ORACLE

Learning R Series

Session 4: Oracle R Enterprise 1.3 Predictive Analytics

Mark Hornick
Oracle Advanced Analytics

Learning R Series 2012

Session	Title
Session 1	Introduction to Oracle's R Technologies and Oracle R Enterprise 1.3
Session 2	Oracle R Enterprise 1.3 Transparency Layer
Session 3	Oracle R Enterprise 1.3 Embedded R Execution
Session 4	Oracle R Enterprise 1.3 Predictive Analytics
Session 5	Oracle R Enterprise 1.3 Integrating R Results and Images with OBIEE Dashboards
Session 6	Oracle R Connector for Hadoop 2.0 New features and Use Cases

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions.

The development, release, and timing of any features or functionality described for Oracle's products remain at the sole discretion of Oracle.

Topics

- ORE Package Overview
- OREeda package
  - ore.lm
  - ore.stepwise
  - ore.neural
- OREdm package
  - Contrasting OREdm with CRAN Package RODM
  - OREdm features
- OREpredict package
- Performance Characteristics
  - OREeda, OREdm & OREpredict
  - Data read and write

ORE Analytics Packages

- OREbase
- OREdm
  - Oracle Data Mining algorithms exposed through R interface
  - Attribute Importance, Decision Trees, GLM, KMeans, Naïve Bayes, SVM
- OREeda
  - ore.lm, ore.stepwise, ore.neural
  - Functions for exploratory data analysis for Base SAS equivalents
- OREgraphics
- OREpredict
  - Score R models in the database
- OREstats
  - In-database statistical computations exposed through R interface
- ORExml

OREeda Package

ore.lm and ore.stepwise

Overview

- ‘ore.lm’ performs least squares regression
- ‘ore.stepwise’ performs stepwise least squares regression
- Uses database data represented by 'ore.frame' objects
- In-database algorithm
  - Estimates model using block update QR decomposition with column pivoting
  - Once coefficients have been estimated, a second pass of the data estimates model-level statistics
  - If collinear terms in model, functions 'ore.lm' and 'ore.stepwise' will not estimate coefficient values for a collinear set of terms
  - For 'ore.stepwise', this collinear set of terms will be excluded throughout the procedure

ore.lm and ore.stepwise

Motivation

- Enable handling data with complex patterns
  - Even for relatively small data sets (e.g., < 1M rows) R may not yield satisfactory results
- Performance
  - Side benefit of handling complex patterns is to dramatically boost performance
  - No need to pull data into memory from database
  - Leverage more powerful database machine

- Provide a stepwise regression that maps to SAS PROC REG

Performance results from Oracle Micro-Processor Tools Environment

https://blogs.oracle.com/R/entry/oracle_r_enterprise_deployment_in

…the ORE capabilities for stepwise regression far surpass similar functionality in tools we considered as alternatives to ORE. The deployment of ORE within the Oracle micro-processor tools environment introduced a technology which significantly expands the data analysis capabilities through the R ecosystem combined with in-database high performance algorithms and opens the door to new applications."

Bilinear model

method	R^2	Number of Regressors	mean(rel_error)	Elapsed Time (seconds)
step	0.9658	86	3.52e-02	2110.0
ore.stepwise	0.9966	124	3.50e-02	32.1
performance difference				**ore.stepwise is approx. 65X faster than step at similar R^2 and relative error as stepwise.**

66x faster

Quadratic model

method	R^2	Number of Regressors	mean(rel_error)	Elapsed Time (seconds)
step	0.9962	154	1.05e-02	12600.0
ore.stepwise	0.9963	210	1.04e-02	69.5
performance difference				**ore.stepwise is approx. 180X faster than step at similar R^2 relative error.**

180x faster

Figure 1: Comparison of results for R's step function and ORE's ore.stepwise function for both bi-linear and quadratic models

lm

For comparison with ore.lm

```
# Fit full model
fit1 <- lm(Employed ~ ., data = longley)
summary(fit1)
```

```
R> fit1 <- lm(Employed ~ ., data = longley)
R> summary(fit1)

Call:
lm(formula = Employed ~ ., data = longley)

Residuals:
     Min       1Q   Median       3Q      Max
-0.41011 -0.15767 -0.02816  0.10155  0.45539

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)  -3.482e+03  8.904e+02  -3.911 0.003560 **
GNP.deflator  1.506e-02  8.492e-02   0.177 0.863141
GNP          -3.582e-02  3.349e-02  -1.070 0.312681
Unemployed   -2.020e-02  4.884e-03  -4.136 0.002535 **
Armed.Forces -1.033e-02  2.143e-03  -4.822 0.000944 ***
Population   -5.110e-02  2.261e-01  -0.226 0.826212
Year          1.829e+00  4.555e-01   4.016 0.003037 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.3049 on 9 degrees of freedom
Multiple R-squared: 0.9955,     Adjusted R-squared: 0.9925
F-statistic: 330.3 on 6 and 9 DF,  p-value: 4.984e-10
```

ore.lm

```
LONGLEY <- ore.push(longley)

# Fit full model
oreFit1 <- ore.lm(Employed ~ ., data = LONGLEY)
summary(oreFit1)
```

```
R> LONGLEY <- ore.push(longley)
R>
R> # Fit full model
R> oreFit1 <- ore.lm(Employed ~ ., data = LONGLEY)
R> summary(oreFit1)

Call:
ore.lm(formula = Employed ~ ., data = LONGLEY)

Residuals:
     Min       1Q   Median       3Q      Max
-0.41011 -0.15980 -0.02816  0.15681  0.45539

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)  -3.482e+03  8.904e+02  -3.911 0.003560 **
GNP.deflator  1.506e-02  8.492e-02   0.177 0.863141
GNP          -3.582e-02  3.349e-02  -1.070 0.312681
Unemployed   -2.020e-02  4.884e-03  -4.136 0.002535 **
Armed.Forces -1.033e-02  2.143e-03  -4.822 0.000944 ***
Population   -5.110e-02  2.261e-01  -0.226 0.826212
Year          1.829e+00  4.555e-01   4.016 0.003037 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.3049 on 9 degrees of freedom
Multiple R-squared: 0.9955,     Adjusted R-squared: 0.9925
F-statistic: 330.3 on 6 and 9 DF,  p-value: 4.984e-10
```

lm and ore.lm results side-by-side

They're identical

```
R> fit1 <- lm(Employed ~ ., data = longley)
R> summary(fit1)

Call:
lm(formula = Employed ~ ., data = longley)

Residuals:
     Min       1Q   Median       3Q      Max
-0.41011 -0.15767 -0.02816  0.10155  0.45539

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)  -3.482e+03  8.904e+02  -3.911 0.003560 **
GNP.deflator  1.506e-02  8.492e-02   0.177 0.863141
GNP          -3.582e-02  3.349e-02  -1.070 0.312681
Unemployed   -2.020e-02  4.884e-03  -4.136 0.002535 **
Armed.Forces -1.033e-02  2.143e-03  -4.822 0.000944 ***
Population   -5.110e-02  2.261e-01  -0.226 0.826212
Year          1.829e+00  4.555e-01   4.016 0.003037 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.3049 on 9 degrees of freedom
Multiple R-squared: 0.9955,     Adjusted R-squared: 0.9925
F-statistic: 330.3 on 6 and 9 DF,  p-value: 4.984e-10
```

```
R> LONGLEY <- ore.push(longley)
R>
R> # Fit full model
R> oreFit1 <- ore.lm(Employed ~ ., data = LONGLEY)
R> summary(oreFit1)

Call:
ore.lm(formula = Employed ~ ., data = LONGLEY)

Residuals:
     Min       1Q   Median       3Q      Max
-0.41011 -0.15980 -0.02816  0.15681  0.45539

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)  -3.482e+03  8.904e+02  -3.911 0.003560 **
GNP.deflator  1.506e-02  8.492e-02   0.177 0.863141
GNP          -3.582e-02  3.349e-02  -1.070 0.312681
Unemployed   -2.020e-02  4.884e-03  -4.136 0.002535 **
Armed.Forces -1.033e-02  2.143e-03  -4.822 0.000944 ***
Population   -5.110e-02  2.261e-01  -0.226 0.826212
Year          1.829e+00  4.555e-01   4.016 0.003037 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.3049 on 9 degrees of freedom
Multiple R-squared: 0.9955,     Adjusted R-squared: 0.9925
F-statistic: 330.3 on 6 and 9 DF,  p-value: 4.984e-10
```

ore.stepwise – example

```
LONGLEY <- ore.push(longley)

# Two stepwise alternatives
oreStep1 <-
  ore.stepwise(Employed ~ .^2, data = LONGLEY,
               add.p = 0.1, drop.p = 0.1)
oreStep1

oreStep2 <-
  step(ore.lm(Employed ~ 1, data = LONGLEY),
       scope = terms(Employed ~ .^2, data = LONGLEY))
oreStep2
```

ore.stepwise – results

```
R> oreStep1 <-
+   ore.stepwise(Employed ~ .^2, data = LONGLEY,
+                add.p = 0.1, drop.p = 0.1)
R> oreStep1

Aliased:
[1] "Unemployed:Armed.Forces" "Unemployed:Population"   "Unemployed:Year"         "Armed.Forces:Population
"
[5] "Armed.Forces:Year"       "Population:Year"

Steps:
                        Add Drop     RSS Rank
1 GNP.deflator:Unemployed <NA> 384.426    2
2               GNP:Year <NA> 218.957    3
3       GNP.deflator:GNP <NA> 130.525    4
4 GNP.deflator:Population <NA>  81.211    5
5       GNP:Armed.Forces <NA>  18.244    6
6                   Year <NA>  14.492    7

Call:
ore.stepwise(formula = Employed ~ .^2, data = LONGLEY, add.p = 0.1,
    drop.p = 0.1)

Coefficients:
            (Intercept)                     Year         GNP.deflator:GNP  GNP.deflator:Unemployed  GNP.de
flator:Population
             -3.539e-01                3.589e-05               -2.978e-03                2.326e-04
        2.303e-05
       GNP:Armed.Forces                 GNP:Year
             6.875e-06                2.007e-04
```

step with ore.lm – results

```
...
R> oreStep2 <-
+   step(ore.lm(Employed ~ 1, data = LONGLEY),
+        scope = terms(Employed ~ .^2, data = LONGLEY))
Start:  AIC=41.17
Employed ~ 1

                Df Sum of Sq     RSS     AIC
+ GNP            1   178.973   6.036 -11.597
+ Year           1   174.552  10.457  -2.806
+ GNP.deflator   1   174.397  10.611  -2.571
+ Population     1   170.643  14.366   2.276
+ Unemployed     1    46.716 138.293  38.509
+ Armed.Forces   1    38.691 146.318  39.411
<none>                       185.009  41.165

Step:  AIC=-11.6
Employed ~ GNP

                Df Sum of Sq     RSS     AIC
+ Unemployed     1     2.457   3.579 -17.960
+ Population     1     2.162   3.874 -16.691
+ Year           1     1.125   4.911 -12.898
<none>                       6.036 -11.597
+ GNP.deflator   1     0.212   5.824 -10.169
+ Armed.Forces   1     0.077   5.959  -9.802
- GNP            1   178.973 185.009  41.165

Step:  AIC=-17.96
Employed ~ GNP + Unemployed

                  Df Sum of Sq     RSS     AIC
+ Armed.Forces     1     0.822   2.757 -20.137
<none>                         3.579 -17.960
+ Year             1     0.340   3.239 -17.556
+ GNP:Unemployed   1     0.182   3.397 -16.795
+ Population       1     0.097   3.482 -16.399
+ GNP.deflator     1     0.019   3.560 -16.044
- Unemployed       1     2.457   6.036 -11.597
- GNP              1   134.714 138.293  38.509
```

```
Step:  AIC=-20.14
Employed ~ GNP + Unemployed + Armed.Forces

                  Df Sum of Sq     RSS     AIC
+ Year             1     1.898   0.859 -36.799
+ GNP:Unemployed   1     0.614   2.143 -22.168
+ Population       1     0.390   2.367 -20.578
<none>                         2.757 -20.137
+ Unemployed:Armed.Forces  1     0.083   2.673 -18.629
+ GNP.deflator     1     0.073   2.684 -18.566
+ GNP:Armed.Forces 1     0.060   2.697 -18.489
- Armed.Forces     1     0.822   3.579 -17.960
- Unemployed       1     3.203   5.959  -9.802
- GNP              1    78.494  81.250  31.999

Step:  AIC=-36.8
Employed ~ GNP + Unemployed + Armed.Forces + Year

                  Df Sum of Sq     RSS     AIC
<none>                         0.8587 -36.799
+ Unemployed:Year  1     0.0749 0.7838 -36.259
+ GNP:Unemployed   1     0.0678 0.7909 -36.115
+ Unemployed:Armed.Forces  1     0.0515 0.8072 -35.788
+ GNP:Armed.Forces 1     0.0367 0.8220 -35.498
+ Population       1     0.0193 0.8393 -35.163
+ GNP.deflator     1     0.0175 0.8412 -35.129
+ Armed.Forces:Year  1     0.0136 0.8451 -35.054
+ GNP:Year         1     0.0084 0.8502 -34.957
- GNP              1     0.4647 1.3234 -31.879
- Year             1     1.8980 2.7567 -20.137
- Armed.Forces     1     2.3806 3.2393 -17.556
- Unemployed       1     4.0491 4.9077 -10.908
R> oreStep2

Call:
ore.lm(formula = Employed ~ GNP + Unemployed + Armed.Forces +
    Year, data = LONGLEY)

Coefficients:
 (Intercept)          GNP    Unemployed  Armed.Forces          Year
  -3.599e+03    -4.019e-02    -2.088e-02    -1.015e-02     1.887e+00
```

