Move the algorithms, not the data!

Mark Hornick
Marcos Arancibia
Oracle Machine Learning Product Management
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, timing, and pricing of any features or functionality described for Oracle’s products may change and remains at the sole discretion of Oracle Corporation.
Why data scientists and data analysts use R and Python

- Powerful
- Extensible
- Graphical
- Extensive statistics
- Ease of installation and use
- Rich ecosystem
  - 1000s of open source packages
  - Millions of users worldwide
- Heavily used by data scientists
- Free
Access latency
Paradigm shift: R/Python → Data Access Language → R/Python
Memory limitation – data size, in-memory processing
Single threaded
Issues for backup, recovery, security
Ad hoc production deployment
Elements affecting enterprise scalability for R and Python

- Data Access, Analysis, and Exploration
- Automation
- Machine Learning
- Production Deployment
- Data & Task Parallelism
Data access, analysis, and exploration

Offload processing to more powerful back-ends using data frame proxies
Avoid data movement

All Processing

Light local Processing

Translate R and Python invocations

SQL
Spark
HiveQL

Data + HPC Environments
Data access, analysis, and exploration

- Maintain language features and interface
- Reference data via proxy objects to eliminate data movement
- Overload functions that translate the open source invocations to the language of data processing engine
- Execute at the data processing engine

Analyze all of your data, faster
No data access latency
Leverage performance optimizations of data processing engine
Proxy objects
Example using OML4R interface

```
> str(iris)
'data.frame': 150 obs. of 5 variables:
$ Sepal.Length: num 5.1 4.9 4.4 4.6 5.4 4.4 4.9 ... 5.0 3.6
$ Sepal.Width : num 3.5 3.2 3.1 3.3 3.4 3.4 3.1 ... 1.5 1.5
$ Petal.Length : num 1.4 1.4 1.7 1.4 1.5 1.4 1.5 ... 1.4 1.5
$ Petal.Width : num 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ... 0.2
$ Species    : Factor w/ 3 levels "setosa","versicolor", "...:

> str(iris)
'data.frame': 150 obs. of 5 variables:
Formal class 'one.frame' [package "OoBase"] with 12 slots
...@ DATA : list()
...@ dataOry: Named chr "SELECT * FROM "iris" WHERE Species = 'setosa'"
...@ attr(, "names")<-	
...@ desc: 'data.frame': 5 obs. of 2 variables:
...@ name: chr "Sepal.Length" "Sepal.Width" "Petal.Length" "Petal.Width" ... 
...@ sClass: chr "numeric" "numeric" "numeric" "numeric" ...
...@ sqlValue: chr " Sepal.Length" " Sepal.Width" " Petal.Length" " Petal.Width"
```

# OML4R - Proxy data.frame

Copyright © 2020 Oracle and/or its affiliates.
### OML4R

```r
library(ORE)

ore.connect("oml_user", ...)

ore.sync()
ore.attach()
ore.ls()
head(ONTIME_S)

TMP_ONTIME <- ore.get("ONTIME_S")

dim(ONTIME_S)
summary(ONTIME_S)
cor(ONTIME_S[,c('ARRDELAY','DEPDELAY')], use="complete.obs")
```

### OML4Py*

```python
import oml
import pandas as pd
import matplotlib.pyplot as plt

import oml

oml.connect("oml_user", ...)

t = oml.sync(table="*", regex_match=True)
t.keys()

oml.dir()

ONTIME_S = t.get("ONTIME_S","none")

ONTIME_S.head()

ONTIME_S.shape()
ONTIME_S.describe()
ONTIME_S[['ARRDELAY','DEPDELAY']].corr()
```

* Python coming soon
hist(DF$ARRDELAY, breaks=100, color="green", 
main='Histogram of Arrival Delay',
xlab='Arrival Delay (minutes)',
ylab='# of flights')

boxplot(SEPAL_WIDTH ~ Species, data=IRIS,
notch=TRUE, ylab='cm',
main='Distribution of IRIS Attributes')
### OML4R

```r
df1 <- data.frame(x1=1:5, y1=letters[1:5])
df2 <- data.frame(x1=5:1, y2=letters[11:15])

ore.drop(table="TEST_DF1")
ore.drop(table="TEST_DF2")

ore.create(df1, table="TEST_DF1")
ore.create(df2, table="TEST_DF2")

merge(TEST_DF1, TEST_DF2, by="x1")
```

