Scalable Data Science and Deep Learning with H2O

Arno Candel, PhD
Chief Architect, H2O.ai
Who Am I?

Arno Candel
Chief Architect,
Physicist & Hacker at H2O.ai

PhD Physics, ETH Zurich 2005
10+ yrs Supercomputing (HPC)
6 yrs at SLAC (Stanford Lin. Accel.)
3.5 yrs Machine Learning
1.5 yrs at H2O.ai

Fortune Magazine
Big Data All Star 2014

Follow me @ArnoCandel
Outline

• Introduction
• H2O Deep Learning Architecture
• Live Demos:
  Flow GUI - Airline Logistic Regression
  Scoring Engine - Million Songs Classification
  R - MNIST Unsupervised Anomaly Detection
  Flow GUI - Higgs Boson Classification
  Sparkling Water - Chicago Crime Prediction
  iPython - CitiBike Demand Prediction
• Outlook
| **In-Memory ML** | Memory-Efficient Data Structures  
|                | Cutting-Edge Algorithms |
| **Distributed** | Use all your Data (No Sampling)  
|                | Accuracy with Speed and Scale |
| **Open Source** | Ownership of Methods - Apache V2  
|                | Easy to Deploy: Bare, Hadoop, Spark, etc. |
| **APIs**       | Java, Scala, R, Python, JavaScript, JSON  
|                | NanoFast Scoring Engine (POJO) |
Team@H2O.ai

GitHub

25,000 commits / 3yrs

h2o = fast statistical, machine learning & math runtime for bigdata

H2O World Conference 2014

Scientific Advisory Council

Stephen Boyd
Professor of EE Engineering, Stanford

Rob Tibshirani
Professor of Health Research and Policy, and Statistics, Stanford

Trevor Hastie
Professor of Statistics, Stanford
Community & Install Base

ML is the new SQL
Prediction is the new Search

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Companies</strong></td>
<td>103</td>
<td>634</td>
<td>2789</td>
</tr>
<tr>
<td><strong>Users</strong></td>
<td>463</td>
<td>2,887</td>
<td>13,237</td>
</tr>
</tbody>
</table>
Accuracy with Speed and Scale

Fast Modeling Engine

- Distributed In-Memory
- Map Reduce/Fork Join
- Columnar Compression
- Matrix Factorization
- Feature Engineering
- Munging
- Supervised Clustering
- Unsupervised Clustering
- Classification Regression
- GLM, Deep Learning
- K-Means, PCA, NB, Cox
- Random Forest / GBM Ensembles

all in-house code

Nano Fast Java Scoring Engines (POJO code generation)
Actual Customer Use Cases

- Real-time marketing (H2O is 10x faster than anything else)
- Ad Optimization (200% CPA Lift with H2O)
- P2B Model Factory (60k models, 15x faster with H2O than before)
- Fraud Detection (11% higher accuracy with H2O Deep Learning - saves millions)

...and many large insurance and financial services companies!
Today’s live demos will be run on yesterday’s nightly build of the next-gen product (alpha, in QA)!
Demo: GLM (Elastic Net) on Airlines

8 nodes on EC2: all cores active. Trains on 116M rows in 17 seconds!

Eastern Airlines: Always on time (No schedule!)
POJO Scoring Engine

Example: First GBM tree

```java
class gbm_model.Tree_0.class_0 {
    static final double score0(double[] data) {
        double pred = (data[0] / C2 + 0) < 45.738606f
            ? (data[1] / C3 + 0) < 45.384472f
                ? (data[2] / C4 + 0) < 44.95825f
                    ? (data[3] / C5 + 0) < 31.05989f
                        ? -0.052555863f : -0.09436895f
                    : -0.08649164f
                : -0.08649144f
            : 332.7656f
                : (data[4] / C5) < 42.09246f
                    ? -0.09416467f : -0.61170127f
            : 24.39137f
        : 44.95825f
            : (data[5] / C4 + 0) < 380.5273f
                : (data[6] / C5 + 0) < 31.617665f
                    : -0.04733852f : -0.07547601f
                : 33.312543f : -0.0566728065f : 6.66485412f
            : 384.6635f
                : (data[7] / C5) < 42.09246f
                    : -0.09416467f : -0.61170127f
                : 24.39137f
        : 44.95825f
            : (data[8] / C3 + 0) < 49.839999f
                : (data[9] / C4 + 0) < 69.74487f
                    : 0.896223487f : 0.086527505f
                : 78.71591f
                    : (data[10] / C5) < 37.46823f : 0.02314875f : -0.056622514f
                : 52.192627f : -0.04426976f : 3.6030795
            : 44.831062f
        : 44.95825f
            : (data[11] / C2 + 0) < 3.254776f
                : 0.66447134f : 0.81897329f
                : (data[12] / C6 + 0) < 0.10146823f : 0.06370063f
            : 22.332417f
                : (data[13] / C9 + 0) < 0.08623718f : 0.14365091f : 0.11135427f
            : (data[14] / C11 + 0) < 798.9386f : 0.87824522f : 0.1226283f)
    }

    return pred;
}
```