ore.neural - overview

- Neural network (NN) is a mathematical model inspired by biological neural networks and in some sense mimics the functioning of a brain
- A neural network consists of an interconnected group of artificial neurons (nodes)
- Modern neural networks are **non-linear statistical data modeling tools**
  - Model complex nonlinear relationships between input and output variables
- Find patterns in data:
  - Function approximation: regression analysis, including time series prediction, fitness approximation, and modeling
  - Classification: including pattern and sequence recognition, novelty detection and sequential decision making
  - Data processing: including filtering, clustering, blind source separation and compression
  - Robotics: including directing manipulators, computer numerical control

ore.neural - characteristics

- Single layer feed-forward neural network models for regression
  - Hidden layer with bipolar activation function
  - Output with linear activation function
- State-of-the-art numerical optimization engine
  - Robustness
  - Accuracy
  - Small number of data reads
- Solve arbitrarily long input data sets
  - Process unlimited number of rows in a table
  - Reads 100k row block, computes, discards block, proceeds to next one
  - R's nnet requires entire dataset stored in memory at once

R Package nnet and. ore.neural

- K9.data set (http://archive.ics.uci.edu/ml/machine-learning-databases/p53/p53_new_2012.zip)
  - Data: 16772 rows, 5409 cols
  - Model: activity ~ val1 .. val500
  - nnet: **did not complete – runs out of memory**

- PAMAP2_Dataset (http://archive.ics.uci.edu/ml/datasets/PAMAP2+Physical+Activity+Monitoring)
  - Data: 3850505 rows
  - Model: val1 ~ val2 + ... + val54
  - nnet: **did not complete – runs out of memory**

ore.neural – error comparison with nnet

```
# nnet example
d <- read.csv('./K9-9.data', header=TRUE)
fit <-nnet(V1 ~., data = d, size=20, linout=TRUE)
p <-predict(fit, newdata=d)
z <- p - d$V1
sum(z^2)

# ore.neural example
d <-read.csv('./K9-9.data', header=TRUE)
dd <- ore.push(d)
fit <- ore.neural(V1 ~., data = dd, hiddenSize=20)
pp <- predict(fit, newdata=dd)
z <- pp - dd$V1
sum(z^2)
```

ore.neural – error comparison with nnet

31160 rows, Hidden layer nodes = 20

K9-5.data – 5 columns

Formula : V1 ~ V2 + V4827 + V4828 + V5409

K9-7.data – 7 columns

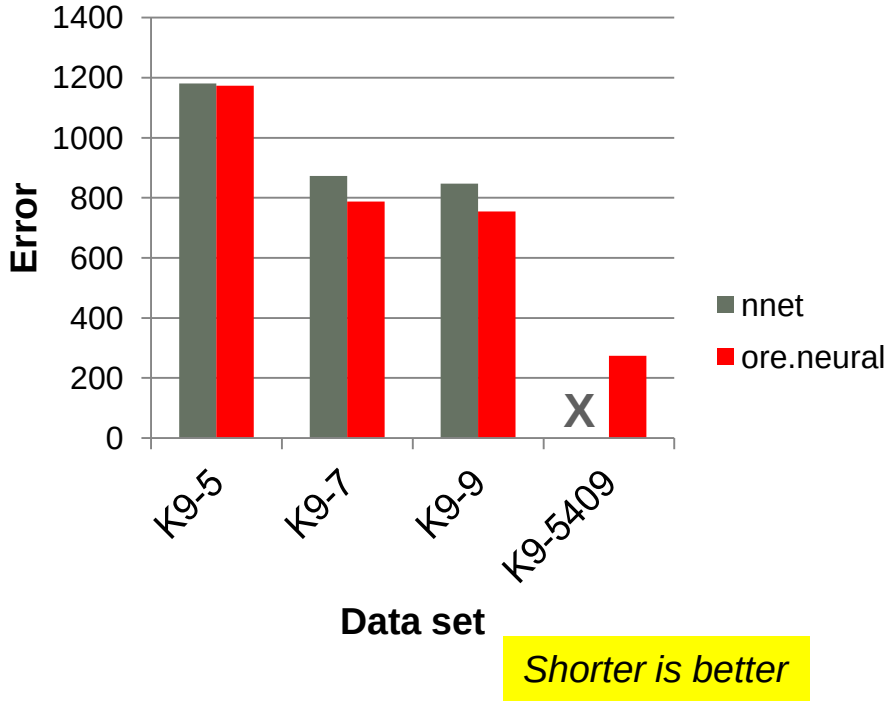
Formula: V1 ~ V2 + V3 + V4827 + V4828 + V4829 + V5409

K9-9.data – 9 columns

Formula V1 ~ V2 + V3 + V4 + V4827 + V4828 + V4829 + V4830 + V5409

K9.data – 5409 columns

Formula V1 ~ .



ore.neural – Example with two targets

```
dd   <- read.csv('~/K9-5.data', header=TRUE)
od   <- ore.push(dd)
fit  <- ore.neural(cbind(V1, V2) ~ V4827 + V4828 + V5409,
                   data = od, hiddenSize=10)
pred <- predict(fit, newdata=od)
res  <- cbind(pred,od)

head(res)
```

ore.neural - Results

```
R> dd   <- read.csv('~/K9-5.data', header=TRUE)
R> od   <- ore.push(dd)
R> fit  <- ore.neural(cbind(V1, V2) ~ V4827 + V4828 + V5409,
+                     data = od, hiddenSize=10)
R> pred <- predict(fit, newdata=od)
R> res  <- cbind(pred,od)
R>
R> head(res)
           o1           o2     V1     V2 V4827  V4828 V5409
1 -0.18541160 -0.019421025 -0.161 -0.014 0.023 -0.001    -1
2 -0.11777635 -0.002044806 -0.158 -0.002 0.010  0.003    -1
3 -0.15176917 -0.013006371 -0.169 -0.025 0.016  0.003    -1
4 -0.11975969 -0.006174594 -0.183 -0.051 0.010  0.010    -1
5 -0.12313956 -0.002837944 -0.154  0.005 0.011  0.001    -1
6 -0.08733031  0.008628771 -0.150  0.016 0.005  0.003    -1
```

OREdm Package

OAA is the broadest enterprise analytical tool-set

- Broadest functional interface to advanced analytics
  - Select tool based on technique required, interface preference (command line or GUI), data volume (small or large), and skill set (R or SQL)
  - ODM provides GUI and SQL access
  - ORE provides command line R and SQL access
- Broadest set of powerful in-database techniques now available to R users via familiar interfaces
  - Avoid data access latency
  - Achieve high performance and scalability
  - ORE exposes key ODM algorithms through R interfaces, as well as providing additional techniques
  - For many problems working with samples is insufficient - e.g. clustering, classification
- Operationalize R workloads developed by R-quants in-database
  - ORE facilitates running that R code in the database
  - ORE allows interfaces to plug R code into SQL as a first step and additionally leverage data- and task-parallelism through embedded R execution

OREdm Features

- Function signatures conforms to R norms
  - Use formula for specifying target and predictor variables
  - Use ore.frame objects as input data set for build and predict
  - Create R objects for model results
  - Use parameter names similar to corresponding R functions
  - Function parameters provide explicit default values to corresponding ODM settings, where applicable
- As in R, models are treated as transient objects
  - Automatically delete ODM model when corresponding R object no longer exists
  - Can be explicitly saved using datastore, via ore.save

OREdm Algorithms

Algorithm	Main R Function	Mining Type / Function
Minimum Description Length	ore.odmAI	Attribute Importance for Classification or Regression
Decision Tree	ore.odmDT	Classification
Generalized Linear Models	ore.odmGLM	Classification Regression
KMeans	ore.odmKMeans	Clustering
Naïve Bayes	ore.odmNB	Classification
Support Vector Machine	ore.odmSVM	Classification Regression Anomaly Detection

ore.odmNB & predict.ore.odmNB

Naïve Bayes

```
ore.odmNB(
  formula,                  # formula specifying attributes for model build
  data,                     # ore.frame of the training dataset
  auto.data.prep = TRUE,    # Setting to perform automatic data preparation
  class.priors = NULL,      # data.frame containing target class priors
  na.action = na.pass)      # Allows missing values (na.pass), or removes rows with
                            #    missing values (na.omit)

predict(
  object,                   # Object of type "ore.naiveBayes"
  newdata,                  # Data used for scoring
  supplemental.cols = NULL, # Columns to retain in output
  type = c("class","raw"),  # "raw" - cond. a-posterior probs for each class returned
                            # "class" - class with max prob
  na.action = na.pass)
```

Example with NaiveBayes

```
# library(ORE)
# ore.connect("rquser","orcl","localhost","rquser",all=TRUE)

data(titanic3,package="PASWR")

t3 <- ore.push(titanic3)
t3$survived <- ifelse(t3$survived == 1, "Yes", "No")

n.rows <- nrow(t3)
set.seed(seed=6218945)
random.sample <- sample(1:n.rows, ceiling(n.rows/2))
t3.train <- t3[random.sample,]
t3.test  <- t3[setdiff(1:n.rows,random.sample),]


priors <- data.frame(
    TARGET_VALUE = c("Yes", "No"),
    PRIOR_PROBABILITY = c(0.1, 0.9))

nb  <- ore.odmNB(survived ~ pclass+sex+age+fare+embarked,
                 t3.train, class.priors=priors)
```

Login to database for transparent access via ORE

Push data to db for transparent access

Recode column from 0/1 to No/Yes keeping data in database

Sample keeping data in database

Create priors for model building

Build model using R formula using transparency layer data

```
nb.res  <- predict (nb, t3.test,"survived")

head(nb.res,10)
with(nb.res, table(survived,PREDICTION, dnn = c("Actual","Predicted")))

library(verification)
res <- ore.pull(nb.res)
perf.auc <- roc.area(ifelse(res$survived == "Yes", 1, 0), res$"'Yes'")
auc.roc <- signif(perf.auc$A, digits=3)
auc.roc.p <- signif(perf.auc$p.value, digits=3)
roc.plot(ifelse(res$survived == "Yes", 1, 0), res$"'Yes'", binormal=T,
    plot="both",
         xlab="False Positive Rate",
         ylab="True Postive Rate", main= "Titanic survival ODM NB model ROC
    Curve")
text(0.7, 0.4, labels= paste("AUC ROC:", signif(perf.auc$A, digits=3)))
text(0.7, 0.3, labels= paste("p-value:", signif(perf.auc$p.value, digits=3)))

summary(nb)

ore.disconnect()
```

Score data using ore.frame with OREdm model object.

