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
• OREdm package
• OREmodels package
  – ore.lm
  – ore.stepwise
  – ore.neural
  – ore.glm
• OREpredict package
• OREeda package
  – ore.esm
ORE Analytics Packages

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

©2014 Oracle – All Rights Reserved
High performance in-database predictive techniques available through ORE packages

OREdm
- Support Vector Machine
- GLM
- k-Means clustering
- OC clustering
- Naïve Bayes
- Decision Trees
- Association Rules
- Attribute Importance

OREmodels
- Neural Networks
- Linear Regression
- Stepwise Regression
- Generalized Linear Model
OREdm Features

• Function signatures conform 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

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Main R Function</th>
<th>Mining Type / Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Association Rules</td>
<td>ore.odmAssocRules</td>
<td>Association Rules</td>
</tr>
<tr>
<td>Minimum Description Length</td>
<td>ore.odmA1</td>
<td>Attribute Importance for Classification or Regression</td>
</tr>
<tr>
<td>Decision Tree</td>
<td>ore.odmDT</td>
<td>Classification</td>
</tr>
<tr>
<td>Generalized Linear Models</td>
<td>ore.odmGLM</td>
<td>Classification Regression</td>
</tr>
<tr>
<td>KMeans</td>
<td>ore.odmKMeans</td>
<td>Clustering</td>
</tr>
<tr>
<td>Naïve Bayes</td>
<td>ore.odmNB</td>
<td>Classification</td>
</tr>
<tr>
<td>Non-negative Matrix Factorization</td>
<td>ore.odmNFM</td>
<td>Feature Extraction</td>
</tr>
<tr>
<td>Orthogonal Partitioning</td>
<td>ore.odmOC</td>
<td>Clustering</td>
</tr>
<tr>
<td>Support Vector Machine</td>
<td>ore.odmSVM</td>
<td>Classification Regression Anomaly Detection</td>
</tr>
</tbody>
</table>

©2014 Oracle – All Rights Reserved
Attribute Importance

- Compute the relative importance of predictor variables for predicting a response (target) variable
- Gain insight into the relevance of variables to guide manual variable selection or reduction, with the goal to reduce predictive model build time and/or improve model accuracy
- Attribute Importance uses a Minimum Description Length (MDL) based algorithm that ranks the relative importance of predictor variables in predicting a specified response (target) variable
- Pairwise only – each predictor with the target
- Supports categorical target (classification) and numeric target (regression)
ore.odmAI

Attribute Importance

ore.odmAI(
    formula, # formula specifying attributes for model build
    data, # ore.frame of the training dataset
    auto.data.prep = TRUE, # Setting to perform automatic data preparation
    na.action = na.pass) # Allows missing values (na.pass), or removes rows with
# missing values (na.omit)
)
Basic Argument Concepts

• **formula**
  - Form `response ~ terms` where 'response' is the numeric or character response vector and 'terms' is a series of terms, i.e., column names, to include in the analysis
  - Multiple terms are specified using '+' between column names
  - Use `response ~ .` if all columns in 'data' should be used for model building. Functions can be applied to 'response' and 'terms' to realize transformations. To exclude columns, use '-' before each column name to exclude.
Basic Argument Concepts

• **auto.data.prep**
  – If TRUE, automatically performs the data transformations required by the algorithm
  – Transformation instructions are embedded in the in-database model

• **Types of transformations**
  – Binning
    • reduces cardinality of continuous and discrete data
    • improve resource utilization and model build response time dramatically without significant loss in model quality
    • can improve model quality by strengthening relationships between attributes
  – Normalization
    • reduces range of numerical data, e.g., between 0 and 1
  – Outlier Treatment
    • an outlier deviates significantly from most other values in the column
    • outliers can skew data and interfere with transformations like normalization or binning
    • outlier treatment methods include trimming or clipping

©2014 Oracle – All Rights Reserved
Basic Argument Concepts

• na.action
  – By default, allows missing values ('na.pass'), or removes rows with missing values ('na.omit')
ore.odmAI - Example

Attribute Importance

R> LONGLEY <- ore.push(longley)
R> head(LONGLEY)

GNP.deflator  GNP Unemployed Armed.Forces Population Year Employed
1947  83.0 234.289 235.6 159.0 107.608 1947  60.323
1948  88.5 259.426 232.5 145.6 108.632 1948  61.122
1949  88.2 258.054 368.2 161.6 109.773 1949  60.171
1950  89.5 284.599 335.1 165.0 110.929 1950  61.187
1951  96.2 328.975 209.9 309.9 112.075 1951  63.221
1952  98.1 346.999 193.2 359.4 113.270 1952  63.639

R> ore.odmAI(Employed ~ ., LONGLEY)

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

Importance:

<table>
<thead>
<tr>
<th>importance rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>Year 0.4901166</td>
</tr>
<tr>
<td>Population 0.4901166</td>
</tr>
<tr>
<td>GNP 0.4901166</td>
</tr>
<tr>
<td>GNP.deflator 0.4901166</td>
</tr>
<tr>
<td>Armed.Forces 0.3648186</td>
</tr>
<tr>
<td>Unemployed 0.1318046</td>
</tr>
</tbody>
</table>
Attribute Importance - results

• importance
  – Relative metric indicating how much the variable contributes to predicting the target
  – Values > 0 contribute to prediction
  – Values <= do not contribute or add noise

• rank
  – Ordering of variables / attributes from most significant to least
Naïve Bayes

- Classification algorithm – simple probabilistic classifier
- Relies on Bayes’ theorem
  \[ P(A|B) = \frac{P(B|A)P(A)}{P(B)} \]
- Assumes independence of predictors
  - May not be the case, but works well in practice
- Conditional probabilities between each predictor and target multiplied to obtain prediction
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
  na.action = na.pass)
Basic Argument Concepts

• **class.priors**
  - Optional user-specified priors for the target classes
  - Specifying prior probabilities offsets distribution differences between training data and real population (scoring data)

• Use when one target value dominates in frequency
  - For example
    • telephone marketing campaign positive responses may be < 2%
    • occurrence of fraud in credit card transactions may be < 1%.
  - A classification model built with so few positive cases may not be able to distinguish characteristics of the two classes, resulting in a model that predicts the frequent class every time → use stratified sampling to balance the data set and set priors
  - Such models may be accurate, but not be very useful
  - Do not rely solely on accuracy when judging the quality of a classification model

• Stratified sampling and anomaly detection are alternatives to compensating for data distribution issues
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")
nrows <- nrow(t3)
set.seed(seed=6218945)
random.sample <- sample(1:nrows, ceiling(nrows/2))
t3.train <- t3[random.sample,]
t3.test <- t3[setdiff(1:nrows,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)

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 Positive 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()
ROC Curve

Call:
  rclassNB(formula = survived ~ pclass + sex + age + fare + embarked,
            data = titanic, class.priors = priors)

Settings:
  value
    prep.auto on

Prior:
  No  Yes
    0.9  0.1

Tables:
  $embarked
    'Cherbourg', 'Queenstown', 'Southampton'
    No 0.1562500 0.04375
    Yes 0.5781250 0.621875

  $fare
    [ ; 51.931249600000001; 51.931249600000001] (51.931249600000001; )
    No 0.91570938 0.6863442
    Yes 0.87007832 0.3263208

  $pclass
    '1st', '2nd', '3rd'
    No 0.34177722 0.6582278
    Yes 0.6346154 0.3653846

  $sex
    female male
    No 0.6709686 0.3290314
    Yes 0.6783351 0.3216649

Levels:
  [1] "No" "Yes"
Naïve Bayes – model object

- **ore.odmNB object**
  - **name** of the model
  - **settings** used to build the model
  - **attributes** used to build the model: name, type (numerical or categorical), data type, data length (size), precision and scale for numeric data, and whether the variable is the target
  - **apriori** table with class distribution for the dependent variable
  - **tables** is a list with one for each predictor variable with conditional probabilities
  - **levels** is a vector of unique target class values
Support Vector Machine

- Suite of algorithms, adaptable for use with a variety of problems and data
- By swapping one *kernel* for another, SVM can fit diverse problem spaces
- Concept
  - Data records with N attributes can be thought of as points in N-dimensional space
  - SVM attempts to separate the points into subsets with homogeneous target values, by hyperplanes in the linear case, and in the non-linear case (Gaussian) by non-linear separators
  - SVM finds the vectors that define the separators giving the widest separation of classes (the “support vectors”).
- SVM solves regression problems by defining an N-dimensional “tube” around the data points, determining the vectors giving the widest separation
- SVM can emulate some traditional methods, such as linear regression and neural networks, but goes far beyond those methods in flexibility, scalability, and speed
  - For example, SVM can act like a neural net in calculating predictions, while a neural net might mistake a local change in direction as a point of minimum error, SVM works to find the global point of minimum error

©2014 Oracle – All Rights Reserved
ore.odmSVM

Support Vector Machine

ore.odmSVM(
    formula,  # specifies attributes for model build
    data,    # ore.frame containing the training dataset
    mining.function,  # Type of model: "classification", "regression"
                      # or "anomaly.detection"
    auto.data.prep = TRUE,  # Setting to perform automatic data preparation
    class.priors = NULL,    # Data frame containing target class priors
    active.learning = TRUE,  # Setting for enabling active learning
    complexity.factor = "system.determined",  # Setting for complexity factor for SVM
    conv.tolerance = 0.0001,  # Setting for convergence tolerance for SVM
    epsilon = "system.determined",  # Setting for epsilon for SVM Regression
    kernel.function = "system.determined",  # Setting for kernel function (SVMS_GAUSSIAN or SVMS_LINEAR)
    kernel.cache.size = 50000000,  # Setting for Gaussian kernel cache size (bytes)
    std.dev = "system.determined",  # Setting for standard deviation for SVM Gaussian kernel
    outlier.rate = 0.1,  # Setting for desired rate of outliers in dataset (1class SVM)
    na.action = na.pass  # Allow missing values in rows by default, or na.omit
)
Basic Argument Concepts

• **class.weights**
• **active.learning** - enabled by default
  – optimization method to control model growth and reduce model build time
  – Without active learning, SVM models grow as the size of the build data set increases, which effectively limits SVM models to small and medium size training sets (less than 100,000 cases)
  – With active learning, SVM models can be built on very large training sets.
  – Active learning forces the SVM algorithm to restrict learning to the most informative training examples and not to attempt to use the entire body of data. In most cases, the resulting models have predictive accuracy comparable to that of a standard (exact) SVM model
  – Active learning provides a significant improvement in both linear and Gaussian SVM models, whether for classification, regression, or anomaly detection. However, active learning is especially advantageous for the Gaussian kernel, because nonlinear models can otherwise grow to be very large and can place considerable demands on memory and other system resources
Basic Argument Concepts

- **complexity.factor**
  - regularization setting that balances complexity of the model against model robustness to achieve good generalization on new data
  - data-driven approach to automatically determine the complexity factor

- **conv.tolerance**
  - convergence tolerance criterion for completing the model training process, default .001

- **epsilon**
  - regularization setting for regression, similar to complexity factor
  - specifies the allowable residuals, or noise, in the data
Basic Argument Concepts

- **kernel.function** – linear or Gaussian
  - A kernel is a function that transforms the input data to a high-dimensional space where the problem is solved. Kernel functions can be linear or nonlinear.
  - The algorithm automatically uses the kernel function that is most appropriate to the data if not specified.
  - Linear kernel when # attributes > 100 in training data, else Gaussian kernel
    - # attributes reflects categorical columns exploded to numeric attributes

- **kernel.cache.size**
  - Memory allocated to Gaussian kernel cache maintained in memory to improve model build time, default 50 MB

- **std.dev**
  - Controls spread of Gaussian kernel function

- **outlier.rate**
  - For anomaly detection
  - Expected outlier rate in anomaly detection, default 0.1
predict.ore.odmSVM <- function (object, newdata, supplemental.cols = NULL, # Columns to retain in the output
    type = c("class","raw"), # "raw" - cond. a-posteriori probs for each class returned,
    # else class with max prob (TBD for compatibility with e1071)
    na.action = na.pass,...) # allow missing values in rows by default, or na.omit

Support Vector Machine

©2014 Oracle – All Rights Reserved
Basic Argument Concepts

• **supplemental.cols**
  - Columns from `newdata` to include as columns in the `ore.frame` prediction result
  - Use to include specific columns in the prediction result for easier analysis

• **type = c("class","raw"), if a classification model…**
  - "raw" provides probability for each class returned
  - "class" returns the class with the maximum probability
  - default `c("class","raw")` returns both
ore.odmSVM – Example

