Graph Analytics and Machine Learning – A Great Combination

Mark Hornick
Oracle Advanced Analytics and Machine Learning
November 3, 2017
Safe Harbor Statement

The following is intended to outline our research activities and 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 remains at the sole discretion of Oracle.
data ➔ Insights
Graphs are everywhere

Inter-relationships between data and networks are growing in importance

- Social Media: Facebook (friends of friends), Twitter, LinkedIn, etc.
- IoT and Sensor networks
- Telecom customer networks
- Healthcare providers, patients, services, etc.

**Computational Graph Analytics (full graph)**
Influencer ID, community detection and ranking, recommendations, structure finding, path finding

**Graph Pattern Matching**
Queries that find sub-graphs fitting relationship patterns
Machine Learning has become mainstream
ML algorithms, AI, Deep Learning uncover patterns and predict outcomes

- Social Media: Facebook, Twitter, LinkedIn demographic and text data
- Product recommendation and customer churn
- IoT and Sensor networks time series
- Image and voice recognition
- Banking, retail, healthcare, manufacturing, human resources, etc.
Graph Analytics

Machine Learning
Graph Analytics

Compute graph metric(s)

Add to structured data

Machine Learning

Build predictive model using graph metric
Graph Analytics

- Compute graph metric(s)
- Explore graph or compute new metrics using ML result

Machine Learning

- Add to structured data
- Build predictive model using graph metric
- Add to graph
- Build model(s) and score or classify data
Graph Analytics

- Compute graph metric(s)
- Explore graph or compute new metrics using ML result

Machine Learning

- Add to structured data
- Add to graph
- Build predictive model using graph metric
- Build model(s) and score or classify data

Approach problem from two perspectives
Money Laundering Pattern Detection

• Graph pattern matching with cycle detection
  – Identify chain of wire transfers, including external entity, between two accounts of single owner
  – Find out how data entities are connected with each other via multiple hops

• ML Anomaly Detection
  – Identify unusual transactions (unsupervised) or predict based on known outcomes (supervised)
  – Cluster actor demographics and analyze those whose behavior most differ within cluster
Retail Product Recommendation

• Graph Matrix Factorization, Personalized Pagerank
  – Given an item, recommend *close* items from user-item or item-feature graph
  – Given a user, identify *close* users who purchased similar items and recommend items popular among those

• ML Classification, Association Rules
  – Combine customer demographics/psychographics with purchase data to predict interest
  – Extract rules of items commonly purchased together

Items similar to this:

Other people also liked:
Fraud Detection in Healthcare

- **Graph Personalized Page Rank**
  - Identify providers who perform services not related to their specialty

- **ML Anomaly Detection**
  - Use 1-Class SVM to find providers with anomalous behaviors
  - Cluster providers and identify those clusters with interesting characteristics
Graph Analytics and Machine Learning

• R has packages for both
• Minimize impedance mismatch with integrated API
• For big data, need scalable and performant tools
Strategy

• Avoid moving data to the client
  – Data access latency
  – Memory constraints
  – Processor / compute limitation

• Leverage parallel algorithm implementations on powerful back-end hardware platforms
  – Oracle Database – the data are already there
  – Oracle R Enterprise – scalable and performant in-database analytics via R
  – Oracle PGX (Parallel Graph AnalytiX) – scalable and performant graph algorithms

• Provide R interface integrating Oracle’s Graph and ML technologies
OAAgraph

• An R package integrating Parallel Graph AnalytiX with Oracle R Enterprise

• Single, unified interface
  – Work with R `data.frame` proxy objects (`ore.frame`) for database data and familiar functions across ML and graph
  – Results available as R `data.frame` proxy objects allowing further processing

• R users take advantage of powerful, complementary technologies available with Oracle Database
  – Highly scalable PGX engine, part of Oracle Spatial and Graph option
  – Integrated with Oracle R Enterprise, part of Oracle Advanced Analytics option
PGX (Parallel Graph AnalytiX)