Note:
- no heap allocation,
- pure decision-making

Fast and easy path to production (batch or real-time)!
H2O Deep Learning

Multi-layer feed-forward Neural Network
Trained with back-propagation (SGD, AdaDelta)

+ distributed processing for big data
  (fine-grain in-memory MapReduce on distributed data)

+ multi-threaded speedup
  (async fork/join worker threads operate at FORTRAN speeds)

+ smart algorithms for fast & accurate results
  (automatic standardization, one-hot encoding of categoricals, missing value imputation, weight & bias initialization, adaptive learning rate, momentum, dropout/l1/L2 regularization, grid search, N-fold cross-validation, checkpointing, load balancing, auto-tuning, model averaging, etc.)

= powerful tool for (un)supervised machine learning on real-world data
H2O Deep Learning Architecture

Main Loop:

map:
- each node trains a copy of the weights and biases with (some or all of) its local data with asynchronous F/J threads
- auto-tuned (default) or user-specified number of rows per MapReduce iteration

reduce:
- model averaging: average weights and biases from all nodes, speedup is at least \( \text{#nodes/log(#rows)} \)

Keep iterating over the data (“epochs”), score at user-given times
H2O Deep Learning beats MNIST

Handwritten digits: $28^2=784$ gray-scale pixel values

---

```r
> library(h2o)
> h2oServer <- h2o.init(ip="mr-0xd1", port=53322)
> train_hex <- h2o.importFile(h2oServer, "mnist/train.csv.gz")
> test_hex <- h2o.importFile(h2oServer, "mnist/test.csv.gz")
> train_hex[,785] <- as.factor(train_hex[,785])
> test_hex[,785] <- as.factor(test_hex[,785])
> dl_model <- h2o.deeplearning(x=c(1:784), y=785, training_frame=train_hex, l1=1e-5,
                              activation="RectifierWithDropout", input_dropout_ratio=0.2,
                              classification_stop=-1, hidden=c(1024, 1024, 2048), epochs=8000)

> h2o.confusionMatrix(dl_model, test_hex)
Confusion Matrix - (vertical: actual; across: predicted):

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>Error</th>
<th>Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>974</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0.00612</td>
<td>6 / 980</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1135</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.00088</td>
<td>1 / 1,135</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1028</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>0.00388</td>
<td>4 / 1,032</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1003</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0.00693</td>
<td>7 / 1,010</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>971</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0.01120</td>
<td>6 / 982</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>882</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0.01121</td>
<td>11 / 982</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>949</td>
<td>0</td>
<td>0</td>
<td>0.00939</td>
<td>9 / 958</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>2</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1019</td>
<td>0</td>
<td>0.00875</td>
<td>9 / 1,028</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>2</td>
<td>960</td>
<td>0.011437</td>
<td>14 / 974</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>3</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0.01189</td>
<td>12 / 1,000</td>
<td></td>
</tr>
</tbody>
</table>
Totals 981 1142 1038 1013 977 891 956 1031 964 1007 0.00830 = 83 / 10,000

Standard 60k/10k data
- No distortions
- No convolutions
- No unsupervised training
- No ensemble

10 hours on 10 16-core servers

World-record!
0.83% test set error

---

A Classic Benchmark!

---

Table 1: Classification error rate comparison: DBN vs. DCN

<table>
<thead>
<tr>
<th></th>
<th>DBN [3] (Hinton’s)</th>
<th>DBN (MSR’s)</th>
<th>DCN (Fine-tuning)</th>
<th>DCN (no Fine-tuning)</th>
<th>Shallow (D)CN (Fine-tuned single layer)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.20%</td>
<td>1.06%</td>
<td>0.83%</td>
<td>0.95%</td>
<td>1.10%</td>
<td></td>
</tr>
</tbody>
</table>
Unsupervised Anomaly Detection