Support Vector Machine

```r
x <- seq(0.1, 5, by = 0.02)
y <- log(x) + rnorm(x, sd = 0.2)
dat <- ore.push(data.frame(x=x, y=y))

# Regression
svm.mod <- ore.odmSVM(y~x, dat, "regression", kernel.function="linear")
summary(svm.mod)
coef(svm.mod)
svm.res <- predict(svm.mod, dat, supplemental.cols="x")
head(svm.res, 6)
```
# Set up data set
m <- mtcars
m$gear <- as.factor(m$gear)
m$cyl <- as.factor(m$cyl)
m$vs <- as.factor(m$vs)
m$ID <- 1:nrow(m)
MTCARS <- ore.push(m)

# Classification
svm.mod <- ore.odmSVM(gear ~ .-ID,
                        MTCARS,"classification")
summary(svm.mod)
coef(svm.mod)
svm.res <- predict(svm.mod, MTCARS,"gear")
head(svm.res)
svm.res <- predict(svm.mod, MTCARS,"gear",type="raw")
head(svm.res)
svm.res <- predict(svm.mod, MTCARS,"gear",type="class")
head(svm.res)
with(svm.res, table(gear,PREDICTION))  # confusion matrix

# Anomaly Detection
svm.mod <- ore.odmSVM(~ .-ID, MTCARS,"anomaly.detection")
summary(svm.mod)
svm.res <- predict(svm.mod, MTCARS,"ID")
head(svm.res)
table(svm.res$PREDICTION)
SVM – model object

• **ore.odmSVM object**
  - **name** of the model
  - **settings** used to build the model
  - **attributes** used to build the model: name, type (numerical or categorical), data type, data length (size), precision and scale for numeric data, and whether the variable is the target
  - **fit.values** is an ore.frame of the actual column and predicted column. For regression, the columns are 'ACTUAL' and 'PREDICTED'. For classification, the columns are 'ACTUAL','PREDICTED','PROBABILITY'. For anomaly detection, the columns are 'PREDICTED' and 'PROBABILITY'.
  - **residuals** For regression models, an ore.numeric vector containing the residual values (PREDICTED - ACTUAL).
  - **formula** is the symbolic description of the model fitted
  - **call** is the invocation parameters of the function

• If built with a linear kernel, the following are also returned
  - **coefficients** of the SVM model, one for each predictor variable. If auto.data.prep is set to TRUE, these coefficients will be in the transformed space (after automatic outlier-aware normalization is applied)
Cluster Description

- Centroids
- Histograms
Cluster Rules

Cluster 1:
0 < age ≤ 35 AND 0 < income ≤ 50K
Cluster 2:
30 < age ≤ 55 AND 40K < income ≤ 80K
Clustering Hierarchy

• Binary tree
  – balanced
  – unbalanced

• Splitting predicates

AGE < 40
AGE ≥ 40

SMOKE
~SMOKE

MALE
FEMALE
K-Means clustering

• Identify distinct segments of a population
• Explain the common characteristics of members of a cluster
• Determine what distinguishes members of one cluster from members of another cluster
• Partitions a set of observations into $k$ partitions, or clusters
• Each observation belongs to the cluster with the nearest centroid or center, which is the mean of the observations variables
• Distance can be computed in various ways, e.g., Euclidean or cosine
ore.odmKMeans

K-Means Clustering

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

- **num.centers** – number of clusters to create, > 1, default 10
- **block.growth** – numeric growth factor for memory to hold cluster data, [1..5], default 2
- **conv.tolerance** – numeric convergence tolerance setting, (0..0.5], default 0.01
- **distance.function**
  - distance function between instances and centroids
  - options: cosine, euclidean, or fast.cosine
  - default: euclidean
- **iterations** – maximum number of iterations, [1..20], default 3
Basic Argument Concepts

• **min.pct.attr.support**
  – minimum percent required for variables to appear in rules, \([0,1]\), default 0.1
  – The fraction of attribute values that must be non-null for variable to be included in rule description for cluster
  – Setting the parameter value too high in data with missing values can result in very short or even empty rules

• **num.bins**
  – number of histogram bins, > 0, default 10
  – specifies the number of bins in the variable histogram produced by k-Means
  – bin boundaries for each variable are computed globally on entire training data set
  – binning method is equi-width
  – all attributes have same number of bins except variables with a single value, which have only one bin

• **split.criterion**
  – split clusters by variance or size, default variance
  – use size for more balanced clusters, e.g., with text mining
ore.odmKMeans

K-Means Clustering

```r
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(lm,mod1)

rhs,cluster.id rhs.support rhs.conf lhs.support lhs.conf lhs.var lhs.var.run.val lhs.var.chr.val lhs.var.support lhs.var.conf
1 1 1 100 1,0 90 0,9 x -0,2084763 <NA> 90 0,20
2 1 1 100 1,0 90 0,9 x 1,802657 <NA> 90 0,20
3 1 1 100 1,0 90 0,9 y -0,2130689 <NA> 91 0,20
4 1 1 100 1,0 90 0,9 y 1,6791639 <NA> 91 0,20
5 2 2 50 0,5 45 0,9 x 0,3193567 <NA> 49 0,25
6 2 2 50 0,5 45 0,9 x 1,802657 <NA> 49 0,25
7 2 2 50 0,5 45 0,9 y 0,7086321 <NA> 45 0,50
8 2 2 50 0,5 45 0,9 y 1,6791639 <NA> 45 0,50
9 3 3 50 0,5 45 0,9 x -0,7561113 <NA> 45 0,50
10 3 3 50 0,5 45 0,9 x 0,3193567 <NA> 45 0,50
11 3 3 50 0,5 45 0,9 y -0,7426619 <NA> 49 0,60
12 3 3 50 0,5 45 0,9 y 0,7086321 <NA> 49 0,60

# '1'

r rhs,cluster.id rhs.support rhs.conf lhs.support lhs.conf lhs.var lhs.var.run.val lhs.var.chr.val lhs.var.support lhs.var.conf predicate
1 1 1 100 1,0 90 0,9 x x 90 0,2 <= 1,9021
1 1 1 100 1,0 90 0,9 x y 90 0,2 >= -0,2085
4 1 1 100 1,0 90 0,9 y x 91 0,2 <= 1,5762
3 1 1 100 1,0 90 0,9 y y 91 0,2 >= -0,2585

# '2'

r rhs,cluster.id rhs.support rhs.conf lhs.support lhs.conf lhs.var lhs.var.run.val lhs.var.chr.val lhs.var.support lhs.var.conf predicate
6 2 2 50 0,5 45 0,9 x x 49 0,25 <= 1,9021
5 2 2 50 0,5 45 0,9 x y 49 0,25 >= 0,3192
8 2 2 50 0,5 45 0,9 y x 45 0,50 <= 1,5762
7 2 2 50 0,5 45 0,9 y y 45 0,50 >= 0,7086

# '3'

r rhs,cluster.id rhs.support rhs.conf lhs.support lhs.conf lhs.var lhs.var.run.val lhs.var.chr.val lhs.var.support lhs.var.conf predicate
10 3 3 50 0,5 45 0,9 x x 45 0,8 <= 0,3192
8 3 3 50 0,5 45 0,9 x y 46 0,8 >= 0,7361
12 3 3 50 0,5 45 0,9 y x 49 0,6 <= 0,7086
11 3 3 50 0,5 45 0,9 y y 43 0,6 >= -0,7427
```

©2014 Oracle – All Rights Reserved
## ore.odmKMeans – results

```
<table>
<thead>
<tr>
<th>cluster_id</th>
<th>variable</th>
<th>bin_id</th>
<th>lower_bound</th>
<th>upper_bound</th>
<th>label count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>x</td>
<td>-0.7391113</td>
<td>-0.2084637</td>
<td>-7.3E+00</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>x</td>
<td>-0.2084637</td>
<td>0.3131387</td>
<td>-2.085E-01</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>x</td>
<td>0.3131387</td>
<td>0.8467337</td>
<td>3.132E-01</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>x</td>
<td>0.8467337</td>
<td>1.3744287</td>
<td>0.460E+00</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>x</td>
<td>1.3744287</td>
<td>1.9026137</td>
<td>1.374E+00</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>y</td>
<td>-0.7426619</td>
<td>-0.2586669</td>
<td>-7.427E-01</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>y</td>
<td>-0.2586669</td>
<td>0.2346387</td>
<td>-2.586E-01</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>y</td>
<td>0.2346387</td>
<td>0.7086331</td>
<td>2.249E-01</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>y</td>
<td>0.7086331</td>
<td>1.1923369</td>
<td>1.132E+00</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>y</td>
<td>1.1923369</td>
<td>1.6761359</td>
<td>1.132E+00</td>
</tr>
<tr>
<td>11</td>
<td>2</td>
<td>x</td>
<td>-0.7391113</td>
<td>-0.2084637</td>
<td>-7.3E+00</td>
</tr>
<tr>
<td>12</td>
<td>2</td>
<td>x</td>
<td>-0.2084637</td>
<td>0.3131387</td>
<td>-2.085E-01</td>
</tr>
<tr>
<td>13</td>
<td>2</td>
<td>x</td>
<td>0.3131387</td>
<td>0.8467337</td>
<td>3.132E-01</td>
</tr>
<tr>
<td>14</td>
<td>2</td>
<td>x</td>
<td>0.8467337</td>
<td>1.3744287</td>
<td>0.460E+00</td>
</tr>
<tr>
<td>15</td>
<td>2</td>
<td>x</td>
<td>1.3744287</td>
<td>1.9026137</td>
<td>1.374E+00</td>
</tr>
<tr>
<td>16</td>
<td>2</td>
<td>y</td>
<td>-0.7426619</td>
<td>-0.2586669</td>
<td>-7.427E-01</td>
</tr>
<tr>
<td>17</td>
<td>2</td>
<td>y</td>
<td>-0.2586669</td>
<td>0.2346387</td>
<td>-2.586E-01</td>
</tr>
<tr>
<td>18</td>
<td>2</td>
<td>y</td>
<td>0.2346387</td>
<td>0.7086331</td>
<td>2.249E-01</td>
</tr>
<tr>
<td>19</td>
<td>2</td>
<td>y</td>
<td>0.7086331</td>
<td>1.1923369</td>
<td>1.132E+00</td>
</tr>
<tr>
<td>20</td>
<td>2</td>
<td>y</td>
<td>1.1923369</td>
<td>1.6761359</td>
<td>1.132E+00</td>
</tr>
</tbody>
</table>
```

### Cluster Histograms

```
Counts per bin by variable

Cluster 1
Cluster 2
Cluster 3
```

©2014 Oracle – All Rights Reserved
K-Means Clustering

```r
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 1.15331e-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  CLUSTER_ID
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
KMeans – model object

• ore.odmKMeans object
  – name …
  – settings …
  – attributes …
  – cluster contain general per-cluster information
  – leaf.cluster.count leaf clusters with support
  – taxonomy is the parent-child cluster relationship
  – centers are per cluster-attribute center (centroid) information
  – formula …
  – call …
Orthogonal Partitioning Clustering

**O-Cluster**

- Uses density-based approach
- Finds natural data clusters
- Creates unbalanced hierarchical trees
- Uses active sampling
O-Cluster
Density-Based Partitioning
When to Use O-Cluster?

- High number of records
  - needed for detailed histogram computation
- High number of attributes
- Presence of noise
- Numeric and categorical attributes
- Multi-modal density data
  - finds “natural” clusters, may not reach max number of clusters set by the user
Orthogonal Partitioning Clustering

- Creates a hierarchical grid-based clustering model
  - creates axis-parallel (orthogonal) partitions in the input attribute space
  - operates recursively
  - resulting hierarchical structure represents irregular grid that tessellates attribute space into clusters
  - resulting clusters define dense areas in the attribute space
- Clusters described by intervals along the attribute axes and corresponding centroids and histograms
- Parameter 'sensitivity' defines a baseline density level
  - Only areas with peak density above this baseline level can be identified as clusters
- O-Cluster separates areas of high density by placing cutting planes through areas of low density
  - O-Cluster needs multi-modal histograms (peaks and valleys)
  - If an area has projections with uniform or monotonically changing density, O-Cluster does not partition it
Orthogonal Partitioning Clustering

• O-Cluster reads the data in batches (the default batch size is 50000)
  – Only read another batch if, based on statistical tests, there may still exist clusters that it has not yet uncovered.
  – Since O-Cluster may stop the model build before it reads all of the data, it is highly recommended that the data be randomized
  – Binary attributes should be declared as categorical
  – O-Cluster maps categorical data to numerical values
  – Recommend to use ODM’s equi-width binning transformation with automated estimation of the required number of bins
  – Outliers can significantly impact clustering algorithms
    • Use a clipping transformation before binning or normalizing
    • Outliers with equi-width binning can prevent O-Cluster from detecting clusters - as a result, the whole population appears to falls within a single cluster.
ore.odmOC

Orthogonal Partitioning Clustering

ore.odmOC(formula, 
    data, 
    auto.data.prep = TRUE, 
    num.centers = 10, 
    max.buffer = 50000, 
    sensitivity = 0.5, 
    na.action = na.pass)

## S3 method for class 'ore.odmOC'
predict(object, 
    newdata, 
    supplemental.cols = NULL, 
    type = c("class","raw"), 
    na.action = na.pass,...)
Basic Argument Concepts

• **num.centers** – number of clusters to create, > 1, default 10

• **max.buffer** – maximum buffer size, >0, default 50000

• **sensitivity** – A fraction that specifies the peak density required for separating a new cluster. The fraction is related to the global uniform density. Value [0,1]. (default: 0.5)
OCluster – model object

- **ore.odmOC object**
  - **name**: name of model in database
  - **settings**: data.frame with settings used to build model
  - **attributes**: data.frame of variable/columns used to build model
  - **clusters**: contain general per-cluster information
  - **leaf.cluster.count**: data.frame of leaf clusters with support
  - **taxonomy**: parent-child cluster relationship
  - **centers**: per cluster-attribute center (centroid) information
  - **centers2**: simplified cluster centroids (means)
  - **histogram**: per cluster attribute histogram information
  - **rules**: rules defining clusters
  - **formula**: formula used to build the model
  - **call**: specific invocation of the function with arguments
ore.odmOC

O-Cluster Clustering