- In-memory graph engine
- Fast, parallel, built-in graph algorithms
- 35+ graph algorithms
- Graph query (pattern-matching) via PGQL
- Custom algorithm compilation (advanced use case)
- PGX also available on Hadoop and NoSQL
## PGX Performance

<table>
<thead>
<tr>
<th>X86 Server</th>
<th>#Nodes</th>
<th>#Edges</th>
<th>Closeness Centrality</th>
<th>Betweenness Centrality</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>PGX</td>
<td>igraph</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>PGX</td>
<td>igraph</td>
</tr>
<tr>
<td><strong>Epinion</strong></td>
<td>75,879</td>
<td>811,480</td>
<td>17 sec</td>
<td>6.27 min</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>45 sec</td>
<td>11.5 min</td>
</tr>
<tr>
<td><strong>Google</strong></td>
<td>107,614</td>
<td>2,4476,570</td>
<td>15 sec</td>
<td>132 min</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>30 min</td>
<td>290 min</td>
</tr>
<tr>
<td><strong>Churner</strong></td>
<td>3,000,000</td>
<td>16,519,402</td>
<td>26 min</td>
<td>5+ days</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>20 hrs</td>
<td>5+ days</td>
</tr>
<tr>
<td><strong>Patent</strong></td>
<td>3,774,768</td>
<td>33,037,894</td>
<td>6.6 hrs</td>
<td>5+ days</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>48 hrs</td>
<td>5+ days</td>
</tr>
<tr>
<td><strong>LiveJ</strong></td>
<td>4,846,609</td>
<td>85,702,474</td>
<td>5.8 hrs</td>
<td>5+ days</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>60 hrs</td>
<td>5+ days</td>
</tr>
</tbody>
</table>

**X86 Server**
- Xeon E5-2660 2.2Ghz
- 2 socket
- x 8 cores
- x 2HT
- 256GB DRAM
# PGX Performance

<table>
<thead>
<tr>
<th>X86 Server</th>
<th>#Nodes</th>
<th>#Edges</th>
<th>Closeness Centrality</th>
<th>Betweenness Centrality</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PGX</td>
<td>igraph</td>
<td>PGX</td>
<td>igraph</td>
</tr>
<tr>
<td>Epinion</td>
<td>75,879</td>
<td>811,480</td>
<td>17 sec</td>
<td>6.27 min</td>
</tr>
<tr>
<td>Google</td>
<td>107,614</td>
<td>2,447,570</td>
<td>15 sec</td>
<td>132 min</td>
</tr>
<tr>
<td>Churner</td>
<td>3,000,000</td>
<td>16,519,402</td>
<td>26 min</td>
<td>5+ days</td>
</tr>
<tr>
<td>Patent</td>
<td>3,774,768</td>
<td>33,037,894</td>
<td>6.6 hrs</td>
<td>5+ days</td>
</tr>
<tr>
<td>LiveJ</td>
<td>4,846,609</td>
<td>85,702,474</td>
<td>5.8 hrs</td>
<td>5+ days</td>
</tr>
</tbody>
</table>

Effectively does not complete!

- **X86 Server**
  - Xeon E5-2660 2.2Ghz
  - 2 socket
  - 8 cores
  - 2 HT
  - 256GB DRAM

---

Copyright © 2017 Oracle and/or its affiliates. All rights reserved.
PGX Performance

• Comparisons against existing graph engines
  – GraphX (Apache Spark)
  – GraphLab (Turi)

• With seven popular algorithms
  – Pagerank (exact and approx), Weakly Connected Components, Single-Source Shortest Path, Hop-Distance (BFS), Eigen Vector, K-Core

• On Two Graph instances
  – Twitter Graph (TWT): 41 million nodes 1.4 billion edges
  – Web Graph (WEB): 77 million nodes 2.9 billion edges

Hardware: Intel(R) Xeon(R) CPU E5-2699 v4 @ 2.20GHz - 256 RAM
Network: Melanox Infiniband (56Gbps)
PGX Performance

- Comparisons against existing graph engines
  - GraphX (Apache Spark)
  - GraphLab (Turi)

