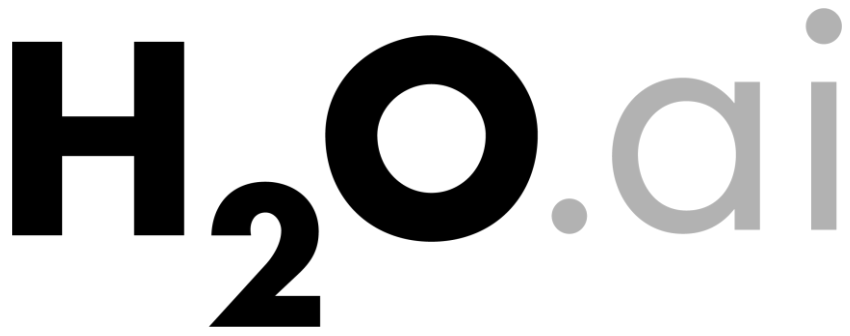
H_2O at Poznan R Meetup

Introduction to H_2O, IoT Use Cases and Deep Water



Jo-fai (Joe) Chow

Data Scientist

joe@h2o.ai

@matlabulous

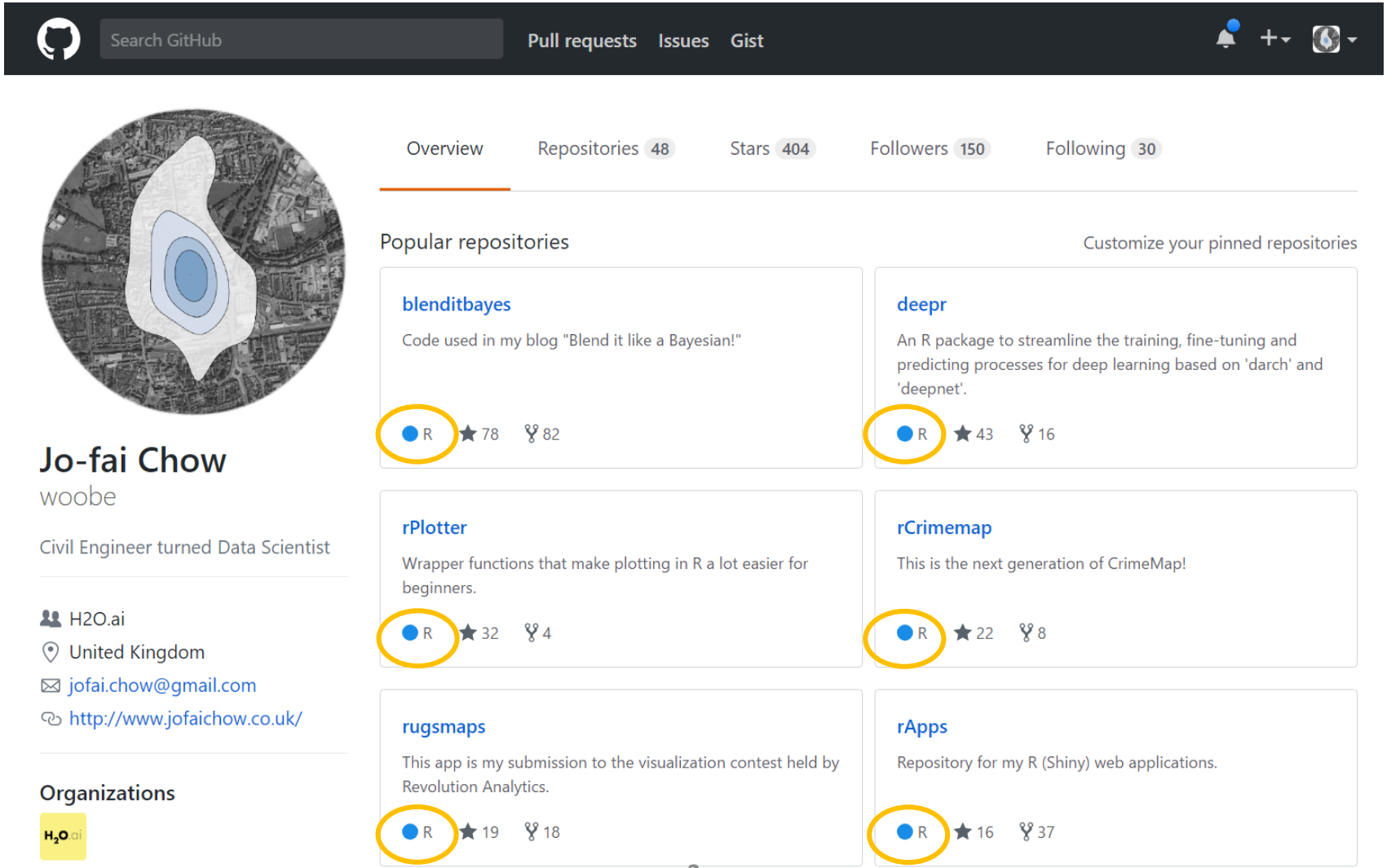
Poznan R

20th April, 2017

About Me

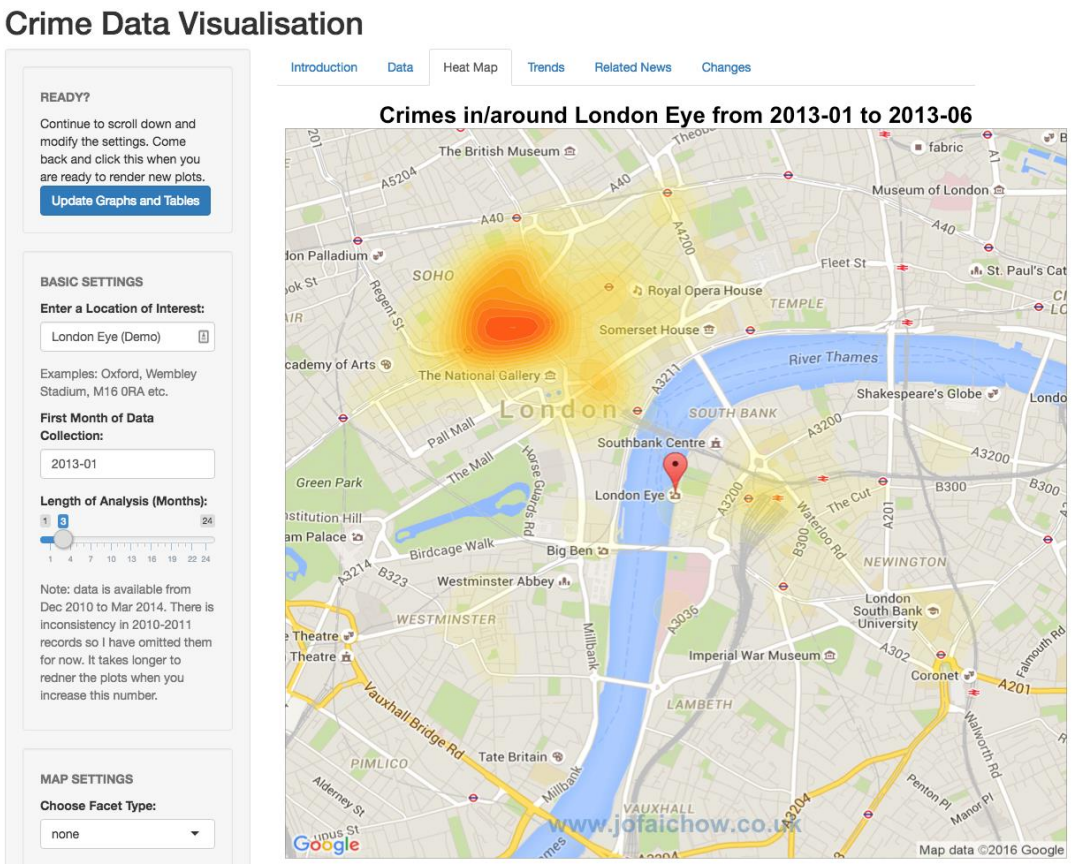
- Civil (Water) Engineer
  - 2010 – 2015
  - Consultant (UK)
    - Utilities
    - Asset Management
    - Constrained Optimization
  - Industrial PhD (UK)
    - Infrastructure Design Optimization
    - Machine Learning + Water Engineering
    - Discovered H_2O in 2014
- Data Scientist
  - 2015
  - Virgin Media (UK)
  - Domino Data Lab (Silicon Valley)
  - 2016 – Present
  - H_2O.ai (Silicon Valley)

About Me

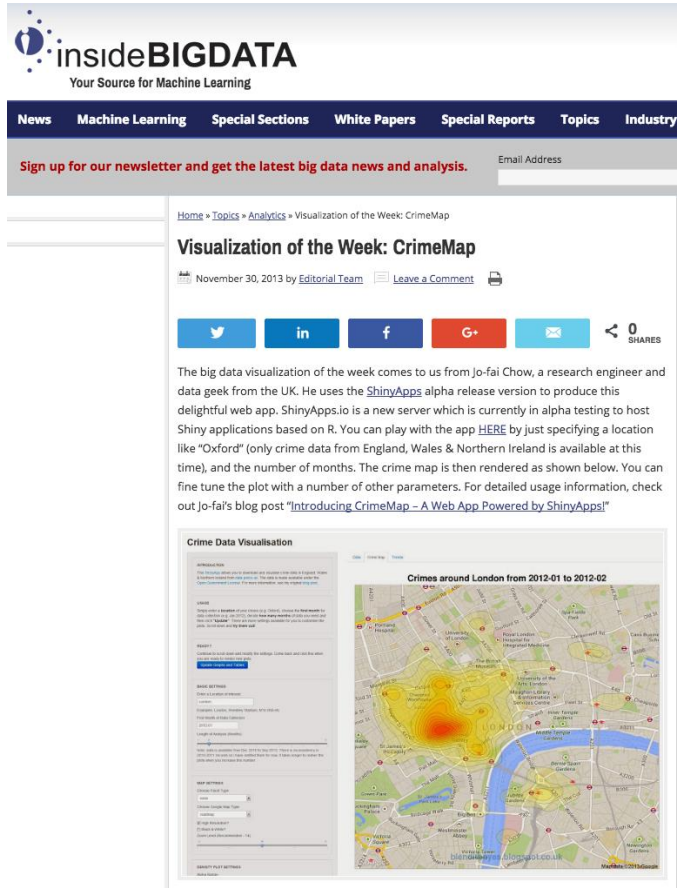


H2O.ai

Side Project #1 – Crime Data Visualization



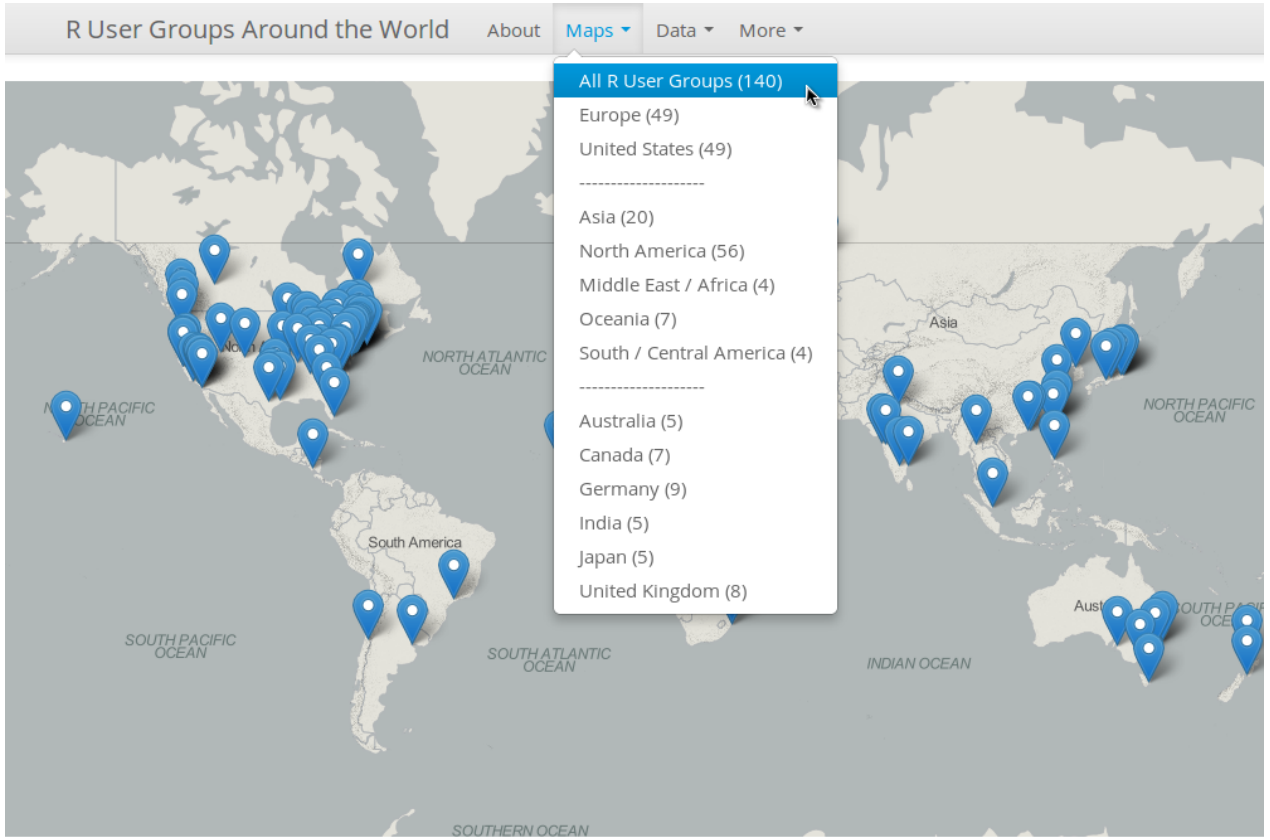
https://github.com/woobe/rApps/tree/master/crimemap

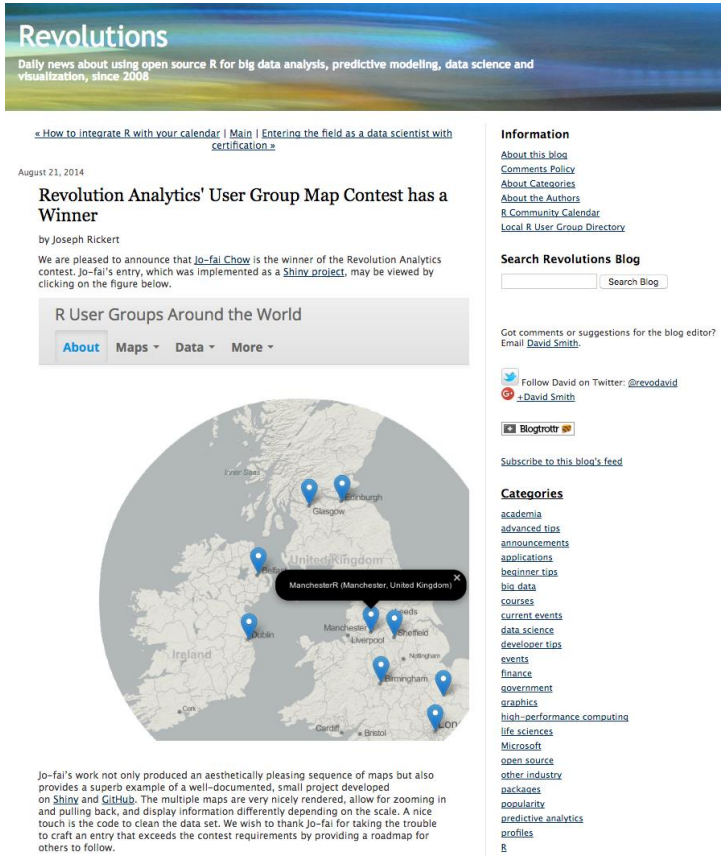


http://insidebigdata.com/2013/11/30/visualization-week-crimemap/

H2O.ai

Side Project #2 – Data Visualization Contest



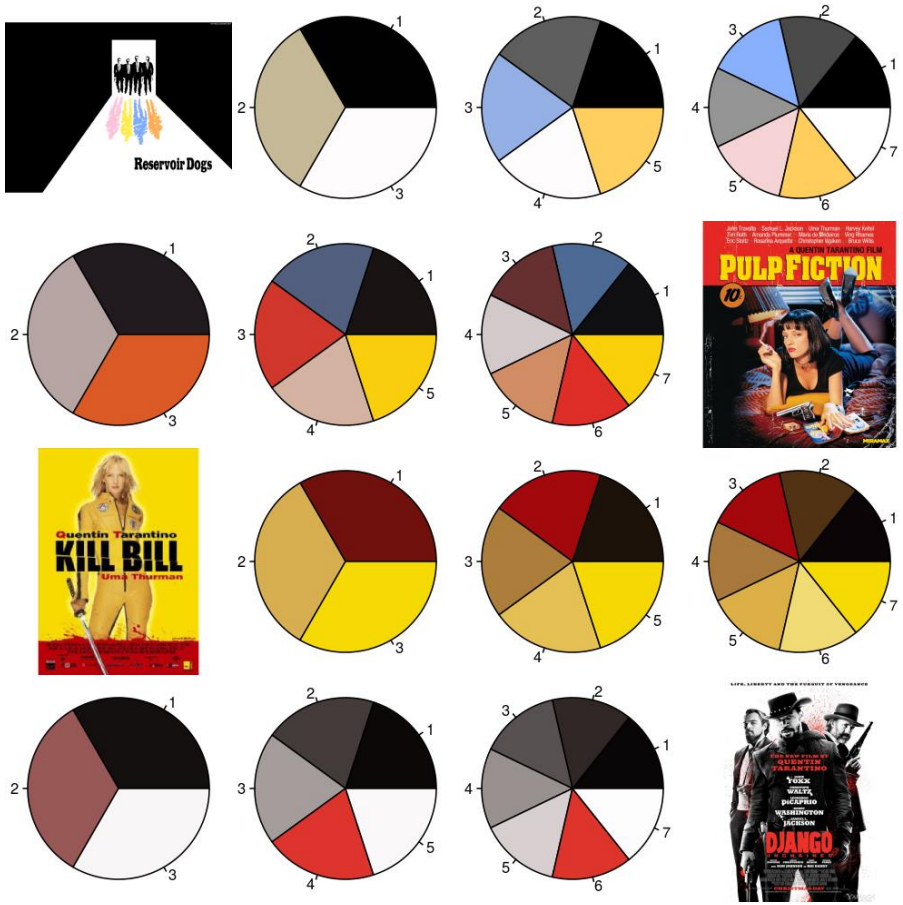


https://github.com/woobe/rugsmaps

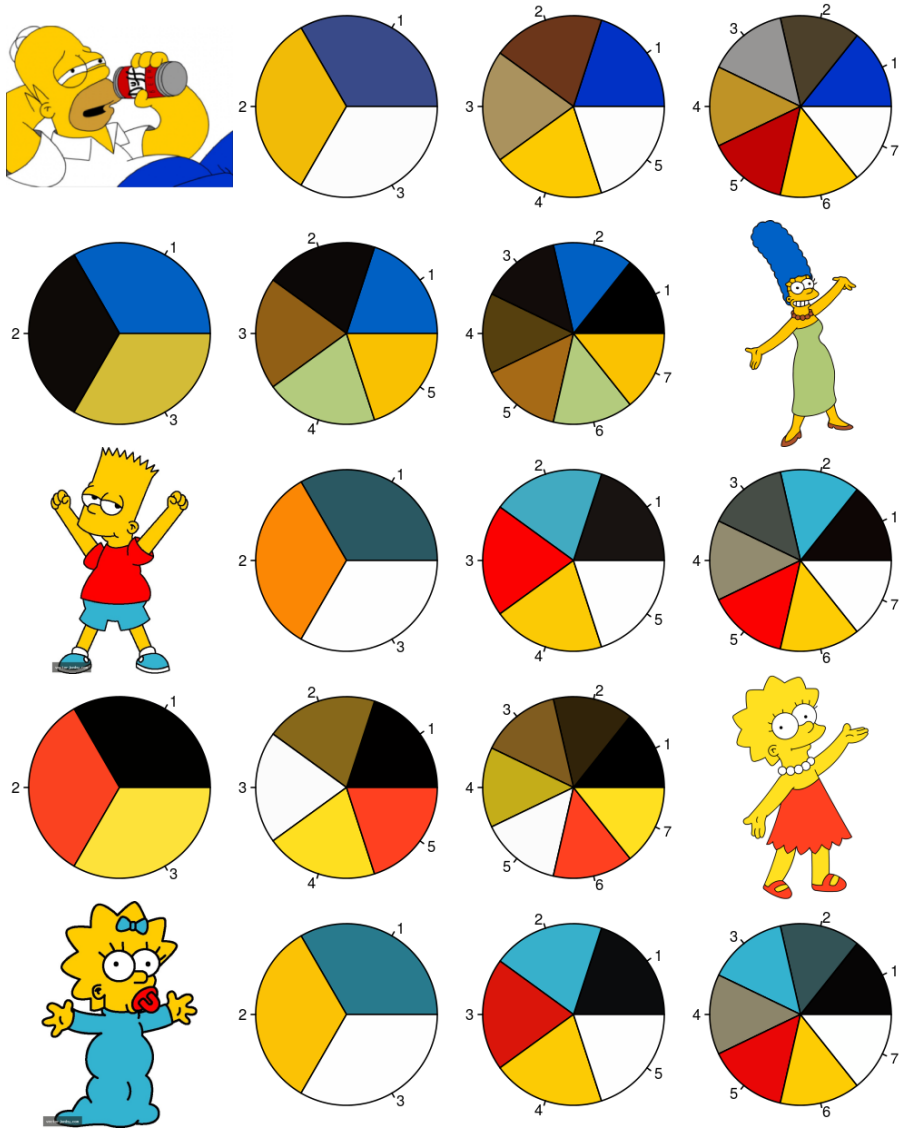
http://blog.revolutionanalytics.com/2014/08/winner-for-revolution-analytics-user-group-map-contest.html

H2O.ai

Side Project #3

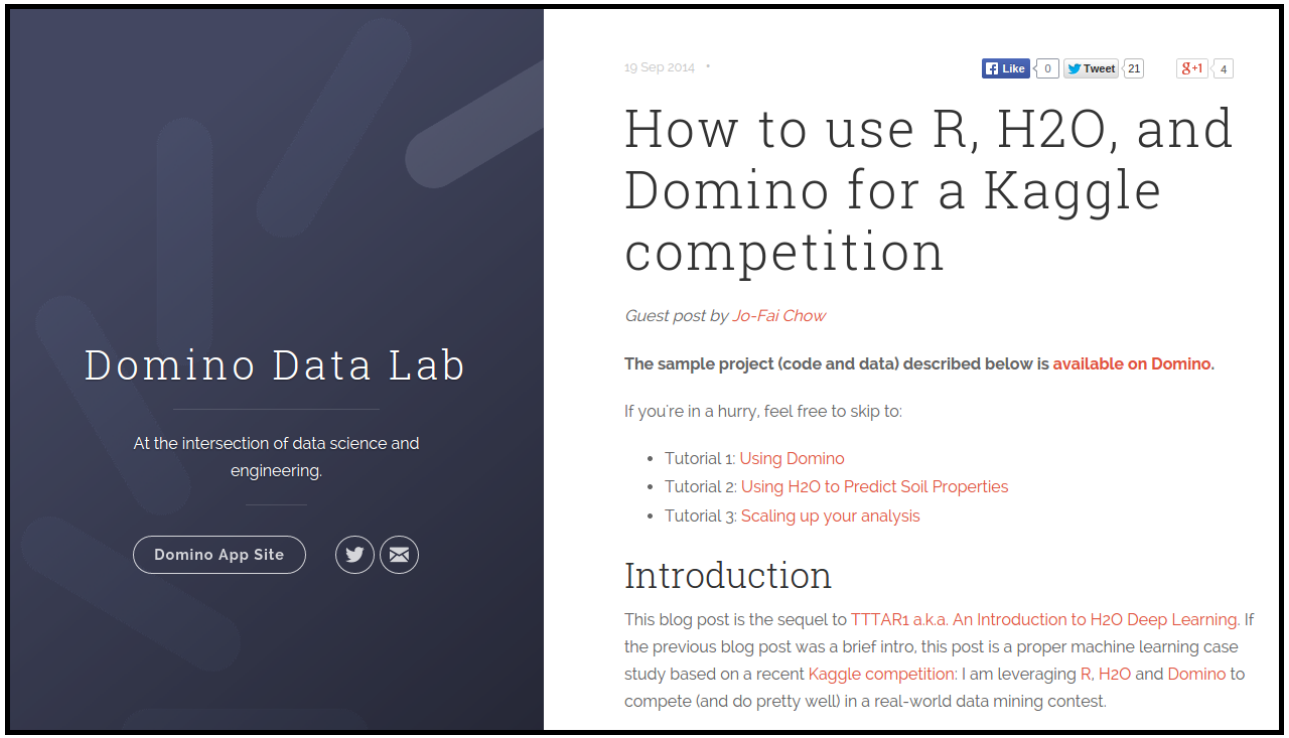


Developing R Packages for Fun
rPlotter (2014)



H2O.ai

Side Project #4 – Kaggle Blog Post



R + H_2O + Domino for Kaggle

Guest Blog Post for Domino & H_2O (2014)

- The Long Story
  - bit.ly/joe_kaggle_story

Agenda

- Introduction
  - Company
  - Machine Learning Platform
- IoT Use Cases
  - Predictive Maintenance
  - Outlier Detection
- Break (25 mins)
- Deep Water
  - Motivation / Benefits
  - Interfaces / Demo

H2O.ai

About H_2O.ai

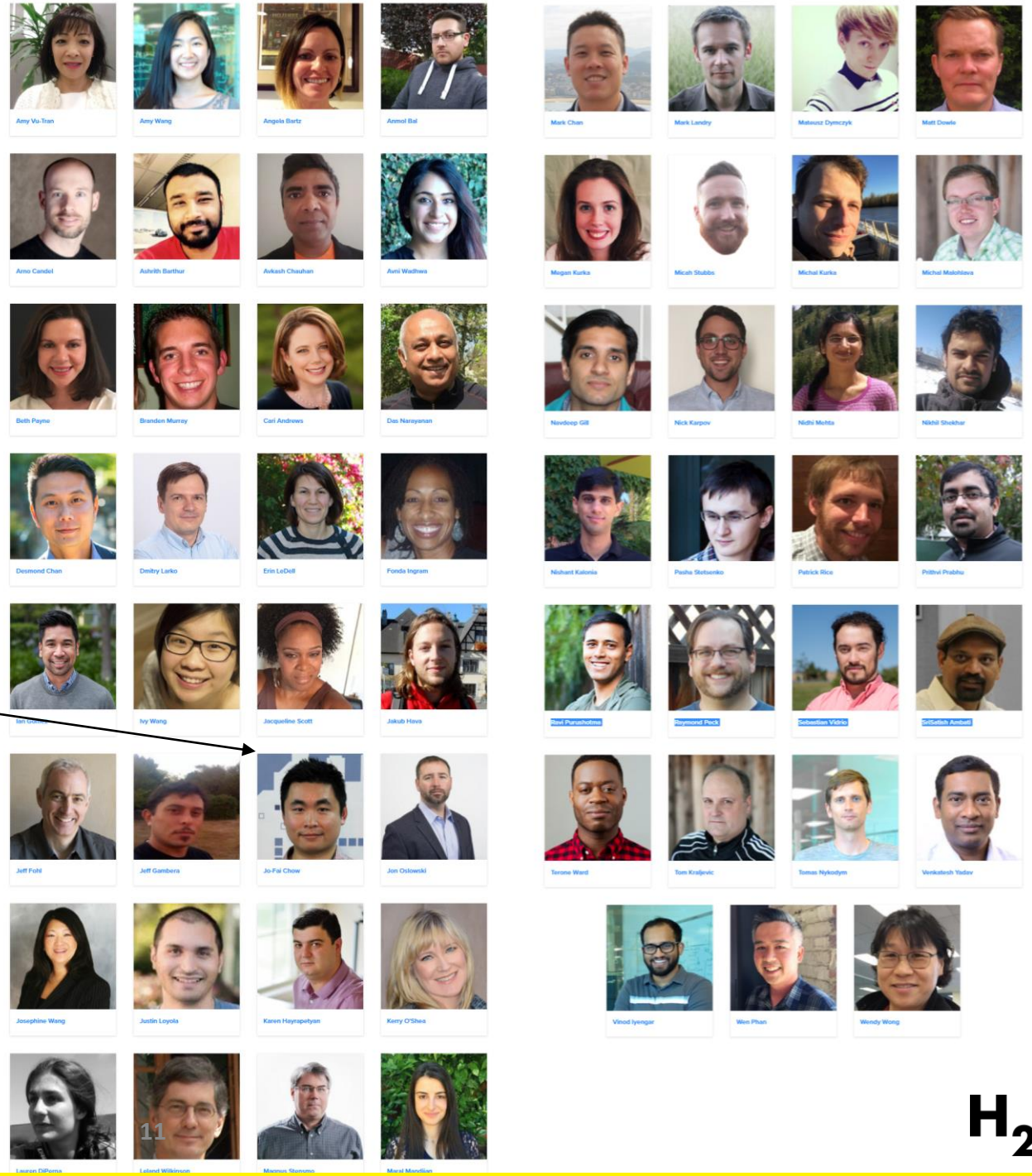
Company Overview

Founded	2011 Venture-backed, debuted in 2012
Products	• H_2O Open Source In-Memory AI Prediction Engine • Sparkling Water • Steam
Mission	Operationalize Data Science, and provide a platform for users to build beautiful data products
Team	70 employees • Distributed Systems Engineers doing Machine Learning • World-class visualization designers
Headquarters	Mountain View, CA