```
# 1) LEARN WHAT'S NORMAL WITH UNSUPERVISED AUTOENCODER
ae_model <- h2o.deeplearning(x=predictors,
    training_frame=train_hex,
    activation="Tanh",
    autoencoder=T,
    hidden=c(50),
    l1=1e-5,
    ignore_const_cols=F,
    epochs=1)

# 2) DETECT OUTLIERS
test_rec_error <- as.data.frame(h2o.anomaly(ae_model, test_hex))
```

<table>
<thead>
<tr>
<th>layer</th>
<th>units</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>784</td>
<td>Input</td>
</tr>
<tr>
<td>2</td>
<td>50</td>
<td>Tanh</td>
</tr>
<tr>
<td>3</td>
<td>784</td>
<td>Tanh</td>
</tr>
</tbody>
</table>

Learns Compressed Identity Function

The good

The bad

The ugly

Try it yourself!
Higgs Boson - Classification Problem

Large Hadron Collider: Largest experiment of mankind!
$13+ billion, 16.8$ miles long, $120$ MegaWatts, $-456F$, $1$PB/day, etc.
Higgs boson discovery (July ’12) led to $2013$ Nobel prize!

Images courtesy CERN / LHC

Higgs vs Background

HIGGS UCI Dataset:

- 21 low-level features AND
- 7 high-level derived features (physics formulae)

Train: 10M rows, Valid: 500k, Test: 500k rows
Deep Learning for Higgs Detection?

Former baseline for AUC: 0.733 and 0.816

<table>
<thead>
<tr>
<th>H2O Algorithm</th>
<th>low-level H2O AUC</th>
<th>all features H2O AUC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generalized Linear Model</td>
<td>0.596</td>
<td>0.684</td>
</tr>
<tr>
<td>Random Forest</td>
<td>0.764 add derived features</td>
<td>0.840</td>
</tr>
<tr>
<td>Gradient Boosted Trees</td>
<td>0.753</td>
<td>0.839</td>
</tr>
<tr>
<td>Neural Net 1 hidden layer</td>
<td>0.760</td>
<td>0.830</td>
</tr>
<tr>
<td>H2O Deep Learning</td>
<td>?</td>
<td></td>
</tr>
</tbody>
</table>
H2O Deep Learning Higgs Demo

Deep DL model on 21 low-level features (no physics formula based extra features)

10 nodes:
AUC = 0.84 after 40 mins
AUC = 0.87+ after 8 hours

Deep Learning learns Physics!
H2O Deep Learning in the News

http://www.datanami.com/2015/05/07/what-police-can-learn-from-deep-learning/

May 7, 2015
What Police Can Learn from Deep Learning
Alex Woodie

Police departments are increasingly turning to predictive analytics to help them fight crime, and the early returns are positive, with double-digit drops in crime rates reported in many cities. So what's next? According to big data analytics experts, police departments could spend their time and money more effectively by giving deep learning algorithms a role in the dispatch room.

Alex, Michal, et al.

http://www.slideshare.net/0xdata/crime-deeplearningkey

http://www.slideshare.net/0xdata/crime-deeplearningkey
Weather + Census + Crime Data

Data munging

Spark SQL join

Split table

Collect models metrics

Evaluate models and score new crimes

Deep Learning

GBM
Integration with Spark Ecosystem

Spark RDD and H2O Frames share same JVM
Sparkling Water Demo

Instructions at h2o.ai/download