### OML4Py*

```python
letters = list(map(chr, range(97, 123)))
df1 = pd.DataFrame({'x1':range(1,6),
                    'x2':letters[1:6]})
df2 = pd.DataFrame({'x1':range(1,6),
                    'x2':letters[11:16]})

oml.drop("TEST_DF1")
oml.drop("TEST_DF2")

TEST_DF1 = oml.create(df1, table="TEST_DF1")
TEST_DF2 = oml.create(df2, table="TEST_DF2")

df2.merge(df1, on='x1')
```
Scalable Machine Learning

Offload model building to parallel software and powerful machines and environments

Data Source — Data Sink

All Processing

Light local Processing

Translate R and Python invocations

SQL
Spark
Tensorflow

Main Processing
Main Processing
Main Processing

Copyright © 2020 Oracle and/or its affiliates.
Scalable Machine Learning

Maintain open source machine learning interface

- OML4R - easy to specify R formula – minimal lines of code including transformations, interaction terms, etc.
- OML4Py – familiar Python predictors/target interface with fit() and predict()
Scalable Machine Learning

Maintain open source machine learning interface
  • OML4R - easy to specify R formula – minimal lines of code including transformations, interaction terms, etc.
  • OML4Py – familiar Python predictors/target interface with fit() and predict()

Bring the algorithm to the data
  • Eliminate or minimize data movement
  • Leverage proxy objects to reference data from R/Python
Scalable Machine Learning

Maintain open source machine learning interface

- OML4R - easy to specify R formula – minimal lines of code including transformations, interaction terms, etc.
- OML4Py – familiar Python predictors/target interface with fit() and predict()

Bring the algorithm to the data

- Eliminate or minimize data movement
- Leverage proxy objects to reference data from R/Python

Parallel, distributed algorithm implementations

- Custom state-of-the-art integrated implementations
- Supplement with open source packages and toolkits
n.rows <- nrow(IRIS)
row.names(IRIS) <- IRIS$Species
my.smpl <- sample(1:n.rows, ceiling(n.rows*0.7))
train.dat <- IRIS[my.smpl,]

# OML4R

test.dat <- IRIS[setdiff(1:n.rows, my.smpl),]

rf_mod <- ore.randomForest(Species~., train.dat)

# OML4Py*

from oml import rf

train_dat, test_dat = IRIS.split()
train_x = train_dat.drop('Species')
train_y = train_dat['Species']

rf_mod = rf()

rf_mod = rf_mod.fit(train_x, train_y)

# OML4R

pred <- predict(rf_mod, test.dat, type="all",
               supplemental.cols=c("SEPAL_LENGTH","Species"))

table(pred$Species, pred$prediction)

# OML4Py*

pred = rf_mod.predict(test_dat.drop('Species'),
                     supplemental_cols = test_dat[:,['SEPAL_LENGTH','Species']])

res_ct =
    pred.crosstab('Species','PREDICTION',pivot=True)
res_ct.sort_values(by='Species')
Benefits of parallelism and *not* moving data

Data remains at Oracle Database Server Machine

- Predict “Total Revenue” of a customer
- 184 million records, 31 numeric predictors
- Data stored in an Oracle Database table

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Threads Used*</th>
<th>Memory required**</th>
<th>Time for Data Loading***</th>
<th>Time just for Computation</th>
<th>Total Elapsed</th>
<th>Relative Performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Open-Source R Linear Model (lm)</td>
<td>1</td>
<td>220Gb</td>
<td>1h3min</td>
<td>43min</td>
<td>1h46min</td>
<td>1x</td>
</tr>
<tr>
<td>OML4R lm (ore.lm)</td>
<td>1</td>
<td>-</td>
<td>-</td>
<td>42.8min</td>
<td>42.8min</td>
<td>2.47X</td>
</tr>
<tr>
<td>OML4R lm (ore.lm)</td>
<td>32</td>
<td>-</td>
<td>-</td>
<td>1min34s</td>
<td>1min34s</td>
<td>67.7X</td>
</tr>
<tr>
<td>OML4R lm (ore.lm)</td>
<td>64</td>
<td>-</td>
<td>-</td>
<td>57.97s</td>
<td>57.97s</td>
<td>110X</td>
</tr>
<tr>
<td>OML4R lm (ore.lm)</td>
<td>128</td>
<td>-</td>
<td>-</td>
<td>41.69s</td>
<td>41.69s</td>
<td>153X</td>
</tr>
</tbody>
</table>