Display first 10 rows of data frame using transparency layer

Compute confusion matrix using transparency layer

Retrieve result from database for using verification package

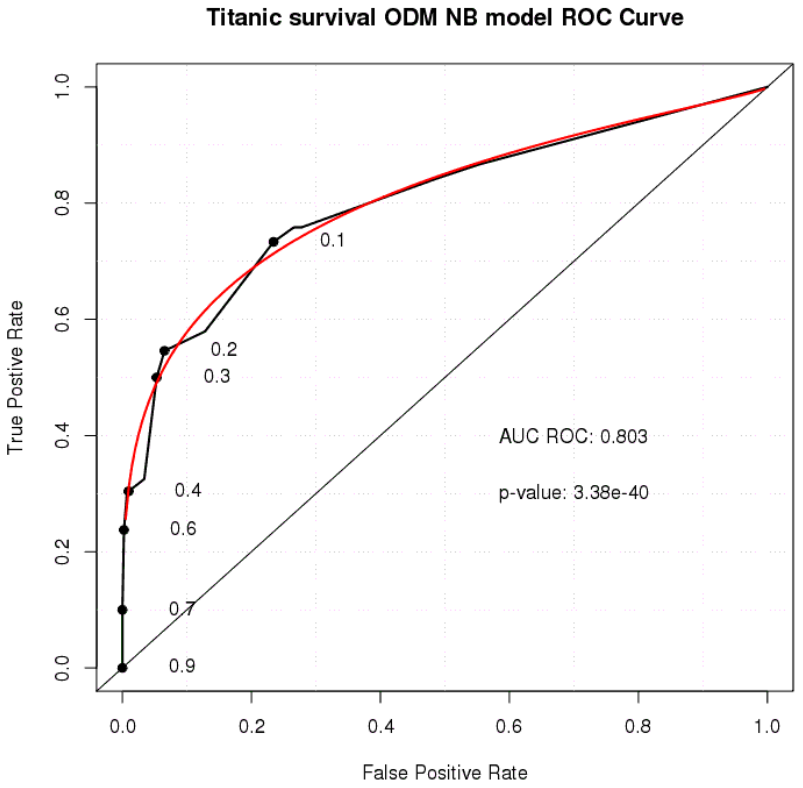
View model object summary

Disconnect from database

Model, train and test objects are automatically removed when session ends or R objects are removed

ROC Curve

```
R> summary(nb)

Call:
ore.odmNB(formula = survived ~ pclass + sex + age + fare + embarked,
    data = t3.train, class.priors = priors)

Settings:
          value
prep.auto    on

Apriori:
 No Yes
0.9 0.1

Tables:
$embarked
    'Cherbourg' 'Queenstown', 'Southampton'
No    0.1569620                 0.8430380
Yes   0.3178295                 0.6821705

$fare
    ( ; 51.931249600000001), [51.931249600000001; 51.931249600000001] (51.931249600000001;  )
No                                                          0.91370558             0.08629442
Yes                                                         0.67307692             0.32692308

$pclass
    '1st', '2nd'     '3rd'
No     0.3417722 0.6582278
Yes    0.6346154 0.3653846

$sex
       female      male
No  0.1670886 0.8329114
Yes 0.6769231 0.3230769


Levels:
[1] "No"  "Yes"
```



ore.odmKMeans

K-Means Clustering

```
ore.odmKMeans(
  formula,
  data,
  auto.data.prep = TRUE,              # Setting to perform automatic data preparation
  num.centers = 10,                   # number of clusters
  block.growth = 2,                   # Numeric growth factor for memory to hold cluster data
  conv.tolerance = 0.01,              # Numeric convergence tolerance setting
  distance.function = "euclidean",    # Distance function: cosine, euclidean, or fast.cosine
  iterations = 3,                     # Maximum number of iterations
  min.pct.attr.support = 0.1,         # Minimum percent required for variables to appear in rules
  num.bins = 10,                      # Number of histogram bins
  split.criterion = "variance",       # Split clusters by variance or size
  na.action = na.pass)                # Allow missing values in rows by default, or na.omit
```

ore.odmKMeans

K-Means Clustering

```
x <- rbind(matrix(rnorm(100, sd = 0.3), ncol = 2),
           matrix(rnorm(100, mean = 1, sd = 0.3), ncol = 2))
colnames(x) <- c("x", "y")
X <- ore.push (data.frame(x))
km.mod1 <- ore.odmKMeans(~., X, num.centers=2, num.bins=5)
summary(km.mod1)
rules(km.mod1)
clusterhists(km.mod1)
histogram(km.mod1)
```

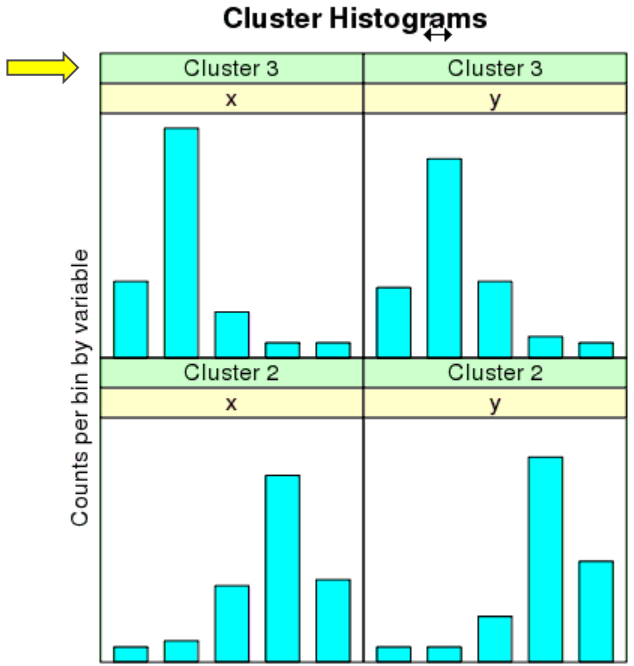
```
R> summary(km.mod1)

Call:
ore.odmKMeans(formula = ~., data = X, num.centers = 2, num.bins = 5)

Settings:
                        value
clus.num.clusters           2
block.growth                2
conv.tolerance           0.01
distance            euclidean
iterations                  3
min.pct.attr.support      0.1
num.bins                    5
split.criterion      variance
prep.auto                  on

Centers:
          x           y
2  1.05630476 1.0455933541
3 -0.01131291 0.0001622473
```

ore.odmKMeans – results

```
R> rules(km.mod1)
   rhs.cluster.id rhs.support rhs.conf lhs.support lhs.conf lhs.var lhs.var.num.val lhs.var.chr.val lhs.var.support lhs.var.conf
1               1         100      1.0          90      0.9       x      -0.2084763           <NA>              90          0.20
2               1         100      1.0          90      0.9       x       1.9020637           <NA>              90          0.20
3               1         100      1.0          90      0.9       y      -0.2588969           <NA>              91          0.20
4               1         100      1.0          90      0.9       y       1.6761630           <NA>              91          0.20
5               2          50      0.5          45      0.9       x       0.3191587           <NA>              49          0.25
6               2          50      0.5          45      0.9       x       1.9020637           <NA>              49          0.25
7               2          50      0.5          45      0.9       y       0.7086331           <NA>              45          0.50
8               2          50      0.5          45      0.9       y       1.6761630           <NA>              45          0.50
9               3          50      0.5          45      0.9       x      -0.7361113           <NA>              45          0.80
10              3          50      0.5          45      0.9       x       0.3191587           <NA>              45          0.80
11              3          50      0.5          45      0.9       y      -0.7426619           <NA>              49          0.60
12              3          50      0.5          45      0.9       y       0.7086331           <NA>              49          0.60
$`1`
  rhs.cluster.id rhs.support rhs.conf lhs.support lhs.conf lhs.var lhs.var.support lhs.var.conf   predicate
2              1         100        1          90      0.9       x              90          0.2   <= 1.9021
1              1         100        1          90      0.9       x              90          0.2  >= -0.2085
4              1         100        1          90      0.9       y              91          0.2   <= 1.6762
3              1         100        1          90      0.9       y              91          0.2  >= -0.2589

$`2`
  rhs.cluster.id rhs.support rhs.conf lhs.support lhs.conf lhs.var lhs.var.support lhs.var.conf  predicate
6              2          50      0.5          45      0.9       x              49         0.25  <= 1.9021
5              2          50      0.5          45      0.9       x              49         0.25  >= 0.3192
8              2          50      0.5          45      0.9       y              45         0.50  <= 1.6762
7              2          50      0.5          45      0.9       y              45         0.50  >= 0.7086

$`3`
   rhs.cluster.id rhs.support rhs.conf lhs.support lhs.conf lhs.var lhs.var.support lhs.var.conf   predicate
10              3          50      0.5          45      0.9       x              45          0.8   <= 0.3192
9               3          50      0.5          45      0.9       x              45          0.8  >= -0.7361
12              3          50      0.5          45      0.9       y              49          0.6   <= 0.7086
11              3          50      0.5          45      0.9       y              49          0.6  >= -0.7427
```

ore.odmKMeans – results

```
R> clusterhists(km.mod1)
   cluster.id variable bin.id lower.bound upper.bound                label count
1           1        x      1 -0.7361113 -0.2084763 -7.361E-01 : -2.085E-01    10
2           1        x      2 -0.2084763  0.3191587 -2.085E-01 :  3.192E-01    36
3           1        x      3  0.3191587  0.8467937  3.192E-01 :  8.468E-01    15
4           1        x      4  0.8467937  1.3744287  8.468E-01 :  1.374E+00    28
5           1        x      5  1.3744287  1.9020637  1.374E+00 :  1.902E+00    11
6           1        y      1 -0.7426619 -0.2588969 -7.427E-01 : -2.589E-01     9
7           1        y      2 -0.2588969  0.2248681 -2.589E-01 :  2.249E-01    30
8           1        y      3  0.2248681  0.7086331  2.249E-01 :  7.086E-01    15
9           1        y      4  0.7086331  1.1923980  7.086E-01 :  1.192E+00    32
10          1        y      5  1.1923980  1.6761630  1.192E+00 :  1.676E+00    14
11          2        x      1 -0.7361113 -0.2084763 -7.361E-01 : -2.085E-01     0
12          2        x      2 -0.2084763  0.3191587 -2.085E-01 :  3.192E-01     1
13          2        x      3  0.3191587  0.8467937  3.192E-01 :  8.468E-01    10
14          2        x      4  0.8467937  1.3744287  8.468E-01 :  1.374E+00    28
15          2        x      5  1.3744287  1.9020637  1.374E+00 :  1.902E+00    11
16          2        y      1 -0.7426619 -0.2588969 -7.427E-01 : -2.589E-01     0
17          2        y      2 -0.2588969  0.2248681 -2.589E-01 :  2.249E-01     0
18          2        y      3  0.2248681  0.7086331  2.249E-01 :  7.086E-01     5
19          2        y      4  0.7086331  1.1923980  7.086E-01 :  1.192E+00    31
20          2        y      5  1.1923980  1.6761630  1.192E+00 :  1.676E+00    14
21          3        x      1 -0.7361113 -0.2084763 -7.361E-01 : -2.085E-01    10
22          3        x      2 -0.2084763  0.3191587 -2.085E-01 :  3.192E-01    35
23          3        x      3  0.3191587  0.8467937  3.192E-01 :  8.468E-01     5
24          3        x      4  0.8467937  1.3744287  8.468E-01 :  1.374E+00     0
25          3        x      5  1.3744287  1.9020637  1.374E+00 :  1.902E+00     0
26          3        y      1 -0.7426619 -0.2588969 -7.427E-01 : -2.589E-01     9
27          3        y      2 -0.2588969  0.2248681 -2.589E-01 :  2.249E-01    30
28          3        y      3  0.2248681  0.7086331  2.249E-01 :  7.086E-01    10
29          3        y      4  0.7086331  1.1923980  7.086E-01 :  1.192E+00     1
30          3        y      5  1.1923980  1.6761630  1.192E+00 :  1.676E+00     0
```



ore.odmKMeans

K-Means Clustering

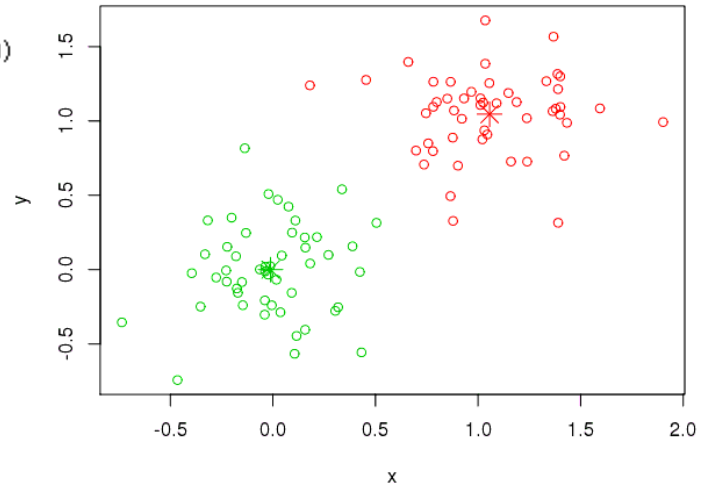
```
km.res1 <- predict(km.mod1,X,type="class",supplemental.cols=c("x","y"))
head(km.res1,3)
```

```
km.res1.local <- ore.pull(km.res1)
plot(data.frame(x=km.res1.local$x, y=km.res1.local$y), col=km.res1.local$CLUSTER_ID)
points(km.mod1$centers2, col = rownames(km.mod1$centers2), pch = 8, cex=2)
```

```
head(predict(km.mod1,X))
head(predict(km.mod1,X,type=c("class","raw"),supplemental.cols=c("x","y")),3)
head(predict(km.mod1,X,type="raw",supplemental.cols=c("x","y")),3)
```

ore.odmKMeans – results

```
R> km.res1 <- predict(km.mod1,X,type="class",supplemental.cols=c("x","y"))
R> head(km.res1,3)
          x          y CLUSTER_ID
1 -0.03999935 -0.3029228          3
2  0.50486611  0.3145332          3
3 -0.20133745  0.3497027          3
R> km.res1.local <- ore.pull(km.res1)
R> plot(data.frame(x=km.res1.local$x, y=km.res1.local$y), col=km.res1.local$CLUSTER_ID)
R> points(km.mod1$centers2, col = rownames(km.mod1$centers2), pch = 8, cex=2)
R>
R> head(predict(km.mod1,X))
        '3'          '2' CLUSTER_ID
1 0.9999998 1.844763e-07          3
2 0.9338791 6.612089e-02          3
3 0.9999185 8.154833e-05          3
4 0.9999520 4.798267e-05          3
5 0.9999885 1.153331e-05          3
6 0.9995041 4.959004e-04          3
R> head(predict(km.mod1,X,type=c("class","raw"),supplemental.cols=c("x","y")),3)
        '3'          '2'          x          y CLUSTER_ID
1 0.9999998 1.844763e-07 -0.03999935 -0.3029228          3
2 0.9338791 6.612089e-02  0.50486611  0.3145332          3
3 0.9999185 8.154833e-05 -0.20133745  0.3497027          3
R> head(predict(km.mod1,X,type="raw",supplemental.cols=c("x","y")),3)
          x          y        '3'          '2'
1 -0.03999935 -0.3029228 0.9999998 1.844763e-07
2  0.50486611  0.3145332 0.9338791 6.612089e-02
3 -0.20133745  0.3497027 0.9999185 8.154833e-05
```