Our Team

Joe



H2O.ai

Scientific Advisory Council

Dr. Trevor Hastie

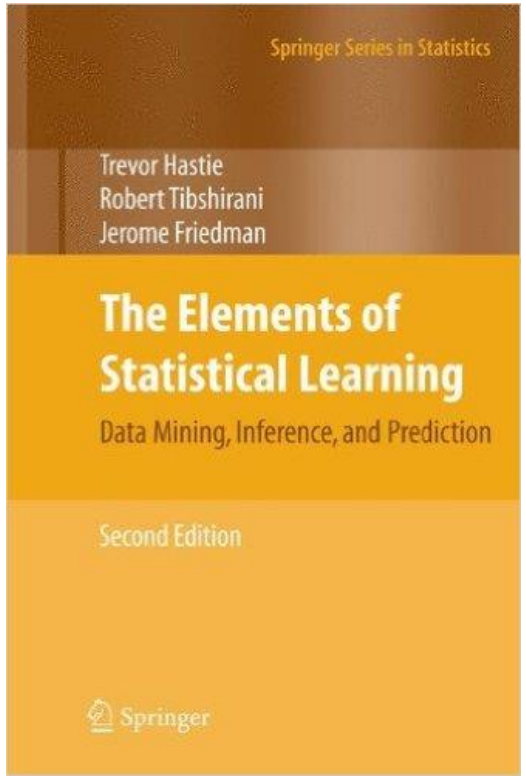
- John A. Overdeck Professor of Mathematics, Stanford University
- PhD in Statistics, Stanford University
- Co-author, *The Elements of Statistical Learning: Prediction, Inference and Data Mining*
- Co-author with John Chambers, *Statistical Models in S*
- Co-author, *Generalized Additive Models*

Dr. Robert Tibshirani

- Professor of Statistics and Health Research and Policy, Stanford University
- PhD in Statistics, Stanford University
- Co-author, *The Elements of Statistical Learning: Prediction, Inference and Data Mining*
- Author, *Regression Shrinkage and Selection via the Lasso*
- Co-author, *An Introduction to the Bootstrap*

Dr. Steven Boyd

- Professor of Electrical Engineering and Computer Science, Stanford University
- PhD in Electrical Engineering and Computer Science, UC Berkeley
- Co-author, *Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers*
- Co-author, *Linear Matrix Inequalities in System and Control Theory*
- Co-author, *Convex Optimization*



H2O.ai

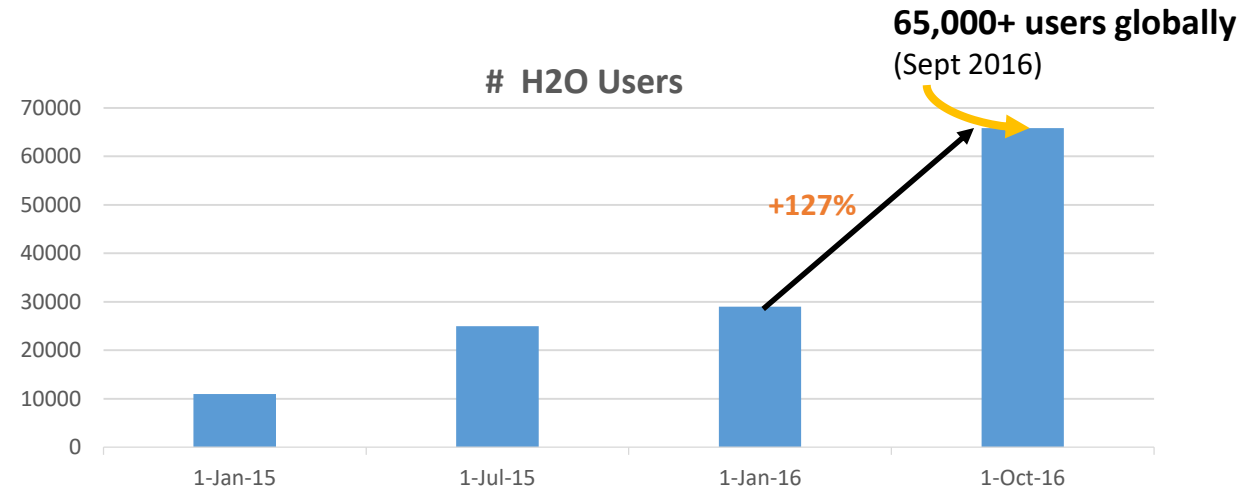
wenphan
@wenphan

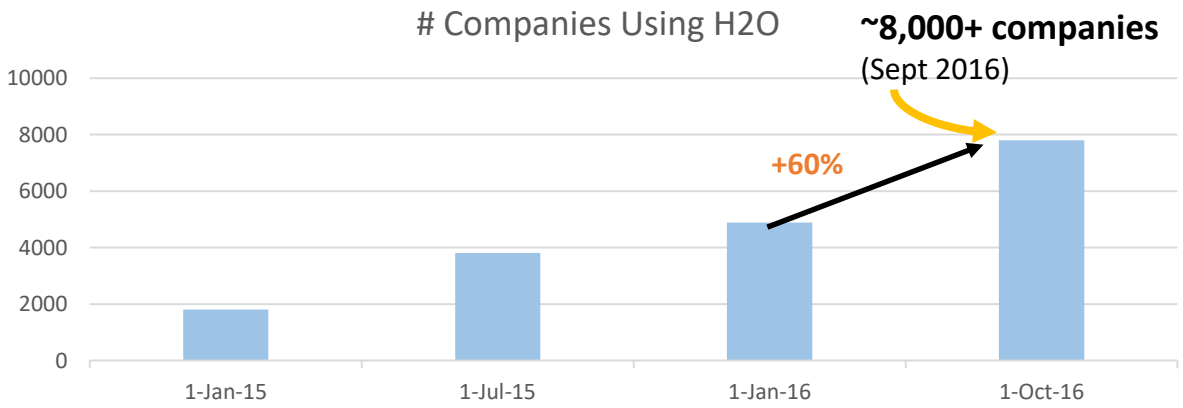
Following

So much brain power in one place: @ArnoCandel and Stanford profs. Boyd, Tibs, and Hastie. Hacking algos at @h2oai HQ

H_2O Community Growth

Tremendous Momentum Globally





Large User Circle

- 65,000+ users from ~8,000 companies in 140 countries. Top 5 from:

1. United States
2. India
3. Japan
4. Germany
5. United Kingdom

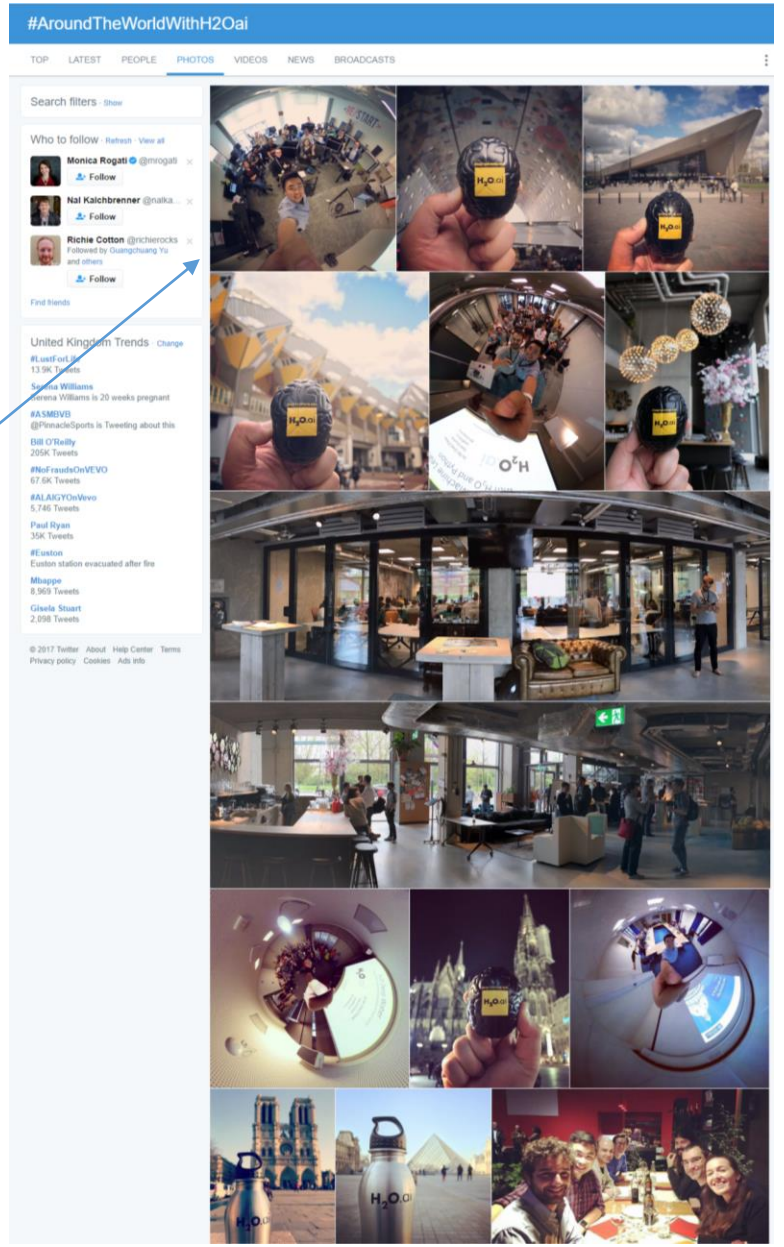
* DATA FROM GOOGLE ANALYTICS EMBEDDED IN THE END USER PRODUCT

H2O.ai

#AroundTheWorldWithH2Oai

Jo-fai (Joe) Chow @matlabulous · Apr 12
Helping #Refugees by teaching them basic #machinelearning skills @h2oai workshop @Restart_Network #Rotterdam
#AroundTheWorldWithH2Oai pic.twitter.com/shJemc3gsv

9 14



H2O.ai

H_2O for Kaggle Competitions

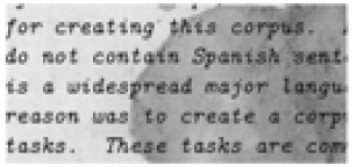
CIFAR-10 Competition Winners: Interviews with Dr. Ben Graham, Phil Culliton, & Zygmunt Zając

Triskelion | 01.02.2015

READ MORE

"I did really like H2O's deep learning implementation in R, though - the interface was great, the back end extremely easy to understand, and it was scalable and flexible. Definitely a tool I'll be going back to."

Kaggle challenge 2nd place winner Colin Priest



Completed • Knowledge • 161 teams

Denoising Dirty Documents

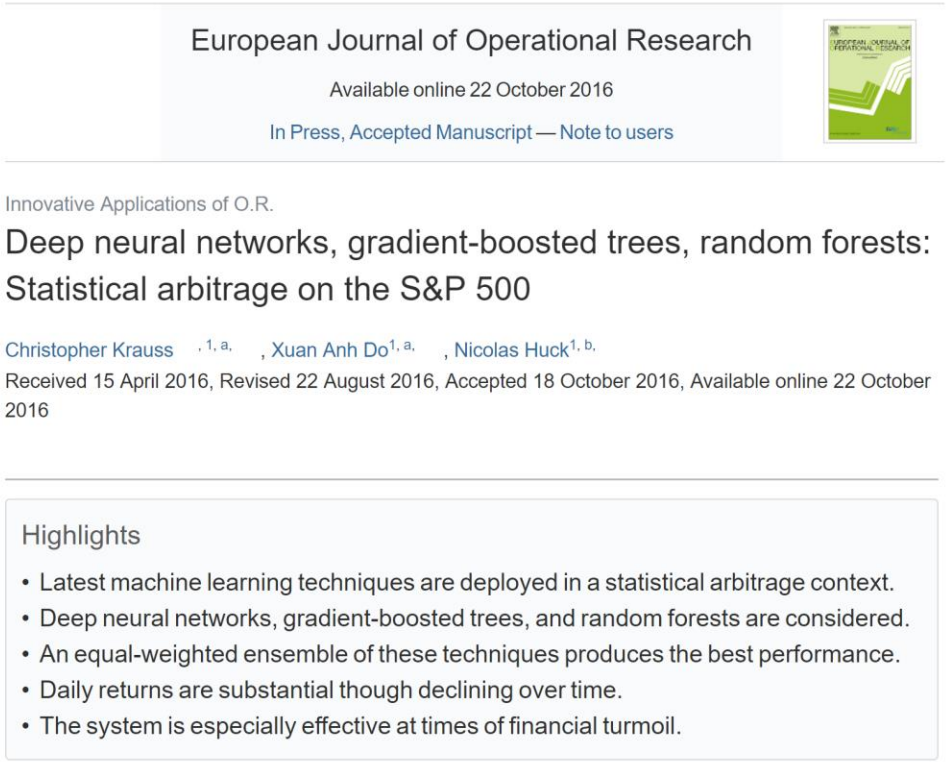
Mon 1 Jun 2015 – Mon 5 Oct 2015 (3 months ago)

READ MORE

"For my final competition submission I used an ensemble of models, including 3 deep learning models built with R and h2o."

H_2O.ai

H_2O for Academic Research



European Journal of Operational Research

Available online 22 October 2016

In Press, Accepted Manuscript — Note to users

Innovative Applications of O.R.

Deep neural networks, gradient-boosted trees, random forests: Statistical arbitrage on the S&P 500

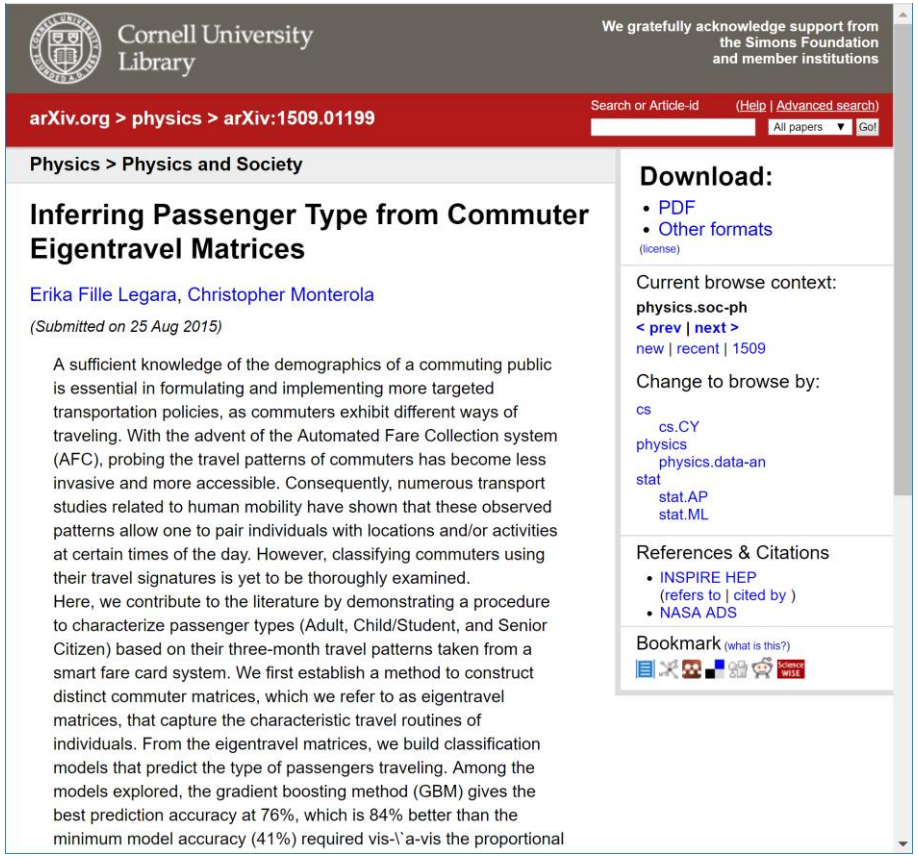
Christopher Krauss [1, a], Xuan Anh Do [1, a], Nicolas Huck [1, b]

Received 15 April 2016, Revised 22 August 2016, Accepted 18 October 2016, Available online 22 October 2016

Highlights

- Latest machine learning techniques are deployed in a statistical arbitrage context.
- Deep neural networks, gradient-boosted trees, and random forests are considered.
- An equal-weighted ensemble of these techniques produces the best performance.
- Daily returns are substantial though declining over time.
- The system is especially effective at times of financial turmoil.

http://www.sciencedirect.com/science/article/pii/S0377221716308657



Cornell University Library

We gratefully acknowledge support from the Simons Foundation and member institutions

arXiv.org > physics > arXiv:1509.01199

Search or Article-id (Help | Advanced search) All papers Go!

Physics > Physics and Society

Inferring Passenger Type from Commuter Eigentravel Matrices

Erika Fille Legara, Christopher Monterola

(Submitted on 25 Aug 2015)

A sufficient knowledge of the demographics of a commuting public is essential in formulating and implementing more targeted transportation policies, as commuters exhibit different ways of traveling. With the advent of the Automated Fare Collection system (AFC), probing the travel patterns of commuters has become less invasive and more accessible. Consequently, numerous transport studies related to human mobility have shown that these observed patterns allow one to pair individuals with locations and/or activities at certain times of the day. However, classifying commuters using their travel signatures is yet to be thoroughly examined. Here, we contribute to the literature by demonstrating a procedure to characterize passenger types (Adult, Child/Student, and Senior Citizen) based on their three-month travel patterns taken from a smart fare card system. We first establish a method to construct distinct commuter matrices, which we refer to as eigentravel matrices, that capture the characteristic travel routines of individuals. From the eigentravel matrices, we build classification models that predict the type of passengers traveling. Among the models explored, the gradient boosting method (GBM) gives the best prediction accuracy at 76%, which is 84% better than the minimum model accuracy (41%) required vis-\`a-vis the proportional

Download:

- PDF
- Other formats

(license)

Current browse context: physics.soc-ph

< prev | next >

new | recent | 1509

Change to browse by:

cs
cs.CY
physics
physics.data-an
stat
stat.AP
stat.ML

References & Citations

- INSPIRE HEP (refers to | cited by)
- NASA ADS

Bookmark (what is this?)

https://arxiv.org/abs/1509.01199

H_2O.ai

Users In Various Verticals Adore H_2O

Figure 1. Magic Quadrant for Data Science Platforms

Source: Gartner (February 2017)

H2O.ai recognized for completeness of vision and ability to execute

We are thrilled to be named a Visionary among the 16 vendors included in Gartner's 2017 Magic Quadrant for Data Science Platforms. As a Visionary we believe we are positioned highest in Ability to Execute for companies of our size and scale.

Since 2011, our mission has been to democratize data science through open source AI and deep learning. Today, H2O.ai is focused on bringing AI to enterprises with a growing community of more than 8,500 organizations that depend on H2O for mission critical applications. H2O.ai was recently named CB Insights AI 100 and is used by 107 of the Fortune 500 companies.

Disclaimer: This graphic was published by Gartner, Inc. as part of a larger research document and should be evaluated in the context of the entire document. The Gartner document is available upon request from H2O.ai.

http://www.h2o.ai/gartner-magic-quadrant/

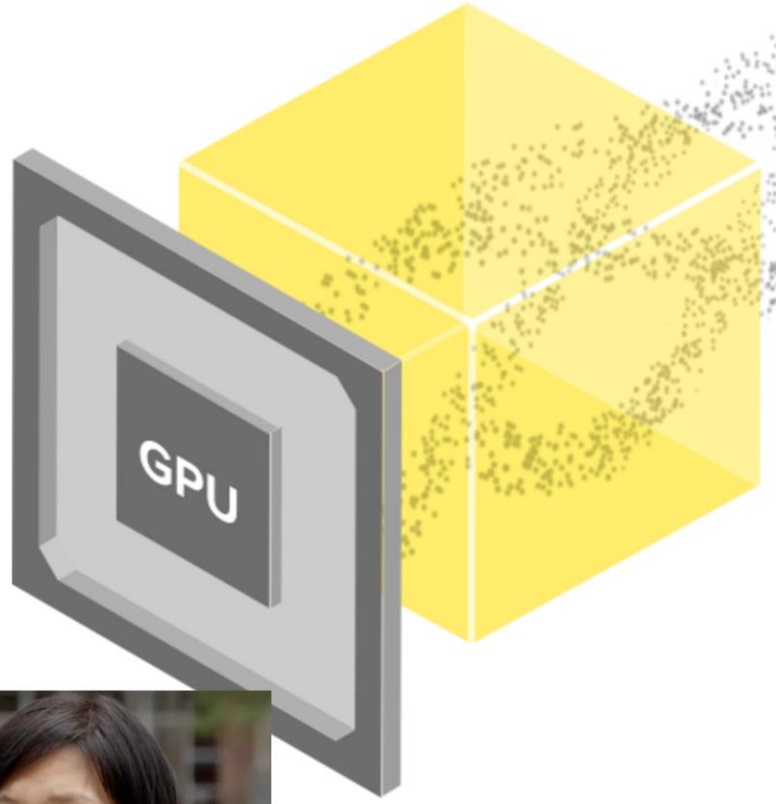
H2O.ai

Check out our website h2o.ai

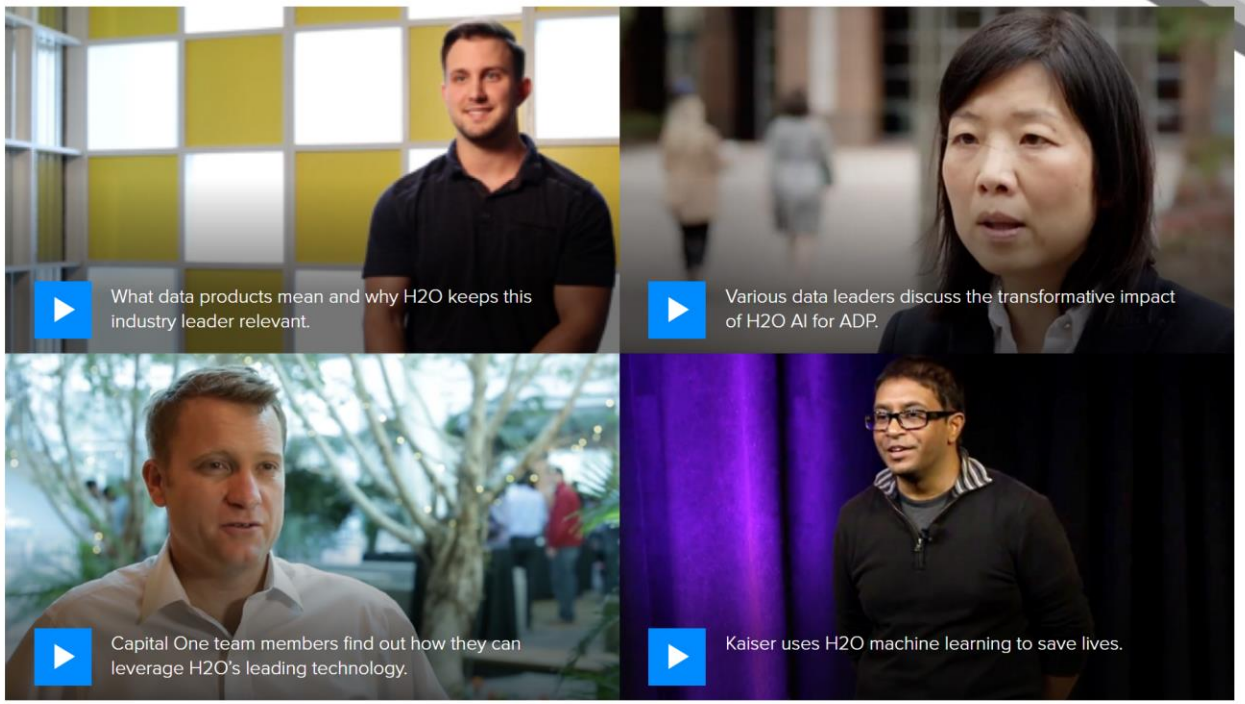
World Record Performance for AI

H2O.ai is accelerating both machine learning and deep learning on GPUs, providing enterprises opportunities to build better models and enable new use cases.

SEE DEEP WATER



H2O in Action



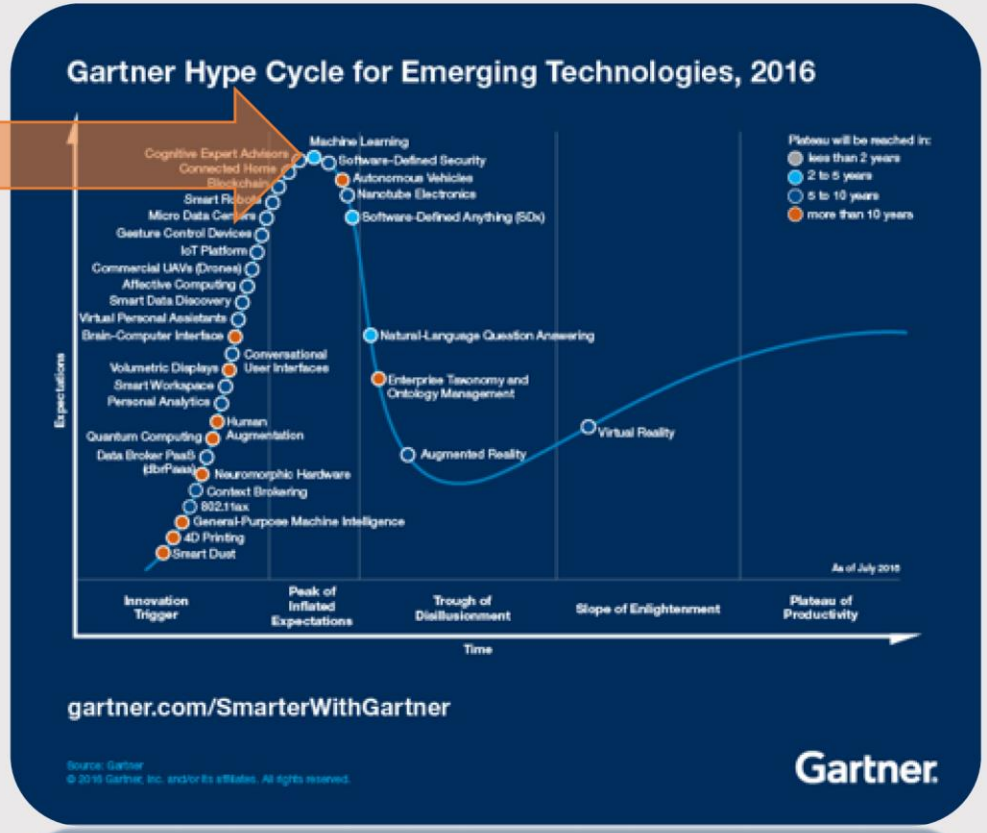
H_2O Machine Learning Platform

What is machine learning?

A field of study that gives computers the ability to learn without being explicitly programmed.

