Best Practices, Education, OML4R

# Using SVD for Dimensionality Reduction

February 6, 2016 | 17 minute read

Mark Hornick
Senior Director, Data Science and Machine Learning

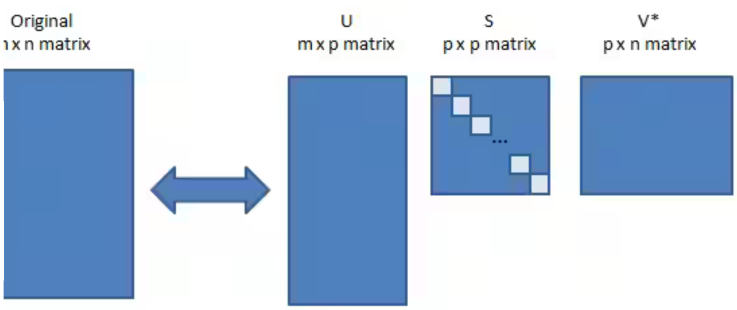
SVD, or Singular Value Decomposition, is one of several techniques that can be used to reduce the dimensionality, i.e., the number of columns, of a data set. Why would we want to reduce the number of dimensions? In predictive analytics, more columns normally means r time required to build models and score data. If some columns have no predictive value, this means wasted time, or worse, those column contribute noise to the model and reduce model quality or predictive accuracy.

Dimensionality reduction can be achieved by simply dropping columns, for example, those that may show up as collinear with others or identified as not being particularly predictive of the target as determined by an attribute importance ranking technique. But it can also be achieved by deriving new columns based on linear combinations of the original columns. In both cases, the resulting transformed data se can be provided to machine learning algorithms to yield faster model build times, faster scoring times, and more accurate models.

While SVD can be used for dimensionality reduction, it is often used in digital signal processing for noise reduction, image compression, other areas.

SVD is an algorithm that factors an *m x n* matrix, *M*, of real or complex values into three component matrices, where the factorization has form *USV**. *U* is an *m x p* matrix. *S* is a *p x p* diagonal matrix. *V* is an *n x p* matrix, with *V** being the transpose of *V*, a *p x n* matrix, or the conjugate transpose if *M* contains complex values. The value *p* is called the rank. The diagonal entries of *S* are referred to as the singular values of *M*. The columns of *U* are typically called the left-singular vectors of *M*, and the columns of *V* are called the right-singular vectors *M*.

Consider the following visual representation of these matrices:

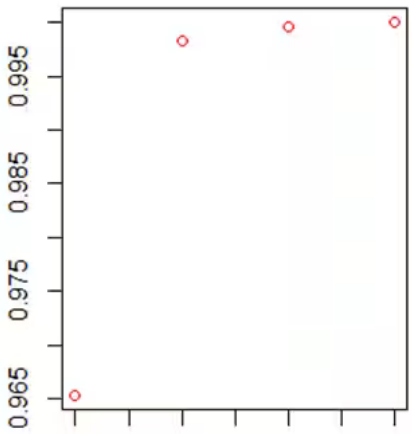



One of the features of SVD is that given the decomposition of *M* into *U*, *S*, and *V*, one can reconstruct the original matrix *M*, or an approximation of it. The singular values in the diagonal matrix *S* can be used to understand the amount of variance explained by each of singular vectors. In R, this can be achieved using the computation:

Copy code sni

```
cumsum(S^2/sum(S^2))
```

When plotted, this provides a visual understanding of the variance captured by the model. The figure below indicates that the first singul vector accounts for 96.5% of the variance, the second with the first accounts for over 99.5%, and so on.




As such, we can use this information to limit the number of vectors to the amount of variance we wish to capture. Reducing the number c vectors can help eliminate noise in the original data set when that data set is reconstructed using the subcomponents of *U*, *S*, and *V*.