```scala
/**
 * To start Sparkling Water please type

cd path/to/sparkling/water
export SPARK_HOME="your/spark-1.2.0-installation"
export MASTER="local-cluster[3,2,4096]"

bin/sparkling-shell --conf spark.executor.memory=3G
*/

// Prepare environment
//
import hex.deeplearning.DeepLearningModel
import hex.tree.gbm.GBMModel
import hex.tree.gbm.GBMModel.GBMParameters.Family
import hex.{Model, ModelMetricsBinomial}
import org.apache.spark.SparkFiles
import org.apache.spark.examples.h2o.DemoUtils._
import org.apache.spark.examples.h2o.{Crime, RefineDateColumn}
import org.apache.spark.h2o._
import org.apache.spark.sql._

// SQL support
implicit val sqlContext = new SQLContext(sc)
import sqlContext._
```
Parse & Munge with H2O, Convert to RDD

```scala
// Start H2O services
implicit val h2oContext = new H2OContext(sc).start()
import h2oContext._

// H2O Data loader using H2O API

def loadData(datafile: String): DataFrame = new DataFrame(new java.net.URI(datafile))

// Loader for weather data

def createWeatherTable(datafile: String): DataFrame = {
    val table = loadData(datafile)
    // Remove first column since we do not need it
    table.remove(0).remove()
    table.update(null)
    table
}
```

H2O Parser: Robust & Fast

Simple Column Selection
Parse & Munge with H2O, Convert to RDD

Munging: Date Manipulations

Conversion to SchemaRDD
Join RDDs with SQL, Convert to H2O

Spark SQL Query Execution

```scala
val crimeWeather = sql("""
SELECT
  a.IUCR, a.Primary_Type, a.Location_Description, a.Community_Area, a.District,
  b.minTemp, b.maxTemp, b.meanTemp,
  c.PERCENT_AGED_UNDER_18_OR_OVER_64, c.PER_CAPITA_INCOME, c.HARDSHIP_INDEX,
  c.PERCENT_OF_HOUSING_CROWDED, c.PERCENT_HOUSEHOLDS_BELOW_POVERTY,
  c.PERCENT_AGED_16__UNEMPLOYED, c.PERCENT_AGED_25__WITHOUT_HIGH_SCHOOL_DIPLOMA
FROM chicagoCrime a
JOIN chicagoWeather b
ON a.Year = b.year AND a.Month = b.month AND a.Day = b.day
JOIN chicagoCensus c
ON a.Community_Area = c.Community_Area_Number"").stripMargin
```

Convert back to H2OFrame

```scala
h2oFrame = crimeWeather
```

Split final data table

```scala
val keys = Array[String]("train.hex", "test.hex")
val ratios = Array[Double](0.8, 0.2)
val frs = splitFrame(h2oFrame, keys, ratios)
val (train, test) = (frs(0), frs(1))
```
Build H2O Deep Learning Model

```scala
def DLModel(train: DataFrame, test: DataFrame, response: String)
  (implicit h2oContext: H2OContext): DeepLearningModel = {
    import h2oContext._
    import hex.deeplearning.DeepLearning

    val dlParams = new DeepLearningParameters()
    dlParams._train = train
    dlParams._valid = test
    dlParams._response_column = response
    dlParams._variable_importances = true
    // Create a job
    val dl = new DeepLearning(dlParams)
    val model = dl.trainModel.get
    model
  }