OREpredict Package

OREpredict Package

- Provide a commercial grade scoring engine
  - High performance
  - Scalable
  - Simplify application workflow
- Use R-generated models to score in-database on ore.frame
- Maximizes use of Oracle Database as compute engine
- Function ore.predict
  - S4 generic function
  - A specific method for each model ORE supports

ore.predict supported algorithms

Class	Package	Description
glm	stats	Generalized Linear Model
negbin	MASS	Negative binomial Generalized Linear Model
hclust	stats	Hierarchical Clustering
kmeans	stats	K-Means Clustering
lm	stats	Linear Model
multinom	nnet	Multinomial Log-Linear Model
nnet	nnet	Neural Network
rpart	rpart	Recursive Partitioning and Regression Tree

Interface function signatures

```
# lm, based on stats:::predict.lm
ore.predict(object, newdata, se.fit = FALSE, scale = NULL,
            df = Inf, interval = c("none","confidence","prediction"),
            level = 0.95, na.action = na.pass, pred.var = NULL,
            weights = NULL, ...)

# glm, based on stats:::predict.glm
ore.predict(object, newdata, type = c("link", "response"),
            se.fit = FALSE, dispersion = NULL, na.action = na.pass,
            ...)

# rpart, based on rpart:::predict.rpart
ore.predict(object, newdata, type = c("vector", "prob",
            "class", "matrix"),na.action = na.pass, ...)
```

```
# matrix (for use in hclust problems)
ore.predict(object, newdata, type = c("classes",
            "distances"), method = "euclidean", p = 2,
             na.action = na.pass, ...)

# kmeans
ore.predict(object, newdata, type = c("classes",
            "distances"), na.action = na.pass, ...)

# nnet, based on nnet:::predict.nnet
ore.predict(object, newdata, type = c("raw", "class"),
            na.action = na.pass, ...)

# multinom, based on nnet:::predict.multinom
ore.predict(object, newdata, type = c("class", "probs"),
            na.action = na.pass, ...)
```

Example using lm

```
irisModel <- lm(Sepal.Length ~ ., data = iris)
IRIS      <- ore.push(iris)
IRISpred  <- ore.predict(irisModel, IRIS, se.fit = TRUE,
                         interval = "prediction")
IRIS <- cbind(IRIS, IRISpred)
head(IRIS)
```

- Build a typical R lm model
- Use ore.predict to score data in Oracle Database using ore.frame, e.g., IRIS

```
R> irisModel <- lm(Sepal.Length ~ ., data = iris)
R> IRIS <- ore.push(iris)
R> IRISpred <- ore.predict(irisModel, IRIS, se.fit = TRUE,
+                          interval = "prediction")
R> IRIS <- cbind(IRIS, IRISpred)
R> head(IRIS)
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species     PRED    SE.PRED LOWER.PRED UPPER.PRED
1          5.1         3.5          1.4         0.2  setosa 5.004788 0.04479188   4.391895   5.617681
2          4.9         3.0          1.4         0.2  setosa 4.756844 0.05514933   4.140660   5.373027
3          4.7         3.2          1.3         0.2  setosa 4.773097 0.04690495   4.159587   5.386607
4          4.6         3.1          1.5         0.2  setosa 4.889357 0.05135928   4.274454   5.504259
5          5.0         3.6          1.4         0.2  setosa 5.054377 0.04736842   4.440727   5.668026
6          5.4         3.9          1.7         0.4  setosa 5.388886 0.05592364   4.772430   6.005342
```

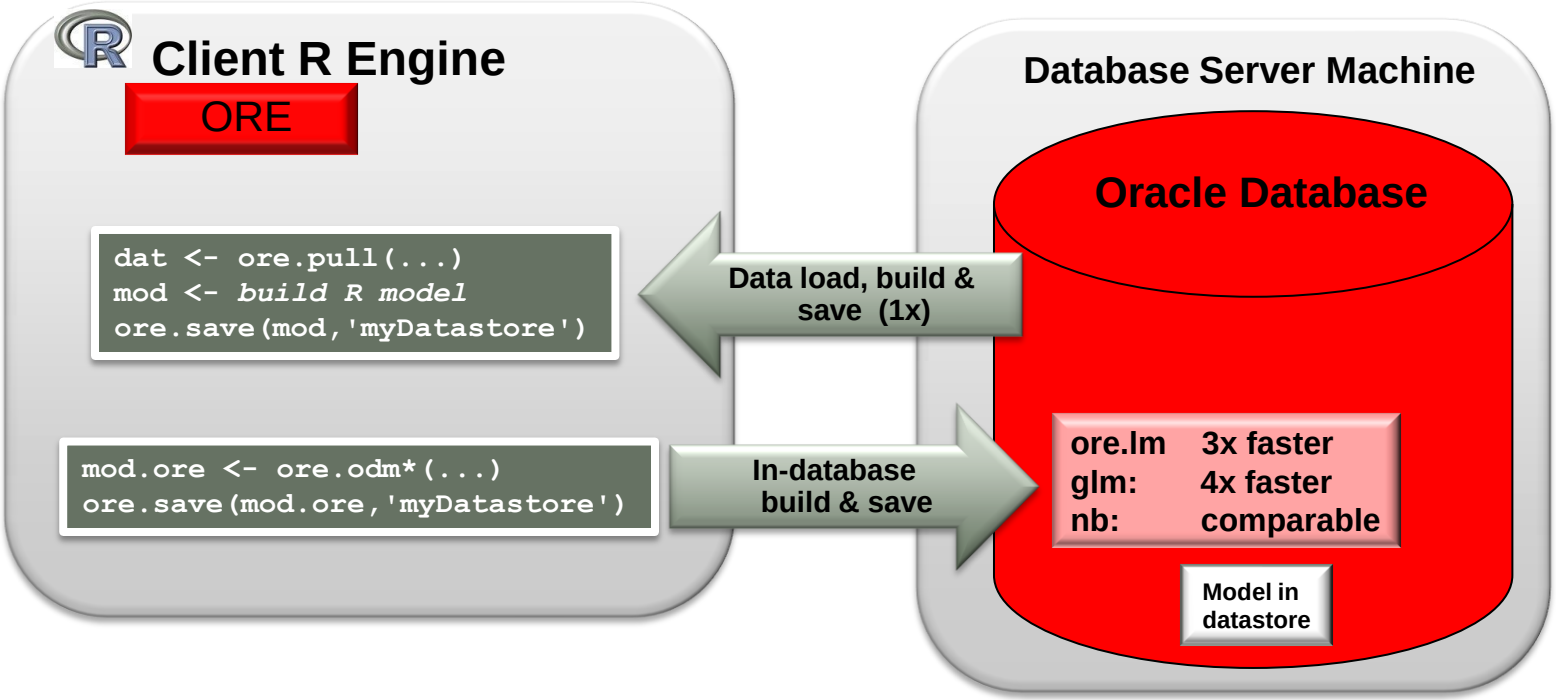
Performance

Note on the data and tests

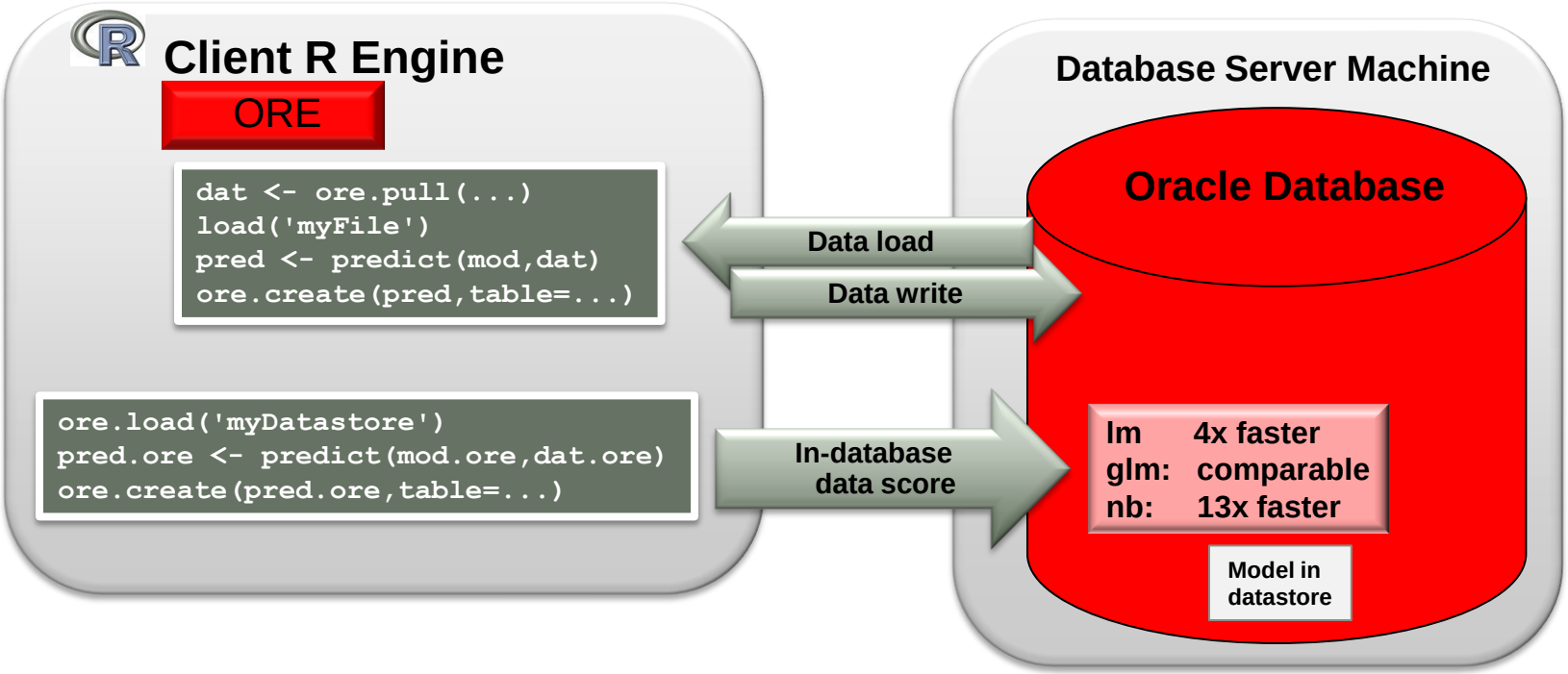
- Data sets relatively small to enable comparison with R
- Create reproducible test cases that don’t require Exadata-size machine
- Leveraged single node of Exadata X3-8