-- Arthur Samuel, 1959

H2O.ai

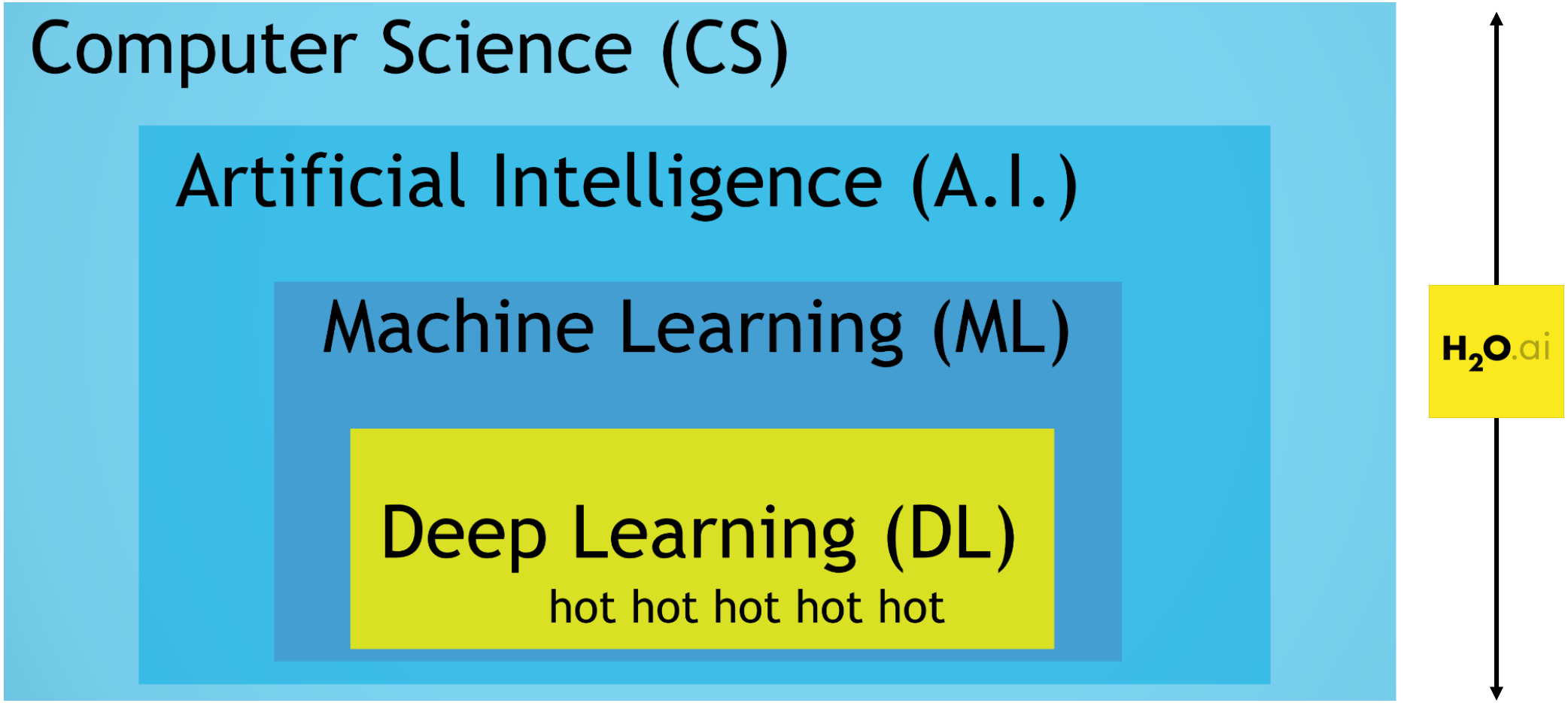


Why now?

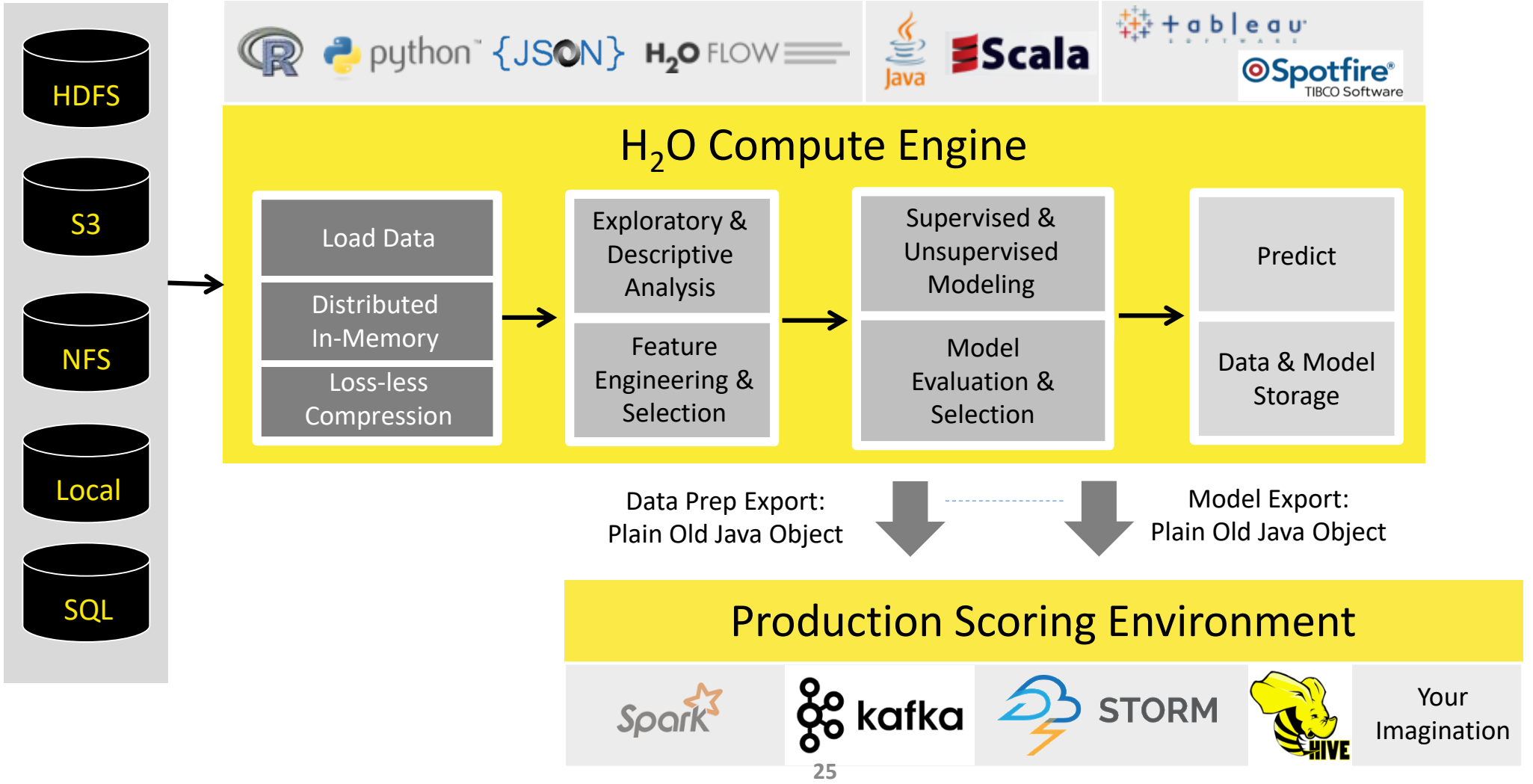
- Data, computers, and algorithms are commodities
- Unstructured data
- Increasing competition in business

H2O.ai

H_2O Overview

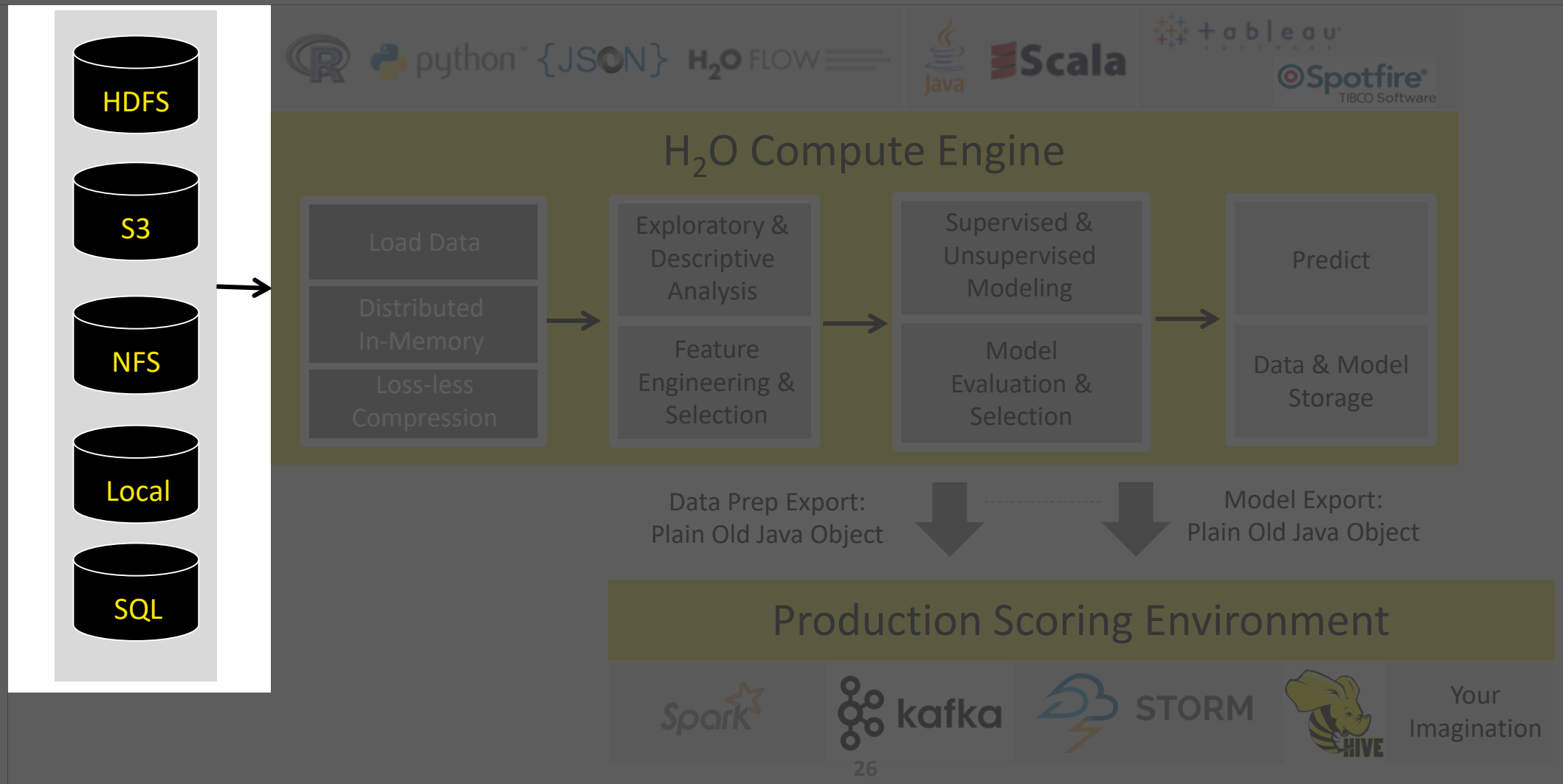


High Level Architecture



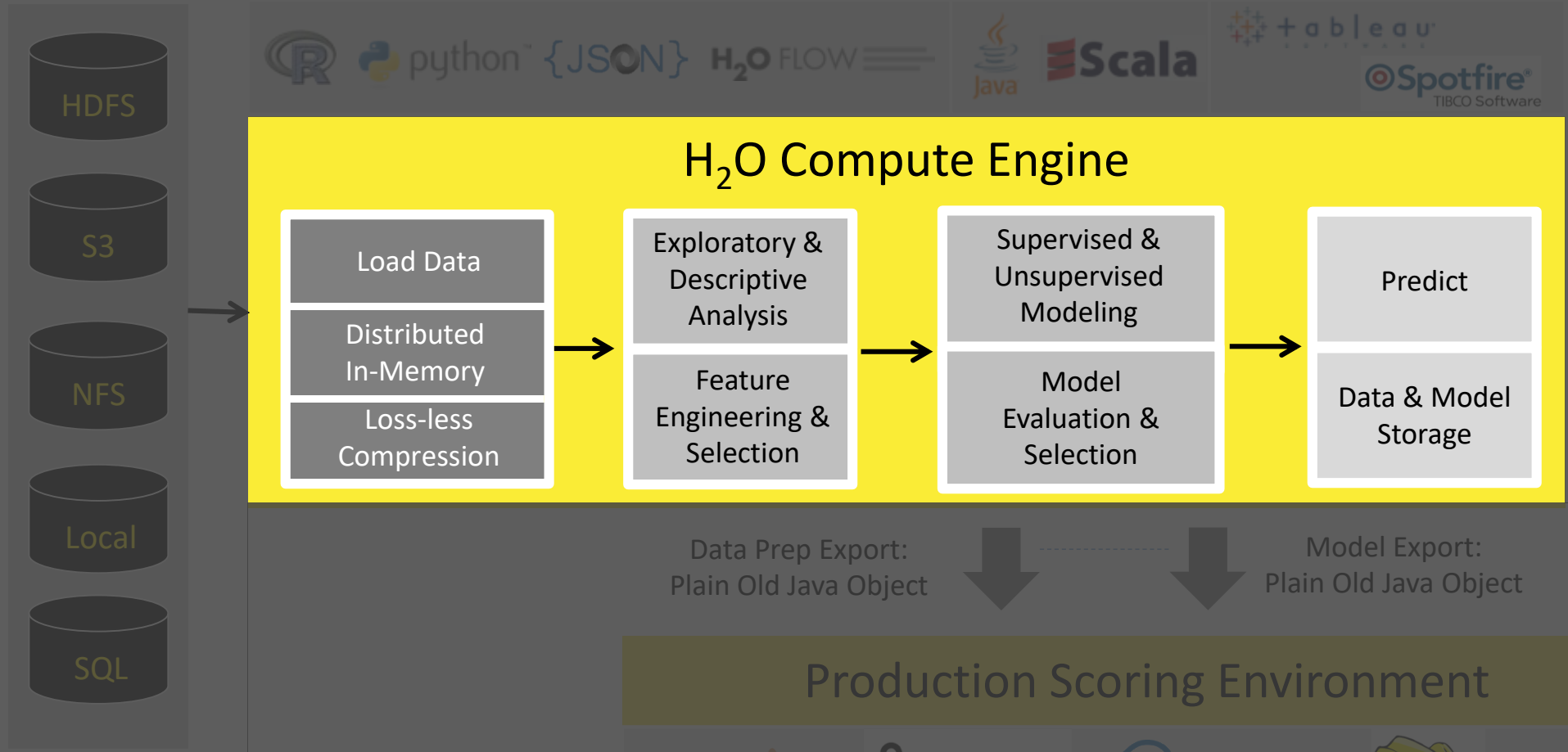
High Level Architecture

Import Data from Multiple Sources



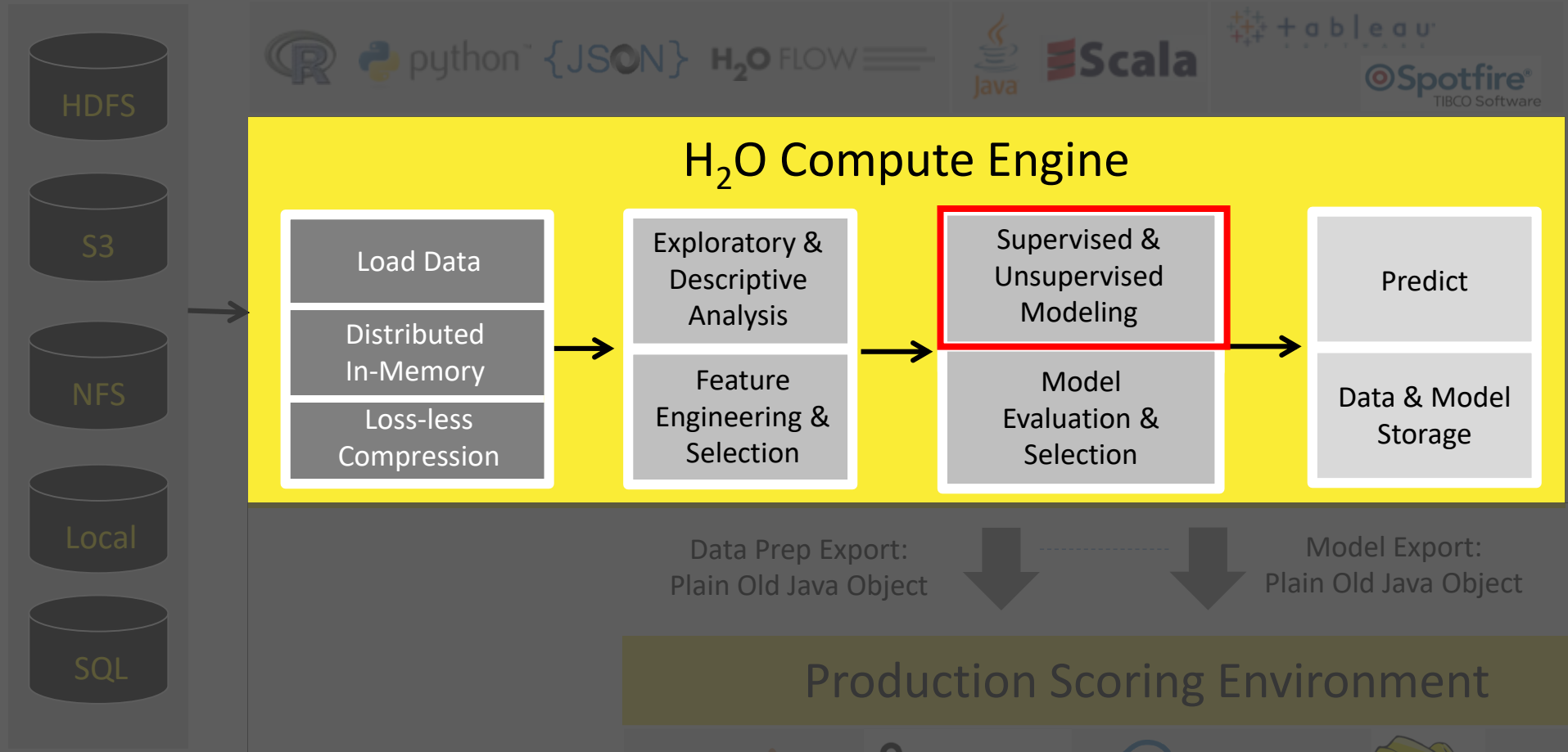
High Level Architecture

Fast, Scalable & Distributed Compute Engine Written in Java



High Level Architecture

Fast, Scalable & Distributed Compute Engine Written in Java



Algorithms Overview

Supervised Learning

Statistical Analysis	• **Generalized Linear Models**: Binomial, Gaussian, Gamma, Poisson and Tweedie • **Naïve Bayes**
Ensembles	• **Distributed Random Forest**: Classification or regression models • **Gradient Boosting Machine**: Produces an ensemble of decision trees with increasing refined approximations
Deep Neural Networks	• **Deep learning**: Create multi-layer feed forward neural networks starting with an input layer followed by multiple layers of nonlinear transformations

Unsupervised Learning

Clustering	• **K-means**: Partitions observations into k clusters/groups of the same spatial size. Automatically detect optimal k
Dimensionality Reduction	• **Principal Component Analysis**: Linearly transforms correlated variables to independent components • **Generalized Low Rank Models:** extend the idea of PCA to handle arbitrary data consisting of numerical, Boolean, categorical, and missing data
Anomaly Detection	• **Autoencoders**: Find outliers using a nonlinear dimensionality reduction using deep learning

H_2O Deep Learning in Action

116M rows, 6GB CSV file
800+ predictors (numeric + categorical)

airlines_all_selected_cols.hex

Actions: View Data | Split... | Build Model... | Predict | Download | Export

Rows	Columns	Compressed Size
116695259	12	2GB

Job

Run Time 00:00:36.712
Remaining Time 00:00:17.188
Type Model
Key deeplearning-dd2f42f7-81f7-42e8-9d98-e34437309828
Description DeepLearning
Status RUNNING
Progress 69%

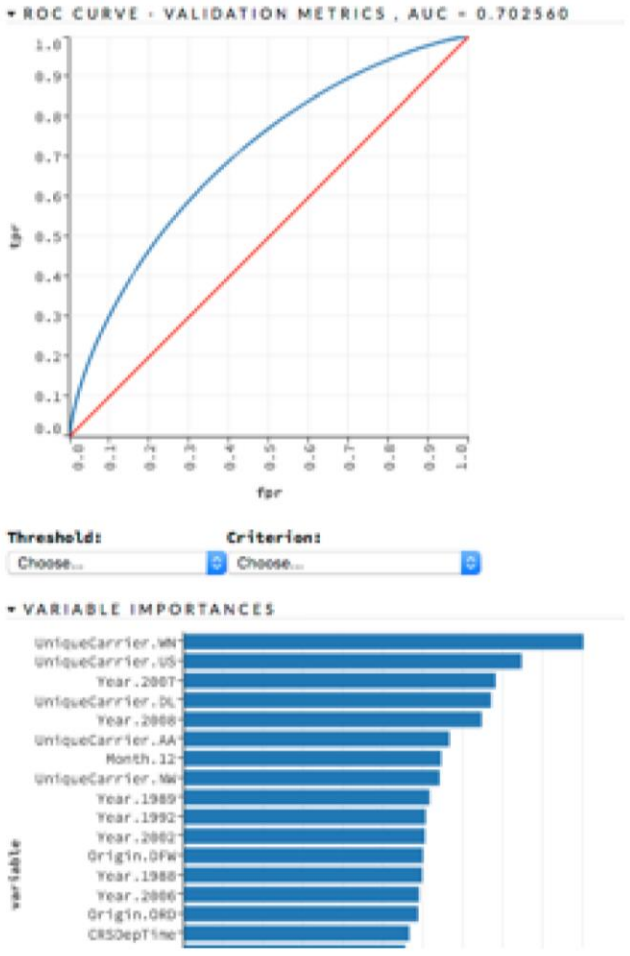
Iterations: 12. Epochs: 0.628821. Speed: 2,243,735 samples/sec. Estimated time left: 21.849 sec

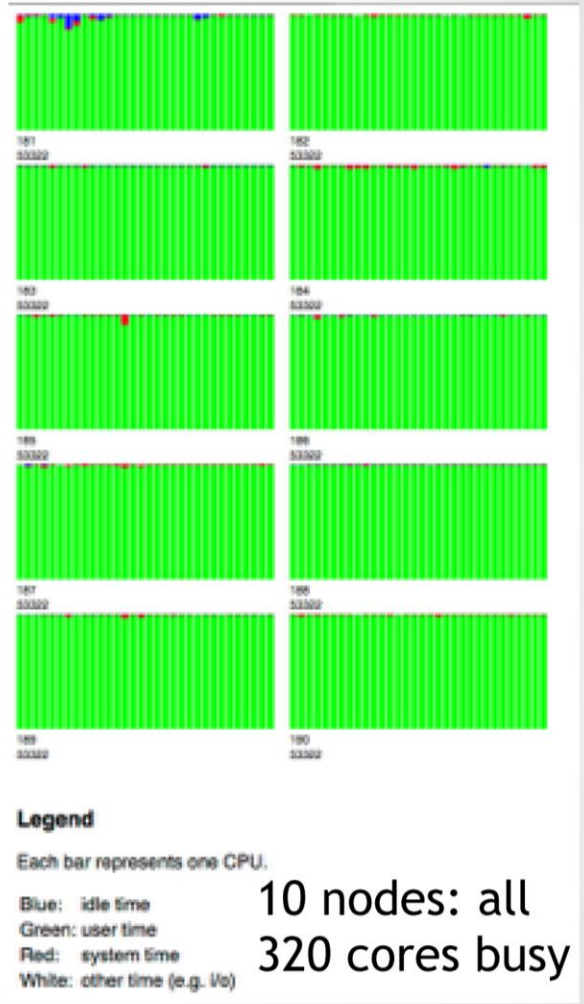
Actions: View | Cancel Job

model trained in <1 min:
2M+ samples/second

▾ OUTPUT - STATUS OF NEURON LAYERS (PREDICTING ISDEPDELAYED, 2-CLASS CLASSIFICATION, BERNOULLI DISTRIBUTION, CROSSENTROPY LOSS, 17,462 WEIGHTS/BIASES, 221.3 KB, 106,585,365 TRAINING SAMPLES, MINI-BATCH SIZE 1)

layer	units	type	dropout	l1	l2	mean_rate	rate_RMS	momentum	mean_weight	weight_RMS	mean_bias	bias_RMS
1	807	Input	0									
2	20	Rectifier	0	0	0	0.0493	0.2020	0	-0.0021	0.2111	-0.9139	1.0036
3	20	Rectifier	0	0	0	0.0197	0.0227	0	-0.1033	0.5362	-1.3988	1.5259
4	20	Rectifier	0	0	0	0.0517	0.0446	0	-0.1575	0.3068	-0.6846	0.6846
5	20	Rectifier	0	0	0	0.0761	0.0844	0	-0.0374	0.2275	-0.2647	0.2481
6	2	Softmax		0	0	0.0161	0.0083	0	0.0741	0.7260	0.4269	0.2056





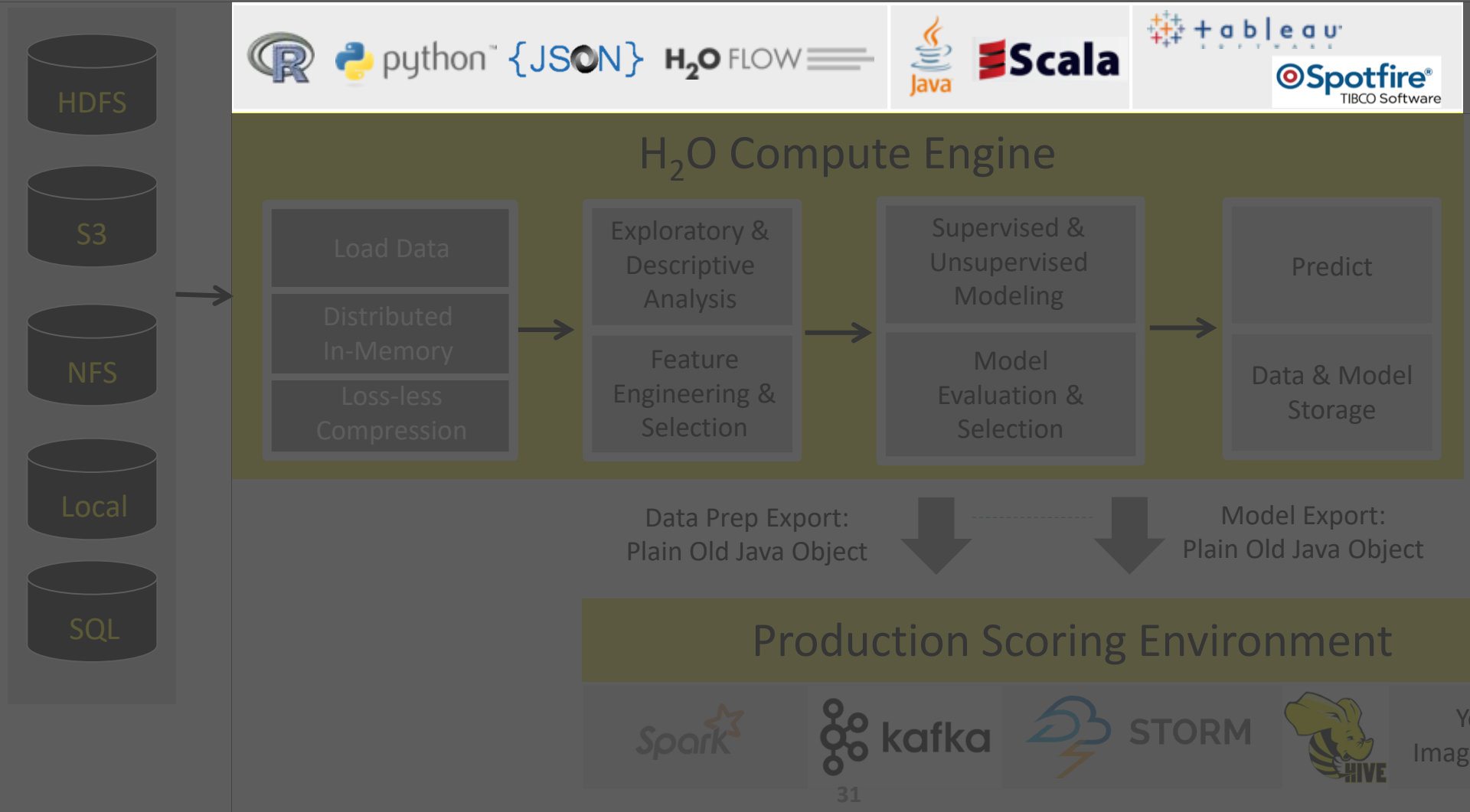
10 nodes: all
320 cores busy

real-time, interactive
model inspection in Flow

H_2O.ai **Deep Learning Model**

High Level Architecture

Multiple Interfaces



H_2O + R

```
# Start and connect to a local H2O cluster
library(h2o)
h2o.init(nthreads = -1)

# Import data from a R data frame
data(iris)
d_iris <- as.h2o(iris)

# Define Targets and Features
target <- "Species"
features <- setdiff(colnames(d_iris), c("Species"))
```