```r
x <- rbind(matrix(rnorm(100, sd = 0.3), ncol = 2),
            matrix(rnorm(100, mean = 2, sd = 0.3), ncol = 2))
colnames(x) <- c("x", "y")
X <- ore.push(data.frame(x))

oc.mod1 <- ore.odmOC(~., X, num.centers=2)
summary(oc.mod1)
rules(oc.mod1)
clusterhists(oc.mod1)
histogram(oc.mod1)
```

Cluster Histograms
ore.odmOC

**O-Cluster Clustering**

```r
oc.res1 <- predict(oc.mod1, X, type="class",
                   supplemental.cols=c("x","y"))
head(oc.res1,3)
oc.res1.local <- ore.pull(oc.res1)

plot(data.frame(x=oc.res1.local$x,
                 y=oc.res1.local$y),
     col=oc.res1.local$CLUSTER_ID)
points(oc.mod1$centers2,
       col = rownames(oc.mod1$centers2),
       pch = 8, cex=2)
```
Decision Tree

- Classification algorithm
  - Predicts a discrete value for each case: 0 or 1, Yes or No, Low Medium or High, with corresponding probability
  - ORE/ODM algorithms based on classification component of well-known C&RT algorithm
  - Enhancement of supplying Surrogate splitting attributes, if possible, at each node

- Uses include
  - Prediction
  - Segmentation
  - Understanding predictions
Decision Tree Example

Segment #1
IF CUST_MO > 14 AND INCOME < $90K, THEN
Prediction = Cell Phone Churner,
Confidence = 100%
Support = 8/39

Segment #3
IF CUST_MO > 7 AND INCOME < $175K, THEN
Prediction = Cell Phone Churner,
Confidence = 83%
Support = 6/39

Source: Inspired from Data Mining Techniques: For Marketing, Sales, and Customer Relationship Management by Michael J. A. Berry, Gordon S. Linoff

©2014 Oracle – All Rights Reserved
ore.odmDT

**Decision Tree**

ore.odmDT(
    formula,                   # formula specifying attributes for model build
    data,                      # ore.frame of the training dataset
    auto.data.prep = TRUE,     # Setting to perform automatic data preparation
    cost.matrix = NULL,        # numerical sq matrix for costs of incorrect prediction
    impurity.metric = "gini",  # gini or entropy
    max.depth = 7,             # maximum depth of tree from root to leaf inclusive [2..20]
    min.rec.split = 20,        # minimum number of cases required to split a node
    min.pct.split = 0.1,       # minimum percent of cases required to split a node
    min.rec.node = 10,         # minimum number of cases required in a child node
    min.pct.node = 0.05,       # minimum percent of cases required in child node
    na.action = na.pass)       # Allows missing values (na.pass), or removes rows with
                              #   missing values (na.omit)
Basic Argument Concepts

- **cost.matrix** – default NULL
- **impurity.metric**
  - options: gini or entropy, default "gini"
  - measure of node purity
  - tree algorithms seek the best test question for splitting data at each node. The best splitter and split value are those that result in the largest increase in target value homogeneity (purity) for the entities in the node
- **max.depth**
  - default 7
  - Criteria for splits: maximum tree depth (the maximum number of nodes between the root and any leaf node, including the leaf node)
- **min.rec.split** – default 20
- **min.pct.split** – default 0.1
- **min.rec.node** – default 10
- **min.pct.node** – default 0.05
```
m <- mtcars
m$gear <- as.factor(m$gear)
m$cyl <- as.factor(m$cyl)
m$vs <- as.factor(m$vs)
m$ID <- 1:nrow(m)
MTCARS <- ore.push(m)
row.names(MTCARS) <- MTCARS

dt.mod <- ore.odmDT(gear ~ ., MTCARS)
summary(dt.mod)

dt.res <- predict (dt.mod, MTCARS,"gear")
# confusion matrix
with(dt.res, table(gear,PREDICTION))
```
Decision Tree – model object

- ore.odmDT object
  - name …
  - settings …
  - attributes …
  - costs a data.frame containing the cost matrix supplied at model build
  - distributions target class distributions at each tree node
  - nodes a data.frame with tree node details, including: parent node id, node id, number of rows assigned to that node, predicted value, split predicate, surrogate variables (if applicable), and full split predicates from current node to root node
  - formula …
  - call …
Generalized Linear Models

Linear Models
- Assumes $Y$ is normally distributed with constant variance
- Linear models fit
  \[ \mu_Y = \beta_0 + \sum_{j=1}^{p} \beta_j X_j \]
- No assumptions about predictors $X_j$ distributions, e.g., need not be normally distributed
- Nonlinear functions on predictors allowed
- Advantages
  - Computational simplicity
  - Interpretable model form
  - Ability to compute certain diagnostic information about the quality of the fit

Generalized Linear Models
- Addresses target variables that are non-normal
  - Assume $Y$ follows distribution from exponential family
  - Specify link function and probability distribution, or variance function
- GLM fits models of the form
  \[ g(\mu_Y) = \beta_0 + \sum_{j=1}^{p} \beta_j X_j \]
- $g(\mu_Y)$ is a function of the conditional mean, a.k.a. link function

http://en.wikipedia.org/wiki/Exponential_family
ore.odmGLM

Generalized Linear Model

ore.odmGLM(
    formula,                   # formula specifying attributes for model build
    data,                      # ore.frame of the training dataset
    weights = NULL,
    type = c("normal", "logistic"),
    na.treatment = c("delete.row", "mean.or.mode"),
    reference = NULL,
    ridge = FALSE,
    ridge.value = NULL,
    ridge.vif = FALSE,
    auto.data.prep = TRUE,     # Setting to perform automatic data preparation
)
Basic Argument Concepts

• **weights**
  - An optional character string representing the column name in the data argument to use as analytical weights in the model fit, Default NULL

• **type**
  - the type of generalized linear model, default “normal”
    • "normal" (Gaussian) - identify link function and variance function = 1 (constant over range of response values)
    • "logistic" (binomial) - logit link function and binomial variance function

• **na.treatment**
  - The missing value treatment; either "delete.row" (delete entire row) or "mean.or.mode" (replace missing values with the mean in numeric predictors and the mode in categorical predictors), Default "delete.row"

• **reference**
  - An optional response variable category to use as the reference value (non-case/failure code) in a binary logistic regression model
  - By default, reference is taken to be the category with the highest prevalence, default NULL
Basic Argument Concepts

• **ridge**
  – Compensates for multicollinearity
  – TRUE to enable ridge estimation of the coefficients, FALSE otherwise, default FALSE
  – Applies both to regression and classification
  – When enabled, no prediction bounds can be produced

• **ridge.value**
  – The value for the ridge parameter used by the algorithm
  – Used when ridge regression explicitly enabled
  – If ridge regression is enabled internally by the algorithm, the ridge parameter is determined by the algorithm, default NULL

• **ridge.vif**
  – (Linear regression only) Optional logical indicator for whether to produce Variance Inflation Factor (VIF) statistics for the ridge estimates
  – VIFs can only be produced if enough Oracle database system resources are available
  – Default FALSE
# Linear regression using the longley data set
LONGLEY <- ore.push(longley)

longfit1 <- ore.odmGLM(Employed ~ ., data = LONGLEY)
summary(longfit1)
# Ridge regression using the longley data set

```r
longfit2 <- ore.odmGLM(Employed ~ ., data = LONGLEY,
                        ridge = TRUE,
                        ridge.vif = TRUE)

summary(longfit2)
```

> # Ridge regression using the longley data set
> longfit2 <- ore.odmGLM(Employed ~ ., data = LONGLEY,
>                        ridge = TRUE,
>                        ridge.vif = TRUE)
> summary(longfit2)

Call:
ore.odmGLM(formula = Employed ~ ., data = LONGLEY, ridge = TRUE,
            ridge.vif = TRUE)

Residuals:
  Min      1Q  Median      3Q     Max
-0.4100 -0.1579 -0.0271  0.1017  0.4575

Coefficients:
     Estimate  VIF
(Intercept) -3.466e+03 0.000
GNP.deflator  1.479e+02 0.077
GNP          -3.535e+02 0.012
Unemployed   -2.013e+02 0.000
Armed.Forces -1.031e+02 0.000
Population   -5.262e+02 0.548
Year         1.821e+00 2.212

Residual standard error: 0.3049 on 9 degrees of freedom
Multiple R-squared: 0.9955,    Adjusted R-squared: 0.9925
F-statistic: 330.2 on 6 and 9 DF,  p-value: 4.986e-10
# Logistic regression using the infert data set
INFERT <- ore.push(infert)
infit1 <- ore.odmGLM(
    case ~ age+parity+education+spontaneous+
    data = INFERT,
    type = "logistic")
infit1

R> # Logistic regression using the infert data set
R> INFERT <- ore.push(infert)
R> infit1 <- ore.odmGLM(case ~ age+parity+education+spontaneous+induced,
    +                      data = INFERT, type = "logistic")
R> infit1

Response:
  case == "1"

Call:  ore.odmGLM(formula = case ~ age + parity + education + spontaneous +
                   induced, data = INFERT, type = "logistic")

Coefficients:
     (Intercept)             age          parity education0-5yrs
          -2.19348         0.03958        -0.82828          1.04424
education12+ yrs spontaneous induced
          -0.35896         2.04590          1.28876

Degrees of Freedom: 247 Total (i.e. Null); 241 Residual
Null Deviance: 316.2
Residual Deviance: 257.8 AIC: 271.8
ore.odmGLM – other functions

**Generalized Linear Model**

```r
residuals(object,
    type = c("deviance", "pearson", "response"), ...)
```

```r
fitted(object, ...)
```

```r
predict(object, newdata, supplemental.cols = NULL,
    confint = FALSE, level = 0.95,
    na.action = na.pass,...)
```