**ORE's parallel, distributed SVD**

With Oracle R Enterprise's parallel distributed implementation of R's svd function, only the *S* and *V* components are returned. More specifically, the diagonal singular values are returned of *S* as the vector *d*. If we store the result of invoking svd on matrix *dat* in *svd.mod*, can be derived from these using *M* as follows:

Copy code sni

```
svd.mod <- svd(dat)
U <- dat %*%  svd.mod$v  %*% diag(1./svd.mod$d)
```

So, how do we achieve dimensionality reduction using SVD? We can use the first *k* columns of *V* and *S* and achieve *U'* with fewer columns

Copy code sni

```
U.reduced <-dat %*% svd.mod$v[,1:k,drop=FALSE] %*% diag((svd.mod$d)[1:k,drop=FALSE])
```

This reduced *U* can now be used as a proxy for matrix *dat* with fewer columns.

The function *dimReduce* introduced below accepts a matrix *x*, the number of columns desired *k*, and a request for any supplemental colu to return with the transformed matrix.

Copy code sni

```
dimReduce <- function(x, k=floor(ncol(x)/2), supplemental.cols=NULL) {
  colIdxs <- which(colnames(x) %in% supplemental.cols)
  colNames <- names(x[,-colIdxs])
  sol <- svd(x[,-colIdxs])
  sol.U <- as.matrix(x[,-colIdxs]) %*% (sol$v)[,1:k,drop=FALSE] %*%
                        diag((sol$d)[1:k,drop=FALSE])
  sol.U = sol.U@data
  res <- cbind(sol.U,x[,colIdxs,drop=FALSE])
```

```
}
```

We will now use this function to reduce the *iris* data set.
To prepare the *iris* data set, we first add a unique identifier, create the database table *IRIS2* in the database, and then assign row names to enable row indexing. We could also make *ID* the primary key using *ore.exec* with the ALTER TABLE statement. Refreshing the *ore.frame* pr object using *ore.sync* reflects the change in primary key.

Copy code sni

```
dat <- iris
dat$ID <- seq_len(nrow(dat))
ore.drop("IRIS2")
ore.create(dat,table="IRIS2")
row.names(IRIS2) <- IRIS2$ID
# ore.exec("alter table IRIS2 add constraint IRIS2 primary key (\"ID\")")
# ore.sync(table = "IRIS2", use.keys = TRUE)
IRIS2[1:5,]
```

Using the function defined above, *dimReduce*, we produce *IRIS2.reduced* with supplemental columns of *ID* and *Species*. This allows us to easily generate a confusion matrix later. You will find that *IRIS2.reduced* has 4 columns.

Copy code sni

```
IRIS2.reduced <- dimReduce(IRIS2, 2, supplemental.cols=c("ID","Species"))
dim(IRIS2.reduced)  # 150 4
```

Next, we will build an *rpart* model to predict *Species* using first the original *iris* data set, and then the reduced data set so we can compare confusion matrices of each. Note that to use R's *rpart* for model building, the data set *IRIS2.reduced* is pulled to the client.

Copy code sni

```
library(rpart)
m1 <- rpart(Species~.,iris)
res1 <- predict(m1,iris,type="class")
table(res1,iris$Species)
#res1         setosa versicolor virginica
#  setosa         50          0         0
#  versicolor      0         49         5
#  virginica       0          1        45
dat2 <- ore.pull(IRIS2.reduced)
m2 <- rpart(Species~.-ID,dat2)
res2 <- predict(m2,dat2,type="class")
table(res2,iris$Species)
# res2          setosa versicolor virginica
#  setosa         50          0         0
#  versicolor      0         47         0
#  virginica       0          3        50
```

We can build a similar model using the in-database decision tree algorithm, via *ore.odmDT*, and get the same results on this particular da
set.