* Open-source R lm() is single threaded

** Data moved into R Session memory, since open-source lm() requires all data to be in-memory

*** Time to load 40Gb of raw data into the open-source R Session’s memory
Data and Task Parallel Execution

- Hand-code logic to spawn R / Python engine
- Partition and feed data to R engines as they become available
- Parallelism controlled manually

Execute user-defined function on back-end servers
Use multiple engines for data- or task-parallel execution
Auto-partition and feed data
Leverage open source packages

Data Source

Data Sink

Script Repository
Object Repository

Spawn / control engines
Provide function + data

Store UDF and invoke
Data and Task Parallel Execution

Easily specify parallelism and data partitioning
  • Simplified API – all-in-one
  • Build and score with large number of open source models
  • Partition data by value or count
  • Invoke user-defined functions in parallel with index input

Automated management of R and Python engines
  • Insulation from hardware details
  • Limit memory and compute resources, as possible

Automated load of data and user-defined functions into R and Python engines

Leverage packages from open source ecosystems
### Comparing OML4R and OML4Py – Table Apply

<table>
<thead>
<tr>
<th><strong>OML4R</strong></th>
<th><strong>OML4Py</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>library(e1071)</td>
<td>def build_nb(dat, dsname):</td>
</tr>
<tr>
<td>buildNB &lt;-</td>
<td>import oml</td>
</tr>
<tr>
<td>function(dat,dsname) {</td>
<td>from sklearn.naive_bayes import GaussianNB</td>
</tr>
<tr>
<td>library(e1071)</td>
<td>from sklearn import preprocessing</td>
</tr>
<tr>
<td>dat$Species &lt;- as.factor(dat$Species)</td>
<td>le = preprocessing.LabelEncoder()</td>
</tr>
<tr>
<td>mod&lt;-naiveBayes(Species ~ ., dat)</td>
<td>raw_labels = dat[&quot;Species&quot;].values</td>
</tr>
<tr>
<td>ore.save(mod,name=dsname,overwrite=TRUE)</td>
<td>le.fit(raw_labels)</td>
</tr>
<tr>
<td>mod</td>
<td>y = le.transform(raw_labels)</td>
</tr>
<tr>
<td></td>
<td>X = dat[&quot;SEPAL_LENGTH&quot;,&quot;SEPAL_WIDTH&quot;, &quot;PETAL_LENGTH&quot;,&quot;PETAL_WIDTH&quot;][.values</td>
</tr>
<tr>
<td></td>
<td>mod = GaussianNB().fit(X, y)</td>
</tr>
<tr>
<td>mod &lt;- ore.tableApply(IRIS, buildNB, dsname='NB_Model-1', ore.connect=TRUE)</td>
<td>oml.ds.save(objs={'mod':mod},name=dsname, overwrite=True)</td>
</tr>
<tr>
<td></td>
<td>mod = oml.table_apply(IRIS, build_nb, dsname = 'NB_Model-1', oml_connect=True)</td>
</tr>
</tbody>
</table>
Comparing OML4R and OML4Py – Row Apply

**OML4R**

```r
scoreNBmodel <- function(dat, dsname) {
  library(e1071)
  ore.load(dsname)
  dat$PRED <- predict(mod, newdata = dat)
  dat
}
```

```r
IRIS_PRED <- IRIS[1,]
IRIS_PRED$PRED <- "A"
res <- ore.rowApply(IRIS, scoreNBmodel,
  dsname = 'NB_Model-1',
  parallel = 4, rows = 10,
  FUN.VALUE = IRIS_PRED,
  ore.connect = TRUE)
```

**OML4Py**

```python
def score_nb_mod(dat, dsname):
    import oml
    from sklearn.naive_bayes import GaussianNB
    objs = oml.ds.load(dsname, to_globals=False)
    mod = objs['mod']
    dat['PREDICTION'] =
        mod.predict(dat.drop('Species', axis=1))
    return dat
```

```python
IRIS_PRED = pd.DataFrame([[(1,1,1,1,'a',1)],
                          columns=['SEPAL_LENGTH','SEPAL_WIDTH',
                                   'PETAL_LENGTH','PETAL_WIDTH',
                                   'Species','PREDICTION'])
res = oml.row_apply(IRIS, score_nb_mod,
                     dsname = 'NB_Model-1',
                     parallel = 4, rows = 10,
                     func_value = IRIS_PRED,
                     oml_connect = True)
```
Deployment