```r
confint(object, parm, level = 0.95, ...)
```

```r
deviance(object, ...)
```

```r
extractAIC(fit, scale = 0, k = 2, ...)
```

```r
logLik(object, ...)
```

```r
nobs(object, ...)
```

**confint**: A logical indicator for whether to produce confidence intervals for the predicted values.

**level**: A numeric value within \([0, 1]\) to use for the confidence level.

**na.action**: Function to use for missing value handling; either 'na.pass' (allow missing values) or 'na.omit' (remove rows with missing values).

**parm**: An optional character vector that specifies which coefficients to include in the set of confidence intervals.

**scale**: An optional numeric scale parameter.

**k**: An optional numeric weight of the equivalent degrees of freedom.
```
res <- predict(infit1, newdata = INFERT, confint=TRUE, level = 0.97)
head(res)

head(residuals(infit1))
extractAIC(infit1)
logLik(infit1)
nobs(infit1)
```

```
R> res <- predict(infit1, newdata = INFERT, confint=TRUE, level = 0.97)
R> head(res)
PREDICTION LOWER.CONF UPPER.CONF
1  0.5721917  0.1767983  0.8928118
2  0.7258536  0.2887066  0.9452694
3  0.1194461  0.5546963  0.9775927
4  0.3684102  0.2546444  0.8958629
5  0.5104286  0.3632442  0.6558268
6  0.6322268  0.4007028  0.8154924
R>
R> head(residuals(infit1))
[1] 1.0566751 0.8005085 2.0614994 1.4131937 1.1597452 0.9576085
R> extractAIC(infit1)
[1]   7.0000 271.7977
R> logLik(infit1)
'log Lik.' -128.8988 (df=7)
R> nobs(infit1)
[1] 248
```
Generalized Linear Model – model object

- ore.odmGLM object
  - name ...
  - settings ...
  - attributes ...
  - coefficients a named vector of coefficients
  - residuals an ore.frame containing three types of residuals: "deviance", "pearson", and "response"
  - fitted.values an ore.vector containing the fitted values
  - rank numeric rank of the fitted model
  - type type of model fit
  - deviance minus twice the maximized log-likelihood, up to a constant
  - aic same version of Akaike's An Information Criterion as used by glm
  - null.deviance deviance for the null (intercept only) model
  - prior.weights weights initially supplied or 1 if none were
  - df.residual residual degrees of freedom
Generalized Linear Model – model object (2)

- **ore.odmGLM object**
  - `df.null` residual degrees of freedom for the null model
  - `y` ore.vector containing the response variable
  - `converged` indicator for whether the model converged
  - `model` ore.frame containing the model frame
  - `na.treatment` how missing values were treated
  - `na.action` number of rows with missing values that were removed
  - `terms` terms object used
  - `data` data argument
  - `nonreference` in logistic regression, the response values that represents success
  - `ridge` ridge argument
  - `auto.data.prep` whether or not auto data preparation should be used
  - `fit.name` internal name for the in-database model
  - `fit.details` model details
  - `formula` ...
  - `call` ...
Association Rules – Market Basket Analysis

- Apriori algorithm (Agrawal and Srikant 1994)
- Finds frequent itemsets and generates association models
  - Finds co-occurrence of items in large volumes of data: both transactional and relational
- Produces rules
  - Set of items in a transactional record implies the existence of another set of items
  - Groups of items form rules if they pass a minimum threshold
  - Thresholds include: how frequently they occur (support) and how often the consequent follows the antecedent (confidence)
- Apriori algorithm is efficient, and scales well with respect to the number of transactions, number of items, and number of itemsets and rules produced
Association (Market Basket Analysis)

Transactional Data and Rule Example

Input Data:

<table>
<thead>
<tr>
<th>User ID</th>
<th>Movies Viewed</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Movie1, Movie2, Movie3</td>
</tr>
<tr>
<td>2</td>
<td>Movie1, Movie4</td>
</tr>
<tr>
<td>3</td>
<td>Movie1, Movie3</td>
</tr>
<tr>
<td>4</td>
<td>Movie2, Movie5, Movie6</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>N</td>
<td>Movie3, Movie4, Movie6</td>
</tr>
</tbody>
</table>

Movie1 and Movie2 $\Rightarrow$ Movie3
with support of .12 and confidence .78
Association Rules

**Support and Confidence**

<table>
<thead>
<tr>
<th>User ID</th>
<th>Movies Viewed</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>{1, 2, 3}</td>
</tr>
<tr>
<td>2</td>
<td>{1, 4}</td>
</tr>
<tr>
<td>3</td>
<td>{1, 3}</td>
</tr>
<tr>
<td>4</td>
<td>{2, 5, 6}</td>
</tr>
</tbody>
</table>

Support (A → B) = \( P(AB) \) = \( \frac{\text{count } (A & B)}{\text{totalCount}} \)

Confidence (A → B) = \( P(AB)/P(A) \) = \( \frac{\text{count } (A & B)}{\text{count } (A)} \)

1 → 3 :
Support = \( \frac{2}{4} = 50\% \)
Confidence = \( \frac{2}{3} = 66\% \)

3 → 1 :
Support = \( \frac{2}{4} = 50\% \)
Confidence = \( \frac{2}{2} = 100\% \)
ore.odmAssocRules

Association Rules

ore.odmAssocRules(formula,  
    data,  
    case.id.column,  
    item.id.column = NULL,  
    item.value.column = NULL,  
    min.support = 0.05,  
    min.confidence = 0.05,  
    max.rule.length = 2,  
    na.action = na.pass)

## S3 method for class 'ore.odmAssocRules'
rules(object, ...)

## S3 method for class 'ore.odmAssocRules'
itemsets(object, ...)

http://docs.oracle.com/cd/E11882_01/datamine.112/e16808/market_basket.htm
Basic Argument Concepts

• **case.id.column**
  - Column name in 'data' that contains unique case identifiers

• **item.id.column**
  - Column in 'data' that contains item IDs. If NULL (default), 'data' treated as single-record case relational table, where each row is considered a transaction and column values of that row are converted to items for that transaction; if specified, treated as transactional or multi-record case table where each row corresponds to an item in the transaction, and the model ignores any columns in 'data' other than item ID and item value.

• **item.value.column**
  - Column name in 'data' that contains the value of the item. (default: NULL)

• **min.support**
  - Numeric value that specifies the minimum support for rules in the model

• **min.confidence**
  - Numeric value that specifies the minimum confidence for rules in the model

• **max.rule.length**
  - Numeric value that specifies the maximum number of items in rule
Association Rules – model object

- ore.odmAssocRules object
  - name – name of in-database model
  - settings - data.frame of settings used to build model
  - attributes - named 'vector' of the types of input item values
  - inputType: The type of input data table. It is "trans","tranWithValue", or "relational" for a multi-record case table, a multi-record case table with the values specified, or a single-record case table, respectively
  - formula: A formula specified by users.
Assocation Rules – model object

- **ore.itemsets object** - returned by itemsets() that describes the property of each itemset
  - ITEMSET_ID: numerical identifier associated with each itemset
  - NUMBER_OF_ITEMS: number of items in the itemset
  - ITEMS: names of items in the itemset
  - SUPPORT: number of transactions containing this itemset

- **ore.rules object** - returned by rules() that describes the property of each rule
  - RULE_ID:
  - NUMBER_OF_ITEMS:
  - LHS: left hand side of rule (antecedent)
  - RHS: right hand side of rule (consequent)
  - SUPPORT:
  - CONFIDENCE:
  - LIFT:
Association Rules

```r
id <- c(1, 1, 1, 2, 2, 2, 3, 3, 3, 3)
item <- c("b", "d", "e", "a", "b", "c", "e", "b", "c", "d", "e")
data.ore <- ore.push(data.frame(ID = id, ITEM = item))

ar.mod1 <- ore.odmAssocRules(~., data.ore, case.id.column = "ID", item.id.column = "ITEM", min.support = 0.6, min.confidence = 0.6, max.rule.length = 3)

# Generate itemsets and rules of the model
itemsets <- itemsets(ar.mod1)
rules <- rules(ar.mod1)

# subsetting
sub.itemsets <- subset(itemsets, min.support=0.7, items=list("b"))
sub.rules <- subset(rules, min.confidence=0.7, lhs=list("b", "c"))

# Convert the rules to the rules object in arules package
rules.arules <- ore.pull(rules)
inspect(rules.arules)
```

R> inspect(rules.arules)

<table>
<thead>
<tr>
<th>lhs</th>
<th>rhs</th>
<th>support</th>
<th>confidence</th>
<th>lift</th>
</tr>
</thead>
<tbody>
<tr>
<td>{b} =&gt; {e}</td>
<td>1.0000000</td>
<td>1.0000000</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>{e} =&gt; {b}</td>
<td>1.0000000</td>
<td>1.0000000</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>{c} =&gt; {e}</td>
<td>0.6666667</td>
<td>1.0000000</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>{d, e} =&gt; {b}</td>
<td>0.6666667</td>
<td>1.0000000</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>{c, e} =&gt; {b}</td>
<td>0.6666667</td>
<td>1.0000000</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>{b, d} =&gt; {e}</td>
<td>0.6666667</td>
<td>1.0000000</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>{b, c} =&gt; {e}</td>
<td>0.6666667</td>
<td>1.0000000</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>{e} =&gt; {d}</td>
<td>0.6666667</td>
<td>1.0000000</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>{e} =&gt; {c}</td>
<td>0.6666667</td>
<td>1.0000000</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>{b, e} =&gt; {d}</td>
<td>0.6666667</td>
<td>1.0000000</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>{b, e} =&gt; {c}</td>
<td>0.6666667</td>
<td>1.0000000</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
```
# Convert itemsets to the itemsets object in arules package

```r
itemsets.arules <- ore.pull(itemsets)
inspect(itemsets.arules)
```

```r
library(arulesViz)
plot(rules.arules, method = "graph", interactive = TRUE)
```

---

```r
R> inspect(itemsets.arules)
```

<table>
<thead>
<tr>
<th>items</th>
<th>support</th>
</tr>
</thead>
<tbody>
<tr>
<td>{b}</td>
<td>1.0000000</td>
</tr>
<tr>
<td>{e}</td>
<td>1.0000000</td>
</tr>
<tr>
<td>{b, e}</td>
<td>1.0000000</td>
</tr>
<tr>
<td>{c}</td>
<td>0.6666667</td>
</tr>
<tr>
<td>{d}</td>
<td>0.6666667</td>
</tr>
<tr>
<td>{b, c}</td>
<td>0.6666667</td>
</tr>
<tr>
<td>{b, d}</td>
<td>0.6666667</td>
</tr>
<tr>
<td>{c, e}</td>
<td>0.6666667</td>
</tr>
<tr>
<td>{d, e}</td>
<td>0.6666667</td>
</tr>
<tr>
<td>{b, c, e}</td>
<td>0.6666667</td>
</tr>
<tr>
<td>{b, d, e}</td>
<td>0.6666667</td>
</tr>
</tbody>
</table>
ore.odmAssocRules – multi-record case with value

**Association Rules**

```r
id <- c(1, 1, 2, 2, 3, 3, 3, 4, 4, 5, 5, 6, 6, 7, 8, 8, 9, 9, 10, 11, 12, 12, 13, 14, 15, 16, 17)
item <- c("a","b","a","b","a","c","d","c","d","c","d","c","d","a","d","e","a",
          "a","d","e","d","e","d","d")
value <- c(1, 1, 1, 1, 1, 1, 2, 1, 2, 1, 2, 1, 2, 3, 1, 2, 3, 2, 2, 3, 2, 2, 3, 3, 3)
data2.ore <- ore.push(data.frame("ID" = id, "ITEM" = item, "VALUE" = value))

ar.mod2 <- ore.odmAssocRules(~., data2.ore, case.id.column = "ID",
                         item.id.column = "ITEM", item.value.column = "VALUE", max.rule.length = 3)

rules <- rules(ar.mod2)
itemsets <- itemsets(ar.mod2)

itemsets.arules <- ore.pull(itemsets)
inspect(itemsets.arules)

rules.arules <- ore.pull(rules)

plot(rules.arules, method = "graph", interactive = TRUE)
```
ore.odmAssocRules

Association Rules

# Relational data in a single-record case table.
ar.mod3 <- ore.odmAssocRules(~., NARROW,
    case.id.column = "ID",
    min.support=0.25, min.confidence=0.15,
    max.rule.length = 2)

rules = rules(ar.mod3)
itemsets = itemsets(ar.mod3)

itemsets.arules <- ore.pull(itemsets)
inspect(itemsets.arules)

rules.arules <- ore.pull(rules)

plot(rules.arules, method = "graph",
     interactive = TRUE)
plot(rules.arules, method = "graph", interactive = TRUE)
Non-negative Matrix Factorization

• State-of-the-art algorithm for Feature Extraction
• Dimensionality reduction technique
  – Creates new features of existing attributes
  – Compare to AI which reduces attributes by taking a subset
  – NMF derives fewer new “features” taking into account interactions among original attributes
• Supports text mining, life sciences, marketing applications
NMF, intuitively…

- Useful where there are many attributes
  - Each has weak predictability, even ambiguous
  - But when taken in combination, produce meaningful patterns, topics, or themes

- Example: Text
  - Same word can predict different documents
    e.g., “hike” can be applied to the outdoors or interest rates
  - NMF introduces context which is essential for predictive power
    e.g., “hike” + “mountain” -> “outdoors sports”
    “hike” + “interest” -> “interest rates”
Conceptual view...