// // Build Deep Learning model
//
// val dlModel = DLModel(train, test, 'Arrest)
```

Train a H2O Deep Learning Model on Data obtained by Spark SQL Query

Predict whether Arrest will be made with AUC of 0.90+
Visualize Results with Flow

Using Flow to interactively plot
Arrest Rate (blue)
vs
Relative Occurrence (red)
per crime type.
R’s data.table now in H2O!

Package ‘data.table’

February 19, 2015

Version 1.9.4
Title Extension of data.frame
Author M Dowle, T Short, S Lianoglou, A Srinivasan with contributions from R Saporta, E Antonyan
Maintainer Matt Dowle <mdowle@mdowle.plus.com>

And here, it took about 2 minutes... The longest part was to extract those first and last observations. So far, it looks like data.table is just perfect to deal with those “large” datasets.

Pinned Tweet
Matt Dowle @MattDowle · Mar 15
Excited to be starting full time at @h2oai in Mountain View tomorrow.
# Explore a typical Data Science workflow with H2O and Python

# Goal: assist the manager of CitiBike of NYC to load-balance the bicycles across the CitiBike network of stations, by predicting the number of bike trips taken from the station every day. Use 10 million rows of historical data, and eventually add weather data.
Group-By Aggregation

In [5]:
# Now do a monster Group-By. Count bike starts per-station per-day. Ends up
# with about 340 stations times 400 days (140,000 rows). This is what we want
# to predict.
by = ["Days","start station name"]
aggregates = {"bikes": ["count", 0, "all"]}
bpd = h2o.group_by(data, cols=by, aggregates=aggregates) # Compute bikes-per-day
bpd.show()
bpd.describe()
bpd.dim()

Displaying 10 row(s):

<table>
<thead>
<tr>
<th>Row ID</th>
<th>Days</th>
<th>start station name</th>
<th>bikes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>15981.0</td>
<td>[u'Fulton St &amp; William St']</td>
<td>89.0</td>
</tr>
<tr>
<td>2</td>
<td>16007.0</td>
<td>[u'W 27 St &amp; 7 Ave']</td>
<td>203.0</td>
</tr>
<tr>
<td>3</td>
<td>15980.0</td>
<td>[u'9 Ave &amp; W 18 St']</td>
<td>137.0</td>
</tr>
<tr>
<td>4</td>
<td>15995.0</td>
<td>[u'Washington Ave &amp; Park Ave']</td>
<td>29.0</td>
</tr>
<tr>
<td>5</td>
<td>15979.0</td>
<td>[u'Bedford Ave &amp; S 9th St']</td>
<td>8.0</td>
</tr>
<tr>
<td>6</td>
<td>16005.0</td>
<td>[u'W 13 St &amp; 6 Ave']</td>
<td>138.0</td>
</tr>
<tr>
<td>7</td>
<td>15979.0</td>
<td>[u'11 Ave &amp; W 27 St']</td>
<td>139.0</td>
</tr>
<tr>
<td>8</td>
<td>15986.0</td>
<td>[u'Central Park S &amp; 6 Ave']</td>
<td>123.0</td>
</tr>
<tr>
<td>9</td>
<td>16004.0</td>
<td>[u'John St &amp; William St']</td>
<td>60.0</td>
</tr>
<tr>
<td>10</td>
<td>15989.0</td>
<td>[u'Allen St &amp; Hester St']</td>
<td>110.0</td>
</tr>
</tbody>
</table>

Rows: 10,450 Cols: 3
Model Building And Scoring

In [9]: # Split the data (into test & train), fit some models and predict on the holdout data
split_fit_predict(bpd)
# Here we see an r^2 of 0.91 for GBM, and 0.71 for GLM. This means given just
# the station, the month, and the day-of-week we can predict 90% of the
# variance of the bike-trip-starts.

Training data has 5 columns and 6247 rows, test has 3164 rows, holdout has 1039

- gbm Model Build Progress: [###############################] 100%
- drf Model Build Progress: [###############################] 100%
- glm Model Build Progress: [###############################] 100%
- deeplearning Model Build Progress: [###############################] 100%

<table>
<thead>
<tr>
<th>Model</th>
<th>R2 TRAIN</th>
<th>R2 TEST</th>
<th>R2 HOLDOUT</th>
<th>Model Training Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>GBM</td>
<td>0.997863069083</td>
<td>0.92453193079</td>
<td>0.9058312743</td>
<td>6.732</td>
</tr>
<tr>
<td>DRF</td>
<td>0.831504093401</td>
<td>0.786203336569</td>
<td>0.780234326364</td>
<td>5.628</td>
</tr>
<tr>
<td>GLM</td>
<td>0.860534716668</td>
<td>0.84755659058</td>
<td>0.833198032239</td>
<td>0.157</td>
</tr>
<tr>
<td>DL</td>
<td>0.97405671444</td>
<td>0.92066423657</td>
<td>0.911478042616</td>
<td>6.762</td>
</tr>
</tbody>
</table>

91% AUC baseline
Joining Bikes-Per-Day with Weather

```python
In [14]:
# Let's drop off the extra time columns to make a easy-to-handle dataset.
wthr4 = wthr3.drop("Year Local").drop("Month Local").drop("Day Local").drop("Hour Local").drop("msec")

