')
```

We use the postResample (http://www.inside-r.org/packages/cran/caret/docs/R2) function to get an accuracy score. A quick note on the predictions. If you do a head on predictions:

```
head(predictions)
```

```
## [1] 1  1  1  1  -1 -1
## Levels: -1 1
```

You will notice that it returns actual predictions (i.e. actual cluster values) instead of probabilities - to get probabilities, use the 'type=prob' instead of 'type=raw' (again see the Modeling 101 - Predicting Binary Outcomes with R, gbm, glmnet, and {caret} (http://amunategui.github.io/binary-outcome-modeling/) walkthrough.)

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

```
##  Accuracy     Kappa
## 0.9516667 0.9032827
```

Not bad, `94.86%` accuracy. Now, let's see how close we can get there without thousands of features!!

Let's reduce the data set using PCA and compare results. As mentioned in the previous lecture, scaling is important, so we scale the entire data set (not the outcome - cluster) then run the prcomp (https://stat.ethz.ch/R-manual/R-patched/library/stats/html/prcomp.html) function. Warning (this step is slow - 12 minutes on my MacBook Air with 8GB):

```
# if your machine can't handle this, try using the smaller data set supplied above (sear
ch for SMALLER DATA SET)
pmatrix <- scale(gisette_nzv)
princ <- prcomp(pmatrix)
```

So, let's extract a data set containing only the first principal component analysis:

```
n.comp <- 1
dfComponents <- predict(princ, newdata=pmatrix)[,1:n.comp]

gisette_df <- cbind(as.data.frame(dfComponents),
              cluster=g_labels$V1)
```

To recap, we have a new data set called `gisette_df` containing only two columns: dfComponents, and cluster (i.e. PCA component 1 and the outcome variable):

```
head(gisette_df)
```

```
##   dfComponents cluster
## 1   0.09733800       1
## 2   0.06140186      -1
## 3   0.03489463       1
## 4   0.03928385       1
## 5  -0.04302395       1
## 6  -0.03183122       1
```

Let's get the accuracy on this data set:

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

# force the outcome
traindf$cluster <- as.factor(traindf$cluster )
fitControl <- trainControl(method="none")
model <- train(cluster~., data=traindf,
               tuneGrid = expand.grid(n.trees = 50, interaction.depth = 3, shrinkage =
0.1, n.minobsinnode=10),
               trControl=fitControl, method="gbm", metric='roc')
```

```
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.3502             nan     0.1000    0.0159
##      2        1.3233             nan     0.1000    0.0134
##      3        1.3004             nan     0.1000    0.0103
##      4        1.2818             nan     0.1000    0.0093
##      5        1.2658             nan     0.1000    0.0075
##      6        1.2512             nan     0.1000    0.0062
##      7        1.2405             nan     0.1000    0.0052
##      8        1.2306             nan     0.1000    0.0041
##      9        1.2230             nan     0.1000    0.0035
##     10        1.2166             nan     0.1000    0.0028
##     20        1.1844             nan     0.1000   -0.0007
##     40        1.1690             nan     0.1000   -0.0007
##     50        1.1642             nan     0.1000   -0.0001
```

```
testdf$cluster <- as.factor(testdf$cluster )

# note: here you need to force our single variable data set 'testdf' to a data frame, ot
herwise R tries to turn it into a vector
predictions <- predict(object=model, newdata=data.frame('dfComponents'=testdf[,setdiff(n
ames(testdf), 'cluster')]), type='raw')

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

```
##  Accuracy     Kappa
## 0.7126667 0.4219162
```

Ouch, our accuracy is only 71.20% but keep in mind that its done with only one variable!!

Let's try two PCA components:

```
n.comp <- 2
dfComponents <- predict(princ, newdata=pmatrix)[,1:n.comp]

gisette_df <- cbind(as.data.frame(dfComponents),
              cluster=g_labels$V1)

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

# force the outcome
traindf$cluster <- as.factor(traindf$cluster )
fitControl <- trainControl(method="none")
model <- train(cluster~., data=traindf,
               tuneGrid = expand.grid(n.trees = 50, interaction.depth = 3, shrinkage =
0.1, n.minobsinnode=10),
               trControl=fitControl, method="gbm", metric='roc')
```