Attributes values

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
<th>g</th>
<th>h</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Target values

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>z</th>
</tr>
</thead>
</table>

Attributes values

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
<th>g</th>
<th>h</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Extracted features

<table>
<thead>
<tr>
<th>f1</th>
<th>f2</th>
<th>f3</th>
<th>f4</th>
</tr>
</thead>
</table>

Target values

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>z</th>
</tr>
</thead>
</table>
Feature Extraction
Face representation with Vector Quantization

VQ

(0,0,0,...,1,...,0,0) × encoding = reconstruction
Feature Extraction
Face representation with Principal Component Analysis

PCA

original

reconstruction

\[
\begin{pmatrix}
(.9,.6, -.5, \ldots, .9, -.3)
\end{pmatrix}
\times
\begin{pmatrix}
\text{encoding}
\end{pmatrix}
= \begin{pmatrix}
\text{reconstruction}
\end{pmatrix}
\]
Feature Extraction
Face representation with NMF

NMF

(0, 5, 3, 0, 1, ..., 3, 0)

encoding

= original

reconstruction
ore.odmNMF

Non-negative Matrix Factorization

ore.odmNMF(formula,
  data,
  auto.data.prep = TRUE,
  num.features = NULL,
  conv.tolerance = NULL,
  num.iter = NULL,
  rand.seed = NULL,
  nonnegative.scoring = TRUE,
  na.action = na.pass)

## S3 method for class 'ore.odmNMF'
predict(object,
  newdata,
  supplemental.cols = NULL,
  type = c("class","raw"),
  na.action = na.pass,...)

http://docs.oracle.com/cd/E11882_01/datamine.112/e16808/algo_nmf.htm
Basic Argument Concepts

- `num.features` – number of features to be extracted
- `conv.tolerance` – convergence tolerance
- `num.iter` – maximum number of iterations
- `rand.seed` – random seed
- `nonnegative.scoring` – non-negative values allowed in scoring
ore.odmNMF
Non-negative Matrix Factorization

```r
training.set <- ore.push(npk[1:18, c("N","P","K")])
scoreing.set <- ore.push(npk[19:24, c("N","P","K")])
nmf.mod <- ore.odmNMF(~., training.set, num.features = 3)
features(nmf.mod)
summary(nmf.mod)
predict(nmf.mod, scoring.set)
```

---

<table>
<thead>
<tr>
<th>FEATURE_ID</th>
<th>ATTRIBUTE_NAME</th>
<th>ATTRIBUTE_VALUE</th>
<th>COEFFICIENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>K</td>
<td>0 3.723468e-01</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>K</td>
<td>1 1.761670e-01</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>N</td>
<td>0 7.469067e-01</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>N</td>
<td>1 1.085058e-02</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>P</td>
<td>0 5.730082e-01</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>P</td>
<td>1 2.797865e-02</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>K</td>
<td>0 4.107375e-01</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>K</td>
<td>1 2.193757e-01</td>
</tr>
<tr>
<td>9</td>
<td>2</td>
<td>N</td>
<td>0 8.065393e-03</td>
</tr>
<tr>
<td>10</td>
<td>2</td>
<td>N</td>
<td>1 8.569538e-01</td>
</tr>
<tr>
<td>11</td>
<td>2</td>
<td>P</td>
<td>0 4.005661e-01</td>
</tr>
<tr>
<td>12</td>
<td>2</td>
<td>P</td>
<td>1 4.124996e-02</td>
</tr>
<tr>
<td>13</td>
<td>3</td>
<td>K</td>
<td>0 1.918852e-01</td>
</tr>
<tr>
<td>14</td>
<td>3</td>
<td>K</td>
<td>1 3.311137e-01</td>
</tr>
<tr>
<td>15</td>
<td>3</td>
<td>N</td>
<td>0 1.547561e-01</td>
</tr>
<tr>
<td>16</td>
<td>3</td>
<td>N</td>
<td>1 1.283887e-01</td>
</tr>
<tr>
<td>17</td>
<td>3</td>
<td>P</td>
<td>0 9.791965e-06</td>
</tr>
<tr>
<td>18</td>
<td>3</td>
<td>P</td>
<td>1 9.113922e-01</td>
</tr>
</tbody>
</table>
```
**ore.odmNMF**

*Non-negative Matrix Factorization*

```r
R> predict(nmf.mod, scoring.set)
  '1'       '2'       '3' FEATURE_ID
19 0.1972489 1.2400782 0.03280919          2
20 0.7298919 0.0000000 1.29438165          3
21 0.1972489 1.2400782 0.03280919          2
22 0.0000000 1.0231268 0.98567623          2
23 0.7298919 0.0000000 1.29438165          3
24 1.5703239 0.1523159 0.00000000          1
```

```r
R> summary(nmf.mod)
Call: 
ore.odmNMF(formula = ~., data = training.set, num.features = 3)

Settings:

<table>
<thead>
<tr>
<th>Setting</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>feat.num.features</td>
<td>3</td>
</tr>
<tr>
<td>nmfs.conv.tolerance</td>
<td>.05</td>
</tr>
<tr>
<td>nmfs.nonnegative.scoring</td>
<td></td>
</tr>
<tr>
<td>nmfs.nonneg.scoring.enable</td>
<td></td>
</tr>
<tr>
<td>nmfs.num.iterations</td>
<td>50</td>
</tr>
<tr>
<td>nmfs.random.seed</td>
<td>-1</td>
</tr>
<tr>
<td>prep.auto</td>
<td>on</td>
</tr>
</tbody>
</table>

Features:

<table>
<thead>
<tr>
<th>FEATURE_ID</th>
<th>ATTRIBUTE_NAME</th>
<th>ATTRIBUTE_VALUE</th>
<th>COEFFICIENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>K</td>
<td>0 3.723468e-01</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>K</td>
<td>1 1.761670e-01</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>N</td>
<td>0 7.469067e-01</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>N</td>
<td>1 1.085058e-02</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>P</td>
<td>0 5.730082e-01</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>P</td>
<td>1 2.797865e-02</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>K</td>
<td>0 4.107375e-01</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>K</td>
<td>1 2.193757e-01</td>
</tr>
<tr>
<td>9</td>
<td>2</td>
<td>N</td>
<td>0 8.065933e-03</td>
</tr>
<tr>
<td>10</td>
<td>2</td>
<td>N</td>
<td>1 8.569538e-01</td>
</tr>
<tr>
<td>11</td>
<td>2</td>
<td>P</td>
<td>0 4.005661e-01</td>
</tr>
<tr>
<td>12</td>
<td>2</td>
<td>P</td>
<td>1 4.124996e-02</td>
</tr>
<tr>
<td>13</td>
<td>3</td>
<td>K</td>
<td>0 1.918852e-01</td>
</tr>
<tr>
<td>14</td>
<td>3</td>
<td>K</td>
<td>1 3.311137e-01</td>
</tr>
<tr>
<td>15</td>
<td>3</td>
<td>N</td>
<td>0 1.547561e-01</td>
</tr>
<tr>
<td>16</td>
<td>3</td>
<td>N</td>
<td>1 1.283887e-01</td>
</tr>
<tr>
<td>17</td>
<td>3</td>
<td>P</td>
<td>0 9.791965e-06</td>
</tr>
<tr>
<td>18</td>
<td>3</td>
<td>P</td>
<td>1 9.113922e-01</td>
</tr>
</tbody>
</table>
```
## OREmodels Algorithms

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Main R Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear Regression</td>
<td>ore.lm</td>
</tr>
<tr>
<td>Stepwise Linear Regression</td>
<td>ore.stepwise</td>
</tr>
<tr>
<td>Generalized Linear Models</td>
<td>ore.glm</td>
</tr>
<tr>
<td>Feedforward Neural Networks</td>
<td>ore.neural</td>
</tr>
</tbody>
</table>
**Regression** – e.g. using lm or ore.lm

*Predict a continuous numerical value*

For a simple dataset with two variables, a line can be used to approximate the values

\[ y = mx + b \]

Build a *model*, i.e., compute coefficients, that can be expressed in terms of values \((m, b)\)

Models aren’t perfect…when used for scoring, or making predictions, they may have an error component

Metrics like Root Mean Square Error (RMSE) are useful for assessing and comparing models

Scoring can be *batch* or *real-time*
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 data, 'ore.lm' and 'ore.stepwise' will not estimate coefficient values for the collinear set of terms
  – For 'ore.stepwise', this collinear set of terms will be excluded throughout the procedure

©2014 Oracle – All Rights Reserved
# Fit full model

```r
?longley
fit1 <- lm(Employed ~ ., data = longley)
summary(fit1)
```

Coefficient “Armed.Forces” significant at \( p < .001 \) indicates for a 1 unit increase in Armed.Forces, Employed decreases by 0.01 units when all other predictors held constant.

Multiple R-squared of 0.9955 indicates the model accounts for 99.55% of the variance in the target.

Adjusted R-squared takes into account the number of predictors to account for chance improvement of R-squared simply be increasing number of predictors.

Residual standard error is the average error in predicting the target.

F-statistic indicates if predictors predict target beyond chance.
ORE.LM

LONGLEY <- ore.push(longley)

# Fit full model
oreFit1 <- ore.lm(Employed ~ ., data = LONGLEY)
summary(oreFit1)
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  0.292681
Unemployed              -2.020e-02  4.884e-03   -4.136  0.0002535 **
Armed.Forces            -1.033e-02  2.143e-03   -4.822  0.0009444 ***
Population              -5.110e-02  2.261e-01   -0.226  0.826212
Year                    -1.823e+00  4.595e-01   -3.995  0.0005227 **
                           ---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 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 8 and 9 DF,  p-value: 4.984e-10
```

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

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

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  0.292681
Unemployed              -2.020e-02  4.884e-03   -4.136  0.0002535 **
Armed.Forces            -1.033e-02  2.143e-03   -4.822  0.0009444 ***
Population              -5.110e-02  2.261e-01   -0.226  0.826212
Year                    -1.823e+00  4.595e-01   -3.995  0.0005227 **
                           ---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 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 8 and 9 DF,  p-value: 4.984e-10
```
Stepwise Regression: `ore.stepwise`

**Motivation**

- Automatically selects predictive variables
- Produces models with fewer terms
- Enable handling data with complex patterns
  - Even for relatively small data sets (e.g., < 1M rows) R may not yield satisfactory results
- Increases 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
ore.stepwise - parameters

ore.stepwise(formula, data, scope,
  direction = c("both", "backward", "forward", "alternate", "none"),
  add.p = 0.50, drop.p = 0.10, nbest = 1, steps = 1000,
  contrasts = NULL, xlev = NULL, ...)

• **scope** – range of models to examine, either single formula object, or list containing lower and upper formula object elements

• **direction** – The stepwise search mode; one of "both" (first try to add a term using the 'add.p' argument value and then try repeatedly to drop terms using the 'drop.p' argument value), "backward", "forward", "alternate" (similar to "both" but only one drop is attempted per add attempt) or "none" with a default of "both"

• **add.p** – F-test p-value threshold for adding term to model

• **drop.p** – F-test p-value threshold for dropping term from model

• **nbest** – number of best models to report at each step

• **steps** – maximum number of steps

• **contrasts** – named list to be supplied to the contrasts.arg argument of model.matrix

• **xlev** – a named list of character vectors specifying the levels for each ore.factor variable

©2014 Oracle – All Rights Reserved
ore.stepwise – example

LONGLEY <- ore.push(longley)

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

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

Build model with interaction terms
ore.stepwise – results

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

Aliased:

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 = LONLEY, add.p = 0.1,
  drop.p = 0.1)

Coefficients:

                    (Intercept)        Year       GNP.deflator:GNP      GNP.deflator:Unemployed      GNP:Armed,forces GNP:Population
GNP:Year             6.875e-06  2.007e-04
GNP:Population       2.303e-05  2.875e-06
GNP:Year             2.303e-05  2.875e-06
Akaike information criterion (AIC)

- Measure of quality of a model
- Used for model selection

Step with `ore.lm` – results

```
RV oneStep2 <-
  + step(ore.lm(Employed ~ 1, data = LONLEY),
  + scope = terms(Employed ~ .^2, data = LONLEY))
Start: AIC=41.17
Employed ~ 1

  DF Sum of Sq  RSS  AIC
  + GNP        1 174.552 10.457 -2.806
  + Year       1 164.355 10.611 -2.671
  + GNP,deflator 1 164.355 10.611 -2.671
  + Population  1 164.355 10.611 -2.671
  + Unemployed  1 164.355 10.611 -2.671
  + Armed.Forces 1 164.355 10.611 -2.671
  <none>       1 164.355 10.611 -2.671

Step: AIC=11.6
Employed ~ GNP

  DF Sum of Sq  RSS  AIC
  + Unemployed 1 2.457 3.579 -17.960
  + Population 1 2.162 3.879 -16.831
  + Year       1 1.125 4.911 -12.998
  <none>       1 6.036 -11.597

+ GNP,deflator 1 2.121 3.582 -10.163
+ Armed.Forces 1 0.077 5.959 -3.802
- GNP        1 178.973 185.008 41.185