In [15]:
# Also, most rain numbers are missing - lets assume those are zero rain days
rain = wthr4["Rain (mm)"]
rain[rain == None] = 0

In [16]:
# -------
# 6 - Join the weather data-per-day to the bike-starts-per-day
print "Merge Daily Weather with Bikes-Per-Day"
bpd_with_weather = bpd.merge(wthr4, allLeft=True, allRite=False)
bpd_with_weather.describe()
bpd_with_weather.show()

Merge Daily Weather with Bikes-Per-Day
Rows: 10,450 Cols: 10

Data Compression Summary:
bitset 0/1
signed byte -128..127
unsigned byte 0..255
1 byte floating point (e.g., 0.493..0.684)
short integers (2 bytes)
sparse doubles
dense double
```
Improved Models with Weather Data

In [17]: # 7 - Test/Train split again, model build again, this time with weather
split_fit_predict(bpd_with_weather)

Training data has 10 columns and 6284 rows, test has 3142 rows, holdout has 1012

<table>
<thead>
<tr>
<th>Model</th>
<th>R2 TRAIN</th>
<th>R2 TEST</th>
<th>R2 HOLDOUT</th>
<th>Model Training Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>GBM</td>
<td>0.997835338379</td>
<td>0.922614027893</td>
<td>0.929365255664</td>
<td>7.827</td>
</tr>
<tr>
<td>DRF</td>
<td>0.895968570144</td>
<td>0.811939811949</td>
<td>0.799997019098</td>
<td>8.916</td>
</tr>
<tr>
<td>GLM</td>
<td>0.843764627796</td>
<td>0.840765558156</td>
<td>0.841326120337</td>
<td>0.146</td>
</tr>
<tr>
<td>DL</td>
<td>0.96673892961</td>
<td>0.91917621918</td>
<td>0.924742944402</td>
<td>5.62</td>
</tr>
</tbody>
</table>

93% AUC after joining bike and weather data
More Info in H2O Booklets

https://leanpub.com/u/h2oai

http://learn.h2o.ai
Competitive Data Science

Mark Landry (joined H2O!) will hold a master class on May 19!

http://www.meetup.com/Silicon-Valley-Big-Data-Science/events/222303884/
Past Kaggle Starter Scripts

still ongoing!
Hyper-Parameter Tuning

```r
# Step 5 - GBM Hyper-Parameter Tuning with Random Search

models <- c()
for (i in 1:10) {
  rand_numtrees <- sample(1:50,1)  # 1 to 50 trees
  rand_max_depth <- sample(5:15,1)  # 5 to 15 max depth
  rand_min_rows <- sample(1:10,1)  # 1 to 10 min rows
  rand_learn_rate <- 0.025*sample(1:10,1)  # 0.025 to 0.25 learning rate
  model_name <- paste0("GBMModel_",i,
    ",_ntrees",rand_numtrees,
    ",_maxdepth",rand_max_depth,
    ",_minrows",rand_min_rows,
    ",_learnrate",rand_learn_rate
  )

  model <- h2o.gbm(x=predictors,
    y=response,
    training_frame=train_holdout.hex,
    validation_frame=valid_holdout.hex,
    destination_key=model_name,
    loss="multinomial",
    ntrees=rand_numtrees,
    max_depth=rand_max_depth,
    min_rows=rand_min_rows,
    learn_rate=rand_learn_rate
  )

  models <- c(models, model)
}
```

93 numerical features
9 output classes
62k training set rows
144k test set rows
Outlook - Algorithm Roadmap

- Ensembles (Erin LeDell et al.)
- Automatic Hyper-Parameter Tuning
- Convolutional Layers for Deep Learning
- Natural Language Processing: tf-idf, Word2Vec
- Generalized Low Rank Models
  - PCA, SVD, K-Means, Matrix Factorization
- Recommender Systems

And many more!

Public JIRAs - Join H2O!
Key Take-Aways

H2O is an open source predictive analytics platform for data scientists and business analysts who need scalable, fast and accurate machine learning.

H2O Deep Learning is ready to take your advanced analytics to the next level.

Try it on your data!

https://github.com/h2oai
H2O Google Group
http://h2o.ai
@h2oai

Thank You!
Questions?

Please remember to evaluate via the GOTO Guide App