Package ‘h2o’ from CRAN or H_2O’s website

Start a local H_2O (Java Virtual Machine) cluster

Simple ‘iris’ example

H_2O + R

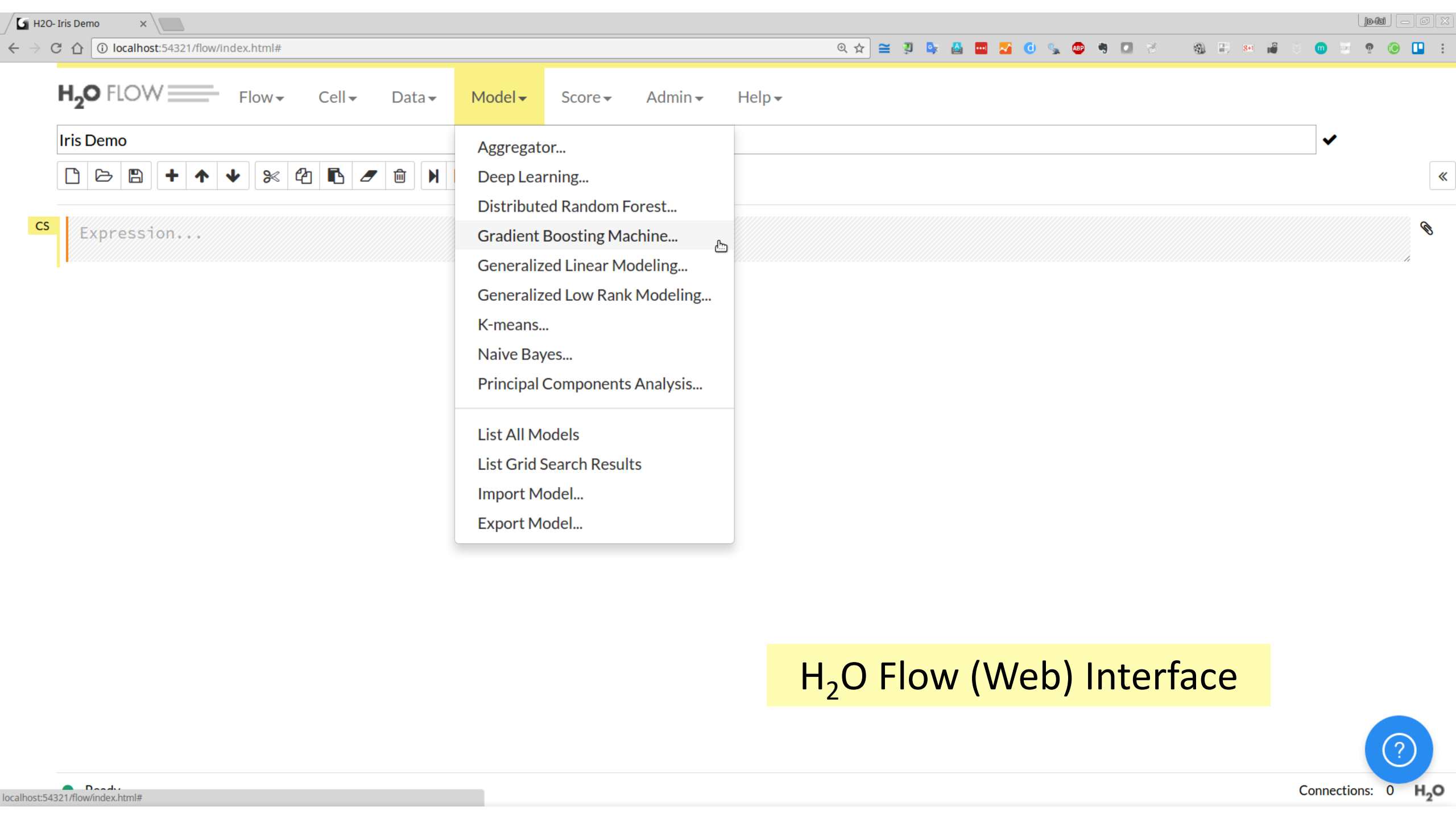
```
# ------------------------------------------------------------
# Train a H2O Model
# ------------------------------------------------------------

# Train three basic H2O models
model_drf <- h2o.randomForest(x = features,
                              y = target,
                              model_id = "iris_random_forest",
                              training_frame = d_iris)

model_gbm <- h2o.gbm(x = features,
                     y = target,
                     model_id = "iris_gbm",
                     training_frame = d_iris)

model_dnn <- h2o.deeplearning(x = features,
                              y = target,
                              model_id = "iris_deep_learning",
                              training_frame = d_iris)
```

H_2O + Python

Gradient Boosting Machines

```
# Build a Gradient Boosting Machines (GBM) model with default settings

# Import the function for GBM
from h2o.estimators.gbm import H2OGradientBoostingEstimator

# Set up GBM for regression
# Add a seed for reproducibility
gbm_default = H2OGradientBoostingEstimator(model_id = 'gbm_default', seed = 1234)

# Use .train() to build the model
gbm_default.train(x = features,
                  y = 'quality',
                  training_frame = wine_train)
```

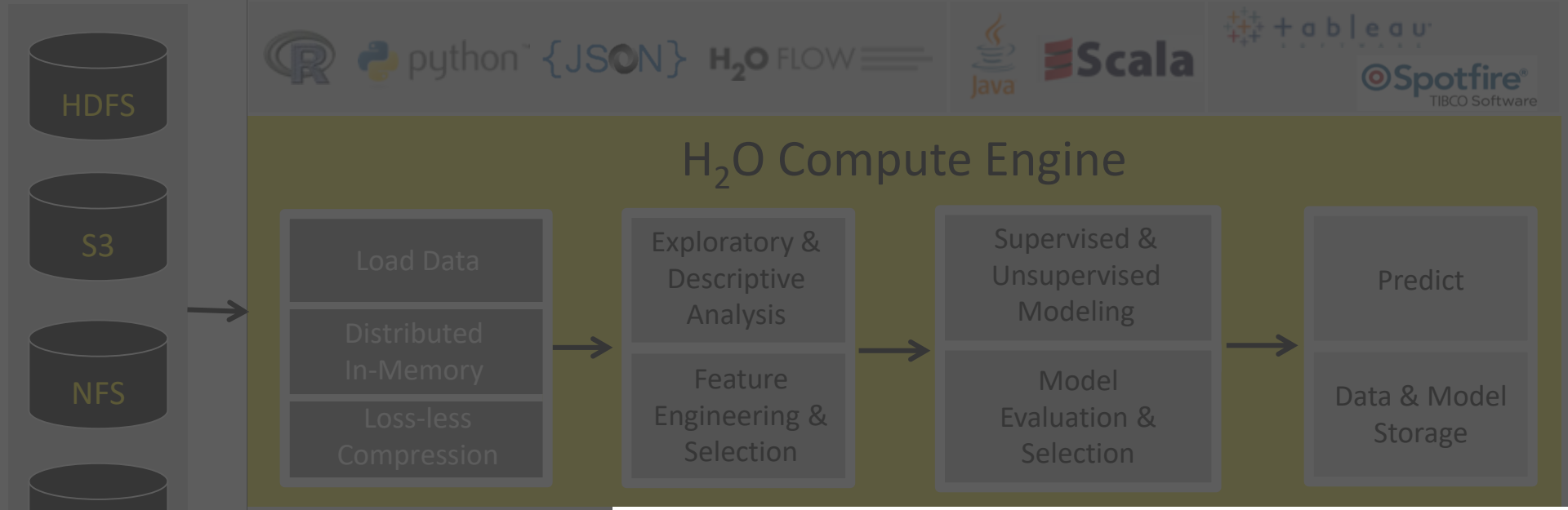
```
gbm Model Build progress: |████████████████████████████████████████████████| 100%
```



H_2O Flow (Web) Interface

High Level Architecture

Export Standalone Models for Production



Data Prep Export:
Plain Old Java Object

Model Export:
Plain Old Java Object

Production Scoring Environment

Spark | kafka | STORM | HIVE | Your Imagination

docs.h2o.ai

Languages

R

- Quick Start Video - R
- R Package Docs
- R Booklet
- Examples and Demos
- R FAQ
- Ensemble R Package Readme
- RSparkling Readme
- Migrating from H2O-2

Python

- Quick Start Video - Python
- Python Module Docs
- Python Booklet
- Examples and Demos
- Python FAQ
- PySparkling Readme 2.0 | 1.6
- skutil Docs

Java

- POJO and MOJO Model Javadoc
- H2O Core Javadoc
- H2O Algorithms Javadoc

Scala

Sparkling Water API	2.0	1.6
Sparkling Water Scaladoc	2.0	1.6
H2O Scaladoc	2.11	2.10

Tutorials, Examples, & Presentations

Tutorials and Blogs

- H2O Tutorials HTML | PDF
- H2O Blogs
- H2O University

Use Case Examples

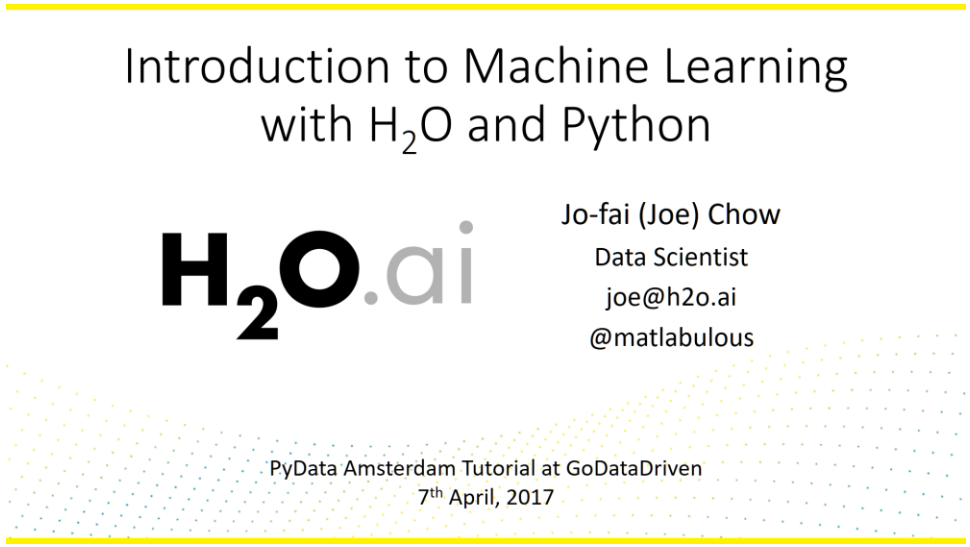
Chicago crime prediction	R	Python	ScalaSW	PySW
Airlines delays prediction	R	Python	ScalaSW	PySW
Lending Club loan prediction	R	Python	ScalaSW	PySW
Ham or Spam	R	Python	ScalaSW	PySW
Prediction with prostate dataset	R	Python	ScalaSW	PySW

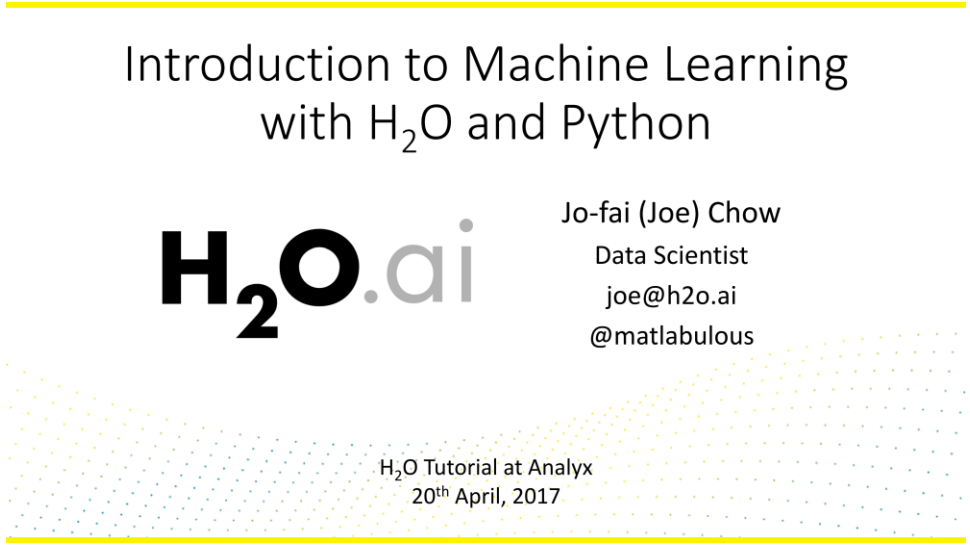
Presentations

- H2O Meetups
- H2O World 2014 Videos
- H2O World 2015 Videos
- Open Tour Chicago Videos
- Open Tour NYC Videos
- Open Tour Dallas Videos

H_2O Tutorials

- Introduction to Machine Learning with H_2O and Python
  - Basic Extract, Transform and Load (ETL)
  - Supervised Learning
  - Parameters Tuning
  - Stacking
- GitHub Repository
  - bit.ly/joe_h2o_tutorials
  - R Code Examples (soon)





H2O.ai

H_2O IoT Use Cases

Predictive Maintenance and Outlier Detection

Predictive Maintenance – SECOM Dataset

UCI Machine Learning Repository
Center for Machine Learning and Intelligent Systems

About Citation Policy Donate a Data Set Contact

Search
Repository Web
Google™

View ALL Data Sets

SECOM Data Set

Download: Data Folder, Data Set Description

Abstract: Data from a semi-conductor manufacturing process

Data Set Characteristics:	Multivariate	**Number of Instances:**	1567	**Area:**	Computer
Attribute Characteristics:	Real	**Number of Attributes:**	591	**Date Donated**	2008-11-19
Associated Tasks:	Classification, Causal-Discovery	**Missing Values?**	Yes	**Number of Web Hits:**	46440

Source:

Authors: Michael McCann, Adrian Johnston

Predictive Maintenance – SECOM Dataset

SECOM Dataset: 1567 examples 591 features, 104 fails

FSmethod (40 features) BER % True + % True - %
S2N (signal to noise) 34.5 +-2.6 57.8 +-5.3 73.1 +2.1
Ttest 33.7 +-2.1 59.6 +-4.7 73.0 +-1.8
Relief 40.1 +-2.8 48.3 +-5.9 71.6 +-3.2
Pearson 34.1 +-2.0 57.4 +-4.3 74.4 +-4.9
Ftest 33.5 +-2.2 59.1 +-4.8 73.8 +-1.8
Gram Schmidt 35.6 +-2.4 51.2 +-11.8 77.5 +-2.3

We want to predict fails in the future.

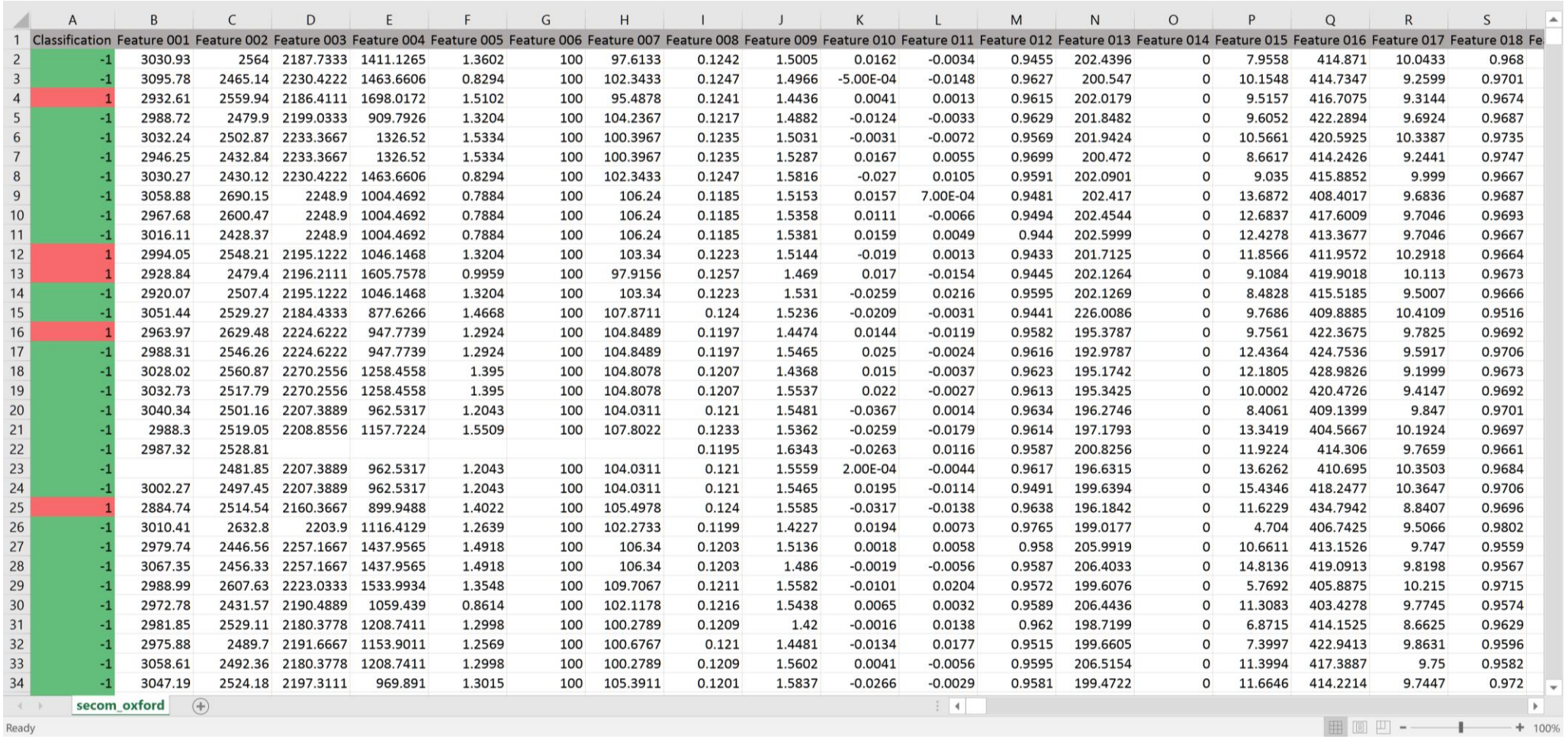
Attribute Information:

Key facts: Data Structure: The data consists of 2 files the dataset file SECOM consisting of 1567 examples each with 591 features a 1567 x 591 matrix and a labels file containing the classifications and date time stamp for each example.

As with any real life data situations this data contains null values varying in intensity depending on the individuals features. This needs to be taken into consideration when investigating the data either through pre-processing or within the technique applied.

The data is represented in a raw text file each line representing an individual example and the features seperated by spaces. The null values are represented by the 'NaN' value as per MatLab.

Predictive Maintenance – SECOM Dataset

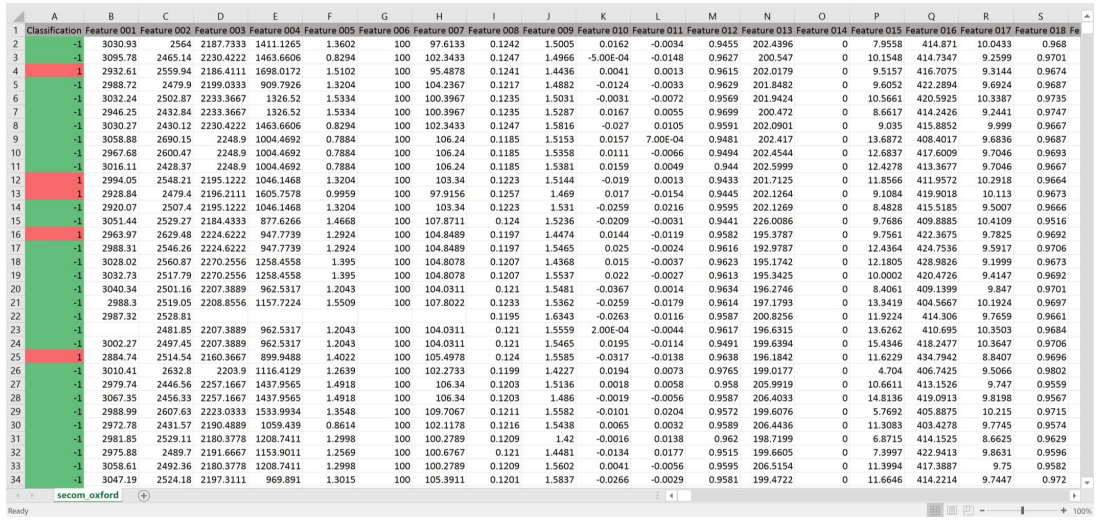


	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
1	Classification	Feature 001	Feature 002	Feature 003	Feature 004	Feature 005	Feature 006	Feature 007	Feature 008	Feature 009	Feature 010	Feature 011	Feature 012	Feature 013	Feature 014	Feature 015	Feature 016	Feature 017	Feature 018	Fe
2	-1	3030.93	2564	2187.7333	1411.1265	1.3602	100	97.6133	0.1242	1.5005	0.0162	-0.0034	0.9455	202.4396	0	7.9558	414.871	10.0433	0.968	
3	-1	3095.78	2465.14	2230.4222	1463.6606	0.8294	100	102.3433	0.1247	1.4966	-5.00E-04	-0.0148	0.9627	200.547	0	10.1548	414.7347	9.2599	0.9701	
4	1	2932.61	2559.94	2186.4111	1698.0172	1.5102	100	95.4878	0.1241	1.4436	0.0041	0.0013	0.9615	202.0179	0	9.5157	416.7075	9.3144	0.9674	
5	-1	2988.72	2479.9	2199.0333	909.7926	1.3204	100	104.2367	0.1217	1.4882	-0.0124	-0.0033	0.9629	201.8482	0	9.6052	422.2894	9.6924	0.9687	
6	-1	3032.24	2502.87	2233.3667	1326.52	1.5334	100	100.3967	0.1235	1.5031	-0.0031	-0.0072	0.9569	201.9424	0	10.5661	420.5925	10.3387	0.9735	
7	-1	2946.25	2432.84	2233.3667	1326.52	1.5334	100	100.3967	0.1235	1.5287	0.0167	0.0055	0.9699	200.472	0	8.6617	414.2426	9.2441	0.9747	
8	-1	3030.27	2430.12	2230.4222	1463.6606	0.8294	100	102.3433	0.1247	1.5816	-0.027	0.0105	0.9591	202.0901	0	9.035	415.8852	9.999	0.9667	
9	-1	3058.88	2690.15	2248.9	1004.4692	0.7884	100	106.24	0.1185	1.5153	0.0157	7.00E-04	0.9481	202.417	0	13.6872	408.4017	9.6836	0.9687	
10	-1	2967.68	2600.47	2248.9	1004.4692	0.7884	100	106.24	0.1185	1.5358	0.0111	-0.0066	0.9494	202.4544	0	12.6837	417.6009	9.7046	0.9693	
11	-1	3016.11	2428.37	2248.9	1004.4692	0.7884	100	106.24	0.1185	1.5381	0.0159	0.0049	0.944	202.5999	0	12.4278	413.3677	9.7046	0.9667	
12	1	2994.05	2548.21	2195.1222	1046.1468	1.3204	100	103.34	0.1223	1.5144	-0.019	0.0013	0.9433	201.7125	0	11.8566	411.9572	10.2918	0.9664	
13	1	2928.84	2479.4	2196.2111	1605.7578	0.9959	100	97.9156	0.1257	1.469	0.017	-0.0154	0.9445	202.1264	0	9.1084	419.9018	10.113	0.9673	
14	-1	2920.07	2507.4	2195.1222	1046.1468	1.3204	100	103.34	0.1223	1.531	-0.0259	0.0216	0.9595	202.1269	0	8.4828	415.5185	9.5007	0.9666	
15	-1	3051.44	2529.27	2184.4333	877.6266	1.4668	100	107.8711	0.124	1.5236	-0.0209	-0.0031	0.9441	226.0086	0	9.7686	409.8885	10.4109	0.9516	
16	1	2963.97	2629.48	2224.6222	947.7739	1.2924	100	104.8489	0.1197	1.4474	0.0144	-0.0119	0.9582	195.3787	0	9.7561	422.3675	9.7825	0.9692	
17	-1	2988.31	2546.26	2224.6222	947.7739	1.2924	100	104.8489	0.1197	1.5465	0.025	-0.0024	0.9616	192.9787	0	12.4364	424.7536	9.5917	0.9706	
18	-1	3028.02	2560.87	2270.2556	1258.4558	1.395	100	104.8078	0.1207	1.4368	0.015	-0.0037	0.9623	195.1742	0	12.1805	428.9826	9.1999	0.9673	
19	-1	3032.73	2517.79	2270.2556	1258.4558	1.395	100	104.8078	0.1207	1.5537	0.022	-0.0027	0.9613	195.3425	0	10.0002	420.4726	9.4147	0.9692	
20	-1	3040.34	2501.16	2207.3889	962.5317	1.2043	100	104.0311	0.121	1.5481	-0.0367	0.0014	0.9634	196.2746	0	8.4061	409.1399	9.847	0.9701	
21	-1	2988.3	2519.05	2208.8556	1157.7224	1.5509	100	107.8022	0.1233	1.5362	-0.0259	-0.0179	0.9614	197.1793	0	13.3419	404.5667	10.1924	0.9697	
22	-1	2987.32	2528.81						0.1195	1.6343	-0.0263	0.0116	0.9587	200.8256	0	11.9224	414.306	9.7659	0.9661	
23	-1		2481.85	2207.3889	962.5317	1.2043	100	104.0311	0.121	1.5559	2.00E-04	-0.0044	0.9617	196.6315	0	13.6262	410.695	10.3503	0.9684	
24	-1	3002.27	2497.45	2207.3889	962.5317	1.2043	100	104.0311	0.121	1.5465	0.0195	-0.0114	0.9491	199.6394	0	15.4346	418.2477	10.3647	0.9706	
25	1	2884.74	2514.54	2160.3667	899.9488	1.4022	100	105.4978	0.124	1.5585	-0.0317	-0.0138	0.9638	196.1842	0	11.6229	434.7942	8.8407	0.9696	
26	-1	3010.41	2632.8	2203.9	1116.4129	1.2639	100	102.2733	0.1199	1.4227	0.0194	0.0073	0.9765	199.0177	0	4.704	406.7425	9.5066	0.9802	
27	-1	2979.74	2446.56	2257.1667	1437.9565	1.4918	100	106.34	0.1203	1.5136	0.0018	0.0058	0.958	205.9919	0	10.6611	413.1526	9.747	0.9559	
28	-1	3067.35	2456.33	2257.1667	1437.9565	1.4918	100	106.34	0.1203	1.486	-0.0019	-0.0056	0.9587	206.4033	0	14.8136	419.0913	9.8198	0.9567	
29	-1	2988.99	2607.63	2223.0333	1533.9934	1.3548	100	109.7067	0.1211	1.5582	-0.0101	0.0204	0.9572	199.6076	0	5.7692	405.8875	10.215	0.9715	
30	-1	2972.78	2431.57	2190.4889	1059.439	0.8614	100	102.1178	0.1216	1.5438	0.0065	0.0032	0.9589	206.4436	0	11.3083	403.4278	9.7745	0.9574	
31	-1	2981.85	2529.11	2180.3778	1208.7411	1.2998	100	100.2789	0.1209	1.42	-0.0016	0.0138	0.962	198.7199	0	6.8715	414.1525	8.6625	0.9629	
32	-1	2975.88	2489.7	2191.6667	1153.9011	1.2569	100	100.6767	0.121	1.4481	-0.0134	0.0177	0.9515	199.6605	0	7.3997	422.9413	9.8631	0.9596	
33	-1	3058.61	2492.36	2180.3778	1208.7411	1.2998	100	100.2789	0.1209	1.5602	0.0041	-0.0056	0.9595	206.5154	0	11.3994	417.3887	9.75	0.9582	
34	-1	3047.19	2524.18	2197.3111	969.891	1.3015	100	105.3911	0.1201	1.5837	-0.0266	-0.0029	0.9581	199.4722	0	11.6646	414.2214	9.7447	0.972	

H2O.ai

Predictive Maintenance – SECOM Dataset

- Dataset Summary
  - Inputs:
    - 591 numerical features
  - Binary Outcome:
    - Pass (-1)
    - Fail (1)
  - Size:
    - 1567 samples