Step: AIC=17.96
Employed ~ GNP + Unemployed

  DF Sum of Sq  RSS  AIC
  + Armed.Forces 1 0.022 2.767 -20.137
  <none>        1 0.340 3.239 -17.586
  + Year       1 0.192 3.397 -16.796
  + GNP,unemployed 1 0.097 3.409 -16.339
  + GNP,deflator 1 0.013 3.560 -15.044
  - Unemployed 1 2.457 6.036 -11.597
  - GNP        1 134.714 130.293 39.509
```

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

  DF Sum of Sq  RSS  AIC
  + Year       1 1.898 0.859 -36.759
  + GNP,unemployed 1 0.611 2.148 -22.168
  + Population  1 0.539 2.367 -20.578
  <none>       1 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 2.693 -18.489
  - Armed.Forces 1 0.022 3.573 -17.960
  + Unemployed 1 3.203 5.939 -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
  + Year       1 0.074 0.703 -36.259
  + GNP,unemployed 1 0.067 0.790 -56.115
  + Unemployed:Armed.Forces 1 0.051 0.842 -35.798
  + GNP,Armed.Forces 1 0.037 0.820 -35.498
  + Population  1 0.039 0.835 -59.165
  + GNP,deflator 1 0.017 0.842 -35.129
  + GNP:Armed.Forces:Year 1 0.013 0.845 -35.054
  + GNP,Year 1 0.008 0.852 -54.957
  - GNP        1 0.464 1.322 -31.679
  - Year      1 1.050 2.757 -20.137
  - Armed.Forces 1 2.306 3.238 -17.556
  - Unemployed 1 4.049 4.917 -10.908
Rv oneStep2

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

Coefficients:
(`(Intercept`)  GNP Unemployed Armed.Forces Year
-5.699e+03 -4.019e-02 -2.086e-02 -1.915e-02 1.897e+00
```
Formula specification option

*Class formula accepts the following options*

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>~</td>
<td>separates response/target variables from explanatory/predictor variables</td>
<td>y ~ x</td>
</tr>
<tr>
<td>+</td>
<td>separates predictors</td>
<td>y ~ a + b + c</td>
</tr>
<tr>
<td>:</td>
<td>specify interaction terms between predictors</td>
<td>y ~ a + c + a:c</td>
</tr>
<tr>
<td>*</td>
<td>Specify all possible interactions between specific predictors</td>
<td>y ~ a * b * c</td>
</tr>
<tr>
<td>^</td>
<td>Specify interactions up to a specific degree</td>
<td>y ~ (a + b + c)^2 ➔ y ~ a + b + c + a:b + a:c + b:c</td>
</tr>
<tr>
<td>.</td>
<td>Represents all other variables beside target variable</td>
<td>y ~ .</td>
</tr>
<tr>
<td>-</td>
<td>Remove the specified predictor(s)</td>
<td>y ~ (a + b + c)^3 ➔ y ~ a + b + c + a:c + a:b:c</td>
</tr>
<tr>
<td>-1</td>
<td>Suppresses the intercept from the model, forcing the regression line through the origin at a = 0</td>
<td>y ~ a + b - 1</td>
</tr>
<tr>
<td>I()</td>
<td>Interpret contents arithmetically</td>
<td>y ~ a + I(b-c)^3 ➔ y ~ a + v, where v = (b-c)^3</td>
</tr>
<tr>
<td>function</td>
<td>Mathematical function</td>
<td>sqrt(y) ~ a + log(b)</td>
</tr>
</tbody>
</table>
Artificial Neural Networks

• Neural network (NN) is a mathematical model inspired by biological neural networks and in some sense mimics the functioning of a brain
  – Consists of an interconnected group of artificial neurons (nodes)
  – 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
Artificial Neural Networks

• Well-suited to data with noisy and complex sensor data
• Problem characteristics
  – Potentially many (numeric) predictors, e.g., pixel values
  – Target may be discrete-valued, real-valued, or a vector of such
  – Training data may contain errors – robust to noise
  – Fast scoring
  – Model transparency not required – models difficult to interpret
• Universal approximator
  – Adding more neurons can lower error to be as small as desired
  – Not always the desired behavior
Steps to Neural Network modeling

- Architecture specification
- Data preparation
- Building the model
  - Stopping criteria: iterations, error on validation set within tolerance
- Viewing statistical results from model
- Improving the model
Architecture Specification

• Input Layer
  – Numerical or categorical
  – No automatic normalization of data
  – Supports up to 1000 actual columns (due to database table limit)
  – No fixed limit on interactions
  – No fixed limit on cardinality of categorical variables

• Hidden Layers
  – Any number of hidden layers - \( k \)
  – All nodes from previous layer are connected to nodes of next
  – Activation function applies to one layer
    • Bipolar Sigmoid default for hidden layers

• Output Layer
  – Currently single numeric target or binary categorical
  – Linear activation function default, all others also supported

• Calculate number of weights
  – \((# \text{ input units}) \times (# L1 \text{ nodes}) + (# L1 \text{ nodes bias}) + (# L1 \text{ nodes}) \times (# L2 \text{ nodes}) + (# L2 \text{ nodes bias}) + \ldots (\# Lk \text{ nodes}) \times (# output nodes)\)

• Initialize weights
  – Change initialization with random seed
  – Set lower and upper bound, typically -0.25, 0.25
Unique aspects of ore.neural

• Hidden layer structure complexity
• #Activation functions - 15
• Support for categorical variables and transformations of all variables – predictors and targets
• Support for logistic regression through entropy activation function
• No competitive CRAN package available for neural networks
• Extraordinary scalability on several dimensions including HYPER SPARSE data sets
  – Scale-up and Scale-out
• Compared to SAS’s HPNeural, ore.neural can work with data sets that do not fit in memory
  – SAS requires complete data set to fit into distributed memory before it can solve any HP* models
Architecture Guidelines

• Start
  – one hidden layer with one neuron/node
  – number of nodes less than $\sqrt{\text{#observations} \times \text{#variables}}$

• Test different number of hidden nodes and number of layers

• Test different activation functions

• Restart (rebuild) model multiple times with different weight initializations to escape local minima, keep model with lowest objective function value, e.g., fit$objValue$

• Perfecting neural networks is an *art*
Data Preparation

- Data preparation may be unnecessary if appropriate activation functions are used - especially for targets (outputs)
  - Bipolar sigmoid can model values from -1 ... 1 range
  - Hyperbolic tangent can model values from -1 ... 1 range
  - Logistic sigmoid can model values from 0 ... 1 range

- Output preparation
  - If target (output) is not scaled (normalized) into ranges above, then linear activation function is appropriate
  - If output activation function is non-linear, targets must be scaled

- Scaling is recommended for faster convergence, however experimentation is key

- For predictors (input data), choose standard R facilities, for instance
  - `data <- iris`
  - `data$Petal.Length <- scale(data$Petal.Length)`
  - To normalize `Petal.Length` around 0
    - `sd(data$Petal.Length)`
      - `[1] 1`
    - `mean(data$Petal.Length)`
      - `[1] -2.895326e-17`
## Activation Functions

- If specified, must include one for each hidden layer and output layer

<table>
<thead>
<tr>
<th>Activation Function</th>
<th>Activation Setting</th>
<th>Definition</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arctangent</td>
<td>atan</td>
<td>$f(x) = \arctan x$</td>
<td></td>
</tr>
<tr>
<td>Bipolar Sigmoid</td>
<td>bSigmoid</td>
<td>$f(x) = (1 - e^{-x})/(1 + e^{-x})$</td>
<td>Use in input data with different signs or unscaled Use on output layer when values [-1, 1]</td>
</tr>
<tr>
<td>Cosine</td>
<td>cos</td>
<td>$f(x) = \cos x$</td>
<td></td>
</tr>
<tr>
<td>Gaussian</td>
<td>gaussian</td>
<td>$f(x) = e^{-x^2}$</td>
<td></td>
</tr>
<tr>
<td>Gauss error</td>
<td>gaussError</td>
<td>$f(x) = 2/\sqrt{\pi} \int e^{-t^2} dt$</td>
<td></td>
</tr>
<tr>
<td>Gompertz</td>
<td>gompertz</td>
<td>$f(x) = e^{-e^{-x}}$</td>
<td></td>
</tr>
<tr>
<td>Linear</td>
<td>linear</td>
<td>$f(x) = x$</td>
<td>Applicable across all data ranges</td>
</tr>
<tr>
<td>Logistic Sigmoid</td>
<td>sigmoid</td>
<td>$f(x) = 1 / (1 + e^{-x})$</td>
<td>Use on output layer when values [0..1]</td>
</tr>
<tr>
<td>Reciprocal</td>
<td>reciprocal</td>
<td>$f(x) = 1 / x$</td>
<td>Value should not include 0 value</td>
</tr>
<tr>
<td>Sigmoid Modulus</td>
<td>sigmoidModulus</td>
<td>$f(x) = x / (1 +</td>
<td>x</td>
</tr>
<tr>
<td>Sigmoid Square Root</td>
<td>sigmoidSqrt</td>
<td>$f(x) = x / (1 + \sqrt{1+x^2})$</td>
<td></td>
</tr>
<tr>
<td>Sine</td>
<td>sin</td>
<td>$f(x) = \sin x$</td>
<td></td>
</tr>
<tr>
<td>Square</td>
<td>Square</td>
<td>$f(x) = x^2$</td>
<td></td>
</tr>
<tr>
<td>Hyperbolic Tangent</td>
<td>tanh</td>
<td>$f(x) = \tanh x$</td>
<td></td>
</tr>
<tr>
<td>Wave</td>
<td>wave</td>
<td>$f(x) = x / (1 + x^2)$</td>
<td></td>
</tr>
</tbody>
</table>
bipolar sigmoid

hyperbolic tangent

arctangent

logistic sigmoid

reciprocal

gaussian
ore.neural

Artificial Neural Network

ore.neural(
  formula,
  data,
  weight = NULL,  # initial vector of weights
  xlev = NULL,    # named list of character vectors specifying levels for each ore.factor var
  sparse = TRUE,  # indicates if sparse matrix solver should be used, TRUE for data sets
                  # with categorical variables
  hiddenSizes = NULL,  # vector of nodes per layer, or none, e.g., 2 layers c(20,5)
  activations = NULL,  # vector activation functions, including one for output
  gradTolerance = 1E-1,  # numerical optimization stopping crit.
  maxIterations = 200L,  # select value >= 5
  objMinProgress = 1E-6,  # Stopping criterion: | f_current - f_previous | / ( 1 + |f| )
  lowerBound = -0.7,  # weight initialization range
  upperBound = 0.7,   # weight initialization range
  nUpdates = 20L,     # number of L-BFGS update pairs
  scaleHessian = TRUE,  # logical whether to scale inverse of Hessian matrix in L-BFGS updates
  trace = FALSE)      # report iteration log for big data solver
Stopping Criteria

• gradTolerance
  – Affects how quickly model can converge
  – Valid values: $> 10^{-9}$
  – If > 1M observations, set to 1
  – If # observations < 1000, set to between .01 and .001

• objMinProgress
  – Valid values $[10^{-1}, 10^{-6}]$
  – Indicates required change from one iteration to next
  – Computed as
    \[ \frac{|f_{current} - f_{previous}|}{1 + |f|} \]

• maxIterations
  – Valid values $\geq 5$
  – Upper limit on the number of iterations
Local Minima

- Local Minima are non-optimal states that can improve no further with current settings and weights
- To determine if a neural network is possibly in a local minima, rebuild model with different weights
  - Change random seen to different value
  - Change upper/lower bound of weight initialization values
  - Select model with best objective function value, e.g., fit$objValue
Local Minima

d <- data.frame(A=c(0,1,0,1),
                B=c(1,0,0,1),
                T=c(1,1,0,0))

# Run the model below 5 ~ 10 times and observe the resulting objective
# function value - the smaller, the better
library(nnet)
fit.nn <- nnet(formula = T ~ A + B, data = d, size=2)
predict(fit.nn,d)

fit.ore <- ore.neural(formula = T ~ A + B, data = ore.push(d),
                       hiddenSizes = c(5000, 10, 10),
                       lowerBound=-1, upperBound=1)
predict(fit.ore,ore.push(d))
Local Minima - results

R> fit.nn <- nnet(formula = T ~ A + B, 
                   data = d, size=2)
# weights:  9
initial value 1.046487
iter 10 value 0.997966
iter 20 value 0.569304
iter 30 value 0.502784
iter 40 value 0.500426
iter 50 value 0.500050
final value 0.500041
converged
R>
R> predict(fit.nn,d)
   [,1]
1   0.499970251
2   0.999986345
3   0.002586049
4   0.500004855