Predictive Model Performance

OREdm performance gains for model building

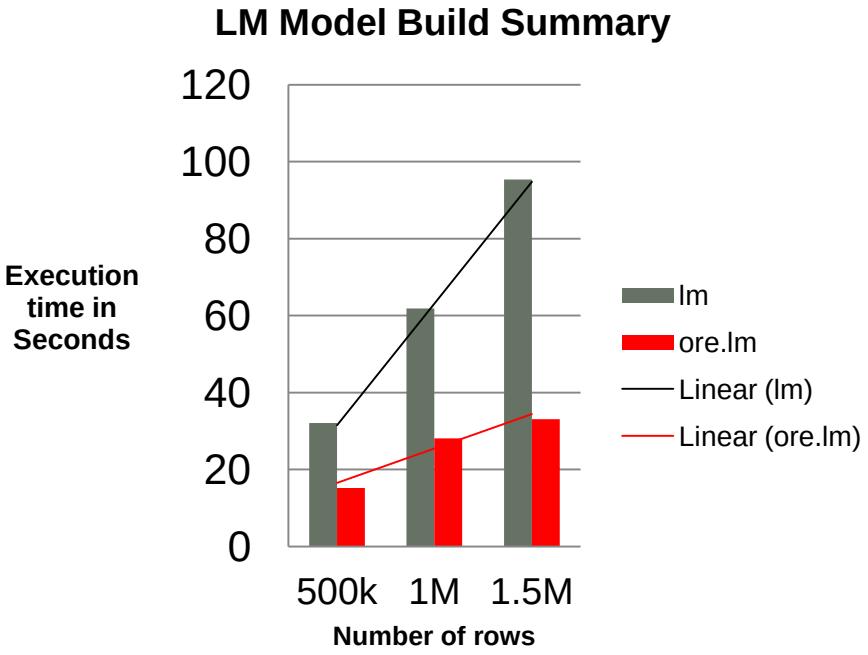


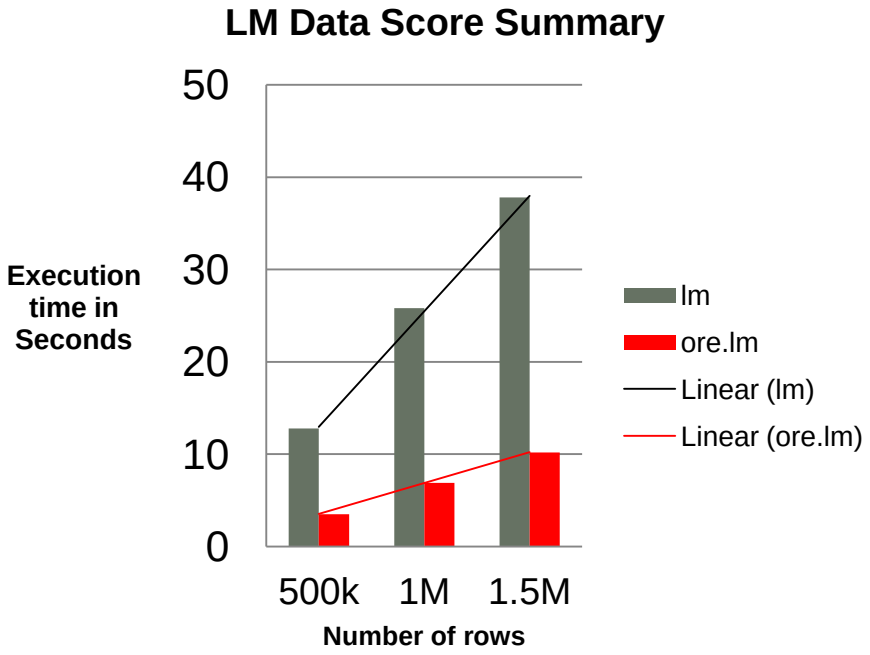
OREdm performance gains for data scoring



lm and ore.lm 500k-1.5M rows – 3 columns

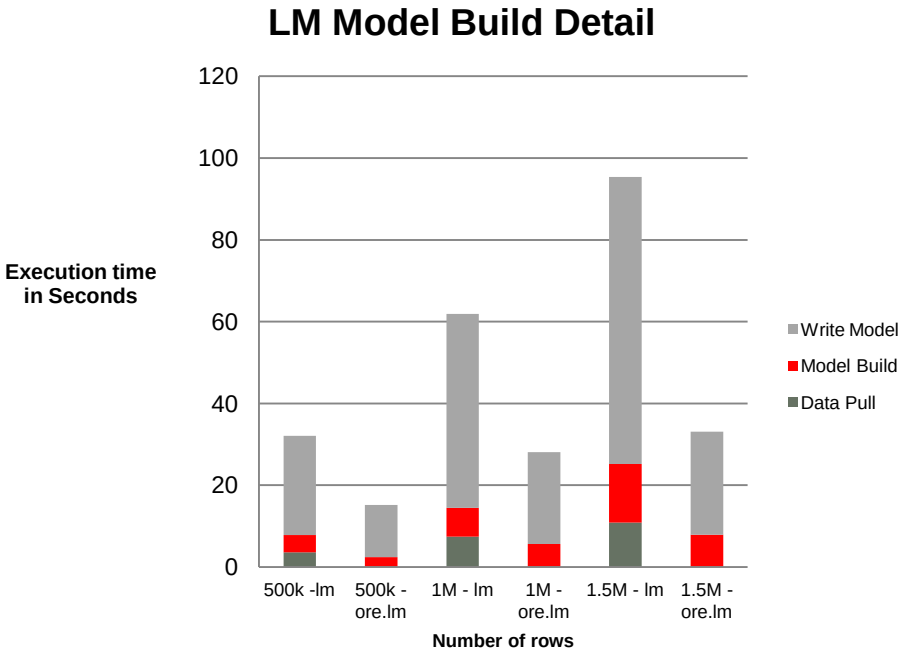
Summary – time in seconds

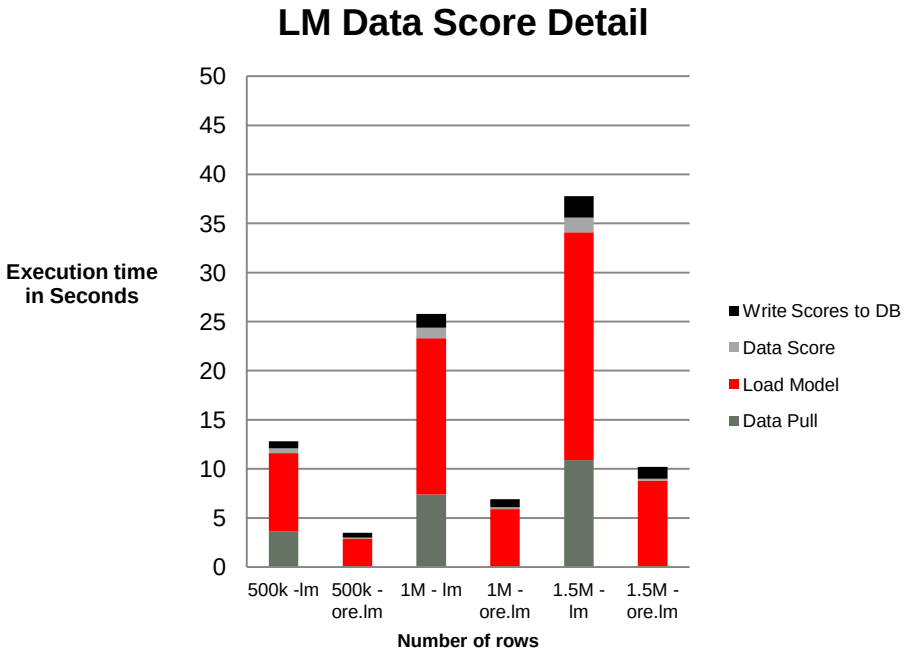




lm and ore.lm 500k-1.5M rows – 3 columns

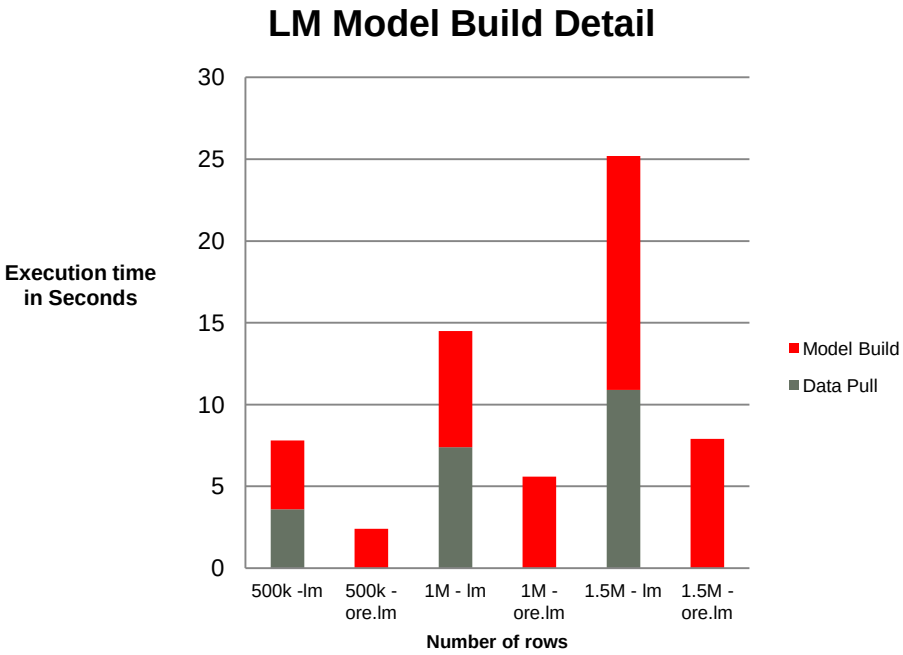
Drill down – time in seconds

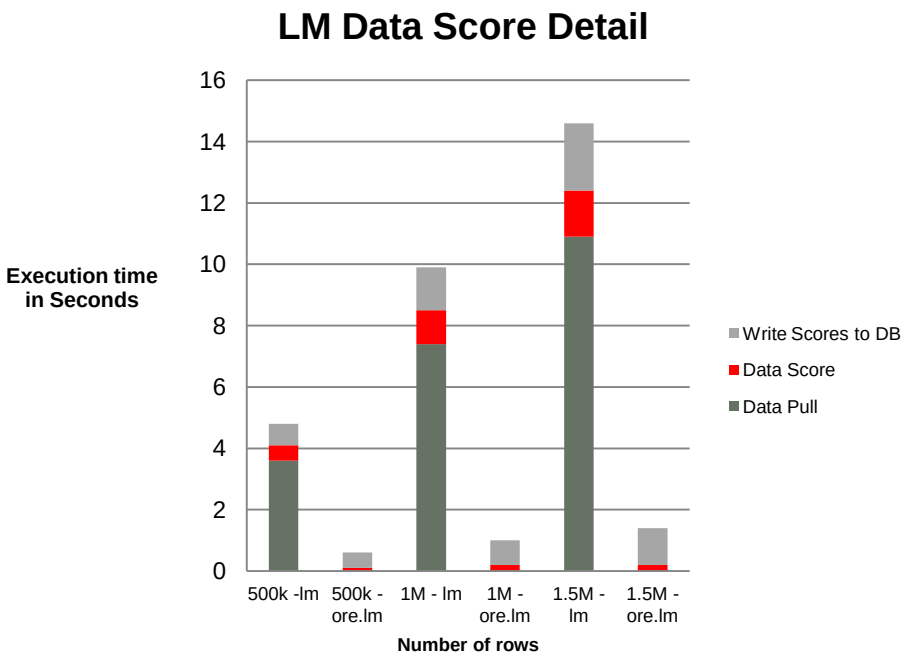




lm and ore.lm 500k-1.5M rows – 3 columns

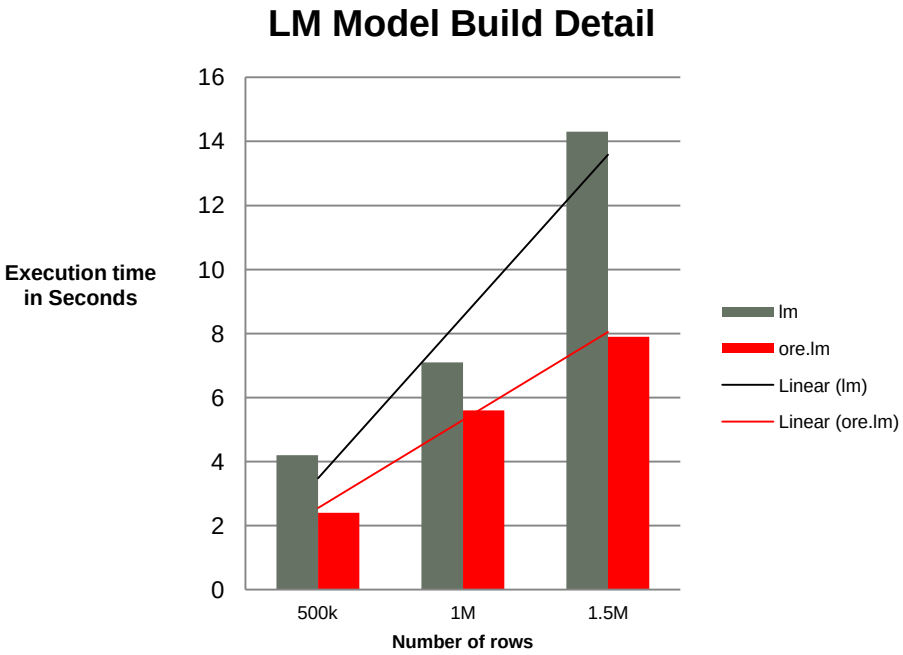
Drill down – time in seconds excluding model save and load