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
1	Classification	Feature 001	Feature 002	Feature 003	Feature 004	Feature 005	Feature 006	Feature 007	Feature 008	Feature 009	Feature 010	Feature 011	Feature 012	Feature 013	Feature 014	Feature 015	Feature 016	Feature 017	Feature 018	Fe
2	-1	3030.93	2564	2187.7333	1411.1265	1.3602	100	97.6133	0.1242	1.5005	0.0162	-0.0034	0.9455	202.4396	0	7.9558	414.871	10.0433	0.968	
3	-1	3095.78	2465.14	2230.4222	1463.6606	0.8294	100	102.3433	0.1247	1.4966	-5.00E-04	-0.0148	0.9627	200.547	0	10.1548	414.7347	9.2599	0.9701	
4	1	2932.61	2559.94	2186.4111	1698.0172	1.5102	100	95.4878	0.1241	1.4436	0.0041	0.0013	0.9615	202.0179	0	9.5157	416.7075	9.3144	0.9674	
5	-1	2988.72	2479.9	2199.0333	909.7926	1.3204	100	104.2367	0.1217	1.4882	-0.0124	-0.0033	0.9629	201.8482	0	9.6052	422.2894	9.6924	0.9687	
6	-1	3032.24	2502.87	2233.3667	1326.52	1.5334	100	100.3967	0.1235	1.5031	-0.0031	-0.0072	0.9569	201.9424	0	10.5661	420.5925	10.3387	0.9735	
7	-1	2946.25	2432.84	2233.3667	1326.52	1.5334	100	100.3967	0.1235	1.5287	0.0167	0.0055	0.9699	200.472	0	8.6617	414.2426	9.2441	0.9747	
8	-1	3030.27	2430.12	2230.4222	1463.6606	0.8294	100	102.3433	0.1247	1.5816	-0.027	0.0105	0.9591	202.0901	0	9.035	415.8852	9.999	0.9667	
9	-1	3058.88	2690.15	2248.9	1004.4692	0.7884	100	106.24	0.1185	1.5153	0.0157	7.00E-04	0.9481	202.417	0	13.6872	408.4017	9.6836	0.9687	
10	-1	2967.68	2600.47	2248.9	1004.4692	0.7884	100	106.24	0.1185	1.5358	0.0111	-0.0066	0.9494	202.4544	0	12.6837	417.6009	9.7046	0.9693	
11	-1	3016.11	2428.37	2248.9	1004.4692	0.7884	100	106.24	0.1185	1.5381	0.0159	0.0049	0.944	202.5999	0	12.4278	413.3677	9.7046	0.9667	
12	1	2994.05	2548.21	2195.1222	1046.1468	1.3204	100	103.34	0.1223	1.5144	-0.019	0.0013	0.9433	201.7125	0	11.8566	411.9572	10.2918	0.9664	
13	1	2928.84	2479.4	2196.2111	1605.7578	0.9959	100	97.9156	0.1257	1.469	0.017	-0.0154	0.9445	202.1264	0	9.1084	419.9018	10.113	0.9673	
14	-1	2920.07	2507.4	2195.1222	1046.1468	1.3204	100	103.34	0.1223	1.531	-0.0259	0.0216	0.9595	202.1269	0	8.4828	415.5185	9.5007	0.9666	
15	-1	3051.44	2529.27	2184.4333	877.6266	1.4668	100	107.8711	0.124	1.5236	-0.0209	-0.0031	0.9441	226.0086	0	9.7686	409.8885	10.4109	0.9516	
16	1	2963.97	2629.48	2224.6222	947.7739	1.2924	100	104.8489	0.1197	1.4474	0.0144	-0.0119	0.9582	195.3787	0	9.7561	422.3675	9.7825	0.9692	
17	-1	2988.31	2546.26	2224.6222	947.7739	1.2924	100	104.8489	0.1197	1.5465	0.025	-0.0024	0.9616	192.9787	0	12.4364	424.7536	9.5917	0.9706	
18	-1	3028.02	2560.87	2270.2556	1258.4558	1.395	100	104.8078	0.1207	1.4368	0.015	-0.0037	0.9623	195.1742	0	12.1805	428.9826	9.1999	0.9673	
19	-1	3032.73	2517.79	2270.2556	1258.4558	1.395	100	104.8078	0.1207	1.5537	0.022	-0.0027	0.9613	195.3425	0	10.0002	420.4726	9.4147	0.9692	
20	-1	3040.34	2501.16	2207.3889	962.5317	1.2043	100	104.0311	0.121	1.5481	-0.0367	0.0014	0.9634	196.2746	0	8.4061	409.1399	9.847	0.9701	
21	-1	2988.3	2519.05	2208.8556	1157.7224	1.5509	100	107.8022	0.1233	1.5362	-0.0259	-0.0179	0.9614	197.1793	0	13.3419	404.5667	10.1924	0.9697	
22	-1	2987.32	2528.81						0.1195	1.6343	-0.0263	0.0116	0.9587	200.8256	0	11.9224	414.306	9.7659	0.9661	
23	-1		2481.85	2207.3889	962.5317	1.2043	100	104.0311	0.121	1.5559	2.00E-04	-0.0044	0.9617	196.6315	0	13.6262	410.695	10.3503	0.9684	
24	-1	3002.27	2497.45	2207.3889	962.5317	1.2043	100	104.0311	0.121	1.5465	0.0195	-0.0114	0.9491	199.6394	0	15.4346	418.2477	10.3647	0.9706	
25	1	2884.74	2514.54	2160.3667	899.9488	1.4022	100	105.4978	0.124	1.5585	-0.0317	-0.0138	0.9638	196.1842	0	11.6229	434.7942	8.8407	0.9696	
26	-1	3010.41	2632.8	2203.9	1116.4129	1.2639	100	102.2733	0.1199	1.4227	0.0194	0.0073	0.9765	199.0177	0	4.704	406.7425	9.5066	0.9802	
27	-1	2979.74	2446.56	2257.1667	1437.9565	1.4918	100	106.34	0.1203	1.5136	0.0018	0.0058	0.958	205.9919	0	10.6611	413.1526	9.747	0.9559	
28	-1	3067.35	2456.33	2257.1667	1437.9565	1.4918	100	106.34	0.1203	1.486	-0.0019	-0.0056	0.9587	206.4033	0	14.8136	419.0913	9.8198	0.9567	
29	-1	2988.99	2607.63	2223.0333	1533.9934	1.3548	100	109.7067	0.1211	1.5582	-0.0101	0.0204	0.9572	199.6076	0	5.7692	405.8875	10.215	0.9715	
30	-1	2972.78	2431.57	2190.4889	1059.439	0.8614	100	102.1178	0.1216	1.5438	0.0065	0.0032	0.9589	206.4436	0	11.3083	403.4278	9.7745	0.9574	
31	-1	2981.85	2529.11	2180.3778	1208.7411	1.2998	100	100.2789	0.1209	1.42	-0.0016	0.0138	0.962	198.7199	0	6.8715	414.1525	8.6625	0.9629	
32	-1	2975.88	2489.7	2191.6667	1153.9011	1.2569	100	100.6767	0.121	1.4481	-0.0134	0.0177	0.9515	199.6605	0	7.3997	422.9413	9.8631	0.9596	
33	-1	3058.61	2492.36	2180.3778	1208.7411	1.2998	100	100.2789	0.1209	1.5602	0.0041	-0.0056	0.9595	206.5154	0	11.3994	417.3887	9.75	0.9582	
34	-1	3047.19	2524.18	2197.3111	969.891	1.3015	100	105.3911	0.1201	1.5837	-0.0266	-0.0029	0.9581	199.4722	0	11.6646	414.2214	9.7447	0.972	

Basic H_2O Usage – GBM with Default Settings

- Link to Jupyter Notebook
  - https://github.com/woobe/h2o_tutorials/blob/master/use_cases/predictive_maintenance/step_01_basics.ipynb

H2O Use Case - Predictive Maintenance

- Source: https://archive.ics.uci.edu/ml/datasets/SECOM
- H2O Basics: train a default Gradient Boosting Machine (GBM) for binary classification.

In [1]:
```
# Load h2o library
suppressPackageStartupMessages(library(h2o))
```

In [2]:
```
# Start and connect to a local H2O cluster
h2o.init(nthreads = -1)
```

```
H2O is not running yet, starting it now...

Note:  In case of errors look at the following log files:
    /tmp/Rtmpf9glXx/h2o_joe_started_from_r.out
    /tmp/Rtmpf9glXx/h2o_joe_started_from_r.err


Starting H2O JVM and connecting: ... Connection successful!

R is connected to the H2O cluster:
    H2O cluster uptime:         3 seconds 160 milliseconds
    H2O cluster version:        3.10.4.4
    H2O cluster version age:    3 days
    H2O cluster name:           H2O_started_from_R_joe_cqn703
    H2O cluster total nodes:    1
    H2O cluster total memory:   5.21 GB
    H2O cluster total cores:    8
    H2O cluster allowed cores:  8
    H2O cluster healthy:        TRUE
    H2O Connection ip:          localhost
    H2O Connection port:        54321
    H2O Connection proxy:       NA
    H2O Internal Security:      FALSE
    R Version:                  R version 3.3.2 (2016-10-31)
```

H_2O's R package

Start a local H_2O Cluster (JVM)
"nthreads = -1" means using all available virtual cores

Information of the H_2O Cluster

H2O.ai

Import data into H_2O cluster (instead of R's memory)

In [3]:
```
# Importing data from local CSV
h_secom <- h2o.importFile(path = "secom.csv", destination_frame = "h_secom")
```

```
  |======================================================================| 100%
```

In [4]:
```
# Print out column names
colnames(h_secom)
```

1. 'Classification'
2. 'Feature 001'
3. 'Feature 002'
4. 'Feature 003'
5. 'Feature 004'
6. 'Feature 005'
7. 'Feature 006'
8. 'Feature 007'
9. 'Feature 008'
10. 'Feature 009'
11. 'Feature 010'

In [5]:
```
# Look at "Classification"
summary(h_secom$Classification, exact_quantiles=TRUE)
```

```
 Classification
 Min.   :-1.0000
 1st Qu.:-1.0000
 Median :-1.0000
 Mean   :-0.8673
 3rd Qu.:-1.0000
 Max.   : 1.0000
```

Convert numerical to categorical values

In [6]:
```
# "Classification" is a column of numerical values
# Convert "Classification" in secom dataset from numerical to categorical value
h_secom$Classification <- as.factor(h_secom$Classification)
```

In [7]:
```
# Look at "Classification" again
summary(h_secom$Classification, exact_quantiles=TRUE)
```

```
 Classification
 -1:1463
 1 : 104
```

In [8]:
```r
# Define target (y) and features (x)
target <- "Classification"
features <- setdiff(colnames(h_secom), target)
print(features)
```

```
 [1] "Feature 001" "Feature 002" "Feature 003" "Feature 004" "Feature 005"
 [6] "Feature 006" "Feature 007" "Feature 008" "Feature 009" "Feature 010"
[11] "Feature 011" "Feature 012" "Feature 013" "Feature 014" "Feature 015"
[16] "Feature 016" "Feature 017" "Feature 018" "Feature 019" "Feature 020"
[21] "Feature 021" "Feature 022" "Feature 023" "Feature 024" "Feature 025"
[26] "Feature 026" "Feature 027" "Feature 028" "Feature 029" "Feature 030"
[31] "Feature 031" "Feature 032" "Feature 033" "Feature 034" "Feature 035"
[36] "Feature 036" "Feature 037" "Feature 038" "Feature 039" "Feature 040"
[41] "Feature 041" "Feature 042" "Feature 043" "Feature 044" "Feature 045"
```

In [9]:
```
# Splitting dataset into training and test
h_split <- h2o.splitFrame(h_secom, ratios = 0.7, seed = 1234)
h_train <- h_split[[1]] # 70%
h_test  <- h_split[[2]] # 30%
```

In [10]:
```
# Look at the size
dim(h_train)
dim(h_test)
```

1. 1105
2. 591

1. 462
2. 591

In [11]:
```
# Check Classification in each dataset
summary(h_train$Classification, exact_quantiles = TRUE)
summary(h_test$Classification, exact_quantiles = TRUE)
```

```
Classification
-1:1028
1 :  77
```

```
Classification
-1:435
1 : 27
```

Split data into training / test in order to evaluate out-of-bag performance later

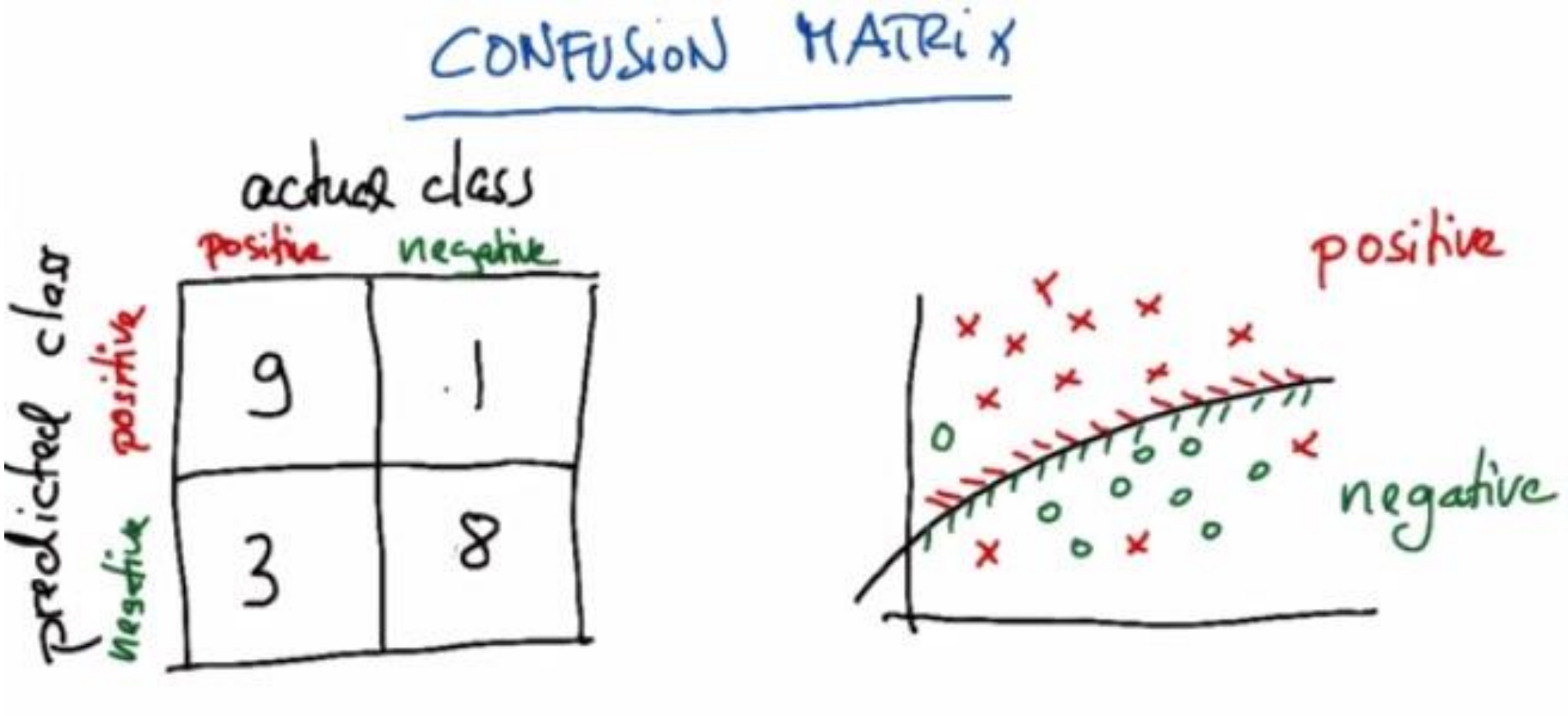
In [12]:
```
# H2O Gradient Boosting Machine with default settings
model <- h2o.gbm(x = features,
                 y = target,
                 training_frame = h_train,
                 seed = 1234)
```

H_2O automatically ignore columns with constant values

```
Warning message in .h2o.startModelJob(algo, params, h2oRestApiVersion):
“Dropping constant columns: [Feature 516, Feature 234, Feature 233, Feature 236, Feature 235, Feature 510,
 Feature 238, Feature 513, Feature 237, Feature 479, Feature 515, Feature 514, Feature 193, Feature 192, Fe
ature 195, Feature 194, Feature 075, Feature 230, Feature 232, Feature 231, Feature 529, Feature 244, Featu
re 365, Feature 401, Feature 400, Feature 006, Feature 403, Feature 402, Feature 405, Feature 404, Feature
 241, Feature 482, Feature 243, Feature 242, Feature 180, Feature 179, Feature 459, Feature 050, Feature 05
3, Feature 450, Feature 210, Feature 331, Feature 452, Feature 330, Feature 451, Feature 191, Feature 070,
 Feature 190, Feature 506, Feature 505, Feature 508, Feature 507, Feature 509, Feature 465, Feature 343, Fe
ature 464, Feature 467, Feature 466, Feature 227, Feature 348, Feature 502, Feature 504, Feature 503, Featu
re 463, Feature 187, Feature 462, Feature 399, Feature 277, Feature 398, Feature 315, Feature 314, Feature
 316, Feature 150, Feature 395, Feature 397, Feature 396, Feature 329, Feature 323, Feature 326, Feature 20
7, Feature 328, Feature 327, Feature 285, Feature 043, Feature 539, Feature 538, Feature 531, Feature 014,
 Feature 376, Feature 530, Feature 258, Feature 379, Feature 533, Feature 257, Feature 499, Feature 532, Fe
ature 535, Feature 259, Feature 534, Feature 537, Feature 415, Feature 536, Feature 371, Feature 370, Featu
re 373, Feature 098, Feature 372, Feature 375, Feature 374, Feature 267, Feature 266, Feature 423, Feature
 380, Feature 261, Feature 382, Feature 260, Feature 381, Feature 142, Feature 263, Feature 262, Feature 26
5, Feature 264].
”
```

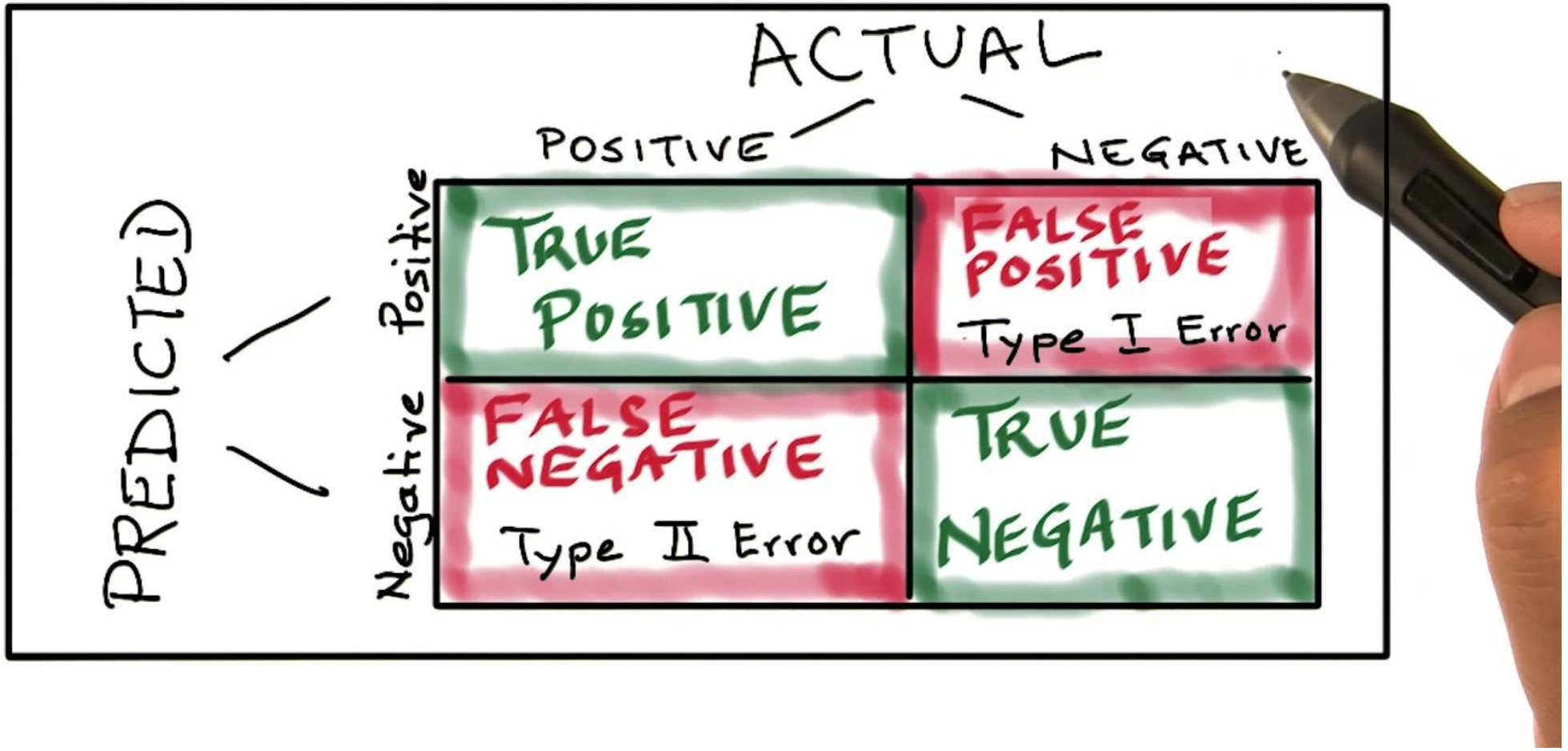
```
  |======================================================================| 100%
```

Classification Performance – Confusion Matrix



H2O.ai

Confusion Matrix



H2O.ai

In [13]:
```
# Print out model summary
summary(model)
```

```
Model Details:
==============

H2OBinomialModel: gbm
Model Key:  GBM_model_R_1492548973134_3
Model Summary:
  number_of_trees number_of_internal_trees model_size_in_bytes min_depth
1              50                        50               11653         5
  max_depth mean_depth min_leaves max_leaves mean_leaves
1         5    5.00000          7         18    12.56000

H2OBinomialMetrics: gbm
** Reported on training data. **

MSE:  0.004654337
RMSE:  0.0682227
LogLoss:  0.03489075
Mean Per-Class Error:  0
AUC:  1
Gini:  1

Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:
         -1  1    Error     Rate
-1     1028  0 0.000000  =0/1028
1         0 77 0.000000    =0/77
Totals 1028 77 0.000000  =0/1105

Maximum Metrics: Maximum metrics at their respective thresholds
                        metric threshold    value idx
1                       max f1  0.510169 1.000000  75
2                       max f2  0.510169 1.000000  75
3                 max f0point5  0.510169 1.000000  75
4                 max accuracy  0.510169 1.000000  75
5                max precision  0.929811 1.000000   0
```

Looking at metrics based on training (in-sample) data only
It doesn't represent out-of-bag performance

```
Variable Importances: (Extract with `h2o.varimp`)
================================================

Variable Importances:
     variable relative_importance scaled_importance percentage
1 Feature 060           13.682457          1.000000   0.059038
2 Feature 065            7.080432          0.517483   0.030551
3 Feature 122            6.408298          0.468359   0.027651
4 Feature 017            5.218538          0.381404   0.022517
5 Feature 133            4.659705          0.340561   0.020106

---
       variable relative_importance scaled_importance percentage
463 Feature 579             0.000000          0.000000   0.000000
464 Feature 582             0.000000          0.000000   0.000000
465 Feature 584             0.000000          0.000000   0.000000
466 Feature 585             0.000000          0.000000   0.000000
467 Feature 586             0.000000          0.000000   0.000000
468 Feature 590             0.000000          0.000000   0.000000
```

Users could reduce number of features based on these findings

In [14]:
```
# Check performance on test set
h2o.performance(model, h_test)
```

```
H2OBinomialMetrics: gbm

MSE:  0.05435156
RMSE:  0.2331342
LogLoss:  0.2154845
Mean Per-Class Error:  0.3260536
AUC:  0.7404427
Gini:  0.4808855

Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:
        -1  1    Error     Rate
-1     393 42 0.096552  =42/435
1       15 12 0.555556   =15/27
Totals 408 54 0.123377  =57/462

Maximum Metrics: Maximum metrics at their respective thresholds
                        metric threshold    value idx
1                       max f1  0.056665 0.296296  53
2                       max f2  0.038989 0.382653  86
3                 max f0point5  0.084880 0.259259  26
4                 max accuracy  0.584555 0.939394   0
5                max precision  0.084880 0.259259  26
6                   max recall  0.007542 1.000000 375
7              max specificity  0.584555 0.997701   0
8             max absolute_mcc  0.056665 0.254007  53
9   max min_per_class_accuracy  0.026387 0.703704 137
10 max mean_per_class_accuracy  0.023328 0.711750 153
```

Metrics based on test (out-of-bag) samples

Making Predictions

In [15]:
```
# Use the model for predictions
yhat_test <- h2o.predict(model, h_test)
```

```
  |======================================================================| 100%
```

In [16]:
```
# Show first 10 rows
head(yhat_test, 10)
```

predict	p-1	p1
-1	0.9891922	0.010807836
-1	0.9667993	0.033200722
-1	0.9404231	0.059576900
-1	0.8932517	0.106748270
-1	0.9900038	0.009996166
-1	0.9528278	0.047172190
-1	0.8492997	0.150700332
-1	0.9855647	0.014435350
-1	0.9465926	0.053407411
-1	0.9617147	0.038285313

H_2O returns predicted class as well as probabilities of each class

Advanced H_2O Usage – Random Grid Search

- Link to Jupyter Notebook
  - https://github.com/woobe/h2o_tutorials/blob/master/use_cases/predictive_maintenance/step_02_random_grid_search.ipynb
- Using Random Grid Search to fine-tune hyper-parameters

Build GBM Models using Random Grid Search and Extract the Best Model

In [12]:
```
# Define the criteria for random grid search
search_criteria = list(strategy = "RandomDiscrete",
                       max_models = 10,
                       seed = 1234)
```

In [13]:
```
# Define the range of hyper-parameters for grid search
hyper_params <- list(
    sample_rate = c(0.6, 0.7, 0.8, 0.9),
    col_sample_rate = c(0.6, 0.7, 0.8, 0.9),
    max_depth = c(4, 5, 6)
)
```

In [14]:
```
# Set up grid search
# Add a seed for reproducibility
rand_grid <- h2o.grid(

    # Core parameters for model training
    x = features,
    y = target,
    training_frame = h_train,
    ntrees = 500,
    learn_rate = 0.05,
    balance_classes = TRUE,
    seed = 1234,

    # Settings for Cross-Validation
    nfolds = 5,
    fold_assignment = "Stratified",

    # Parameters for early stopping
    stopping_metric = "mean_per_class_error",
    stopping_rounds = 15,
    score_tree_interval = 1,

    # Parameters for grid search
    grid_id = "rand_grid",
    hyper_params = hyper_params,
    algorithm = "gbm",
    search_criteria = search_criteria

)
```

API for performing a Random Grid Search

In [15]:
```
# Sort and show the grid search results
rand_grid <- h2o.getGrid(grid_id = "rand_grid", sort_by = "mean_per_class_error", decreasing = FALSE)
print(rand_grid)
```

```
H2O Grid Details
================

Grid ID: rand_grid
Used hyper parameters:
  -  col_sample_rate
  -  max_depth
  -  sample_rate
Number of models: 10
Number of failed models: 0

Hyper-Parameter Search Summary: ordered by increasing mean_per_class_error
   col_sample_rate max_depth sample_rate         model_ids mean_per_class_error
1              0.6         4         0.8 rand_grid_model_2   0.3678874627318207
2              0.6         4         0.7 rand_grid_model_1  0.37603592905149325
3              0.6         6         0.6 rand_grid_model_5   0.3794658648744252
4              0.8         4         0.8 rand_grid_model_6   0.3818725049269796
5              0.7         4         0.7 rand_grid_model_8   0.3851066248926171
6              0.8         4         0.7 rand_grid_model_9    0.389534589923695
7              0.8         4         0.6 rand_grid_model_7  0.38985042195158925
8              0.9         6         0.7 rand_grid_model_4   0.4093562079943403
9              0.8         6         0.6 rand_grid_model_0  0.41636136237303556
10             0.9         6         0.8 rand_grid_model_3  0.42382763151245645
```

Grid search results sorted by specific metric. Best model on top.