Copy code sni

```
m2.1 <- ore.odmDT(Species~.-ID, IRIS2.reduced)
res2.1 <- predict(m2.1,IRIS2.reduced,type="class",supplemental.cols = "Species")
table(res2.1$PREDICTION, res2.1$Species)
# res2          setosa versicolor virginica
#  setosa          50          0         0
#  versicolor       0         47         0
#  virginica        0          3        50
```

A more interesting example is based on the digit-recognizer data which can be located on the Kaggle website here. In this example, we fir
use Support Vector Machine as the algorithm with default parameters on split train and test samples of the original training data. This all
us to get an objective assessment of model accuracy. Then, we preprocess the train and test sets using the in-database SVD algorithm ar
reduce the original 785 predictors to 40. The reduced number of variables specified is subject to experimentation. Degree of parallelism f
SVD was set to 4.

The results highlight that reducing data dimensionality can improve overall model accuracy, and that overall execution time can be significantly faster. Specifically, using *ore.odmSVM* for model building saw a 43% time reduction and a 4.2% increase in accuracy by preprocessing the train and test data using SVD.

However, it should be noted that not all algorithms are necessarily aided by dimensionality reduction with SVD. In a second test on the sa
data using *ore.odmRandomForest* with 25 trees and defaults for other settings, accuracy of 95.3% was achieved using the original train ar
test sets. With the SVD reduced train and test sets, accuracy was 93.7%. While the model building time was reduced by 80% and scoring t
reduced by 54%, if we factor in the SVD execution time, however, using the straight random forest algorithm does better by a factor of tw

**Details**

For this scenario, we modify the *dimReduce* function introduced above and add another function *dimReduceApply*. In *dimReduce*, we save
model in an ORE Datastore so that the same model can be used to transform the test data set for scoring. In *dimReduceApply*, that same
model is loaded for use in constructing the reduced *U* matrix.

Copy code sni

```
dimReduce <- function(x, k=floor(ncol(x)/2), supplemental.cols=NULL, dsname="svd.model") {
  colIdxs <- which(colnames(x) %in% supplemental.cols)
  if (length(colIdxs) > 0) {
    sol <- svd(x[,-colIdxs])
    sol.U <- as.matrix(x[,-colIdxs]) %*% (sol$v)[,1:k,drop=FALSE] %*%
            diag((sol$d)[1:k,drop=FALSE])
    res <- cbind(sol.U@data,x[,colIdxs,drop=FALSE])
   # names(res) <- c(names(sol.U@data),names(x[,colIdxs]))
    res
  } else {
    sol <- svd(x)
    sol.U <- as.matrix(x) %*% (sol$v)[,1:k,drop=FALSE] %*%
                  diag((sol$d)[1:k,drop=FALSE])
    res <- sol.U@data
  }
  ore.save(sol, name=dsname, overwrite=TRUE)
  res
}
```

```
  ore.load(dsname)
  if (length(colIdxs) > 0) {
    sol.U <- as.matrix(x[,-colIdxs]) %*% (sol$v)[,1:k,drop=FALSE] %*%
                diag((sol$d)[1:k,drop=FALSE])
    res <- cbind(sol.U@data,x[,colIdxs,drop=FALSE])
    # names(res) <- c(names(sol.U@data),names(x[,colIdxs]))
    res
  } else {
    sol.U <- as.matrix(x) %*% (sol$v)[,1:k,drop=FALSE] %*%
                 diag((sol$d)[1:k,drop=FALSE])
    res <- sol.U@data
  }
  res
}
```

Here is the script used for the digit data:

Copy code sni

```
# load data from file
train <- read.csv("D:/datasets/digit-recognizer-train.csv")
dim(train)       # 42000   786
train$ID <- 1:nrow(train)             # assign row id
ore.drop(table="DIGIT_TRAIN")
ore.create(train,table="DIGIT_TRAIN") # create as table in the database
dim(DIGIT_TRAIN)  # 42000   786
# Split the original training data into train and
# test sets to evaluate model accuracy
set.seed(0)
dt <- DIGIT_TRAIN
ind <- sample(1:nrow(dt),nrow(dt)*.6)
group <- as.integer(1:nrow(dt) %in% ind)
row.names(dt) <- dt$ID
sample.train <- dt[group==TRUE,]
sample.test  <- dt[group==FALSE,]
dim(sample.train)  # 25200   786
dim(sample.test)   # 16800   786
# Create train table in database
ore.create(sample.train, table="DIGIT_SAMPLE_TRAIN")
# Create test table in database
ore.create(sample.test,  table="DIGIT_SAMPLE_TEST")
# Add persistent primary key for row indexing
# Note: could be done using row.names(DIGIT_SAMPLE_TRAIN) <- DIGIT_SAMPLE_TRAIN$ID
ore.exec("alter table DIGIT_SAMPLE_TRAIN add constraint
           DIGIT_SAMPLE_TRAIN primary key (\"ID\")")
ore.exec("alter table DIGIT_SAMPLE_TEST  add constraint
            DIGIT_SAMPLE_TEST  primary key (\"ID\")")
ore.sync(table = c("DIGIT_SAMPLE_TRAIN","DIGIT_SAMPLE_TRAIN"), use.keys = TRUE)
# SVM model
m1.svm <- ore.odmSVM(label~.-ID, DIGIT_SAMPLE_TRAIN, type="classification")
pred.svm <- predict(m1.svm, DIGIT_SAMPLE_TEST,
                    supplemental.cols=c("ID","label"),type="class")
cm <- with(pred.svm, table(label,PREDICTION))
library(caret)
confusionMatrix(cm)
# Confusion Matrix and Statistics
#
# PREDICTION
# label    0    1    2    3    4    5    6    7    8    9
#     0 1633    0    4    2    3    9   16    2    7    0
```

```
#     4    5    9   10    0 1508    0   10    4   14   85
#     5   24   12   14   52   28 1314   26    6   49   34
#     6   10    2    7    1    8   26 1603    0    6    0
#     7   10    8   27    4   21    8    1 1616    4   70
#     8   12   45   14   40    7   47   13   10 1377   30
#     9   12   10    6   19   41   15    2   54   15 1447
#
# Overall Statistics
#
# Accuracy : 0.9114
# 95% CI : (0.907, 0.9156)
# No Information Rate : 0.1167
# P-Value [Acc > NIR] : < 2.2e-16
#...
options(ore.parallel=4)
sample.train.reduced <- dimReduce(DIGIT_SAMPLE_TRAIN, 40, supplemental.cols=c("ID","label"))
sample.test.reduced <- dimReduceApply(DIGIT_SAMPLE_TEST, 40, supplemental.cols=c("ID","label"
ore.drop(table="DIGIT_SAMPLE_TRAIN_REDUCED")
ore.create(sample.train.reduced,table="DIGIT_SAMPLE_TRAIN_REDUCED")
ore.drop(table="DIGIT_SAMPLE_TEST_REDUCED")
ore.create(sample.test.reduced,table="DIGIT_SAMPLE_TEST_REDUCED")
m2.svm <- ore.odmSVM(label~.-ID,
                      DIGIT_SAMPLE_TRAIN_REDUCED, type="classification")
pred2.svm <- predict(m2.svm, DIGIT_SAMPLE_TEST_REDUCED,
                      supplemental.cols=c("label"),type="class")
cm <- with(pred2.svm, table(label,PREDICTION))
confusionMatrix(cm)
# Confusion Matrix and Statistics
#
# PREDICTION
# label    0    1    2    3    4    5    6    7    8    9
#     0 1652    0    3    3    2    7    4    1    3    1
#     1    0 1887    8    2    2    1    1    3    3    2
#     2    3    4 1526   11   20    3    7   21   27    7
#     3    0    3   29 1595    3   38    4   16   34   12
#     4    0    4    8    0 1555    2   11    5    9   51
#     5    5    6    2   31    6 1464   13    6   10   16
#     6    2    1    5    0    5   18 1627    0    5    0
#     7    2    6   22    7   10    2    0 1666    8   46
#     8    3    9    9   34    7   21    9    7 1483   13
#     9    5    2    8   17   30   10    3   31   20 1495
#
# Overall Statistics
#
# Accuracy : 0.9494
# 95% CI : (0.946, 0.9527)
# No Information Rate : 0.1144
# P-Value [Acc > NIR] : < 2.2e-16
#...
# CASE 2 with Random Forest
m2.rf <- ore.randomForest(label~.-ID, DIGIT_SAMPLE_TRAIN,ntree=25)
pred2.rf <- predict(m2.rf, DIGIT_SAMPLE_TEST, supplemental.cols=c("label"),type="response")
cm <- with(pred2.rf, table(label,prediction))
confusionMatrix(cm)
# Confusion Matrix and Statistics
#
# prediction
# label    0    1    2    3    4    5    6    7    8    9
#     0 1655    0    1    1    2    0    7    0    9    1
#     1    0 1876   12    8    2    1    1    2    6    1
#     2    7    4 1552   14   10    2    5   22   10    3
#     3    9    5   33 1604    1   21    4   16   27   14
#     4    1    4    3    0 1577    1    9    3    3   44
#     5    9    6    2   46    3 1455   18    1    9   10
#     6   13    2    3    0    6   14 1621    0    3    1
```

```
#      9    9    2    7   23   32    5    1   15   12 1515
#
# Overall Statistics
#
# Accuracy : 0.9527
# 95% CI : (0.9494, 0.9559)
# No Information Rate : 0.1138
# P-Value [Acc > NIR] : < 2.2e-16
#...
m1.rf <- ore.randomForest(label~.-ID, DIGIT_SAMPLE_TRAIN_REDUCED,ntree=25)
pred1.rf <- predict(m1.rf, DIGIT_SAMPLE_TEST_REDUCED,
                     supplemental.cols=c("label"),type="response")
cm <- with(pred1.rf, table(label,prediction))
confusionMatrix(cm)
# Confusion Matrix and Statistics
#
# prediction
# label    0    1    2    3    4    5    6    7    8    9
#      0 1630    0    4    5    2    8   16    3    5    3
#      1    0 1874   17    4    0    5    2    2    4    1
#      2   15    2 1528   17   10    5   10   21   16    5
#      3    7    1   32 1601    4   25   10    8   34   12
#      4    2    6    6    3 1543    2   17    4    4   58
#      5    9    1    5   45   12 1443   11    3   15   15
#      6   21    3    8    0    5   15 1604    0    7    0
#      7    5   11   33    7   17    6    1 1649    2   38
#      8    5   13   27   57   14   27    9   12 1404   27
#      9   10    2    6   22   52    8    5   41   12 1463
#
# Overall Statistics
#
# Accuracy : 0.9368
# 95% CI : (0.9331, 0.9405)
# No Information Rate : 0.1139
# P-Value [Acc > NIR] : < 2.2e-16
#...
```

