Please ask questions via the mobile app!

Engage
Fast Analytics on Big Data with H20

0xdata.com, h2o.ai

Tomas Nykodym, Petr Maj
Team
About H2O and 0xdata

- H2O is a platform for distributed in memory predictive analytics and machine learning
- Pure Java, Apache v2 Open Source
- Easy deployment with a single jar, automatic cloud discovery
- https://github.com/0xdata/h2o
- https://github.com/0xdata/h2o-dev
- Google group h2ostream
- ~15000 commits over two years, very active developers
Overview

- H2O Architecture
- GLM on H2O
  - demo
- Random Forest
H2O Architecture
Practical Data Science

- Data scientists are not necessarily trained as computer scientists
- A “typical” data science team is about 20% CS, working mostly on UI and visualization tools
- An example is Netflix
  - Statisticians prototype in R
  - When done, developers recode the code in Java and Hadoop
What we want from modern machine learning platform

<table>
<thead>
<tr>
<th>Requirements</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fast &amp; Interactive</td>
<td>In-Memory</td>
</tr>
<tr>
<td>Big Data (no sampling)</td>
<td>Distributed</td>
</tr>
<tr>
<td>Flexibility</td>
<td>Open Source</td>
</tr>
<tr>
<td>Extensibility</td>
<td>API/SDK</td>
</tr>
<tr>
<td>Portability</td>
<td>Java, REST/JSON</td>
</tr>
<tr>
<td>Infrastructure</td>
<td>Cloud or On-Premise Hadoop or Private Cluster</td>
</tr>
</tbody>
</table>
H2O Architecture

- **Frontends**
  - REST API, R, Python, Web Interface

- **Algorithms**
  - GBM, Random Forest, GLM, PCA, K-Means, Deep Learning

- **Core**
  - Distributed Tasks
  - Map/Reduce
  - Distributed in memory K/V Store
  - Column Compressed Data
  - Memory Managed

- **Data Sources**
  - HDFS, S3, NFS, Web Upload
Distributed Data Taxonomy

Vector
Distributed Data Taxonomy

The vector may be very large ~ billions of rows

- Store compressed (often 2-4x)
- Access as Java primitives with on the fly decompression
- Support fast Random access
- Modifiable with Java memory semantics
Distributed Data Taxonomy

- Large vectors must be distributed over multiple JVMs
  - Vector is split into chunks
  - Chunk is a unit of parallel access
  - Each chunk ~ 1000 elements
  - Per chunk compression
  - Homed to a single node
  - Can be spilled to disk
  - GC very cheap
Distributed Data Taxonomy

A row is always stored in a single JVM

- Similar to R frame
- Adding and removing columns is cheap
- Row-wise access
Distributed Data Taxonomy

- **Elem** - a java double
- **Chunk** - a collection of thousands to millions of elems
- **Vec** - a collection of Chunks
- **Frame** - a collection of Vecs
- **Row i** - i’th elements of all the vecs in a frame
Distributed Fork/Join
Distributed Fork/Join

- Task is distributed in a tree pattern
- Results are reduced at each inner node
- Returns with a single result when all subtasks done
Distributed Fork/Join

- On each node the task is parallelized over home chunks using Fork/Join
- No blocked thread using continuation passing style
Distributed Code

- Simple tasks
  - Executed on a single remote node
- Map/Reduce
  - Two operations
    - map(x) -> y
    - reduce(y, y) -> y
  - Automatically distributed amongst the cluster and worker threads inside the nodes
Distributed Code

double sumY2 = new MRTask2(){
    double map(double x){
        return x*x;
    }
    double reduce(double x, double y){
        return x + y;
    }
}.doAll(vec);
Demo

GLM
CTR Prediction Contest

- Kaggle contest - clickthrough rate prediction
- Data
  - 11 days worth of clickthrough data from Avazu
  - ~ 8GB, ~ 44 million rows
  - Mostly categoricals
- Large number of features (predictors), good fit for linear models
Linear Regression

- Least Squares Fit
Logistic Regression

- Least Squares Fit
Logistic Regression

- GLM Fit
Generalized Linear Modelling

- Solved by iterative reweighted least squares
- Computation in two parts
  - Compute $X^TX$
  - Compute inverse of $X^TX$ (Cholesky Decomposition)
- Assumption
  - Number of rows >> number of cols
  - (use strong rules to filter out inactive columns)
- Complexity
  - $N_{rows} \times \frac{N_{cols2}}{N^3P} + \frac{N_{cols3}}{P}$
Generalized Linear Modelling

- Solved by iterative reweighted least squares
- Computation in two parts
  - Compute $X^T X$ [Distributed]
  - Compute inverse of $X^T X$ (Cholesky Decomposition)
- Assumption
  - Number of rows >> number of cols
  - (use strong rules to filter out inactive columns)
- Complexity
  - $N_{rows} \times \frac{N_{cols^2}}{N \times P} + \frac{N_{cols^3}}{P}$
Random Forest
How Big is Big?

- Data set size is relative
  - Does the data fit in one machine’s RAM
  - Does the data fit in one machine’s disk
  - Does the data fit in several machine’s RAM
  - Does the data fit in several machine’s disk
Why so Random?