In [16]:
```
# Extract the best model from random grid search
best_model_id <- rand_grid@model_ids[[1]] # top of the list
best_model <- h2o.getModel(best_model_id)
print(best_model)
```

```
Model Details:
==============

H2OBinomialModel: gbm
Model ID:  rand_grid_model_2
Model Summary:
  number_of_trees number_of_internal_trees model_size_in_bytes min_depth
1              61                       61               15119         4
  max_depth mean_depth min_leaves max_leaves mean_leaves
1         4    4.00000         11         16    13.77049
```

Extract the best model

```
H2OBinomialMetrics: gbm
** Reported on cross-validation data. **
** 5-fold cross-validation on training data (Metrics computed for combined holdout predictions) **

MSE:  0.06572745
RMSE:  0.2563737
LogLoss:  0.2767561
Mean Per-Class Error:  0.3678875
AUC:  0.6739855
Gini:  0.3479711

Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:
        -1   1    Error       Rate
-1     819 209 0.203307  =209/1028
1       41  36 0.532468    =41/77
Totals 860 245 0.226244  =250/1105

Maximum Metrics: Maximum metrics at their respective thresholds
                        metric threshold    value idx
1                        max f1  0.028906 0.223602 174
2                        max f2  0.011951 0.328467 297
3                  max f0point5  0.082901 0.204778  44
4                  max accuracy  0.164703 0.929412   0
5                 max precision  0.084812 0.229167  40
6                    max recall  0.001585 1.000000 391
7                max specificity  0.164703 0.999027   0
8               max absolute_mcc  0.028906 0.161951 174
9    max min_per_class_accuracy  0.018432 0.610390 243
10 max mean_per_class_accuracy  0.028906 0.632113 174

Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/F>, xva
l=<T/F>)`
Cross-Validation Metrics Summary:
                               mean          sd   cv_1_valid  cv_2_valid
accuracy                   0.85497105 0.028622998    0.8867925  0.78629035
auc                        0.70107853 0.040595975   0.79950494  0.64812684
err                        0.14502896 0.028622998   0.11320755  0.21370968
```

Performance metrics based on 5-fold cross-validation

Comparison – Default vs. Tuned

Still not perfect but better than default settings

```
# Check performance on test set
h2o.performance(model, h_test)
```

```
H2OBinomialMetrics: gbm

MSE:  0.05435156
RMSE:  0.2331342
LogLoss:  0.2154845
Mean Per-Class Error:  0.3260536
AUC:  0.7404427
Gini:  0.4808855

Confusion Matrix (vertical: actual; acr
        -1  1    Error     Rate
-1     393 42 0.096552  =42/435
1       15 12 0.555556   =15/27
Totals 408 54 0.123377  =57/462
```

GBM with Default Settings

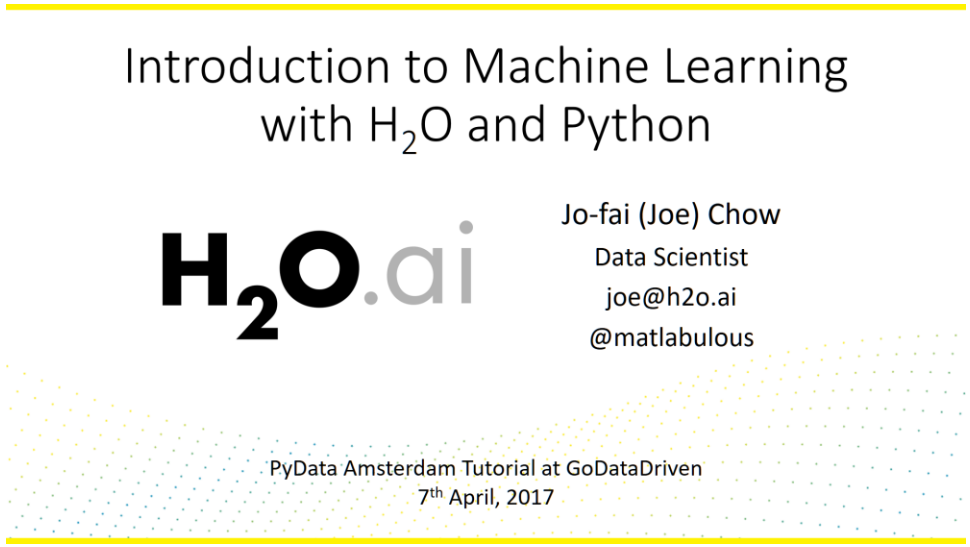
```
# Check performance on test set
h2o.performance(best_model, h_test)
```

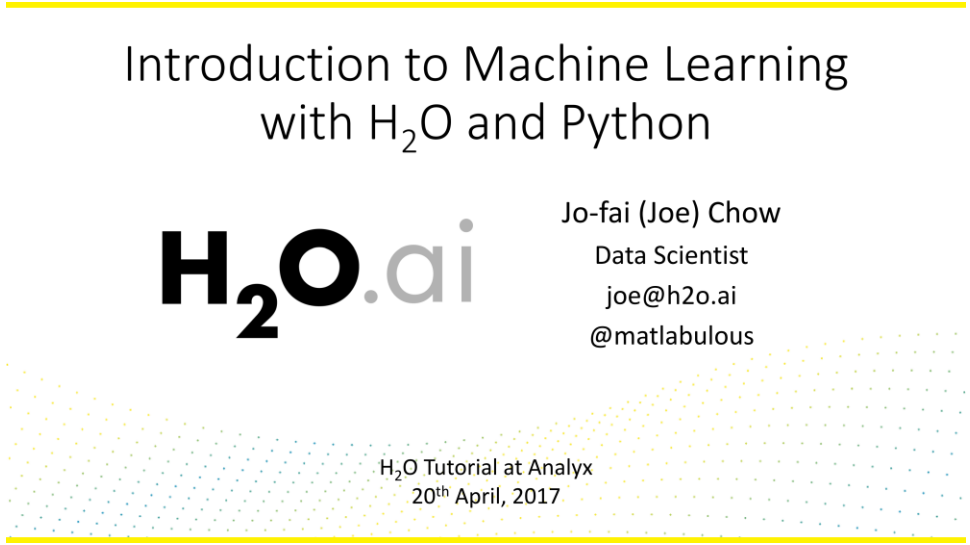
```
H2OBinomialMetrics: gbm

MSE:  0.05339771
RMSE:  0.2310794
LogLoss:  0.2138174
Mean Per-Class Error:  0.2693487
AUC:  0.7553427
Gini:  0.5106854

Confusion Matrix (vertical: actual; acr
        -1  1    Error     Rate
-1     394 41 0.094253  =41/435
1       12 15 0.444444   =12/27
Totals 406 56 0.114719  =53/462
```

GBM with Random Grid Search

H_2O Tutorials

- Introduction to Machine Learning with H_2O and Python
  - Basic Extract, Transform and Load (ETL)
  - Supervised Learning
  - Parameters Tuning
  - Stacking
- GitHub Repository
  - bit.ly/joe_h2o_tutorials
  - R Code Examples (soon)





H2O.ai

Outlier Detection

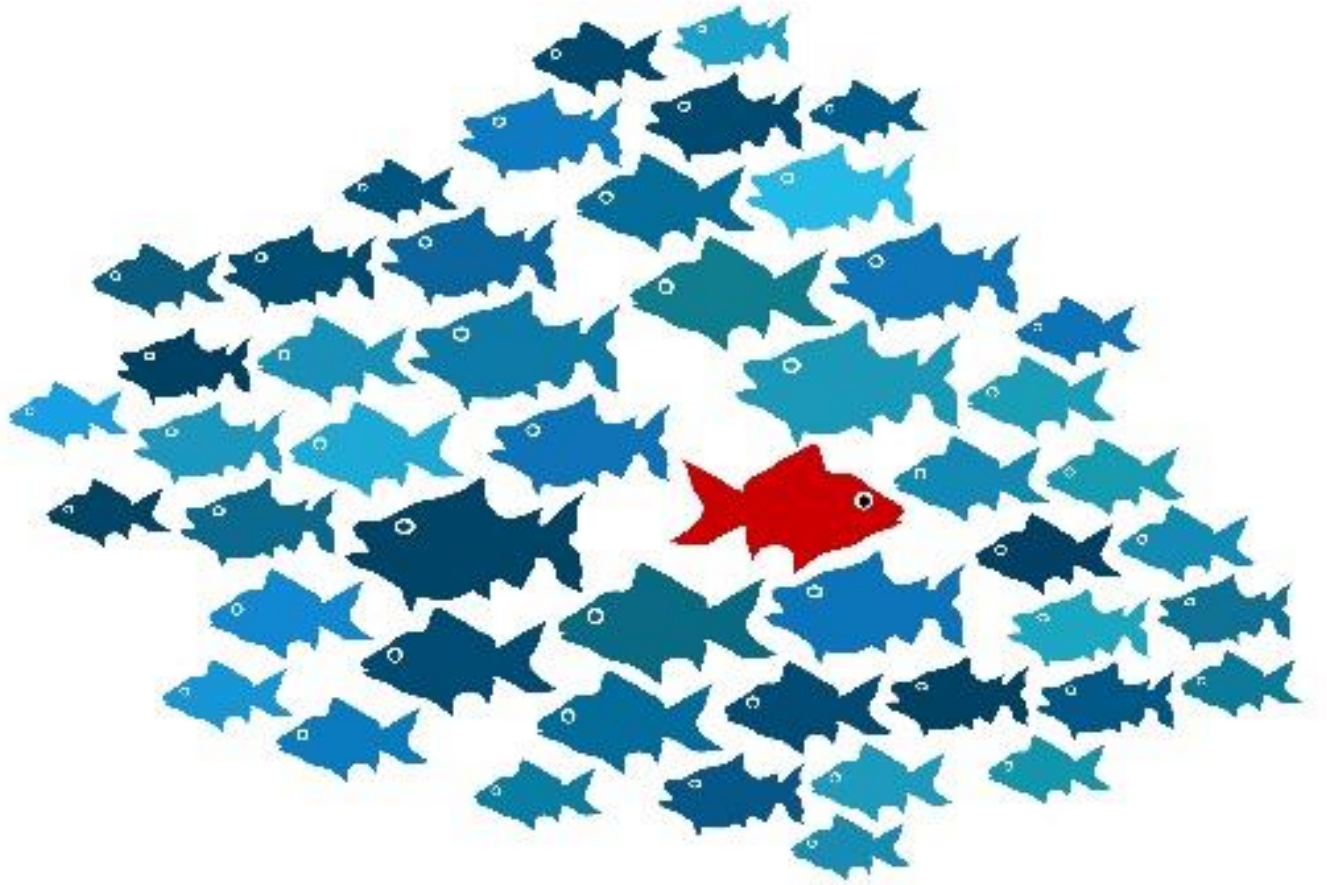


Photo credit: www.dbta.com

Outlier Detection – MNIST Dataset

Photo credit: http://www.opendeep.org/v0.0.5/docs/tutorial-classifying-handwritten-mnist-images

Outlier Detection – MNIST Dataset

- 784 Inputs
  - 28 x 28 = 784 pixels
- 1 Output
  - 0, 1, 2, 3, 4, 5, 6, 7, 8 or 9
- File (from Kaggle)
  - Train (42k Records)
  - kaggle_mnist_train.csv.gz
- Source
  - https://www.kaggle.com/c/digit-recognizer/data

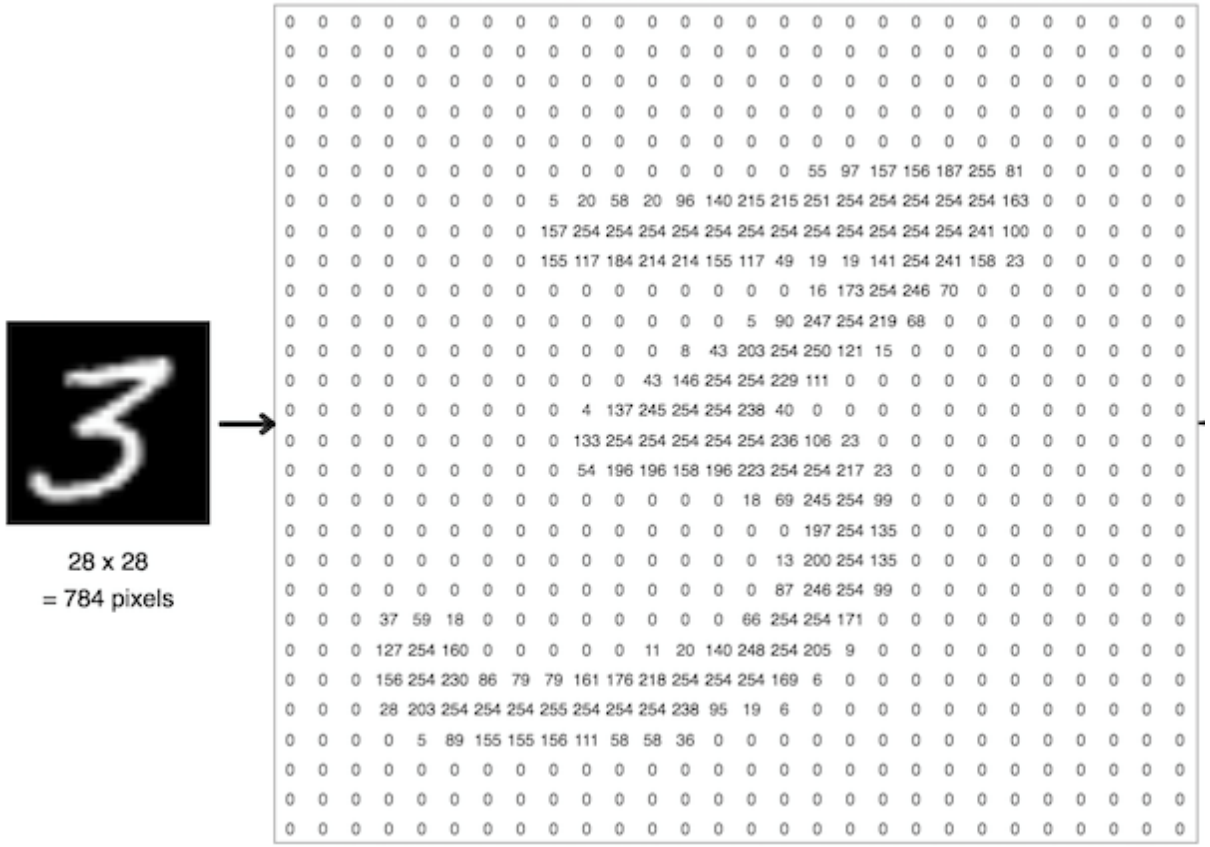
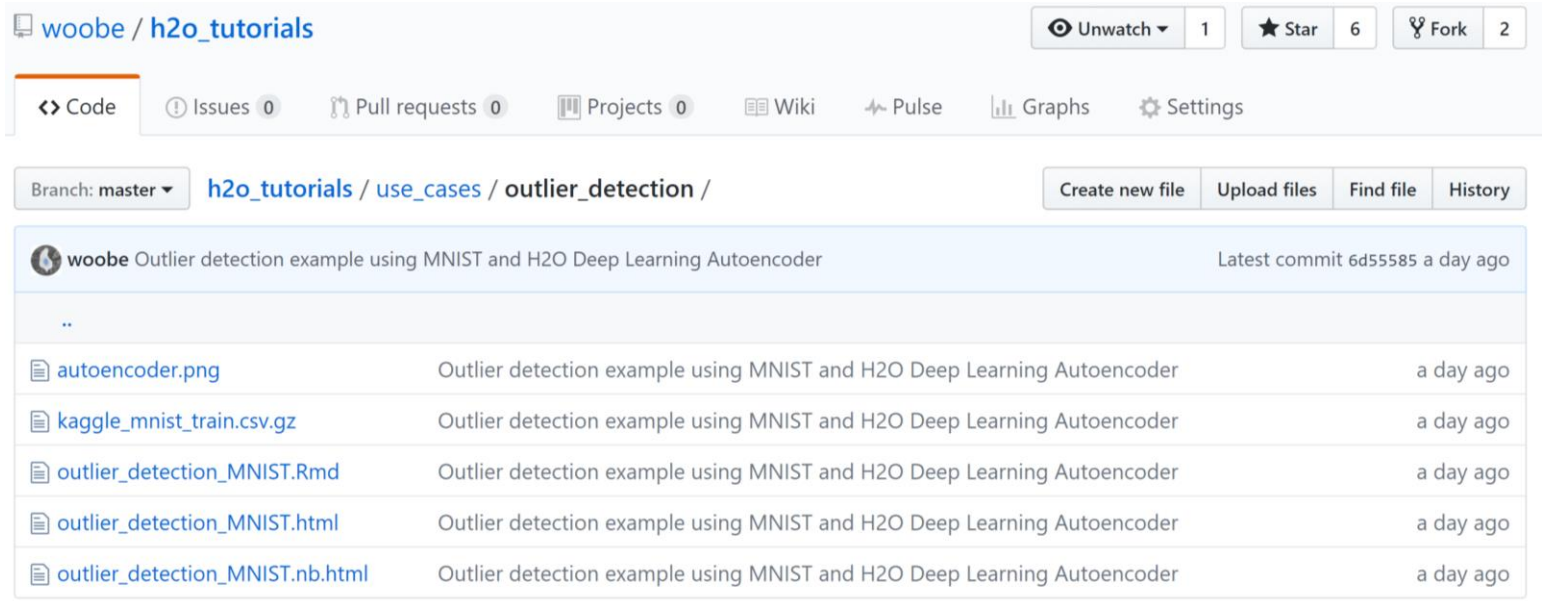


Photo credit: https://ml4a.github.io/ml4a/neural_networks/

H2O.ai

Advanced H_2O Usage – Random Grid Search

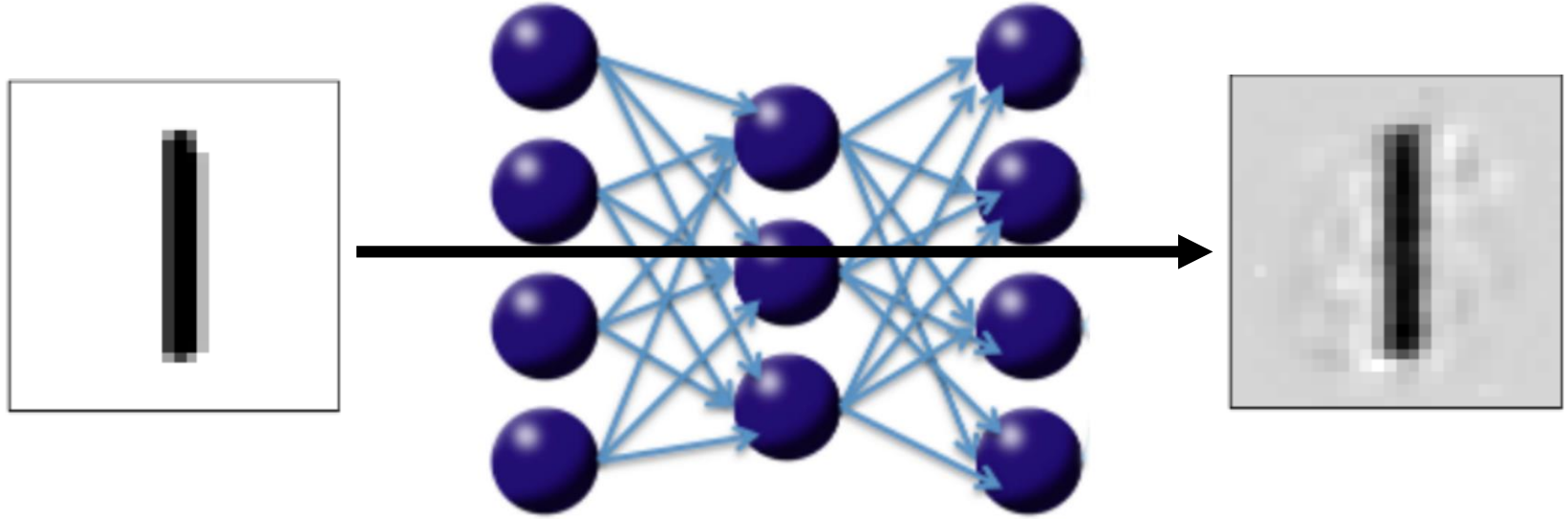
- Link to Data and Code
  - https://github.com/woobe/h2o_tutorials/tree/master/use_cases/outlier_detection



Outlier Detection on MNIST with H2O Deep Learning

This tutorial shows how a Deep Learning Auto-Encoder model can be used to find outliers in a dataset.

Consider the following three-layer neural network with one hidden layer and the same number of input neurons (features) as output neurons. The loss function is the MSE between the input and the output. Hence, the network is forced to learn the identity via a nonlinear, reduced representation of the original data. Such an algorithm is called a deep autoencoder; these models have been used extensively for unsupervised, layer-wise pretraining of supervised deep learning tasks, but here we consider the autoencoder's application for discovering anomalies in data.



We use the well-known MNIST dataset of hand-written digits, where each row contains the 28^2=784 raw gray-scale pixel values from 0 to 255 of the digitized digits (0 to 9).

Import data
(directly from a gz)

```
# Load MNIST (Kaggle Version)
# The data consists of 784 (=28^2) pixel values per row, with (gray-scale) values from 0 to 255.
# The first column is the response (a label in 0,1,2,...,9).
mnist <- h2o.importFile(path = "kaggle_mnist_train.csv.gz")
```

```
# We do unsupervised training, so we can drop the response column.
mnist <- mnist[, -1]
```

Remove the label (not needed
for unsupervised learning)

Finding outliers - ugly hand-written digits

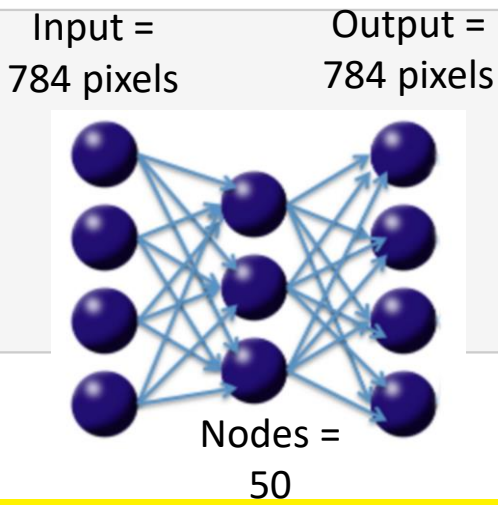
We train a Deep Learning Auto-Encoder to learn a compressed (low-dimensional) non-linear representation of the dataset, hence learning the intrinsic structure of the training dataset. The auto-encoder model is then used to transform all test set images to their reconstructed images, by passing through the lower-dimensional neural network. We then find outliers in a test dataset by comparing the reconstruction of each scanned digit with its original pixel values. The idea is that a high reconstruction error of a digit indicates that the test set point doesn't conform to the structure of the training data and can hence be called an outlier.

Step 1 - Learn what's normal from the training data

Train unsupervised Deep Learning autoencoder model on the training dataset. For simplicity, we train a model with 1 hidden layer of 50 Tanh neurons to create 50 non-linear features with which to reconstruct the original dataset. We learned from the Dimensionality Reduction tutorial that 50 is a reasonable choice. For simplicity, we train the auto-encoder for only 1 epoch (one pass over the data). We explicitly include constant columns (all white background) for the visualization to be easier.

Note that the response column is ignored (it is only required because of a shared DeepLearning code framework).

```
autoencoder_model <- h2o.deeplearning(x = 1:784,
                                      training_frame = mnist,
                                      hidden = c(50),
                                      epoch = 1,
                                      activation = "Tanh",
                                      autoencoder = TRUE,
                                      ignore_const_cols = FALSE)
```



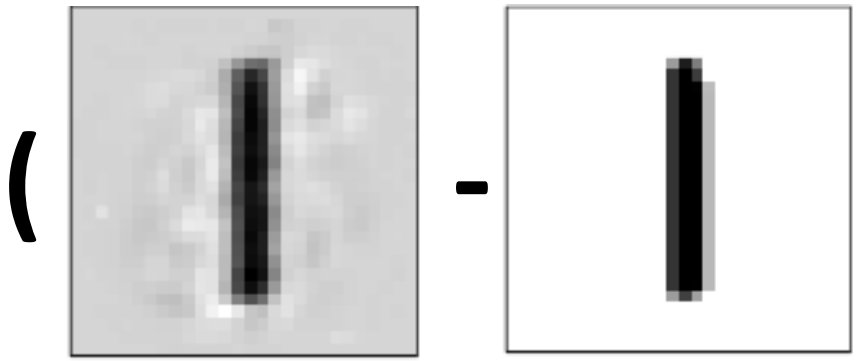
H2O.ai

Quantify the average reconstruction error per sample

Step 2 - Find outliers in the data

The Anomaly app computes the per-row reconstruction error for the MNIST data set. It passes MNIST data through the autoencoder model and computes mean square error (MSE) for each row.

```
# Use h2o.anomaly(...) to calculate per-row reconstruction error
rec_error <- h2o.anomaly(object = autoencoder_model, data = mnist)
rec_error <- as.data.frame(rec_error)
```



(-) / no. of pixels = Avg. Error

Step 3 - Visualize the good, the bad and the ugly

We will need a helper function for plotting handwritten digits (adapted from http://www.r-bloggers.com/the-essence-of-a-handwritten-digit/).

```
# Helper function
plotDigit <- function(mydata, rec_error) {
  len<-nrow(mydata)
  N<-ceiling(sqrt(len))
  op <- par(mfrow=c(N,N),pty='s',mar=c(1,1,1,1),xaxt='n',yaxt='n')
  for (i in 1:nrow(mydata)) {
    colors<-c('white','black')
    cus_col<-colorRampPalette(colors=colors)
    z<-array(mydata[i,],dim=c(28,28))
    z<-z[,28:1]
    image(1:28,1:28,z,main=paste0("rec_error: ", round(rec_error[i],4)),col=cus_col(256))
  }
  on.exit(par(op))
}

plotDigits <- function(data, rec_error, rows) {
 row_idx <- order(rec_error[,1],decreasing=F)[rows]
 my_rec_error <- rec_error[row_idx,]
 my_da <- as.matrix(as.data.frame(data))
     my_data <- my_da[row_idx,]
     plotDigit(my_data, my_rec_error)
}
```

Helper functions for plotting graphs only (not related to H_2O algorithms)

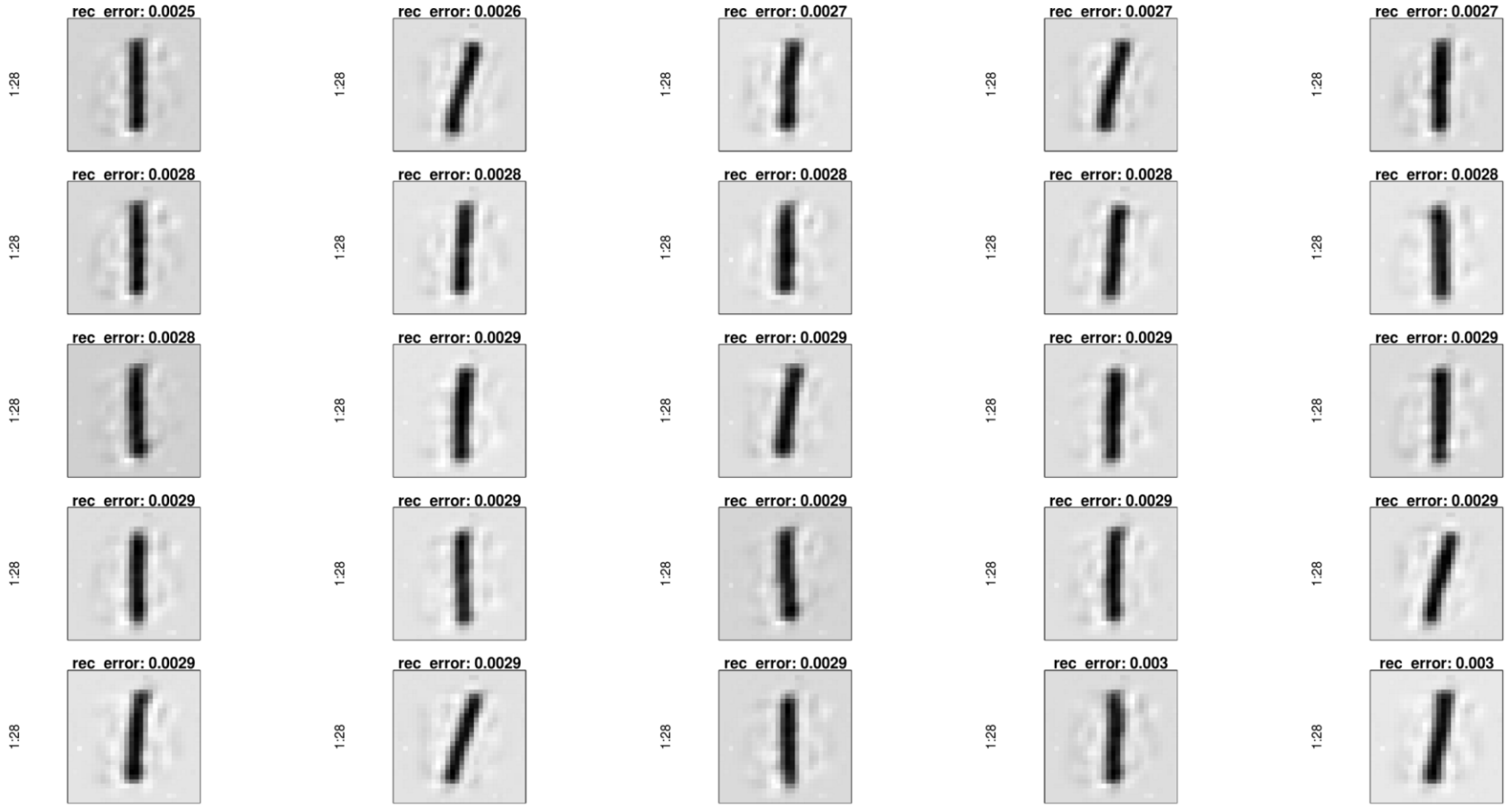
Let's look at samples with low/median/high reconstruction errors. We will now visualize the original MNIST image and their reconstructions obtained by propagating them through the narrow neural net.

```
recon <- h2o.predict(object = autoencoder_model, newdata = mnist)
```

The Good

Let's plot the 25 digits with lowest reconstruction error. First we plot the reconstruction, then the original scanned images.

```
plotDigits(recon, rec_error, c(1:25))
```



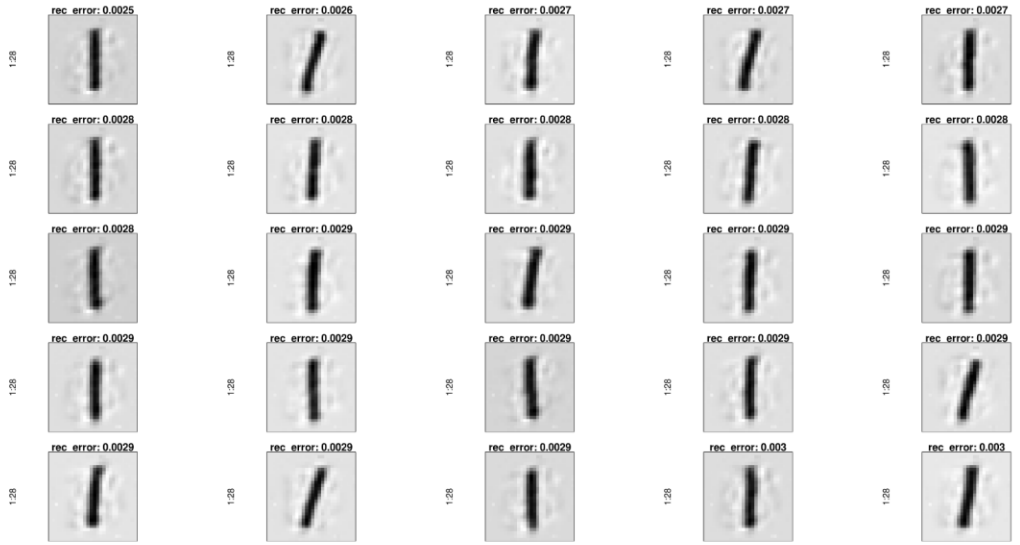
Reconstruction

Original Image

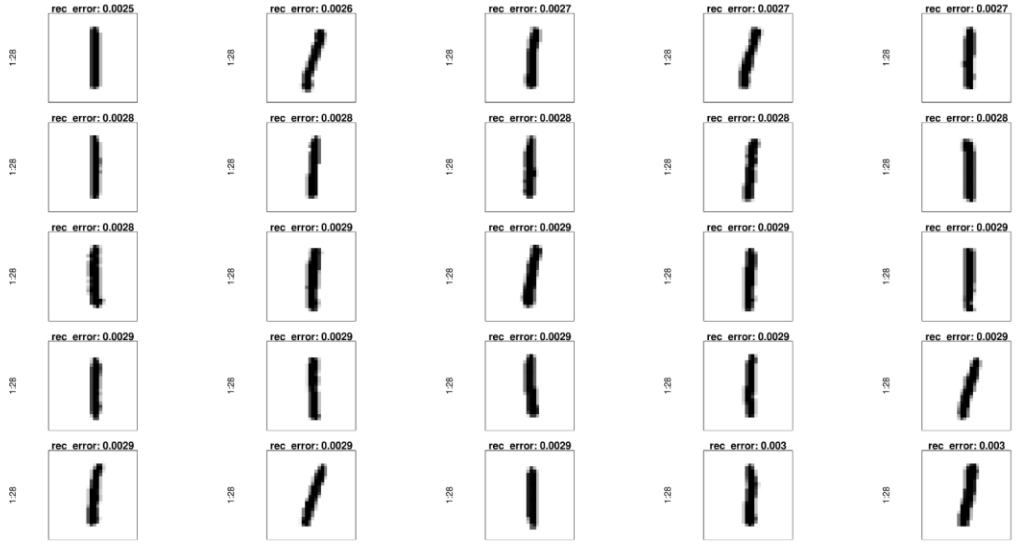
The Good

Let's plot the 25 digits with lowest reconstruction error. First we plot the reconstruction, then the original scanned images.

```
plotDigits(recon, rec_error, c(1:25))
```




```
plotDigits(mnist, rec_error, c(1:25))
```



Clearly, a well-written digit 1 appears in both the training and testing set, and is easy to reconstruct by the autoencoder with minimal reconstruction error. Nothing is as easy as a straight line.