R> fit.ore <- ore.neural(formula = T ~ A + B, data = ore.push(d), 
                          +   hiddenSizes = c(5000, 10, 10), 
                          +   lowerBound=-1, upperBound=1)
R> predict(fit.ore,ore.push(d))
pred_T
   1   0.913355525
   2   1.035549253
   3  -0.020444140
   4  -0.001179834
Optimization argument: nUpdates

- Optimization parameter for *lbfgs* solver
- Indicates number of matrix adjustments to occur before updating Hessian matrix
- Recommended ranges
  - Usual models: 7..25
  - If # weights > 1M: 3..25
  - If highly non-linear behavior, use > 10
- If you’re unfamiliar with underlying techniques, don’t touch
Optimization argument: scaleHessian

- Optimization parameter
- If you’re unfamiliar with underlying techniques, don’t touch
- Default $TRUE$
Example

IRIS <- ore.push(iris)

fit <- ore.neural(Petal.Length ~ Petal.Width + Sepal.Length,
                   data = IRIS, hiddenSizes = c(20, 5),
                   activations = c('bSigmoid', 'tanh', 'linear'))

print(fit)

R> print(fit)
Number of input units 2
Number of output units 1
Number of hidden layers 2
Objective value 6.431877E+00
Solution status Optimal
Hidden layer [1] number of neurons 20, activation 'bSigmoid'
Hidden layer [2] number of neurons 5, activation 'tanh'
Output layer number of neurons 1, activation 'linear'
Optimization solver L-BFGS
Scale Hessian inverse 1
Number of L-BFGS updates 20

ans <- predict(fit, newdata = IRIS,
               supplemental.cols = 'Petal.Length')
localPredictions <- ore.pull(ans)

# Inspect some predictions
head(localPredictions)

# Compute RMSE
ore.rmse <- function (pred, obs) {
    sqrt(mean(pred-obs)^2)
}

R> ore.rmse(localPredictions$pred_Petal.Length,
           localPredictions$Petal.Length)
[1] 0.00148768

©2014 Oracle – All Rights Reserved
Example – linear regression

No hidden structure in network

```r
fit <- ore.neural(Petal.Length ~ Petal.Width + Sepal.Length, data = IRIS)

print(fit)
```

# Print fit object
R> print(fit)

Number of input units 2
Number of output units 1
Number of hidden layers 0
Objective value 1.311757E+01
Solution status Optimal
Output layer number of neurons 1, activation 'linear'
Optimization solver L-BFGS
Scale Hessian inverse 1
Number of L-BFGS updates 20
Model Details: Solution Status

- optimal
  - meets all stopping criteria
- numerical difficulties encountered
  - TBD
- maximum iterations reached
  - more iterations may be needed to improve model
- insufficient memory
  - could not build model with current settings due to memory
- no progress
  - change in objective function insufficient to make process
- unbounded
  - one of model parameters (weights) is greater than 1E+24 (check input data, unlikely to happen)
Model Details: Objective Value

• Error statistic on the model
• ore.neural tries to minimize this value
• Calculated as \( \text{sum}((\text{predicted} - \text{actual})^2) \)
ore.neural vs. nnet

• ORE is
  – Scalable
  – Choose activation functions
  – Generic topology
    • unrestricted number of hidden layers, including none
  – Parallel implementation
ore.glm

Generalized Linear Model

ore.glm(formula,                 # 'formula' object representing the model to be fit
data,                    # 'ore.frame' object specifying the data for the model
weights,                 # optional 'ore.number' object specifying the model’s analytic weights
family = gaussian(),      # 'family' object specifying the generalized linear model family details.
                         # Same type of object used for 'glm' function in the 'stats' package
start = NULL,            # optional 'numeric' vector specifying initial coefficient estimates in
                         # the linear predictor
control = list(...),     # optional 'list' object containing a list of fit control parameters to
                         # be interpreted by the 'ore.glm.control' function
contrasts = NULL,         # optional named 'list' to be supplied to 'contrasts.arg' argument
                         # of 'model.matrix'
xlev = NULL,             # optional named 'list' of 'character' vectors specifying the 'levels'
                         # for each 'ore.factor' variable
ylev = NULL,             # optional 'character' vector to specify the response variable levels
                         # in 'binomial' generalized linear models
yprob = NULL,            # optional numeric value between 0 and 1 specifying overall probability
                         # of 'y != ylev[1]' in 'binomial' linear models
...)

©2014 Oracle – All Rights Reserved
ore.glm examples

Generalized Linear Model

library(rpart)

# Logistic regression
KYPHOSIS <- ore.push(kyphosis)
kyphFit1 <- ore.glm(Kyphosis ~ ., data = KYPHOSIS, family = binomial())
kyphFit2 <- glm(Kyphosis ~ ., data = kyphosis, family = binomial())
summary(kyphFit1)
summary(kyphFit2)

# Poisson regression
SOLDER <- ore.push(solder)
solFit1 <- ore.glm(skips ~ ., data = SOLDER, family = poisson())
solFit2 <- glm(skips ~ ., data = solder, family = poisson())
summary(solFit1)
summary(solFit2)
ore.glm results

Generalized Linear Model

R> summary(kyphFit1)

Call:
ore.glm(formula = Kyphosis ~ ., data = KYPHOSIS, family = binomial())

Deviance Residuals:

Min       1Q   Median       3Q      Max
-2.3124   -0.5484   -0.3632   -0.1659    2.1613

Coefficients:

Estimate  Std. Error   z value   Pr(>|z|)
(Intercept) -2.036934   1.449622    -1.405   0.15998
Age          0.010930   0.006447    -1.696   0.08997 .
Number       0.410601   0.224870    -1.826   0.06786 .
Start        -0.206510   0.067700    -3.050   0.00229 **

---

Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

Null deviance: 83.234  on 80  degrees of freedom
Residual deviance: 61.380  on 77  degrees of freedom
AIC: 69.38

Number of Fisher Scoring iterations: 4

R> summary(kyphFit2)

Call:
glm(formula = Kyphosis ~ ., family = binomial(), data = kyphosis)

Deviance Residuals:

Min       1Q   Median       3Q      Max
-2.3124   -0.5484   -0.3632   -0.1659    2.1613

Coefficients:

Estimate  Std. Error   z value   Pr(>|z|)
(Intercept) -2.036934   1.449575    -1.405   0.15996
Age          0.010930   0.006446    -1.696   0.08996 .
Number       0.410601   0.224861    -1.826   0.06785 .
Start        -0.206510   0.067699    -3.050   0.00229 **

---

Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

Null deviance: 83.234  on 80  degrees of freedom
Residual deviance: 61.380  on 77  degrees of freedom
AIC: 69.38

Number of Fisher Scoring iterations: 5
ore.glm results

R> summary(solFit1)
Call:
ore.glm(formula = skips ~ ., data = SOLDER, family = poisson())
Deviance Residuals:
        Min       1Q   Median       3Q      Max
-3.4105   -1.0897  -0.4408   0.6406   3.7927
Coefficients:
                      Estimate Std. Error z value Pr(>|z|)
(Intercept)          -1.25506    0.10069  -12.465  < 2e-16 ***
OpeningM              0.25851    0.06656   3.884  0.000103 ***
OpeningS              1.89394    0.05363  35.305  < 2e-16 ***
SolderThin            1.09973    0.03864  28.465  < 2e-16 ***
MaskA3                0.42819    0.07547   5.674  1.40e-08 ***
MaskB3                1.20225    0.06697  17.953  < 2e-16 ***
MaskB6                1.86648    0.06310  29.580  < 2e-16 ***
PadTypeD6             -0.36865    0.07138  -5.164  2.41e-07 ***
PadTypeD7             -0.09844    0.06620  -1.487  0.137001
PadTypeL6             0.26236    0.06071   4.321  1.55e-05 ***
PadTypeL7             -0.66845    0.07841  -8.525  < 2e-16 ***
PadTypeL8             -0.49021    0.07406  -6.619  3.61e-11 ***
PadTypeL9             -0.63645    0.07759  -8.203  2.35e-16 ***
PadTypeW4             -1.10000    0.06640  -1.657  0.097591
PadTypeW9             -1.43759    0.10419  -13.798  < 2e-16 ***
Panel                 0.11818    0.02056   5.749  8.97e-09 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
(Dispersion parameter for poisson family taken to be 1)
Null deviance: 6855.7  on 719  degrees of freedom
Residual deviance: 1165.4  on 703  degrees of freedom
AIC: 2781.6
Number of Fisher Scoring iterations: 4

R> summary(solFit2)
Call:
glm(formula = skips ~ ., family = poisson(), data = solder)
Deviance Residuals:
        Min       1Q   Median       3Q      Max
-3.4105   -1.0897  -0.4408   0.6406   3.7927
Coefficients:
                      Estimate Std. Error z value Pr(>|z|)
(Intercept)          -1.25506    0.10069  -12.465  < 2e-16 ***
OpeningM              0.25851    0.06656   3.884  0.000103 ***
OpeningS              1.89394    0.05363  35.305  < 2e-16 ***
SolderThin            1.09973    0.03864  28.465  < 2e-16 ***
MaskA3                0.42819    0.07547   5.674  1.40e-08 ***
MaskB3                1.20225    0.06697  17.953  < 2e-16 ***
MaskB6                1.86648    0.06310  29.580  < 2e-16 ***
PadTypeD6             -0.36865    0.07138  -5.164  2.41e-07 ***
PadTypeD7             -0.09844    0.06620  -1.487  0.137001
PadTypeL4             0.26236    0.06071   4.321  1.55e-05 ***
PadTypeL6             -0.66845    0.07841  -8.525  < 2e-16 ***
PadTypeL7             -0.49021    0.07406  -6.619  3.61e-11 ***
PadTypeL8             -0.27115    0.06939  -3.907  9.33e-05 ***
PadTypeL9             -0.63645    0.07759  -8.203  2.35e-16 ***
PadTypeW4             -1.10000    0.06640  -1.657  0.097591
PadTypeW9             -1.43759    0.10419  -13.798  < 2e-16 ***
Panel                 0.11818    0.02056   5.749  8.97e-09 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
(Dispersion parameter for poisson family taken to be 1)
Null deviance: 6855.7  on 719  degrees of freedom
Residual deviance: 1165.4  on 703  degrees of freedom
AIC: 2781.6
Number of Fisher Scoring iterations: 5

©2014 Oracle – All Rights Reserved
Example: Bag of Little Bootstraps
The “Bagging” Concept

Data for Building

Data Samples

S1

S2

Sn

Build Models

M1

M2

Mn

Individual Model Predictions

P1

P2

Pn

Voting or Averaging

Final Prediction

P

Data to Score

©2014 Oracle – All Rights Reserved
“Bagging” Execution Model

Two options: client-controlled and database-controlled

Client R Engine

Multiple invocations of ore.lm to Oracle Database

ORE package

Transparency Layer

Oracle Database

In-db stats

User tables

Database Server Machine

Client R Engine

Single invocation to Oracle Database using ore.indexApply

ORE package

Transparency Layer

Oracle Database

In-db stats

User tables

R Script Repository

extproc

©2014 Oracle – All Rights Reserved
Setting up the data

```r
set.seed(2345)
x <- seq(0.1, 100, by = 0.02)
y <- log(x) + rnorm(x, sd = 1.0)
plot(x,y)

ID <- 1:length(x)
DAT <- ore.push(data.frame(ID=ID, X=x, Y=y))
myDAT <- DAT
row.names(myDAT) <- myDAT$ID
dim(myDAT)
```
Function: bootstrapSample

```r
bootstrapSample <- function (data, num.samples, size) {
  num.rows <- nrow(data)
  sample.indexes <- lapply(1:num.samples,
    function(x) sample(num.rows,
      size, replace=TRUE))

  create.sample <- function (sample.indexes, data) {
    ore.push (data[sample.indexes,])
  }

  samples <- lapply(sample.indexes, create.sample, data)
  samples
}
```
Sampling

```
sample(20, 5, replace=FALSE)
sample(20, 15, replace=TRUE)

inputData <- myDAT
num.samples <- 5
sample.size <- 500
samples.myDAT <- bootstrapSample (inputData,     
                                  num.samples,     
                                  sample.size)

lapply(samples.myDAT,head)
```
Function: bagReg

```r
bagReg <- function (data, formula, num.models, sample.size) {
  samples <- bootstrapSample (data, num.samples = num.models,
                             sample.size)

  build.ore.lm <- function (data, formula, ...){
    ore.lm(formula, data, ...)
  }

  models <- lapply(samples, build.ore.lm, formula)
  models
}
```
Building a regression model

```r
myMod <- ore.lm (Y~X, inputData)
myMod

myPred <- predict(myMod, inputData[1:10,])
myPred

formula_1 <- Y ~ X
sample.size <- 500
num.models <- 25
models <- bagReg (inputData, formula_1, num.models, sample.size)
models[[1]]
models[[25]]
```
Function: predict.bagReg

predict.bagReg <- function (models, data, supp.cols) {

  score.ore.lm <- function (model, data, supp.cols) {
    res <- data.frame(data[,supp.cols])
    res$PRED <- predict (model, data)
    res
  }

  predictions <- lapply (models, score.ore.lm, data, supp.cols)
  scores <- predictions[[1L]][,c(supp.cols)]
  predValues <- lapply(predictions, function(y) y[, "PRED"])
  scores$PRED_MIN <- do.call(pmin, predValues)
  scores$PRED <- rowMeans(do.call(ore.frame, predValues))
  scores$PRED_MAX <- do.call(pmax, predValues)
  scores
}