- Introducing
  - Random Forest
  - Bagging
  - Out of bag error estimate
  - Confusion matrix

Consider a supervised learning problem with a simple data set with two classes and two features $x$ in $[1,4]$ and $y$ in $[5,8]$.

We can build a classification tree to predict new observations.
Classification Trees

- Classification trees often overfit the data
Random Forest

- Overfitting is avoided by building multiple randomized and far less precise (partial) trees
  - All these trees in fact underfit
- Result is obtained by a vote over the ensemble of the decision trees
  - Different voting strategies possible
Random Forest

- Each tree sees a different part of the training set and captures the information it contains.
Random Forest

- Each tree sees a different random selection of the training set (without replacement)

- At each split, a random subset of features is selected over which the decision should maximize gain
  - Gini Impurity
  - Information gain
Random Forest

- Each tree sees a different random selection of the training set (without replacement)
  - Bagging
- At each split, a random subset of features is selected over which the decision should maximize gain
  - Gini Impurity
  - Information gain
Random Forest

- Each tree sees a different random selection of the training set (without replacement)
  - Bagging
- At each split, a random subset of features is selected over which the decision should maximize gain

![Decision tree diagram]

- Gini Impurity $I_G(f) = \sum_{i=1}^{m} f_i(1 - f_i) = \sum_{i=1}^{m} (f_i - f_i^2) = \sum_{i=1}^{m} f_i - \sum_{i=1}^{m} f_i^2 = 1 - \sum_{i=1}^{m} f_i^2$
- Information gain
Random Forest

- Each tree sees a different random selection of the training set (without replacement)
  - Bagging

- At each split, a random subset of features is selected over which the decision should maximize gain

- Gini Impurity

- Information gain
  \[ I_E(f) = -\sum_{i=1}^{m} f_i \log_2 f_i \]
Validating the trees

- We can exploit the fact that each tree sees only a subset of the training data.
- Each tree in the forest is validated on the training data it has never seen.
Validating the trees

- We can exploit the fact that each tree sees only a subset of the training data.
- Each tree in the forest is validated on the training data it has never seen.
Validating the trees

- We can exploit the fact that each tree sees only a subset of the training data.
- Each tree in the forest is validated on the training data it has never seen.

Data used to construct the tree.
Validating the trees

- We can exploit the fact that each tree sees only a subset of the training data.
- Each tree in the forest is validated on the training data it has never seen.
Validating the trees

- We can exploit the fact that each tree sees only a subset of the training data.
- Each tree in the forest is validated on the training data it has never seen.

Errors (Out of Bag Error)
Validating the Forest

- Confusion Matrix is build for the forest and training data
  - During a vote, trees trained on the current row are ignored

<table>
<thead>
<tr>
<th>actual/assigned</th>
<th>Red</th>
<th>Green</th>
</tr>
</thead>
<tbody>
<tr>
<td>Red</td>
<td>15</td>
<td>5</td>
</tr>
<tr>
<td>Green</td>
<td>1</td>
<td>10</td>
</tr>
</tbody>
</table>
Distributing and Parallelizing

- How do we sample?
- How do we select splits?
- How do we estimate OOBE?
Distributing and Parallelizing

- How do we sample?
- How do we select splits?
- How do we estimate OOBE?
- When random data sample fits in memory, RF building parallelize extremely well
  - Parallel tree building is trivial
  - Validation requires trees to be collocated with data
    - Moving trees to data
    - (large training datasets can produce huge trees!)
Random Forest in H2O

- Trees must be built in parallel over randomized data samples
- To calculate gains, feature sets must be sorted at each split
Random Forest in H2O

- Trees must be built in parallel over randomized data samples
  - H2O reads data and distributes them over the nodes
  - Each node builds trees in parallel on a sample of the data that fits locally
- To calculate gains, feature sets must be sorted at each split
Random Forest in H2O

- Trees must be built in parallel over randomized data samples
- To calculate gains, feature sets must be sorted at each split
  - the values are discretized -> instead of sorting features are represented as arrays of their cardinality
  - `{ (2, red), (3.4, red), (5, green), (6.1, green) }` becomes `{ (1, red), (2, red), (3, green), (4, green) }`
- But trees can be very large (~100k splits)
Random Forest in H2O

- Trees must be built in parallel over randomized data samples
- To calculate gains, feature sets must be sorted at each split
  - the values are discretized -> instead of sorting features are represented as arrays of their cardinality
  - \{ (2, red), (3.4, red), (5, green), (6.1, green) \} becomes \{ (1, red), (2, red), (3, green), (4, green) \}
- But trees can be very large (~100k splits)
Lessons Learned

- Java Random is not really random
  - Small seeds give very bad random sequences resulting in poor RF performance
    - And we of course started with a deterministic seed of 42:)
  - But determinism is important for debugging
- Linux kernel drops TCP connections silently when under stress
  - Sender opens connection, sends, closes w/o exceptions, but receiver never sees the data
  - Need to recycle TCP connections and use TCP reliable delay
- Good Diagnostics to detect hardware issues is needed
  - Specific UDP packet drops with 100% chance
Demo
Continued
Q & A
Thank you
Please evaluate this talk via the mobile app!

Engage