Data frames
Images
Objects

R and Python Scripts

Application
Business Logic
C, C++, Java, SQL

Results as table: structured, image, XML
Automate parallel and concurrent execution

Use familiar SQL or REST protocols for invoking R and Python
Integrate with applications and dashboards that already use SQL or REST

Data Sources

Application
Business Logic
C, C++, Java, SQL

SQL – R/Python Engine
R/Python Scripts
Data

RBDMS
HDFS
NoSQL
HIVE
Deployment

- Invoke user-defined R and Python functions easily from environments that readily use SQL or REST
- Automated mapping of data structures and types
- Seamlessly return data frames, images, XML as database rowsets
- Schedule execution of user-defined functions for “lights-out” operation
Create user-defined functions from SQL (or use from R/Python)

**OML4R**

```sql
BEGIN
    sys.rqScriptDrop('RandomRedDots');
    sys.rqScriptCreate('RandomRedDots',
        'function(){
            id <- 1:10
            plot( 1:100, rnorm(100), pch = 21,
                bg = "red", cex = 2,
                main="Random Red Dots" )
            data.frame(id=id, val=id / 100)
        }');
END;
```

**OML4Py*”

```python
BEGIN
    sys.pyqScriptDrop('RandomRedDots');
    sys.pyqScriptCreate('RandomRedDots',
        'def RandomRedDots():
            import numpy as np
            import pandas as pd
            import matplotlib.pyplot as plt
            d = {'id': range(1,10),
                 'val': [x/100 for x in range(1,10)]}
            df = pd.DataFrame(data=d)
            fig = plt.figure(1)
            ax = fig.add_subplot(111)
            ax.scatter(range(0,100),
                        np.random.rand(100),c='r')
            fig.suptitle("Random Red Dots")
            return df', NULL, TRUE);
END;
```
Invoke user-defined functions from SQL

**OML4R**

- `select ID, IMAGE from table(rqEval(NULL,'PNG','RandomRedDots'))`
- `select ID, VAL from table(rqEval(NULL,
  'select 1 id, 1 val from dual',
  'RandomRedDots'))`
- `select dbms_lob.substr(VALUE,4000,1) from table(rqEval(NULL,'XML','RandomRedDots'))`
- # In R, invoke same function by name
  `ore.doEval(FUN.NAME='RandomRedDots')`

**OML4Py*`

- `select ID, IMAGE from table(pyqEval(NULL,'PNG','RandomRedDots'))`
- `select ID, VAL from table(pyqEval(NULL,
  'select 1 id, 1 val from dual',
  'RandomRedDots'))`
- `select dbms_lob.substr(VALUE,4000,1) from table(pyqEval(NULL,'XML','RandomRedDots'))`
- # In Python, invoke same function by name
  `res = oml.do_eval(func='RandomRedDots')`
Automation

User builds and evaluates many models using multiple algorithms with various settings — trial and error approach

Build Model → Evaluate Model

Data Source → Candidate Models

Automated ML

ML-enhanced functions automatically select and tune models

Increase model quality
Increase data scientist productivity
Reduce overall compute time
Invoke single function to find “best” algorithm and model
Automated hyperparameter tuning selects “best” algorithm settings

User builds and evaluates many models using multiple algorithms with various settings — trial and error approach
AutoML – *new* with OML4Py

Increase data scientist productivity – reduce overall compute time

Auto Algorithm Selection
- Identify in-database algorithm that achieves highest model quality
- Find best model faster than with exhaustive search

Auto Feature Selection
- Reduce # of features by identifying most predictive
- Improve performance and accuracy

Auto Tune Hyperparameters
- Significantly improve model accuracy
- Avoid manual or exhaustive search techniques

Enables non-expert users to leverage Machine Learning
Auto Algorithm Selection
Identify best algorithm to achieve maximum score metric guided by ML
Find best model many times faster than with exhaustive search techniques