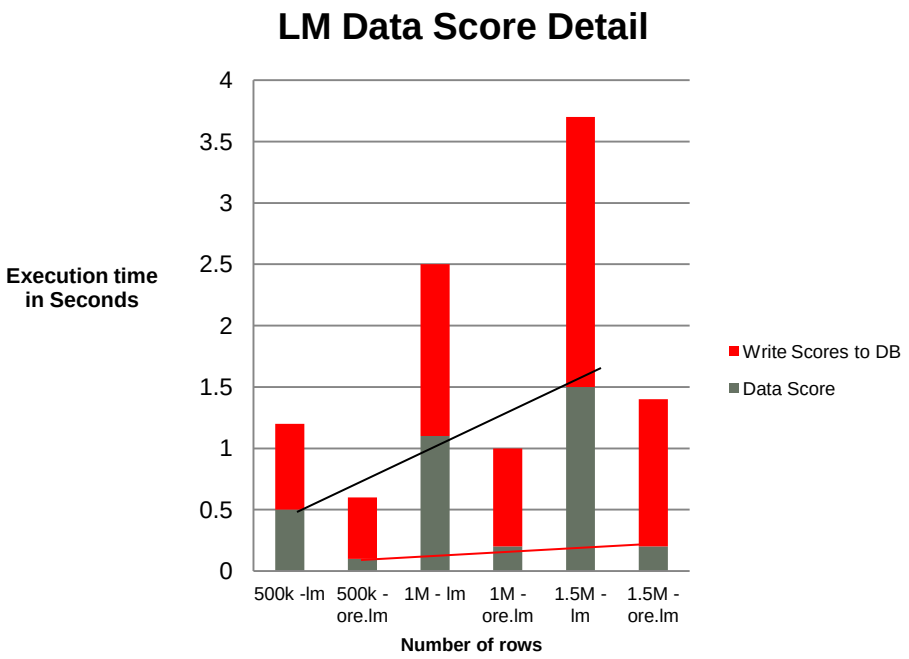




lm and ore.lm 500k-1.5M rows – 3 columns

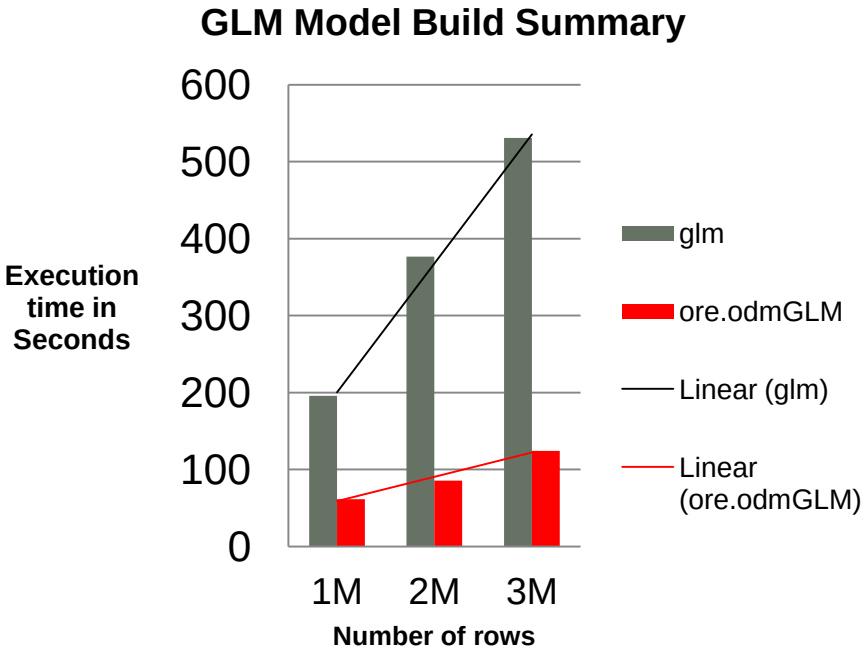
Drill down – time in seconds also excluding data pull