**Run Times**

The following numbers reflect the execution times for select operations of the above script. Hardware was a Lenovo Thinkpad with Intel i5 processor and 16 GB RAM.

| Data Set | Operation | Execution Time | | |
|---|---|---|---|---|
| | | SVM | RF | SVD |
| DIGIT_SAMPLE_TRAIN | Build | 349.61 | 88.31 | 88.87 |
| DIGIT_SAMPLE_TEST | Predict | 6.85 | 20.62 | 1.67 |
| | **Total** | **356.46** | **108.93** | **90.54** |
| DIGIT_SAMPLE_TRAIN_REDUCED | Build | 18.77 | 17.55 | |
| DIGIT_SAMPLE_TEST_REDUCED | Predict | 0.77 | 9.52 | |
| | **Total** | **19.54** | **27.07** | |

| Data Set | Operation | Execution Time |
|---|---|---|
| DIGIT_SAMPLE_TRAIN_REDUCED | Create Table | 54.79 |
| DIGIT_SAMPLE_TEST _REDUCED | Create Table | 35.9 |
| | **Subtotal** | **90.69** |

**Mark Hornick**

Senior Director, Data Science and Machine Learning

Mark Hornick is senior director of product management for Oracle Machine Learning. Mark has more than 20 years of experience integrating and leveraging machine learning with Oracle software as well as working wit