```
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.3496             nan     0.1000    0.0163
##      2        1.3216             nan     0.1000    0.0138
##      3        1.2983             nan     0.1000    0.0105
##      4        1.2793             nan     0.1000    0.0090
##      5        1.2627             nan     0.1000    0.0072
##      6        1.2464             nan     0.1000    0.0069
##      7        1.2331             nan     0.1000    0.0063
##      8        1.2212             nan     0.1000    0.0050
##      9        1.2121             nan     0.1000    0.0039
##     10        1.2042             nan     0.1000    0.0036
##     20        1.1582             nan     0.1000    0.0000
##     40        1.1267             nan     0.1000    0.0002
##     50        1.1179             nan     0.1000   -0.0006
```

```
testdf$cluster <- as.factor(testdf$cluster )
predictions <- predict(object=model, newdata=testdf[,setdiff(names(testdf), 'cluster')],
 type='raw')

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

```
##  Accuracy     Kappa
## 0.7193333 0.4354248
```

So, 71.76%, as you can see, for this particular data set, one PCA variable isn't enough and we have to add more. Let's try 10! Note that `dfComponents` will take a little longer to be built:

```
n.comp <- 10
dfComponents <- predict(princ, newdata=pmatrix)[,1:n.comp]

gisette_df <- cbind(as.data.frame(dfComponents),
              cluster=g_labels$V1)

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

# force the outcome
traindf$cluster <- as.factor(traindf$cluster )
fitControl <- trainControl(method="none")
model <- train(cluster~., data=traindf,
               tuneGrid = expand.grid(n.trees = 50, interaction.depth = 3, shrinkage =
0.1, n.minobsinnode=10),
               trControl=fitControl, method="gbm", metric='roc')
```

```
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.3195             nan     0.1000    0.0313
##      2        1.2658             nan     0.1000    0.0263
##      3        1.2107             nan     0.1000    0.0263
##      4        1.1644             nan     0.1000    0.0226
##      5        1.1206             nan     0.1000    0.0205
##      6        1.0778             nan     0.1000    0.0204
##      7        1.0414             nan     0.1000    0.0148
##      8        1.0037             nan     0.1000    0.0183
##      9        0.9750             nan     0.1000    0.0138
##     10        0.9455             nan     0.1000    0.0131
##     20        0.7447             nan     0.1000    0.0058
##     40        0.5320             nan     0.1000    0.0029
##     50        0.4714             nan     0.1000    0.0024
```

```
testdf$cluster <- as.factor(testdf$cluster )
predictions <- predict(object=model, newdata=testdf[,setdiff(names(testdf), 'cluster')],
 type='raw')

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

```
##  Accuracy     Kappa
## 0.9300000 0.8599243
```

Yowza!! 92.73% accuracy!! Not bad going from a 4500+ feature data set down to one with only 10. So, let's try 20, see where that takes us:

```
n.comp <- 20
dfComponents <- predict(princ, newdata=pmatrix)[,1:n.comp]

gisette_df <- cbind(as.data.frame(dfComponents),
              cluster=g_labels$V1)

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

# force the outcome
traindf$cluster <- as.factor(traindf$cluster )
fitControl <- trainControl(method="none")
model <- train(cluster~., data=traindf,
               tuneGrid = expand.grid(n.trees = 50, interaction.depth = 3, shrinkage =
0.1, n.minobsinnode=10),
               trControl=fitControl, method="gbm", metric='roc')
```

```
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.3195             nan     0.1000    0.0313
##      2        1.2658             nan     0.1000    0.0263
##      3        1.2107             nan     0.1000    0.0263
##      4        1.1644             nan     0.1000    0.0226
##      5        1.1206             nan     0.1000    0.0205
##      6        1.0778             nan     0.1000    0.0204
##      7        1.0414             nan     0.1000    0.0148
##      8        1.0037             nan     0.1000    0.0183
##      9        0.9750             nan     0.1000    0.0138
##     10        0.9455             nan     0.1000    0.0131
##     20        0.7403             nan     0.1000    0.0072
##     40        0.5211             nan     0.1000    0.0028
##     50        0.4590             nan     0.1000    0.0014
```

```
testdf$cluster <- as.factor(testdf$cluster )
predictions <- predict(object=model, newdata=testdf[,setdiff(names(testdf), 'cluster')],
 type='raw')

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

```
##  Accuracy     Kappa
## 0.9303333 0.8605418
```

93.10%!! Recall that the full data set gave us an accuracy of 94.86% so we're almost there! The first PCA is the best and it slowly goes down from there. I'll let you keep trying by adding additional columns and seeing if/how it affects the accuracy at predicting the `cluster` outcome (but I'll give you a hint, the climb gets steep from here, adding 50 components, only yields a 0.03% improvement).

Have fun!