- With seven popular algorithms
  - Pagerank (exact and approx), Weakly Connected Components, Single-Source Shortest Path, Hop-Distance (BFS), Eigen Vector, K-Core

- On Two Graph instances
  - Twitter Graph (TWT): 41 million nodes 1.4 billion edges
  - Web Graph (WEB): 77 million nodes 2.9 billion edges

Hardware: Intel(R) Xeon(R) CPU E5-2699 v4 @ 2.20GHz - 256 RAM
Network: Melanox Infiniband (56Gbps)
PGX Performance

- Comparisons against existing graph engines: GraphX (Spark), GraphLab
- With seven popular algorithms: PageRank (exact and approx), Weakly Connected Components, Single-Source Shortest Path, Hop Distance (BFS), EigenVector, K-Core
- On Two Graph instances: Twitter Graph (TWT): 41 million nodes, 1.4 Billion edges, Web Graph (WEB): 77 million nodes, 2.9 Billion edges

Hardware: Intel(R) Xeon(R) CPU E5-2699 v4 @ 2.20GHz - 256 RAM
Network: Melanox Infiniband (56Gbps)
Oracle R Enterprise

• Use Oracle Database as a high performance compute environment
• Transparency layer
  – Leverage proxy objects (ore.frames) - data remains in the database
  – Overload R functions that translate functionality to SQL
  – Use standard R syntax to manipulate database data
• Parallel, distributed ML algorithms
  – Scalability and performance
  – Exposes in-database machine learning algorithms from ODM
  – Additional R-based algorithms executing and database server
• Embedded R execution
  – Store and invoke R scripts in Oracle Database
  – Data-parallel, task-parallel, and non-parallel execution
  – Invoke R scripts at Oracle Database server from R or SQL
  – Use open source CRAN packages
# ORE Performance

Execution of a Linear Model to Predict “Total Revenue” of a customer based on 31 numerical values as predictors, on 184M records

Data already 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 for Computation</th>
<th>Total</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>Oracle R Enterprise 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>Oracle R Enterprise lm (ore.lm)</td>
<td>32</td>
<td>-</td>
<td>-</td>
<td>1min34s</td>
<td>1min34s</td>
<td>67.7X</td>
</tr>
<tr>
<td>Oracle R Enterprise 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>Oracle R Enterprise 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 the R Session's memory, since open-source lm() requires all data to be in-memory

***How long it takes to load 40Gb of raw data into the open-source R Session's memory
# Connect R client to Oracle Database using ORE
R> ore.connect(..)
# Connect R client to Oracle Database using ORE
R> ore.connect()

# Connect to PGX server using OAAgraph
R> oaa.graphConnect(...)

OAAGraph with Oracle Database
Data Sources

• Graph data represented as two tables
  – Nodes with properties
  – Edges with properties

Node Table

<table>
<thead>
<tr>
<th>Node ID</th>
<th>Node Prop 1 (name)</th>
<th>Node Prop 2 (age)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1238</td>
<td>John</td>
<td>39</td>
</tr>
<tr>
<td>1299</td>
<td>Paul</td>
<td>41</td>
</tr>
<tr>
<td>4818</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Edge Table

<table>
<thead>
<tr>
<th>From Node</th>
<th>To Node</th>
<th>Edge Prop 1 (relation)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1238</td>
<td>1299</td>
<td>Likes</td>
</tr>
<tr>
<td>1299</td>
<td>4818</td>
<td>FriendOf</td>
</tr>
<tr>
<td>1299</td>
<td>6637</td>
<td>FriendOf</td>
</tr>
</tbody>
</table>
# Load graph into PGX:
# Graph load happens at the server side.
# Returns OAAgraph object – a proxy
# for the graph in PGX
R> mygraph <-
oaa.graph (EdgeTable, NodeTable, ...)