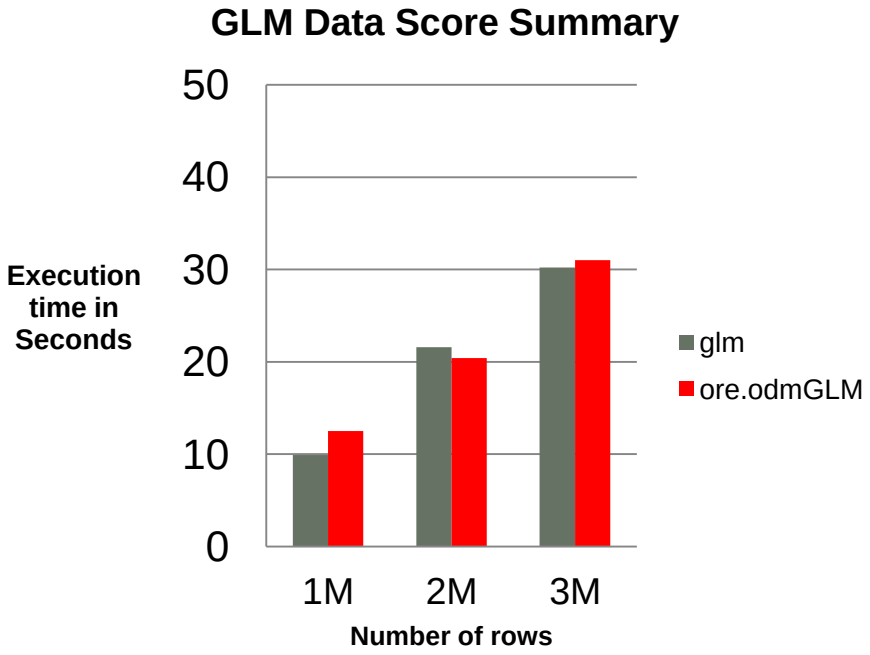




glm and ore.odmGLM 1M-3M rows – 3 columns

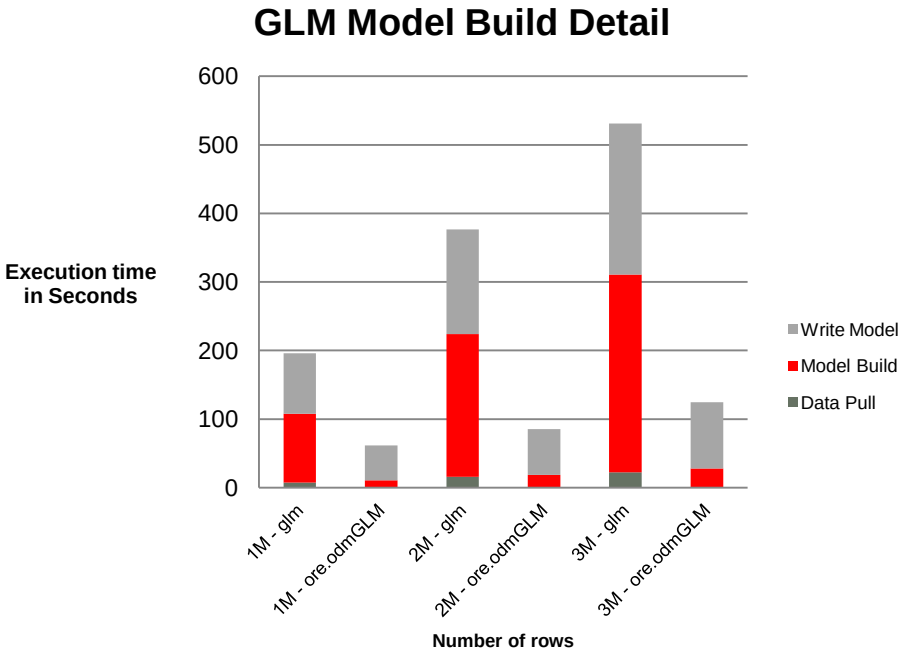
Summary – time in seconds

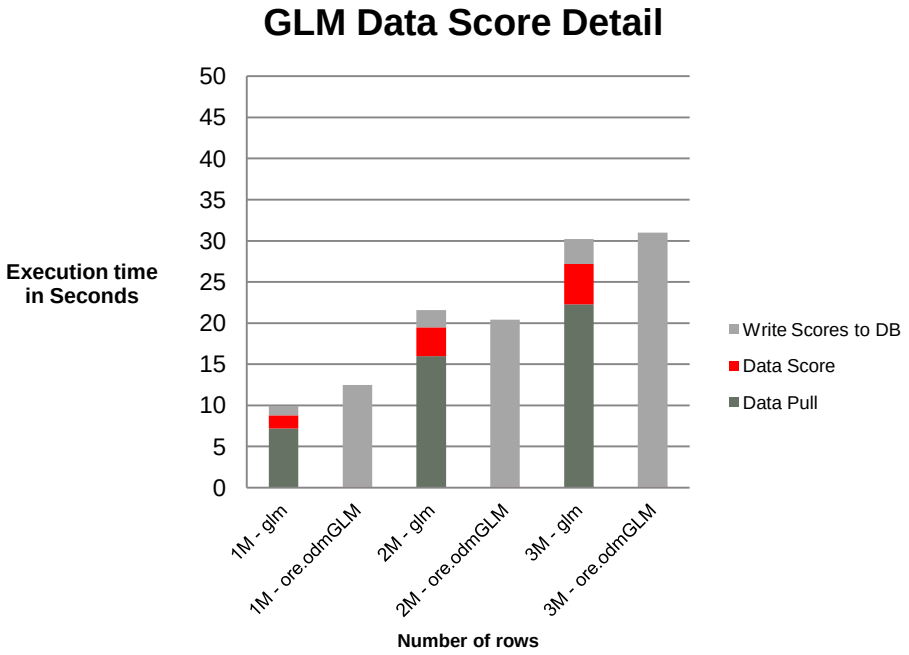




glm and ore.odmGLM 1M-3M rows – 3 columns

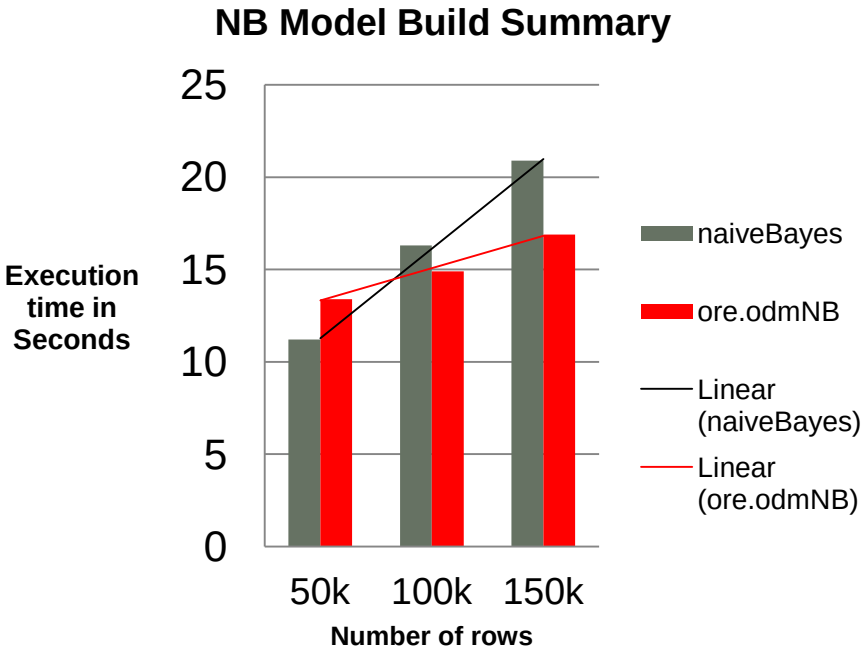
Drill down – time in seconds

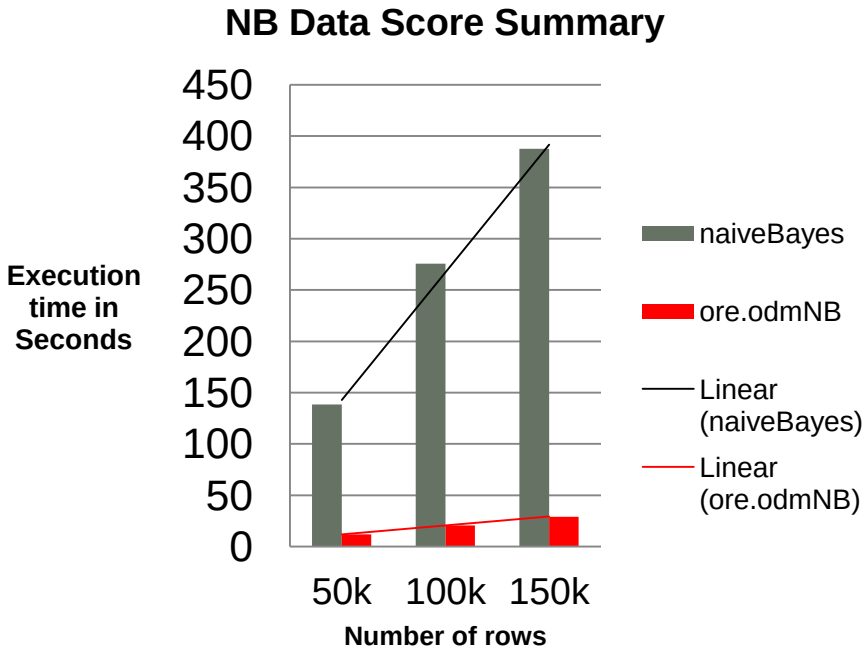




Naïve Bayes 50k-150k rows – 20 columns

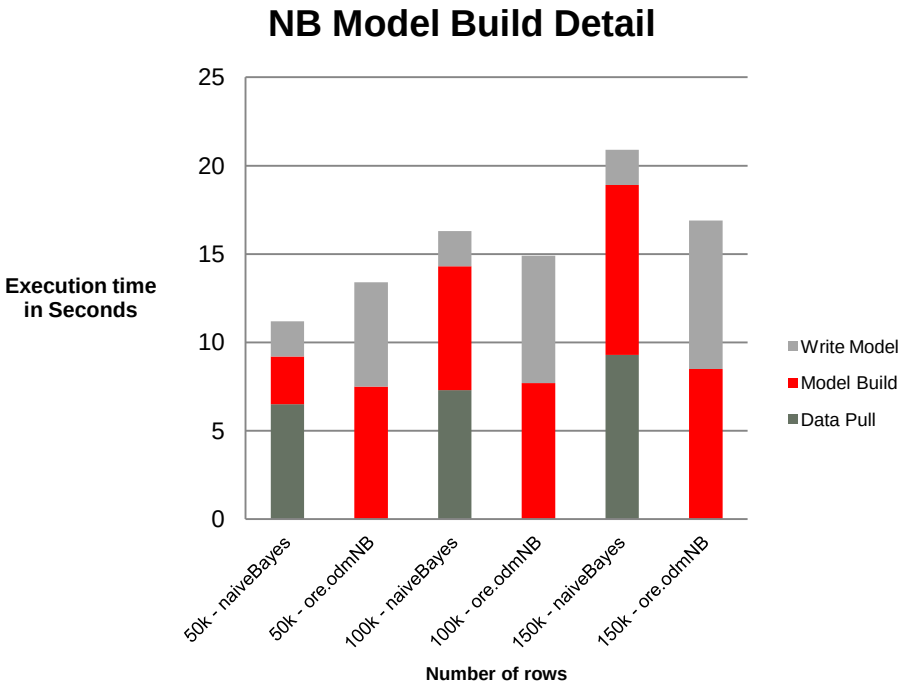
Summary – time in seconds

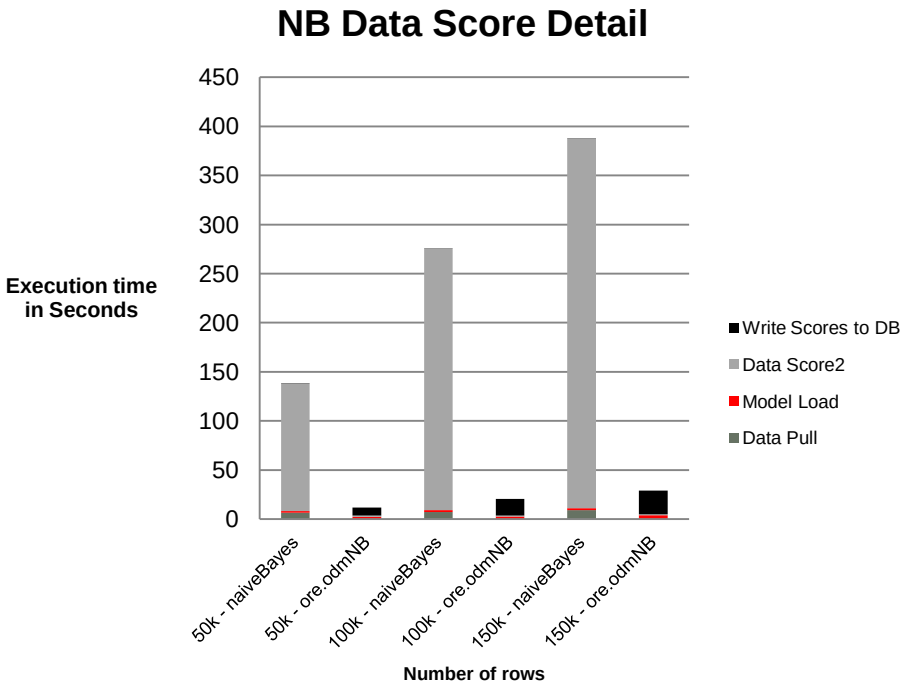




Naïve Bayes 50k-150k rows– 20 columns

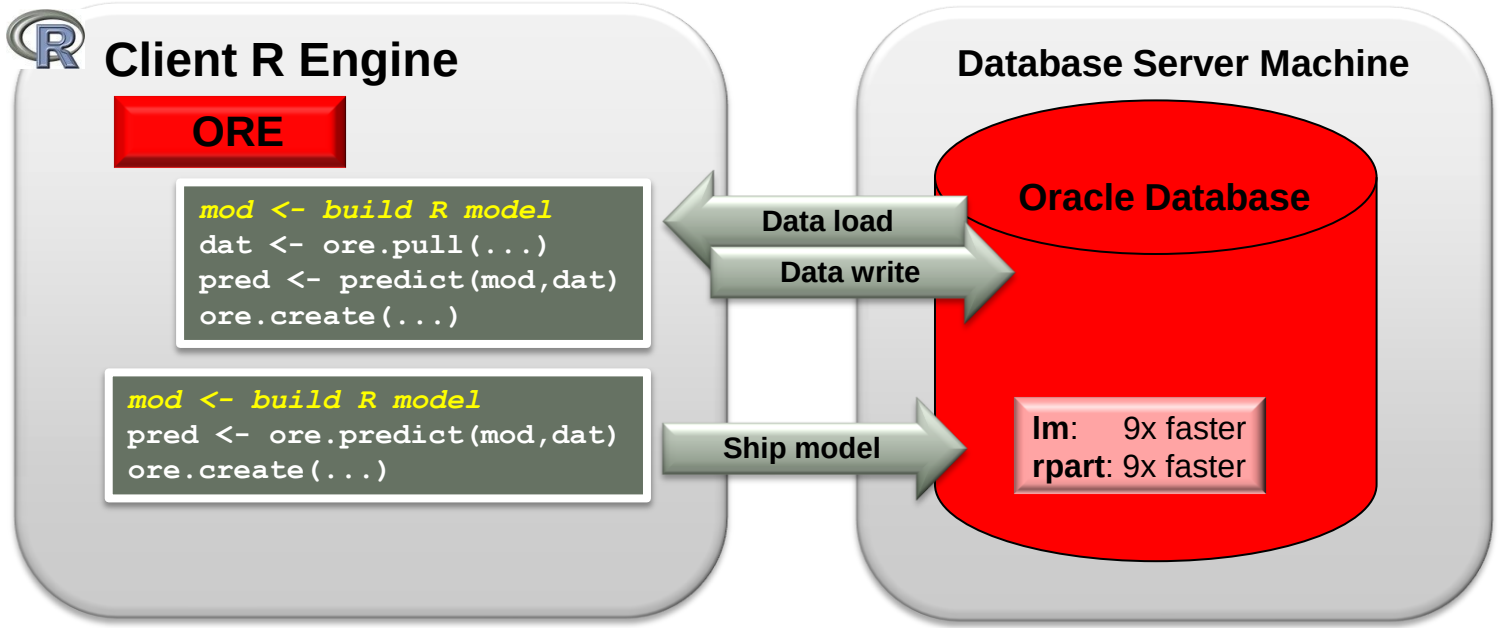
Drill down – time in seconds





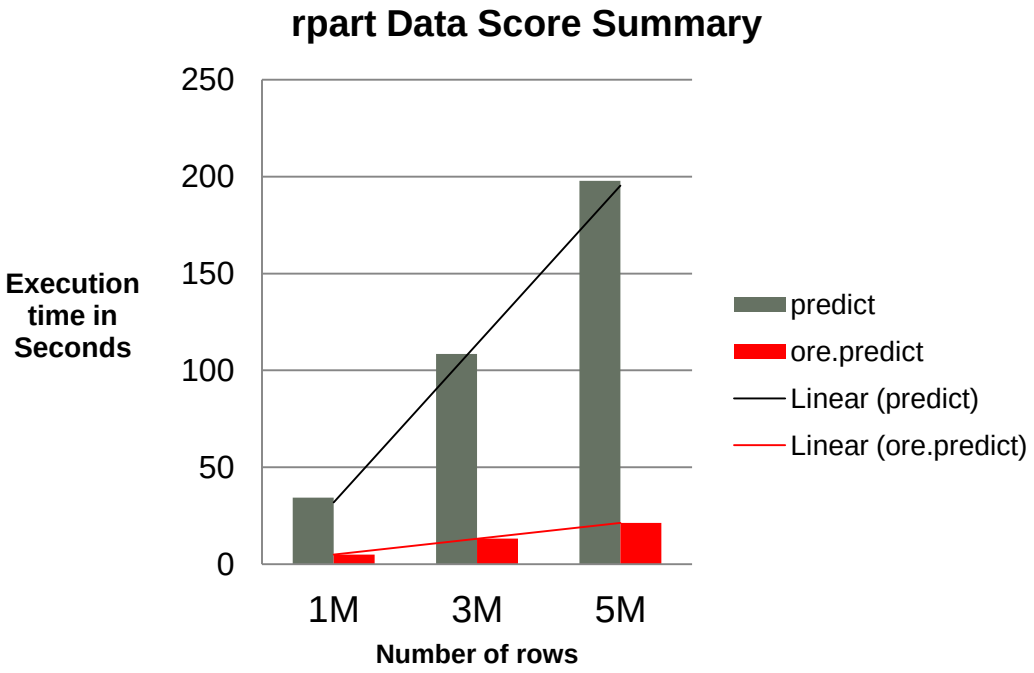
OREpredict Performance

OREpredict performance gains – 9x faster



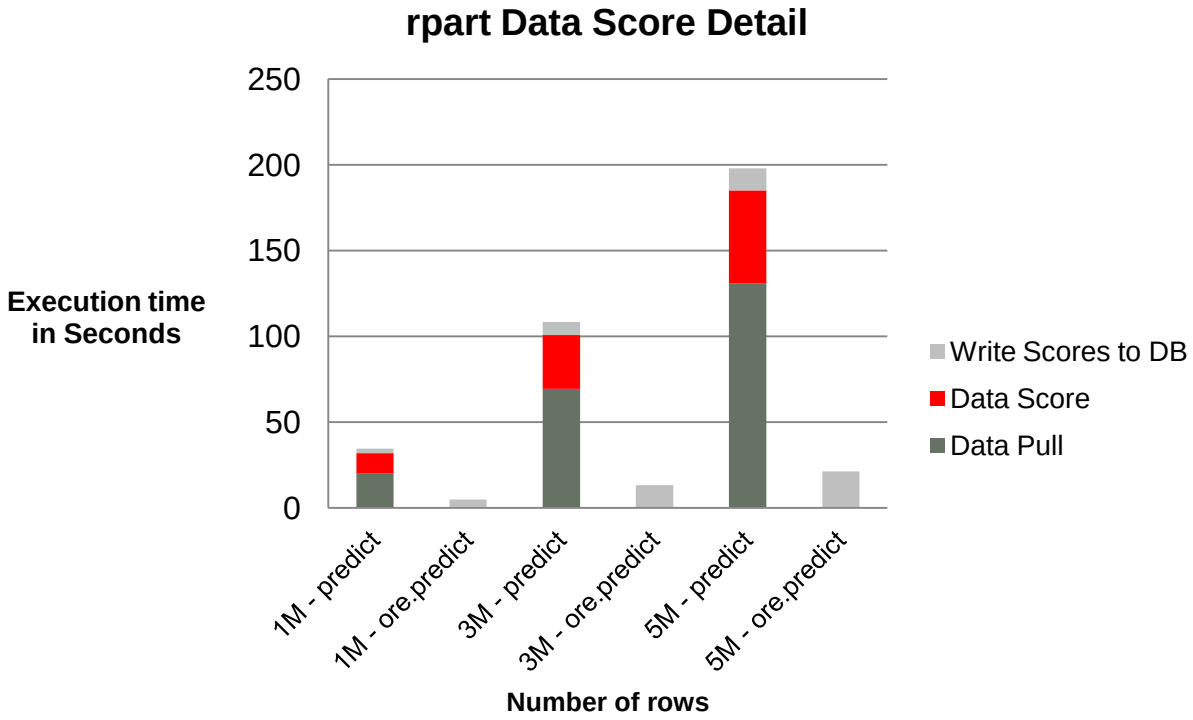
rpart 1M-5M rows – 20 col

Summary – time in seconds



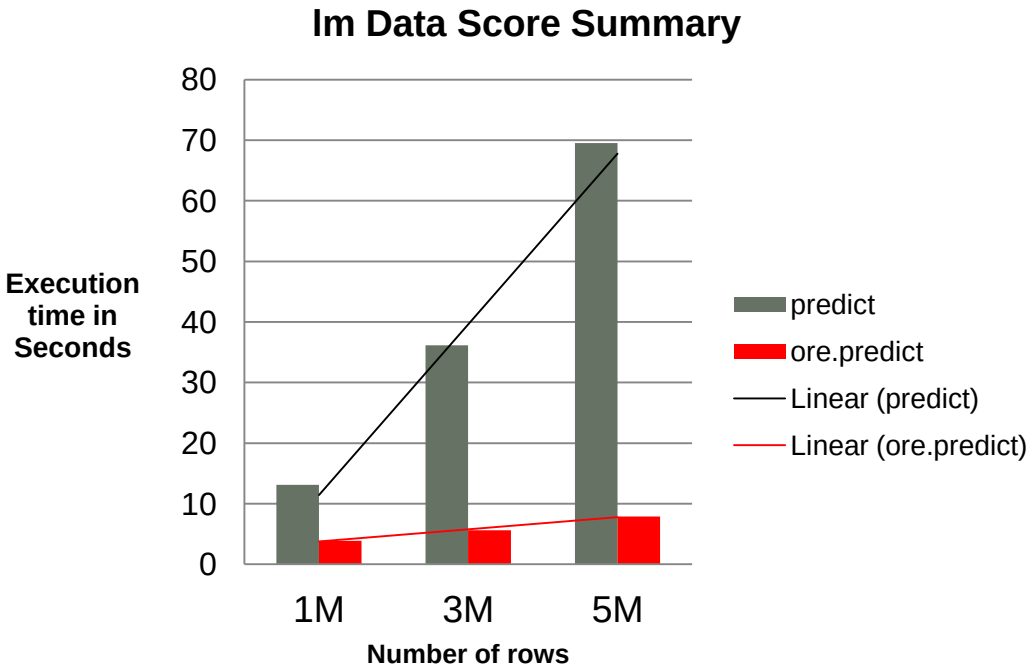
rpart 1M-5M rows – 20 col

Drill down – time in seconds



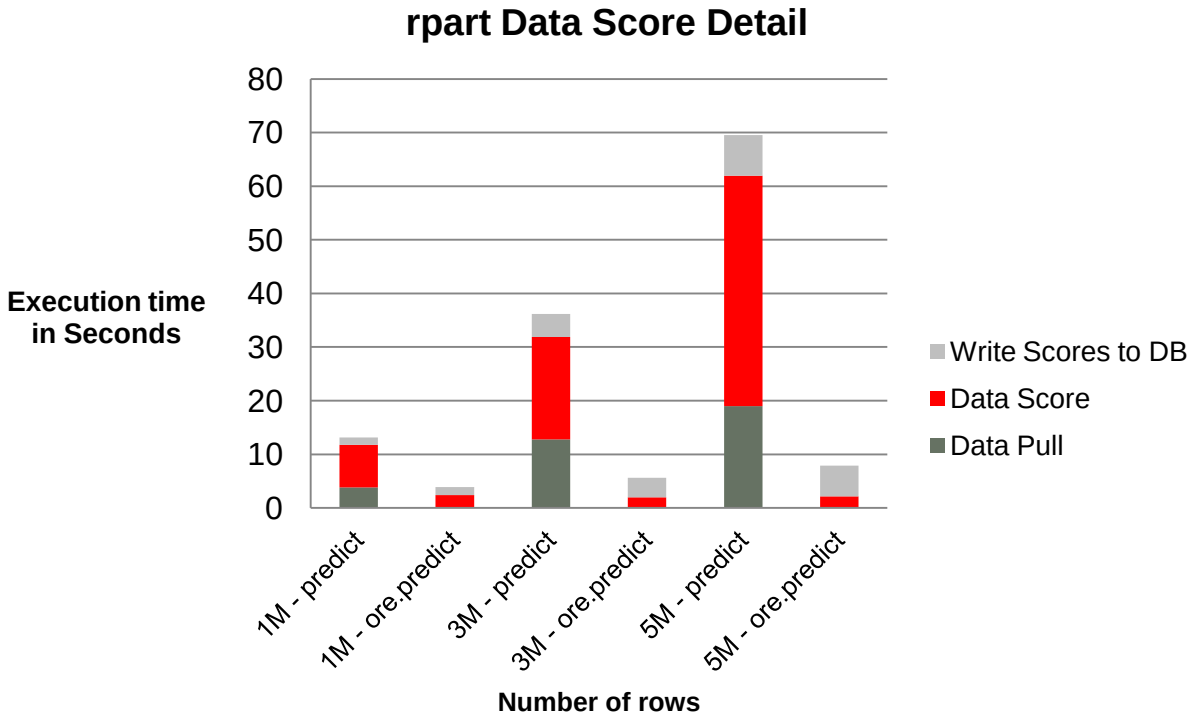
lm 1M-5M rows – 4 col

Summary – time in seconds



lm 1M-5M rows – 4 col

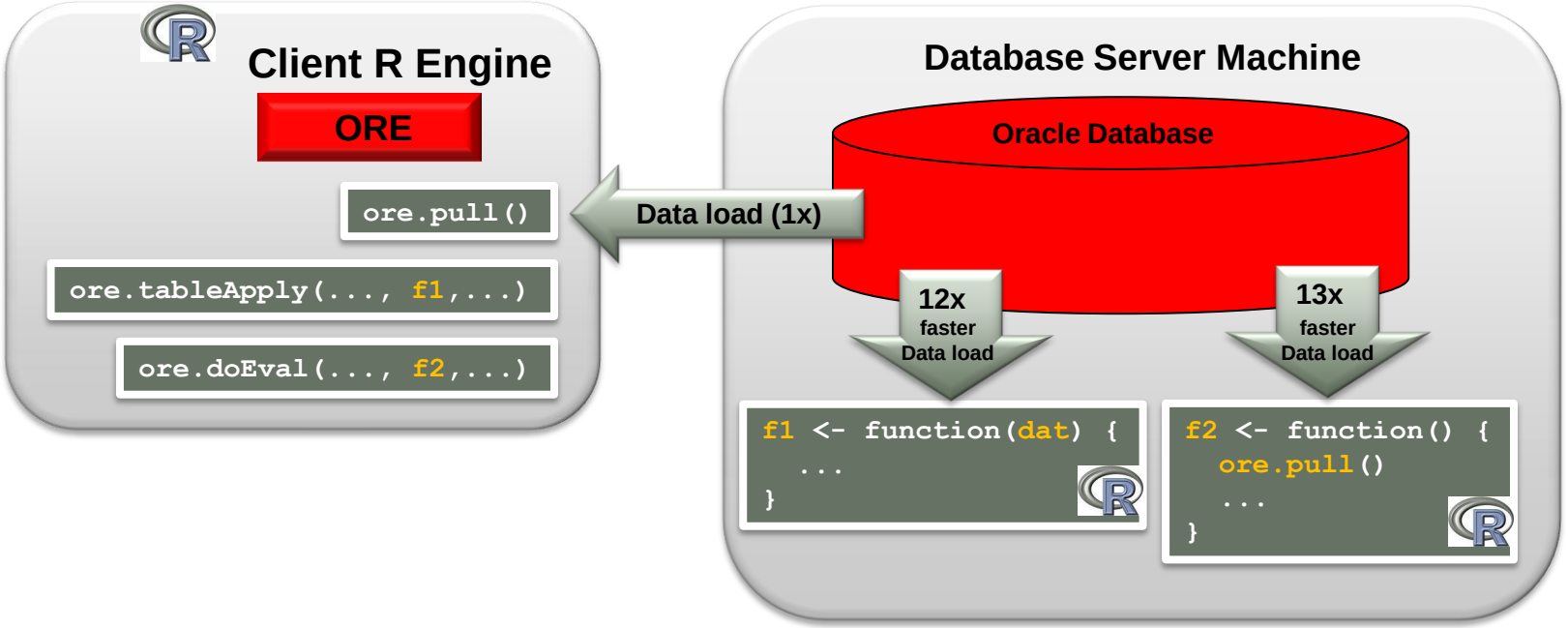
Drill down – time in seconds



Data Load Performance

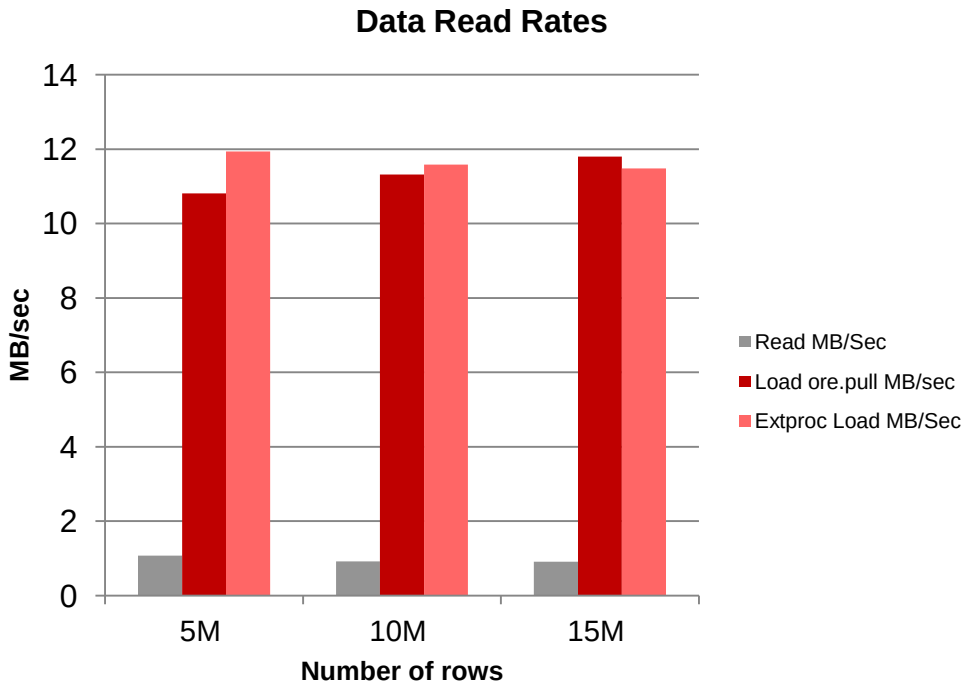
Embedded R Execution data load performance gains