OML4Py

```python
ms = AlgorithmSelection(mining_function = 'classification',
                        score_metric = 'accuracy',
                        parallel = 4)

best_model = ms.select(X_train, y_train)

all_models = ms.select(X_train, y_train, tune=False)
```
Auto Feature Selection
Speed-up ML pipeline by selecting most relevant features

OML4Py

```python
fs = FeatureSelection(mining_function = 'classification',
                      score_metric = 'accuracy',
                      parallel = 4)

selected_features = fs.reduce('dt', X_train, y_train)

X_train = X_train[:, selected_features]
```
OML4Py Auto Feature Selection: examples
Reduce # of features by identifying most relevant, improving performance and accuracy

OpenML dataset 312 with 1925 rows, 299 columns

OpenML dataset 40996 with 56K rows, 784 columns

ML training time

<table>
<thead>
<tr>
<th>Training time (seconds)</th>
<th>299</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>299</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>33x reduction</td>
<td>97%</td>
<td></td>
</tr>
</tbody>
</table>

Prediction Accuracy

<table>
<thead>
<tr>
<th>Accuracy</th>
<th>299</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.93</td>
<td>0.94</td>
<td></td>
</tr>
<tr>
<td>+4%</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Prediction Accuracy

<table>
<thead>
<tr>
<th>Accuracy</th>
<th>784</th>
<th>309</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.65</td>
<td>0.75</td>
<td></td>
</tr>
<tr>
<td>+18%</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1.3X time reduction to build SVM Gaussian model
60% reduction
Auto Tune
Significantly improve model accuracy guided by ML
Avoid manual or exhaustive search techniques

OML4Py

```python
at = Autotune(mining_function = 'classification',
              score_metric = 'accuracy',
              parallel = 4)

results = at.tune('dt', X_train, y_train)

best_mod = results['best_model']
all_evals = results['all_evals']
```
Mean improvement 1.7%
Across ~300 datasets
8-24% improvement for several datasets

Auto Tune:
Evaluation for OML Neural Network

OML Neural Network - Default vs. Tuned Accuracy
Enabling R and Python for the Enterprise

1. Leverage new, more powerful ML back-ends and libraries more easily as they arise
2. Enable data- and task-parallel execution quickly and easily for big data processing
3. Offload processing to powerful back-ends for the heavy lifting... transparent scalability
4. Immediately leverage data scientist R and Python scripts and results in production environments
5. Enhance user productivity through automation
Oracle Machine Learning for R / Python
Coming soon to Oracle Autonomous Database

• Use Oracle Database as HPC environment
• Use in-database parallel and distributed machine learning algorithms
• Manage R and Python scripts and objects in Oracle Database
• Integrate open source results into applications and dashboards via SQL
• In OML4Py, automated machine learning – AutoML
Oracle Machine Learning for R / Python
Coming soon to Oracle Autonomous Database

Transparency layer
- Leverage proxy objects so data remain in database
- Overload native functions translating functionality to SQL
- Use familiar R / Python syntax to manipulate database data

Parallel, distributed algorithms
- Scalability and performance
- Exposes in-database algorithms available from OML4SQL

Embedded execution
- Manage and invoke R or Python scripts in Oracle Database
- Data-parallel, task-parallel, and non-parallel execution
- Use open source packages to augment functionality

OML4Py Automated Machine Learning - AutoML
- Model selection, feature selection, hyper-parameter tuning
For more information...

oracle.com/machine-learning

Oracle Machine Learning

The Oracle Machine Learning product family enables scalable data science projects. Data scientists, analysts, developers, and IT can achieve data science project goals faster while taking full advantage of the Oracle platform.

Oracle Machine Learning consists of complementary components supporting scalable machine learning algorithms for in-database and big data environments, notebook technology, SQL and R APIs, and Hadoop/Spark environments.

See AskTOM OML Office Hours

Ask The Oracle Masters
Oracle Cloud Infrastructure

New Free Tier with Always Free Oracle Autonomous Database and Oracle APEX

oracle.com/cloud/free

Always Free
Services you can use for unlimited time

30-Day Free Trial
Get $500 in free credits

LEARN MORE
Oracle Cloud's New Free Tier and Always Free Oracle Autonomous Database
Speaker: Todd Bottger
Session ID: PRO6695
Wednesday, Sept 18th at 10:00am PT
Moscone South #203
Mark Hornick
Marcos Arancibia

Thank you!