Reconstruction

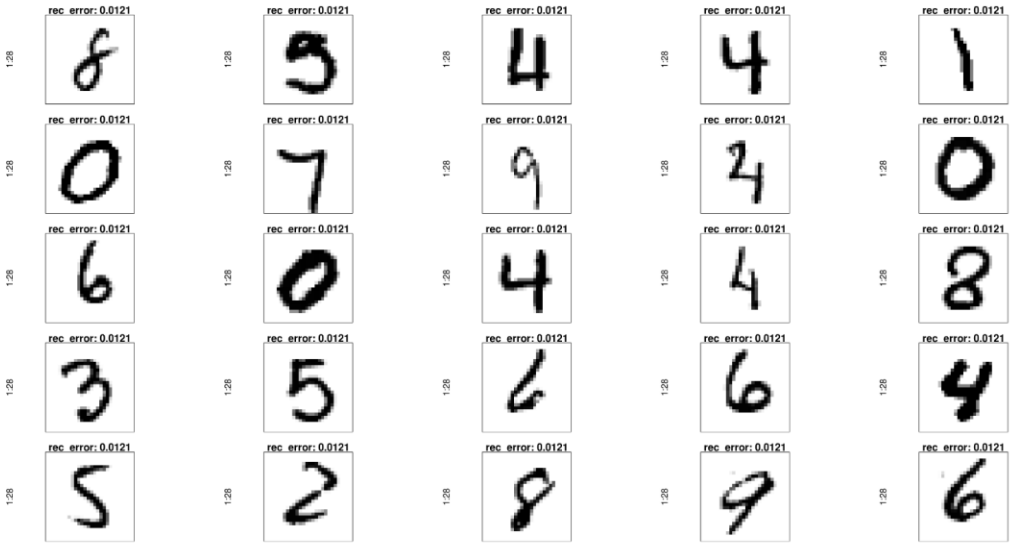
Original Image

The Bad

Now let's look at the 25 digits with median reconstruction error.

```
plotDigits(recon, rec_error, c(21001:21025))
```


```
plotDigits(mnist,   rec_error, c(21001:21025))
```



These test set digits look "normal" - it is plausible that they resemble digits from the training data to a large extent, but they do have some particularities that cause some reconstruction error.

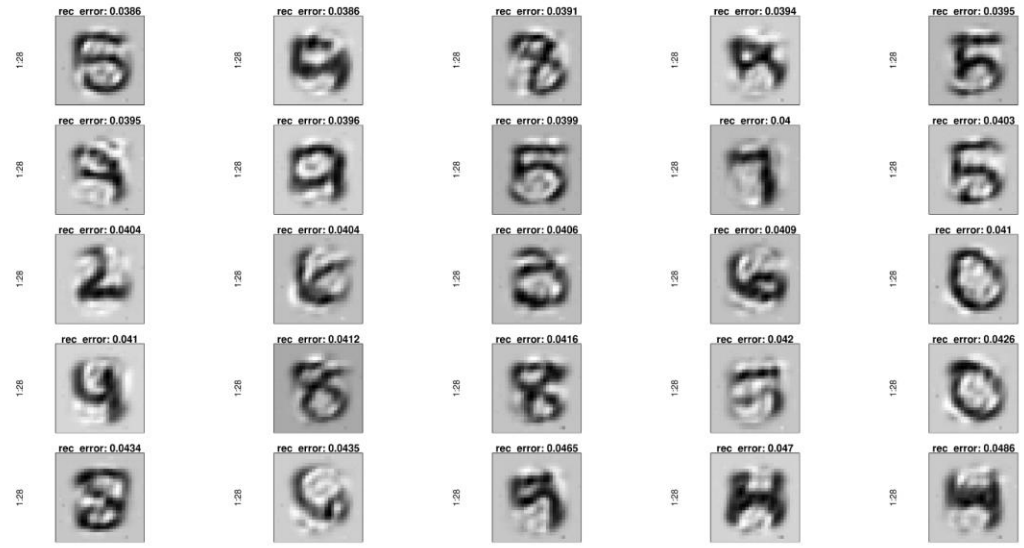
Reconstruction

Original Image

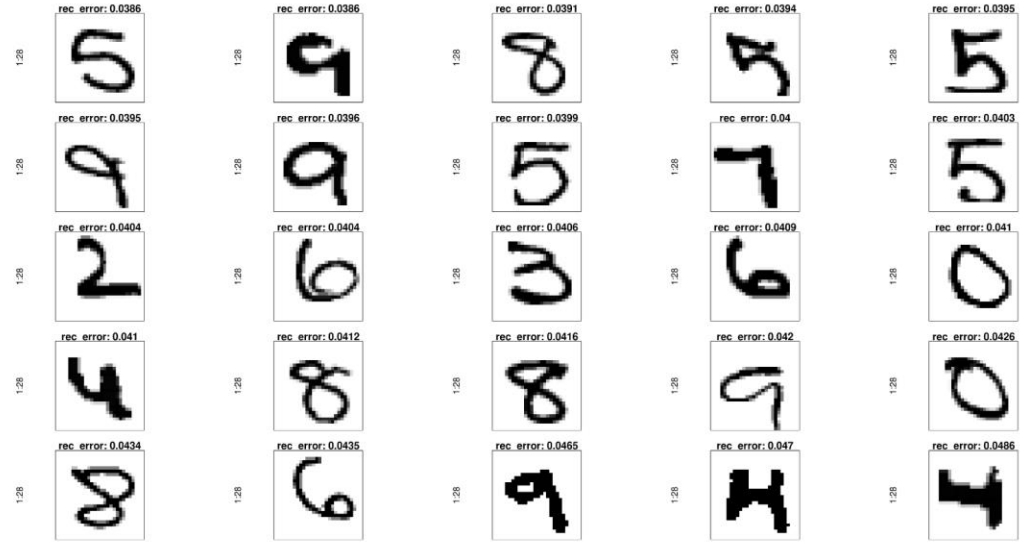
The Ugly

And here are the biggest outliers - The 25 digits with highest reconstruction error!

```
plotDigits(recon, rec_error, c(41976:42000))
```




```
plotDigits(mnist,   rec_error, c(41976:42000))
```



Now here are some pretty ugly digits that are plausibly not commonly found in the training data - some are even hard to classify by humans.

Conclusions

We were able to find outliers with H2O Deep Learning Auto-Encoder models. We would love to hear your usecase for Anomaly detection.

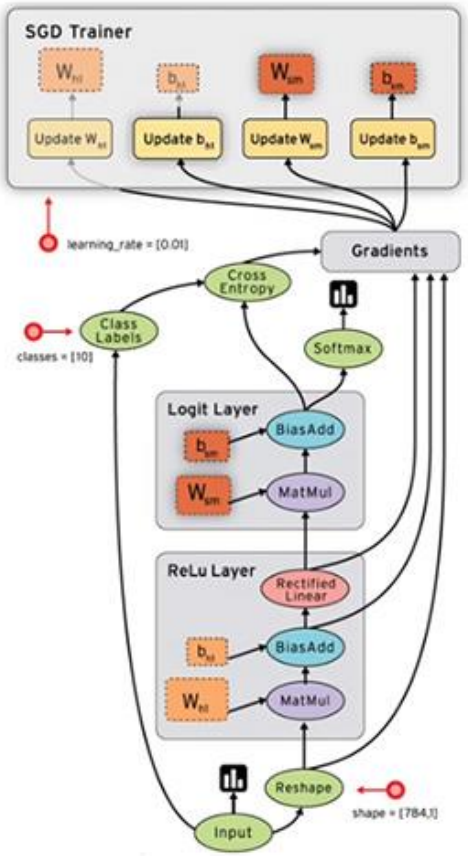
Note: Every run of DeepLearning results in different results since we use Hogwild! parallelization with intentional race conditions between threads. To get reproducible results at the expense of speed for small datasets, set `reproducible = TRUE` and specify a seed.

H_2O Deep Water

H_2O's Integration with TensorFlow, mxnet and Caffe

TensorFlow

- Open source machine learning framework by Google
- Python / C++ API
- TensorBoard
  - Data Flow Graph Visualization
- Multi CPU / GPU
  - v0.8+ distributed machines support
- Multi devices support
  - desktop, server and Android devices
- Image, audio and NLP applications
- **HUGE** Community
- Support for Spark, Windows …



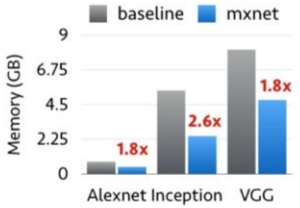
https://github.com/tensorflow/tensorflow

dmlc mxnet for Deep Learning

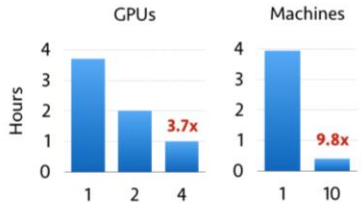
build passing | docs latest | license Apache 2.0

Portable

Efficient



Scalable



MXNet is a deep learning framework designed for both *efficiency* and *flexibility*. It allows you to ***mix*** the flavours of symbolic programming and imperative programming to ***maximize*** efficiency and productivity. In its core, a dynamic dependency scheduler that automatically parallelizes both symbolic and imperative operations on the fly. A graph optimization layer on top of that makes symbolic execution fast and memory efficient. The library is portable and lightweight, and it scales to multiple GPUs and multiple machines.

MXNet is also more than a deep learning project. It is also a collection of blue prints and guidelines for building deep learning system, and interesting insights of DL systems for hackers.

MXNet now chosen by Amazon as Deep Learning Framework

By Geneva Clark | 2016-11-24

19 0

Share this magazine

Amazon has announced that it has chosen MXNet as its deep learning framework of choice for its web services(AWS). Amazon extensively uses machine learning in areas like fraud detection, abusive review detection, and book classification. Amazon also uses it in application areas such as text and speech recognition, autonomous drones etc...

https://github.com/dmlc/mxnet
https://www.zeolearn.com/magazine/amazon-to-use-mxnet-as-deep-learning-framework

H2O.ai

Caffe

- **C**onvolution **A**rchitecture **F**or **F**eature **E**xtraction (CAFFE)
- Pure C++ / CUDA architecture for deep learning
- Command line, Python and MATLAB interface
- Model Zoo
  - Open collection of models

https://docs.google.com/presentation/d/1UeKXVgRvvxg9OUdh_UiC5G71UMscNPlvArsWER41PsU/

H2O.ai

H_2O Deep Learning

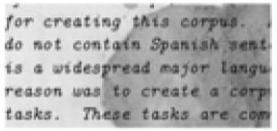
CIFAR-10 Competition Winners: Interviews with Dr. Ben Graham, Phil Culliton, & Zygmunt Zając

Triskelion | 01.02.2015

READ MORE

"I did really like H2O's deep learning implementation in R, though - the interface was great, the back end extremely easy to understand, and it was scalable and flexible. Definitely a tool I'll be going back to."

Kaggle challenge 2nd place winner Colin Priest



Completed • Knowledge • 161 teams

Denoising Dirty Documents

Mon 1 Jun 2015 – Mon 5 Oct 2015 (3 months ago)

READ MORE

"For my final competition submission I used an ensemble of models, including 3 deep learning models built with R and h2o."

H_2O.ai

Both TensorFlow and H_2O are widely used

The usage of Hadoop/Big Data tools grew to 39%, up from 29% in 2015 (and 17% in 2014), driven by Apache Spark, MLlib (Spark Machine Learning Library) and H2O.
See also

- KDnuggets interview with Spark Creator Matei Zaharia
- KDnuggets interview with Arno Candel, H2O.ai on How to Quick Start Deep Learning with H2O

http://www.kdnuggets.com

H2O and TensorFlow are tied

usage of Deep Learning tools in past year

Deep Learning Tools & Platforms: Caffe (63)	0%
Cuda-convnet (22)	0%
Convnet.js (8)	0%
carch (2)	0%
Deeplearning4j (46)	0%
H2O (190)	1%
MATLAB Deep Learning Toolbox (57)	0%
Microsoft CNTK (25)	0%
mxnet (16)	0%
Nervana (2)	0%
Tensorflow (190)	1%
Theano ecosystem including Keras, Lasagne, Pylearn2 (140)	1%
Torch (28)	0%
Veles (2)	0%
Other Deep Learning Tools (104)	1%

H_2O.ai

TensorFlow , MXNet, Caffe and H_2O DL democratize the power of deep learning.

H_2O platform democratizes artificial intelligence & big data science.

There are other open source deep learning libraries like Theano and Torch too.

Let's have a party, this will be fun!

H_2O.ai


H2O.ai
Products
Industries
Customers
Community
Documentation
Download
Deep Water
H2O.ai
Caffe
mxnet
TensorFlow

Deep Water

Next-Gen Distributed Deep Learning with H_2O

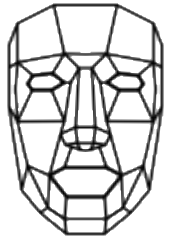
One Interface - GPU Enabled - Significant Performance Gains

Inherits All H_2O Properties in Scalability, Ease of Use and Deployment

TensorFlow Caffe H_2O.ai mxnet

Deep Water

H_2O integrates with existing GPU backends for significant performance gains



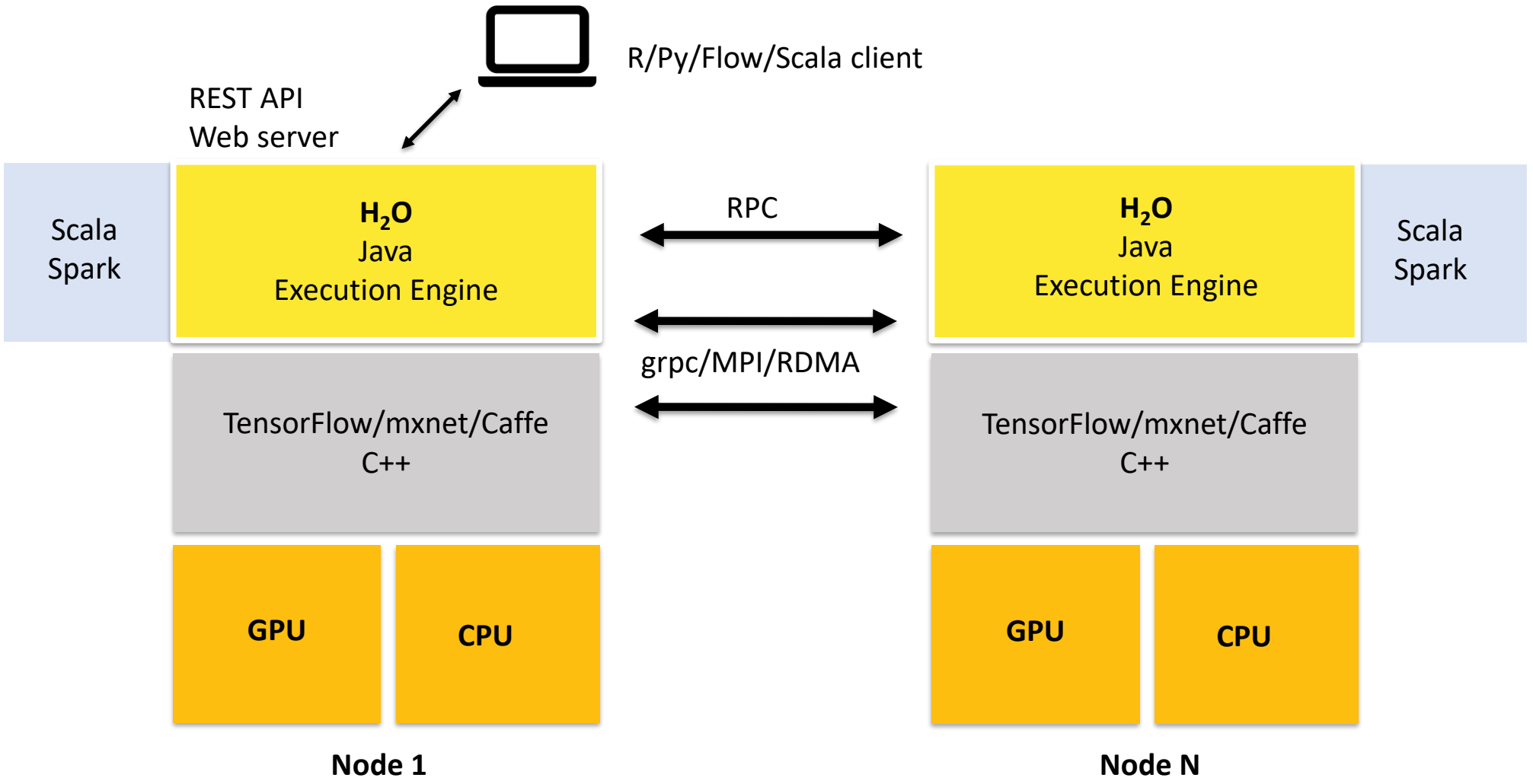
Convolutional Neural Networks enabling **Image, video, speech recognition**

Recurrent Neural Networks enabling **natural language processing, sequences, time series,** and more

Hybrid Neural Network Architectures enabling **speech to text translation, image captioning, scene parsing** and more

H_2O.ai

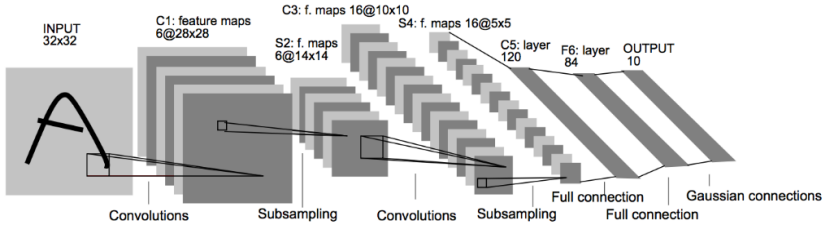
Deep Water Architecture



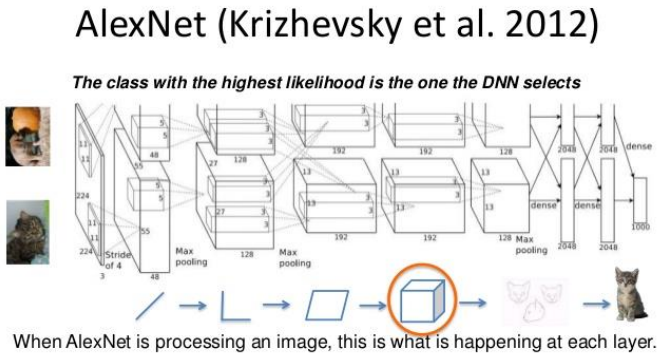
H2O.ai

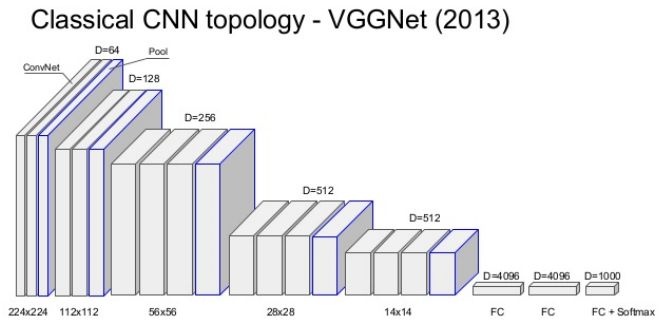
Available Networks in Deep Water

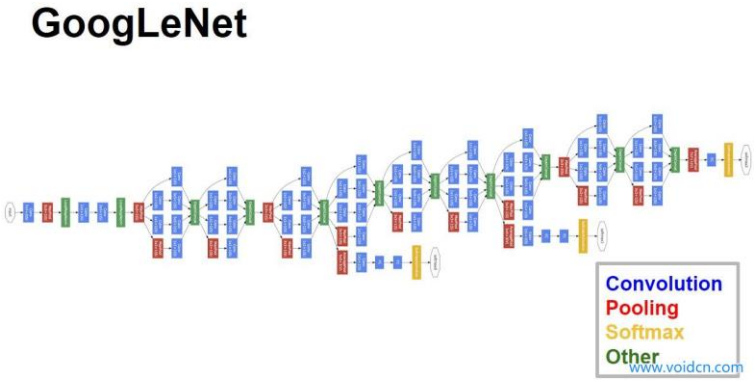
- LeNet
- AlexNet
- VGGNet
- Inception (GoogLeNet)
- ResNet (Deep Residual Learning)
- Build Your Own

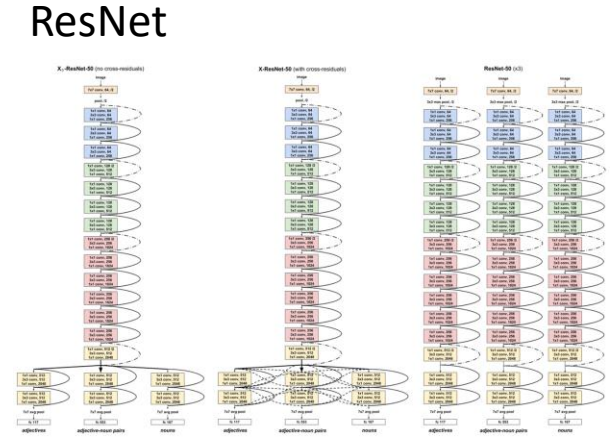


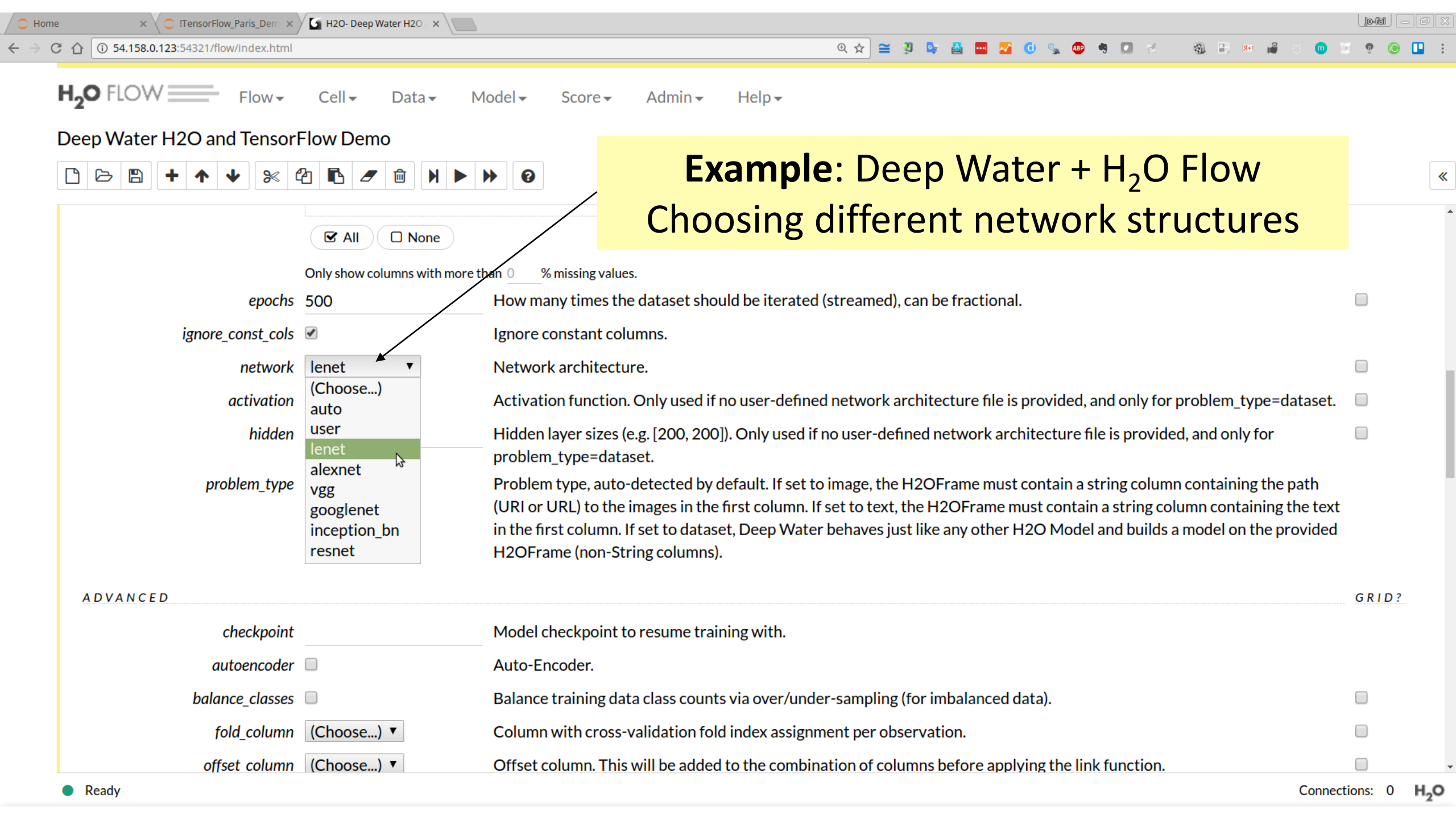
CNN called LeNet by Yann LeCun (1998)