Using Embedded R Execution greatly reduces overall execution time



Data Read 5M-15M rows – 11 cols

Row Count	MB	Local Read MB/Sec	Extproc Load MB/Sec	Load ore.pull MB/sec
5M	80MB	1.07	11.94	10.81
10M	160MB	0.92	11.59	11.32
15M	240MB	0.91	11.48	11.80



Summary

- ORE provides a rich set of predictive analytics capabilities
  - In-database algorithms from ODM
  - New ORE algorithms
  - Ability to supplement using R-based CRAN packages
  - Ability to score data in-database using R models
- In-database model building and scoring yields performance and scalability
  - Data movement at volume can be expensive
  - Improved algorithm implementations have desirable side effects

Resources

- **Blog:** https://blogs.oracle.com/R/
- **Forum:** https://forums.oracle.com/forums/forum.jspa?forumID=1397
- **Oracle R Distribution:** http://www.oracle.com/technetwork/indexes/downloads/r-distribution-1532464.html
- **ROracle:** http://cran.r-project.org/web/packages/ROracle
- **Oracle R Enterprise:** http://www.oracle.com/technetwork/database/options/advanced-analytics/r-enterprise
- **Oracle R Connector for Hadoop:** http://www.oracle.com/us/products/database/big-data-connectors/overview


ORACLE®