Client

R Client

OLE

OAAgraph

Database Server

PGX Server

node

edge

In Database
Running Graph Algorithm

# e.g. compute Pagerank for every node
# in the graph
# Execution occurs in PGX server side
R> result1<- pagerank (mygraph, ... )
Iterating remote values with cursor

# e.g. compute Pagerank for each node in the graph
# Execution occurs in PGX server side
R> result1 <- pagerank (mygraph, ... )
Iterating remote values with cursor

# e.g. compute Pagerank for each node in the graph
# Execution occurs in PGX server side
R> result1 <- pagerank (mygraph, ... )

# Return value is a “cursor” object
# for the computed result:
# client can get local data frames by oaa.next()
R> df <- oaa.next(result1, 10)
Querying the graph

# Query graph via SQL syntax pattern specification
R> q_result <- oaa.cursor(mygraph, "SELECT n.name, m.name, n.pagerank, m.pagerank
   WHERE (n WITH pagerank < 0.1) -> (m),
   n.pagerank > m.pagerank
   ORDER BY n.pagerank")
# Returns a cursor to examine results
R> df <- oaa.next(q_result, 10)
Querying the graph

# Query graph via SQL syntax pattern specification
R> q_result <- oaa.cursor(mygraph,
  "SELECT n.name, m.name, n.pagerank, m.pagerank
  WHERE (n WITH pagerank < 0.1) -> (m),
  n.pagerank > m.pagerank
  ORDER BY n.pagerank")
# Returns a cursor to examine results
R> df <- oaa.next(q_result, 10)
Querying the graph

# Query graph via SQL syntax pattern specification
R> q_result <- oaa.cursor(mygraph, 
  "SELECT n.name, m.name, n.pagerank, m.pagerank 
  WHERE (n WITH pagerank < 0.1) -> (m), 
  n.pagerank > m.pagerank 
  ORDER BY n.pagerank")
# Returns a cursor to examine results
R> df <- oaa.next(q_result, 10)
Exporting the result to DB

# Export result to DB as Table(s)
R> oaa.create(mygraph, nodeTableName = "node",
              nodeProperties = c("pagerank", ...),
              ... )
Exporting the result to DB

R Client
- ORE
- OAAgraph

# Export result to DB as Table(s)
R> oaa.create(mygraph, nodeTableName = "node",
              nodeProperties = c("pagerank", ... ),
              ... )
Continuing analysis with ORE ML

# Use ORE Machine Learning on
# the exported table proxy object ore.frames
R> model <- ore.odmKMmeans(formula = ~.,
                           data = NODES,
                           num.centers = 5,...)
R> scores <- predict(model, NODES, ...)
...
Fraud Detection in Healthcare Billing

• Public dataset US Center for Medicare and Medicaid Services (CMS) for 2012
  – Aggregated medical transactions: 9,153,272 records with 29 variables
  – Transactions between 880,644 medical providers and CMS with total amounts > $77B for the year
  – Per provider/service aggregate counts, and submitted/allowed/payment mean/sd

• Using data in database table
  – Analyze the data using R/ORE
  – Create NODE and EDGE tables for PGX
  – Use *personalized page rank* graph algorithm to identify anomalies
  – Perform ML-based anomaly detection

Setting up the graph

Create a bipartite graph from table data

- Node – LHS as health care **providers** and RHS as health **services**
- Edge – a specific medical provider (LHS vertex) who provided a specific health service (RHS vertex)
- There exists a two-hop (undirected) path between two medical providers, for each of the common health services they both provide
Finding anomalies using graph  (simplified approach)

• The more common services two providers provide, the closer the medical providers are to each other in the graph

• Since medical providers of same specialty likely perform many common services, these vertices are expected to be closer to one another than the others

• Providers exceptionally close to other providers with different specialty are considered anomalous

• Compute personalized page rank per specialty, then cross-check values to identify anomalies

• Look up the highest ranked vertices to determine if they belong to the current specialty, if not, this is an anomaly
Demo

OAAGraph
Summary

• OAAGraph provides powerful, scalable graph analytics enabled from R in Oracle Database

• Augment ML model building with graph metrics

• Augment graph analytics with ML scores and predictions

• Explore and integrate complementary approaches
Learn More about Oracle’s R Technologies...

http://oracle.com/goto/R