Practical Data Science: Reducing High Dimensional Data in R

Manuel Amunategui - amunategui@gmail.com

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 (**warning:** this is a large file).

We use the RCurl library to download the data files. We do the same for the labels:

```
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))
```

```
## [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 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:
# giseteRaw <-giseteRaw[1:2000,]
# g_labels <-data.frame('V1'=g_labels[1:2000,] )

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

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

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

```
##    freqRatio percentUnique zeroVar  nzv
## V1     48.25        5.2167   FALSE TRUE
## V2   1180.80        1.3667   FALSE TRUE
## V3     41.32        6.1500   FALSE TRUE
## V4   5991.00        0.1667   FALSE TRUE
## V5    980.00        1.5333   FALSE TRUE
## V6    140.00        3.5167   FALSE TRUE
```

We remove features with less than 0.1% variance:

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

```
## [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). GBM uses boosted trees. It is also one of the models supported by the great library caret.

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}.

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 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),
               trControl=fitControl, method="gbm",
metric='roc')
```

```
## Warning: variable 3915: V4230 has no variation.
```

```
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.2640             nan     0.1000    0.0589
##      2        1.1633             nan     0.1000    0.0493
##      3        1.0790             nan     0.1000    0.0411
##      4        1.0074             nan     0.1000    0.0342
##      5        0.9451             nan     0.1000    0.0306
##      6        0.8877             nan     0.1000    0.0273
##      7        0.8369             nan     0.1000    0.0246
##      8        0.7939             nan     0.1000    0.0207
##      9        0.7528             nan     0.1000    0.0197
##     10        0.7183             nan     0.1000    0.0166
##     20        0.5050             nan     0.1000    0.0086
##     40        0.3418             nan     0.1000    0.0013
##     50        0.2932             nan     0.1000    0.0008
```

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

We use the postResample 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} walkthrough.)

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

```
## Accuracy    Kappa
##   0.9470   0.8939
```

Not bad, 94.70% 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 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 (search 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. first PCA component and the outcome variable):

```
head(gisette_df)
```

```
##   dfComponents cluster
## 1      0.09734       1
## 2      0.06140      -1
## 3      0.03489       1
## 4      0.03928       1
## 5     -0.04302       1
## 6     -0.03183       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),
               trControl=fitControl, method="gbm",
metric='roc')
```

```
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.3497             nan     0.1000    0.0164
##      2        1.3229             nan     0.1000    0.0122
##      3        1.3001             nan     0.1000    0.0108
##      4        1.2822             nan     0.1000    0.0098
##      5        1.2660             nan     0.1000    0.0078
##      6        1.2535             nan     0.1000    0.0065
##      7        1.2415             nan     0.1000    0.0055
##      8        1.2314             nan     0.1000    0.0042
##      9        1.2233             nan     0.1000    0.0032
##     10        1.2157             nan     0.1000    0.0029
##     20        1.1826             nan     0.1000   -0.0001
##     40        1.1640             nan     0.1000   -0.0001
##     50        1.1585             nan     0.1000   -0.0003
```

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

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

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

```
## Accuracy    Kappa
##   0.7140   0.4244
```

Ouch, our accuracy is only 71.40% 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),
               trControl=fitControl, method="gbm",
metric='roc')
```

```
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.3492             nan     0.1000    0.0166
##      2        1.3221             nan     0.1000    0.0124
##      3        1.2981             nan     0.1000    0.0106
##      4        1.2793             nan     0.1000    0.0096
##      5        1.2616             nan     0.1000    0.0082
##      6        1.2478             nan     0.1000    0.0068
##      7        1.2353             nan     0.1000    0.0052
##      8        1.2238             nan     0.1000    0.0050
##      9        1.2136             nan     0.1000    0.0044
##     10        1.2043             nan     0.1000    0.0034
##     20        1.1557             nan     0.1000    0.0005
##     40        1.1232             nan     0.1000    0.0002
##     50        1.1152             nan     0.1000    0.0001
```

```
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
##   0.7177   0.4318
```

So, 71.77%, 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),
               trControl=fitControl, method="gbm",
metric='roc')
```

```
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.3163             nan     0.1000    0.0321
##      2        1.2588             nan     0.1000    0.0263
##      3        1.2064             nan     0.1000    0.0253
##      4        1.1584             nan     0.1000    0.0226
##      5        1.1167             nan     0.1000    0.0198
##      6        1.0774             nan     0.1000    0.0186
##      7        1.0371             nan     0.1000    0.0191
##      8        1.0012             nan     0.1000    0.0173
##      9        0.9734             nan     0.1000    0.0128
##     10        0.9476             nan     0.1000    0.0120
##     20        0.7450             nan     0.1000    0.0068
##     40        0.5350             nan     0.1000    0.0035
##     50        0.4723             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
##   0.9273   0.8546
```

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),
               trControl=fitControl, method="gbm",
metric='roc')
```

```
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.3163             nan     0.1000    0.0321
##      2        1.2588             nan     0.1000    0.0263
##      3        1.2064             nan     0.1000    0.0253
##      4        1.1584             nan     0.1000    0.0226
##      5        1.1140             nan     0.1000    0.0210
##      6        1.0739             nan     0.1000    0.0197
##      7        1.0327             nan     0.1000    0.0196
##      8        0.9987             nan     0.1000    0.0156
##      9        0.9704             nan     0.1000    0.0131
##     10        0.9413             nan     0.1000    0.0136
##     20        0.7303             nan     0.1000    0.0086
##     40        0.5201             nan     0.1000    0.0033
##     50        0.4579             nan     0.1000    0.0015
```

```
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
##   0.9310   0.8619
```

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!