©2014 Oracle – All Rights Reserved
Predict and evaluate with the bagged model

```r
supp.cols <- c("ID","Y")
scores <- predict.bagReg (models, inputData, supp.cols)
tail(scores,30)

ore.rmse <- function (pred, obs) sqrt(mean(pred-obs)^2)

with(scores, ore.rmse (PRED, Y))

plot(x,y)
res <- lapply (models, abline,col="red")
```
Exadata storage tier scoring for R models

• Fastest way to operationalize R-based models for scoring in Oracle Database

• Go from model to SQL scoring in one step
  – No dependencies on PMML or any other plugins

• R packages supported out-of-the-box include
  – glm, glm.nb, hclust, kmeans, lm, multinom, nnet, rpart

• Models can be managed in-database using ORE datastore
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

<table>
<thead>
<tr>
<th>Class</th>
<th>Package</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>glm</td>
<td>stats</td>
<td>Generalized Linear Model</td>
</tr>
<tr>
<td>negbin</td>
<td>MASS</td>
<td>Negative binomial Generalized Linear Model</td>
</tr>
<tr>
<td>hclust</td>
<td>stats</td>
<td>Hierarchical Clustering</td>
</tr>
<tr>
<td>kmeans</td>
<td>stats</td>
<td>K-Means Clustering</td>
</tr>
<tr>
<td>lm</td>
<td>stats</td>
<td>Linear Model</td>
</tr>
<tr>
<td>multinom</td>
<td>nnet</td>
<td>Multinomial Log-Linear Model</td>
</tr>
<tr>
<td>nnet</td>
<td>nnet</td>
<td>Neural Network</td>
</tr>
<tr>
<td>rpart</td>
<td>rpart</td>
<td>Recursive Partitioning and Regression Tree</td>
</tr>
</tbody>
</table>
# 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, ...)

©2014 Oracle – All Rights Reserved
Example using `lm`

```r
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`
### Example using glm

- Build an R glm model
- Use ore.predict to score data in Oracle Database using ore.frame, e.g., INFERT

```r
R> infertModel <-
+  glm(case ~ age + parity + education + spontaneous + induced,
+      data = infert, family = binomial())
R> INFERT <- ore.push(infert)
R> INFERTpred <- ore.predict(infertModel, INFERT, type = "response",
+                           se.fit = TRUE)
R> INFERT <- cbind(INFERT, INFERTpred)
R> head(INFERT)

<table>
<thead>
<tr>
<th>education</th>
<th>age</th>
<th>parity</th>
<th>induced</th>
<th>case</th>
<th>spontaneous</th>
<th>stratum</th>
<th>pooled.stratum</th>
<th>PRED</th>
<th>SE.PRED</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-5yrs</td>
<td>26</td>
<td>6</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>0.5721916</td>
<td>0.20630954</td>
</tr>
<tr>
<td>0-5yrs</td>
<td>42</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>0.7258539</td>
<td>0.17196245</td>
</tr>
<tr>
<td>0-5yrs</td>
<td>39</td>
<td>6</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>4</td>
<td>0.1194459</td>
<td>0.08617462</td>
</tr>
<tr>
<td>0-5yrs</td>
<td>34</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>4</td>
<td>2</td>
<td>0.3684102</td>
<td>0.17295285</td>
</tr>
<tr>
<td>6-11yrs</td>
<td>35</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>5</td>
<td>32</td>
<td>0.5104285</td>
<td>0.06944005</td>
</tr>
<tr>
<td>6-11yrs</td>
<td>36</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>6</td>
<td>36</td>
<td>0.6322269</td>
<td>0.10117919</td>
</tr>
</tbody>
</table>
```

©2014 Oracle – All Rights Reserved
OREeda Package
Time Series Exponential Smoothing

- Used to produce smoothed data for presentation or for forecasting, i.e., making predictions
- Assigns exponentially decreasing weights over time
- Commonly applied to financial market and economic data

- Simplest form

$$s_1 = x_0$$
$$s_t = \alpha x_{t-1} + (1 - \alpha) s_{t-1} = s_{t-1} + \alpha(x_{t-1} - s_{t-1}), \ t > 1$$

http://en.wikipedia.org/wiki/Exponential_smoothing
ore.esm function signature

```r
ore.esm(x,
    interval = NULL,
    model = "simple",
    accumulate = "NONE",
    setmissing = "PREV",
    rows = 1000000,
    parallel = FALSE,
    optim.start = c(alpha=0.3, beta=0.1),
    optim.control = list())
```

```r
## S3 method for class 'ore.esm'
predict(object, n.ahead = 12L, ...)
```

```r
forecast.ore.esm(object, h = 12L, ...)
```
ore.esm arguments

• **x** – An ordered 'ore.vector' of time series data or transactional data. The ordering column could be either integers from 1 to the length of the time series or of type 'ore.datetime'.

• **interval** – The interval of the time series, or the time interval by which the transactional data is to be accumulated. If the ordering column of the argument 'x' is of type 'ore.datetime', 'interval' must be specified.
  – Possible values: "YEAR", "QTR", "MONTH", "WEEK", "DAY", "HOUR", "MINUTE", "SECOND"

• **model** – The exponential smoothing model name. Possible values: "simple", "double"

• **accumulate** – The method of accumulation. Possible values:
  – NONE No accumulation occurs. Argument 'x' is required to be equally spaced time series observations.
  – TOTAL Accumulation based on the sum of the observed values
  – AVERAGE Accumulation based on the average of the observed values. The value could be abbreviated to "AVG".
  – MINIMUM Accumulation based on the minimum of the observed values. The value could be abbreviated to "MIN".
  – MAXIMUM Accumulation based on the maximum of the observed values. The value could be abbreviated to "MAX".
  – NOBS Accumulation based on the number of observations
  – NMISS Accumulation based on the number of missing observations.
ore.esm arguments

- **setmissing**: The method of treating missing values. Possible values:
  - AVERAGE: Missing values are set to the average of the accumulated values. The value could be abbreviated to "AVG"
  - MINIMUM: Missing values are set to the minimum of the accumulated values. The value could be abbreviated to "MIN"
  - MAXIMUM: Missing values are set to the maximum of the accumulated values. The value could be abbreviated to "MAX"
  - MEDIAN: Missing values are set to the median of the accumulated values. The value could be abbreviated to "MED"
  - FIRST: Missing values are set to the first accumulated non-missing value
  - LAST: Missing values are set to the last accumulated non-missing value
  - PREVIOUS: Missing values are set to the previous accumulated non-missing value. The value could be abbreviated to "PREV"
  - NEXT: Missing values are set to the next accumulated non-missing value.

- **rows**: The maximum number of rows in each chunk in the embedded R solution. If this value is greater than the length of ‘x’, an in-memory sequential solution (faster for small datasets) is used. Otherwise, a scalable embedded R solution is used.

- **parallel**: The preferred degree of parallelism to use in the embedded R job. The value could be one of the following: a positive integer greater than or equal to ‘2’ for a specific degree of parallelism, ‘TRUE’ for the 'x' argument's default parallelism, 'FALSE' or ‘1’ for no parallelism.

- **optim.start**: A vector with named components 'alpha' and 'beta' containing the starting values for the optimizer. Ignored in the 'simple' model case.

- **optim.control**: Optional list with additional control parameters passed to 'optim' in the 'double' model case. Ignored in the 'simple' model case.
predict and forecast arguments for ore.esm

predict(object, n.ahead = 12L, ...)  
forecast.ore.esm(object, h = 12L, ...)

- **object**: object of type 'ore.esm'
- **n.ahead**: number of time periods to forecast
- **h**: number of time periods to forecast
```r
library(TTR)
# Get data for selected stocks in XTS format
stocks <- c("orcl","ibm","sap","msft","appl")
list.data <- vector("list",length(stocks))
for(s in stocks) {
  xts.data <- getYahooData(s, 20010101, 20141024)
  df.data <- data.frame(xts.data)
  df.data$date <- index(xts.data)
  df.data$symbol <- s
  df.data$Split <- NULL
  list.data[[s]] <- df.data
}
stock.data <- data.frame(do.call("rbind",list.data))
ore.drop("STOCKS")
ore.create(stock.data,table="STOCKS")
rownames(STOCKS) <- STOCKS$date
head(STOCKS)
```
Stock Data with ore.esm

```r
orcl.stocks <- ore.pull(STOCKS[STOCKS$symbol=="orcl",c("date","Close","symbol")])

ts.orcl.stocks <- ts(orcl.stocks$Close)

ts.sm.orcl <- ts(SMA(ts.orcl.stocks,n=30),frequency=365, start=c(2008,1))

plot(orcl.stocks$date,orcl.stocks$Close,type="l",col="red",xlab="Date",ylab="US $", main="ORCL Stock Close CLIENT-side Smoothed Series n=30 days")
lines(orcl.stocks$date,ts.sm.orcl,col="blue")
legend("topleft", c("Closing","MA(30) of Closing"),
   col=c("red","blue"),lwd=2,title = "Series",bty="n")

orcl.stock <- STOCKS[STOCKS$symbol=="orcl",c("date","Close")]

dESM.mod <- ore.esm(orcl.stock$Close, "DAY", model = "double")
dESM.predict <- predict(dESM.mod, 30)
plot(orcl.stock,type="l")
lines(dESM.predict,col="red",lwd=4)
```

©2014 Oracle – All Rights Reserved
Supplemental functions to return intermediate states

ore.dESM.states <- function(esm.mod, x) {
  x <- ore.pull(x)
  alpha <- esm.mod$smoothing.param[1]
  beta <- esm.mod$smoothing.param[2]
  len = length(x)
  S.ref0 <- vector(length = len, mode="numeric")
  b.ref0 <- vector(length = len, mode="numeric")
  S.ref0[1] <- x[1]
  for (i.row in 2:len) {
    S.ref0[i.row] <- alpha*x[i.row-1] + (1-alpha)*(S.ref0[i.row-1]+b.ref0[i.row-1])
    b.ref0[i.row] <- beta*(S.ref0[i.row]-S.ref0[i.row-1]) + (1-beta)*b.ref0[i.row-1]
  }
  S.hat <- vector(length = len)
  S.hat[1] <- x[1]
  S.hat[2:len] <- S.ref0[1:(len-1)] + b.ref0[1:(len-1)]
  S.hat
}

ore.sESM.states <- function(esm.mod, x) {
  x <- ore.pull(x)
  alpha <- esm.mod$smoothing.param
  len <- length(x)
  S.ref0 = vector(length = len, mode="numeric")
  S.ref0[1] = x[1]
  for (i.row in 2:len)
    S.ref0[i.row] <- alpha*x[i.row-1] + (1-alpha)*S.ref0[i.row-1]*S.ref0[i.row-1]
  S.ref0
Using supplemental functions

dESM.mod <- ore.esm(orcl.stock$Close, "DAY",
                      model = "double",
                      optim.start=c(alpha=3.5,beta=2.5))

dESM.predict <- predict(dESM.mod, 30)
S.hat <- ore.dESM.states(dESM.mod, orcl.stock$Close)
plot(orcl.stock,type="l",lwd=3)
lines(dESM.predict,col="red",lwd=2)
lines(orcl.stock[,1], S.hat, col='red',lwd=1)

row.idx = 2800:3223
plot(orcl.stock[row.idx,], type="l",lwd=3)
lines(orcl.stock[row.idx,1], S.hat[row.idx], col='red')
lines(dESM.predict,col="red",lwd=2)
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

- **Book:** Using R to Unlock the Value of Big Data, by Mark Hornick and Tom Plunkett
- **Blog:** https://blogs.oracle.com/R/
- **Forum:** https://forums.oracle.com/forums/forum.jspa?forumID=1397

- Oracle R Distribution
- ROracle
- Oracle R Enterprise
- Oracle R Advanced Analytics for Hadoop