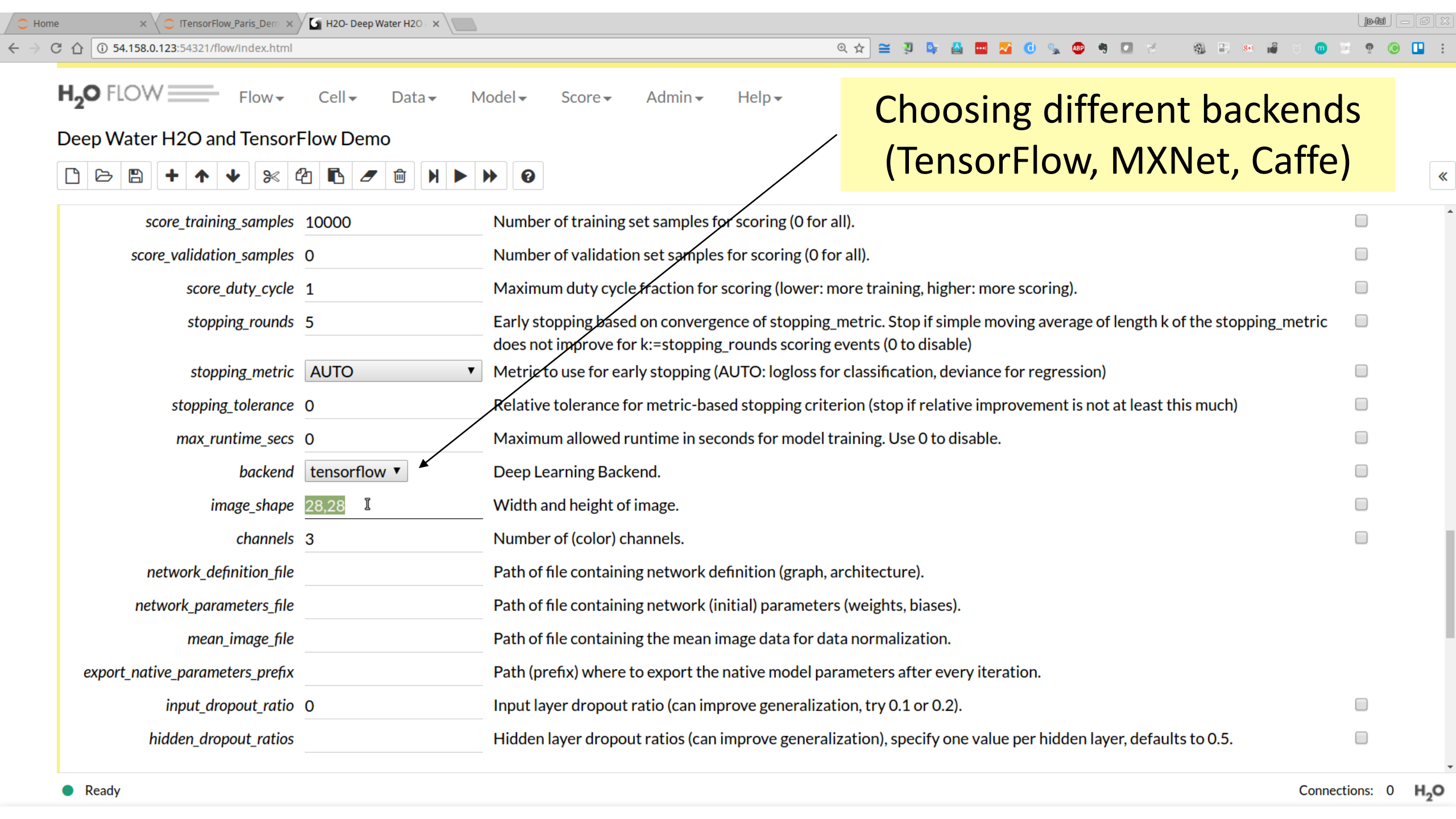









Home
ITensorFlow_Paris_Dem
H2O- Deep Water H2O
54.158.0.123:54321/flow/index.html
H2O FLOW
Flow
Cell
Data
Model
Score
Admin
Help
Deep Water H2O and TensorFlow Demo
Example: Deep Water + H2O Flow
Choosing different network structures
All
None
Only show columns with more than 0 % missing values.
epochs
500
How many times the dataset should be iterated (streamed), can be fractional.
ignore_const_cols
Ignore constant columns.
network
lenet
Network architecture.
activation
Activation function. Only used if no user-defined network architecture file is provided, and only for problem_type=dataset.
hidden
Hidden layer sizes (e.g. [200, 200]). Only used if no user-defined network architecture file is provided, and only for problem_type=dataset.
problem_type
Problem type, auto-detected by default. If set to image, the H2OFrame must contain a string column containing the path (URI or URL) to the images in the first column. If set to text, the H2OFrame must contain a string column containing the text in the first column. If set to dataset, Deep Water behaves just like any other H2O Model and builds a model on the provided H2OFrame (non-String columns).
(Choose...)
auto
user
lenet
alexnet
vgg
googlenet
inception_bn
resnet
ADVANCED
GRID?
checkpoint
Model checkpoint to resume training with.
autoencoder
Auto-Encoder.
balance_classes
Balance training data class counts via over/under-sampling (for imbalanced data).
fold_column
(Choose...)
Column with cross-validation fold index assignment per observation.
offset_column
(Choose...)
Offset column. This will be added to the combination of columns before applying the link function.
Ready
Connections: 0
H2O


Home
ITensorFlow_Paris_Dem
H2O- Deep Water H2O
54.158.0.123:54321/flow/index.html
H2O FLOW
Flow
Cell
Data
Model
Score
Admin
Help
Deep Water H2O and TensorFlow Demo
Choosing different backends
(TensorFlow, MXNet, Caffe)
score_training_samples 10000 Number of training set samples for scoring (0 for all).
score_validation_samples 0 Number of validation set samples for scoring (0 for all).
score_duty_cycle 1 Maximum duty cycle fraction for scoring (lower: more training, higher: more scoring).
stopping_rounds 5 Early stopping based on convergence of stopping_metric. Stop if simple moving average of length k of the stopping_metric does not improve for k:=stopping_rounds scoring events (0 to disable)
stopping_metric AUTO Metric to use for early stopping (AUTO: logloss for classification, deviance for regression)
stopping_tolerance 0 Relative tolerance for metric-based stopping criterion (stop if relative improvement is not at least this much)
max_runtime_secs 0 Maximum allowed runtime in seconds for model training. Use 0 to disable.
backend tensorflow Deep Learning Backend.
image_shape 28,28 Width and height of image.
channels 3 Number of (color) channels.
network_definition_file Path of file containing network definition (graph, architecture).
network_parameters_file Path of file containing network (initial) parameters (weights, biases).
mean_image_file Path of file containing the mean image data for data normalization.
export_native_parameters_prefix Path (prefix) where to export the native model parameters after every iteration.
input_dropout_ratio 0 Input layer dropout ratio (can improve generalization, try 0.1 or 0.2).
hidden_dropout_ratios Hidden layer dropout ratios (can improve generalization), specify one value per hidden layer, defaults to 0.5.
Ready
Connections: 0
H2O

Unified Interface (Deep Water + R)

```
model <- h2o.deepwater(x=path, y=response,
                       training_frame=df, epochs=50,
                       learning_rate=1e-3, network = "lenet")
model
```

Choosing different network structures

Unified Interface (Deep Water + Python)

Choosing different network structures

```
model = H2ODeepWaterEstimator(epochs      = 500,
                              network     = "lenet",
                              image_shape = [28,28],  ## provide image size
                              channels    = 3,
                              backend     = "tensorflow",
                              model_id    = "deepwater_tf_simple")

model.train(x = [0], # file path e.g. xxx/xxx/xxx.jpg
            y = 1, # label cat/dog/mouse
            training_frame = frame)

model.show()
```

Change backend to “mxnet”, “caffe” or “auto”

```
deepwater Model Build progress: |████████████████████████████████████████| 100%
Model Details
=============
H2ODeepWaterEstimator :  Deep Water
Model Key:  deepwater_tf_simple
```

Easy Stacking with other H_2O Models

Model Stacking

Now we have three different models, we are ready to carry out model stacking.

In [47]:
```
# Create a list to include all the models for stacking
models <- list(model_dw, model_gbm, model_drf)
```

In [48]:
```
# Define a metalearner (one of the H2O supervised machine learning algorithms)
metalearner <- "h2o.glm.wrapper"
```

In [49]:
```
# Use h2o.stack() to carry out metalearning
stack <- h2o.stack(models = models,
                   response_frame = h_train$medv,
                   metalearner = metalearner)
```

```
[1] "Metalearning"
```

In [50]:
```
# Finally, we evaluate the predictive performance on the ensemble as well as individual models.
h2o.ensemble_performance(stack, newdata = h_test)
```

```
Base learner performance, sorted by specified metric:
        learner      MSE
1 h2o_deepwater 8.377644
2       h2o_gbm 8.106541
3       h2o_drf 7.443517


H2O Ensemble Performance on <newdata>:
----------------
Family: gaussian

Ensemble performance (MSE): 5.80436983051916
```

Ensemble of Deep Water, Gradient Boosting Machine & Random Forest models

H2O, Sparkling Water, Steam, & Deep Water Documentation

Getting Started | Data Science Algorithms | Languages | Tutorials, Examples, & Presentations | For Developers | For the Enterprise

docs.h2o.ai

Getting Started

H2O

What is H2O?
H2O User Guide
H2O Book (O'Reilly)
Recent Changes
Open Source License (Apache V2)

Quick Start Video - Flow Web UI
Quick Start Video - R
Quick Start Video - Python

Download H2O

Sparkling Water

What is Sparkling Water?
Sparkling Water Booklet
PySparkling Readme 2.0 | 1.6
RSparkling Readme
Open Source License (Apache V2)

Quick Start Video - Scala
Quick Start Video - Python

Download Sparkling Water

Steam

What is Steam?
Steam User Guide
Recent Changes
Open Source License (AGPL)

Download Steam

Deep Water (preview)

Deep Water Readme
Deep Water AMI Guide
Open Source License (Apache V2)

Launch Deep Water AMI
(choose g2.2xlarge)

Q & A

FAQ
Community Forum
h2ostream Google Group
Issue Tracking (JIRA)
Gitter
Stack Overflow
Cross Validated

For Supported Enterprise Customers

Enterprise Support Web | Email

Branch: master ▾ h2o-3 / examples / deeplearning / notebooks / Create new file | Upload files | Find file | History

mstensmo changing the name of deeplearning_credit_card_default_risk_prediction... ... Latest commit 5568350 11 days ago

..

images	Add cat/dog/mouse lenet example.	3 months ago
README.md	Update README.md	2 months ago
deeplearning_anomaly_detection.ip...	Update notebooks, introduce local paths to ~/h2o-3/	3 months ago
deeplearning_benchmark_mnist.ipynb	Update lenet test to remove all. Update MNIST benchmark with comments.	3 months ago
deeplearning_cat_dog_mouse_incep...	Add credit card default risk model, update other notebooks.	
deeplearning_cat_dog_mouse_lenet....	Add credit card default risk model, update other notebooks.	
deeplearning_cat_dog_mouse_lenet...	Add back model.plot() and scoring history.	
deeplearning_cifar10_vgg.ipynb	Rename notebooks.	
deeplearning_credit_card_default_ri...	changing the name of deeplearning_credit_card_default_risk_prediction...	
deeplearning_ensemble_boston_ho...	Ensemble demo using GBM, DRF and Deep Water (#676)	17 days ago
deeplearning_grid_iris.ipynb	Add two new notebooks: Lenet for R and iris grid for python	3 months ago
deeplearning_grid_iris_R.ipynb	Update R py notebook.	3 months ago
deeplearning_image_reconstruction...	Update notebooks, introduce local paths to ~/h2o-3/	3 months ago
deeplearning_mnist_convnet.ipynb	Update notebooks, introduce local paths to ~/h2o-3/	3 months ago
deeplearning_mnist_introduction.ip...	Add missing file.	3 months ago
deeplearning_tensorflow_cat_dog_...	Add tensorflow example (#529)	2 months ago
deeplearning_tensorflow_mnist.ipynb	Added MNIST example for TensorFlow	a month ago

Deep Water Example notebooks

https://github.com/h2oai/h2o-3/tree/master/examples/deeplearning/notebooks

H2O.ai

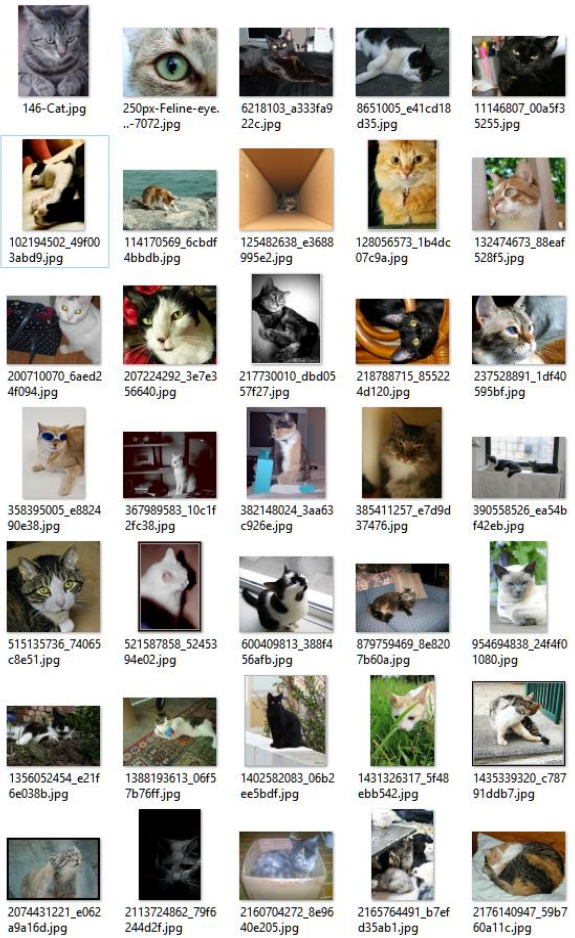
Deep Water Cat/Dog/Mouse Demo

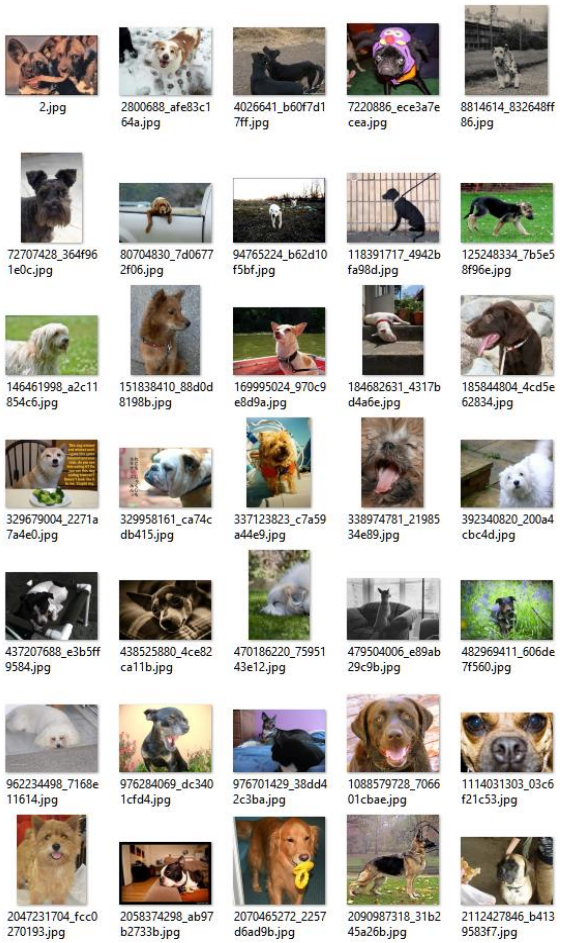
Deep Water R Demo

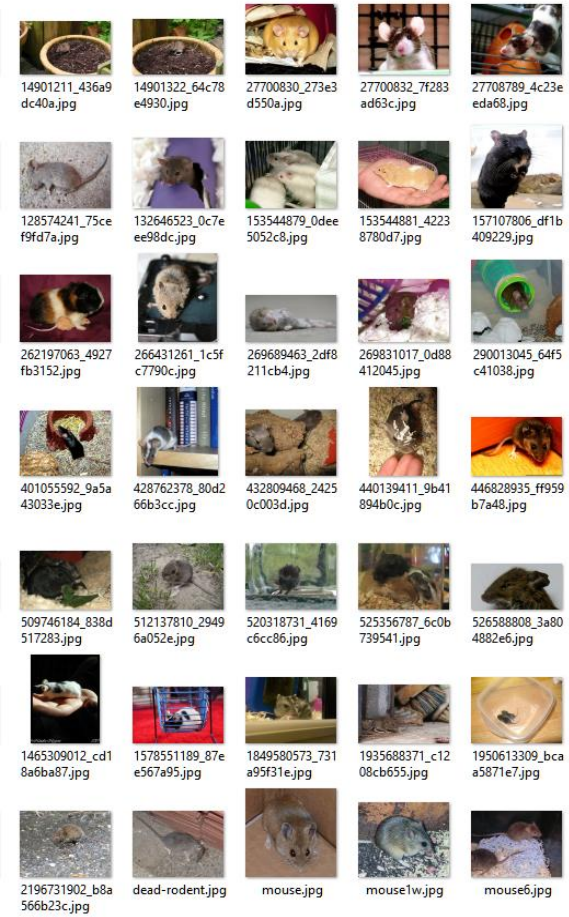
- H_2O + MXNet + TensorFlow
  - Dataset – Cat/Dog/Mouse
  - MXNet & TF as GPU backend
  - Train LeNet (CNN) models
  - R Demo

- Code and Data
  - github.com/h2oai/deepwater

Data – Cat/Dog/Mouse Images







H2O.ai

Data – CSV

	A	B
1	bigdata/laptop/deepwater/imagenet/cat/102194502_49f003abd9.jpg	cat
2	bigdata/laptop/deepwater/imagenet/cat/11146807_00a5f35255.jpg	cat
3	bigdata/laptop/deepwater/imagenet/cat/1140846215_70e326f868.jpg	cat
4	bigdata/laptop/deepwater/imagenet/cat/114170569_6cbdf4bbdb.jpg	cat
5	bigdata/laptop/deepwater/imagenet/cat/1217664848_de4c7fc296.jpg	cat
6	bigdata/laptop/deepwater/imagenet/cat/1241603780_5e8c8f1ced.jpg	cat
7	bigdata/laptop/deepwater/imagenet/cat/1241612072_27ececbdef.jpg	cat
8	bigdata/laptop/deepwater/imagenet/cat/1241613138_ef1d82973f.jpg	cat
9	bigdata/laptop/deepwater/imagenet/cat/1244562192_35becd66bd.jpg	cat
10	bigdata/laptop/deepwater/imagenet/cat/125482638_e3688995e2.jpg	cat
11	bigdata/laptop/deepwater/imagenet/cat/128056573_1b4dc07c9a.jpg	cat
12	bigdata/laptop/deepwater/imagenet/cat/12945197_75e607e355.jpg	cat
13	bigdata/laptop/deepwater/imagenet/cat/132474673_88eaf528f5.jpg	cat
14	bigdata/laptop/deepwater/imagenet/cat/1350530984_ecf3039cf0.jpg	cat
15	bigdata/laptop/deepwater/imagenet/cat/1351606235_c9fbebf634.jpg	cat
16	bigdata/laptop/deepwater/imagenet/cat/1356052454_e21f6e038b.jpg	cat
17	bigdata/laptop/deepwater/imagenet/cat/1388193613_06f57b76ff.jpg	cat

Deep Water – Basic Usage

Live Demo if Possible

Start and Connect to H_2O Deep Water Cluster

- Download Latest Nightly Build
  - https://s3.amazonaws.com/h2o-deepwater/public/nightly/latest/h2o.jar

- In Terminal
  - cd to the folder containing h2o.jar
  - java –jar h2o.jar *(this is the default command)*
  - java –jar –Xmx16g h2o.jar *(this is the command to allocate 16GB of memory)*

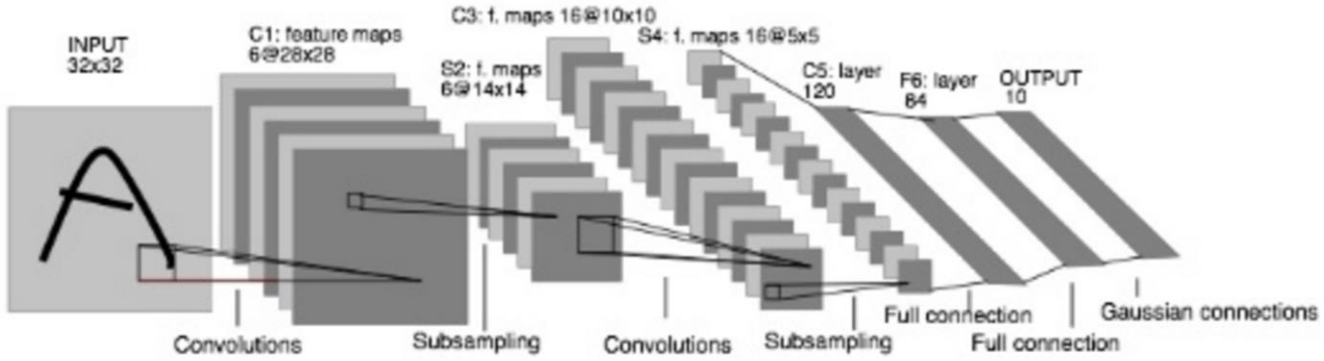
- In R
  - library(h2o) *(latest stable release from h2o.ai website or CRAN)*
  - h2o.connect(ip = “xxx.xxx.xxx.xxx”, strict_version_check = FALSE)

Import CSV

```
df <- h2o.importFile("/home/ubuntu/h2o-3/bigdata/laptop/deepwater/imagenet/cat_dog_mouse.csv")
print(head(df))
path = 1 ## must be the first column
response = 2
```

```
  |============================================================| 100%
                                                          C1  C2
1  bigdata/laptop/deepwater/imagenet/cat/102194502_49f003abd9.jpg cat
2   bigdata/laptop/deepwater/imagenet/cat/11146807_00a5f35255.jpg cat
3 bigdata/laptop/deepwater/imagenet/cat/1140846215_70e326f868.jpg cat
4  bigdata/laptop/deepwater/imagenet/cat/114170569_6cbdf4bbdb.jpg cat
5 bigdata/laptop/deepwater/imagenet/cat/1217664848_de4c7fc296.jpg cat
6 bigdata/laptop/deepwater/imagenet/cat/1241603780_5e8c8f1ced.jpg cat
```

Train a CNN (LeNet) Model on GPU



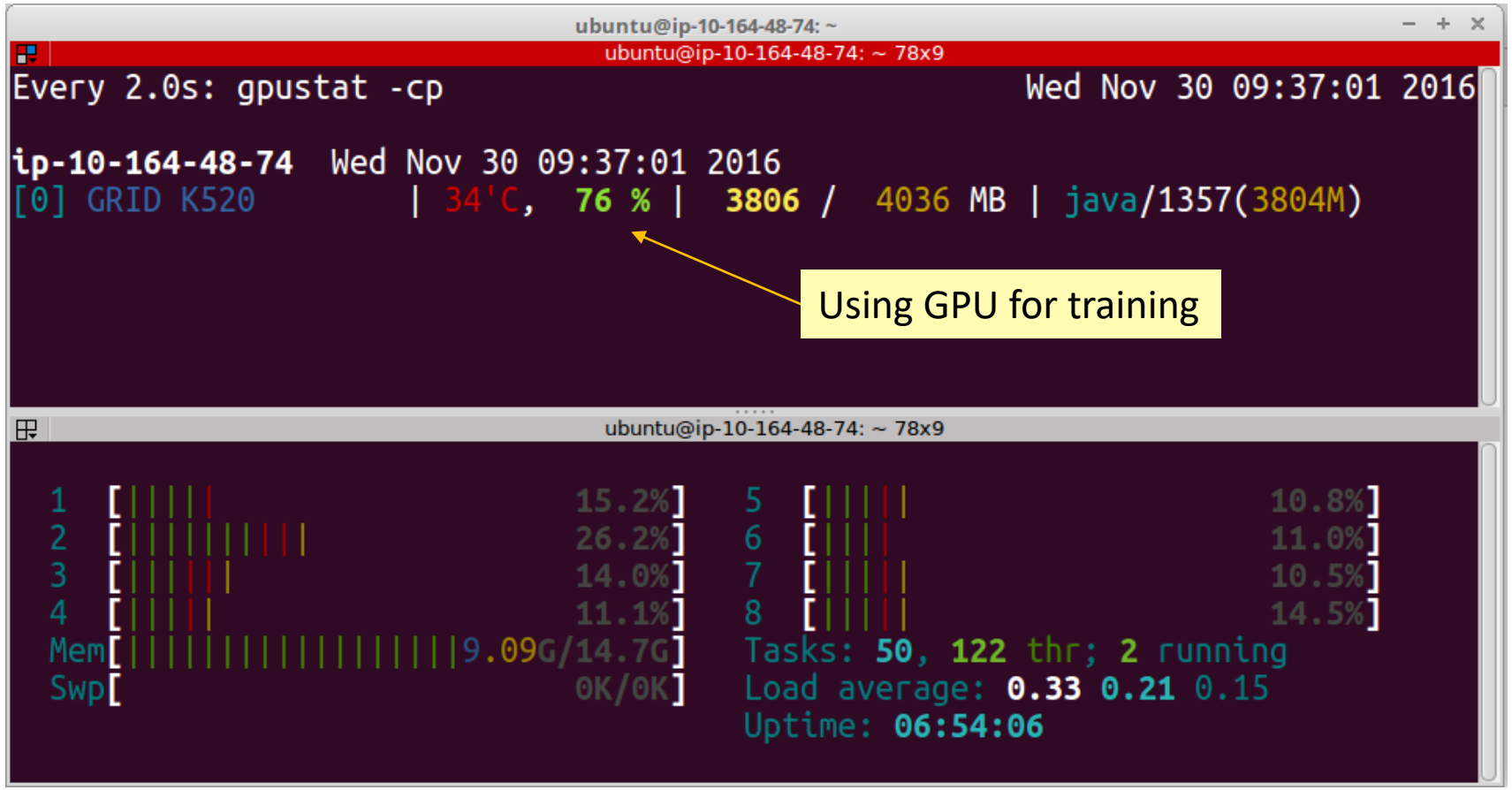
LeNet: a layered model composed of convolution and subsampling operations followed by a holistic representation and ultimately a classifier for handwritten digits. [Yann LeCun; LeNet]

We'll use a GPU to train such a LeNet model in seconds

To build a LeNet image classification model in H2O, simply specify network = "lenet":

```
model <- h2o.deepwater(x=path, y=response,
                       training_frame=df, epochs=50,
                       learning_rate=1e-3, network = "lenet")
```

Train a CNN (LeNet) Model on GPU



H2O.ai

Model

```
Model Details:
==============

H2OMultinomialModel: deepwater
Model ID:  DeepWater_model_R_1477378862430_2
Status of Deep Learning Model: lenet, 1.6 MB, predicting C2, 3-class classif
s, mini-batch size 32
  input_neurons     rate momentum
1          2352 0.000986 0.990000


H2OMultinomialMetrics: deepwater
** Reported on training data. **
** Metrics reported on full training frame **

Training Set Metrics:
=====================

Extract training frame with `h2o.getFrame("cat_dog_mouse.hex_sid_95f8_1")`
MSE: (Extract with `h2o.mse`) 0.131072
RMSE: (Extract with `h2o.rmse`) 0.3620386
Logloss: (Extract with `h2o.logloss`) 0.4176429
Mean Per-Class Error: 0.1165104
Confusion Matrix: Extract with `h2o.confusionMatrix(<model>,train = TRUE)`)
========================================================================
Confusion Matrix: vertical: actual; across: predicted
       cat dog mouse  Error       Rate
cat     75   4    11 0.1667 =  15 / 90
dog      4  75     6 0.1176 =  10 / 85
mouse    3   3    86 0.0652 =   6 / 92
Totals  82  82   103 0.1161 = 31 / 267
```

H2O.ai

Deep Water – Custom Network

If you'd like to build your own LeNet network architecture, then this is easy as well. In this example script, we are using the 'mxnet' backend. Models can easily be imported/exported between H2O and MXNet since H2O uses MXNet's format for model definition.

In [5]:
```
get_symbol <- function(num_classes = 1000) {
  library(mxnet)
  data <- mx.symbol.Variable('data')
  # first conv
  conv1 <- mx.symbol.Convolution(data = data, kernel = c(5, 5), num_filter = 20)

  tanh1 <- mx.symbol.Activation(data = conv1, act_type = "tanh")
  pool1 <- mx.symbol.Pooling(data = tanh1, pool_type = "max", kernel = c(2, 2), stride = c(2, 2))

  # second conv
  conv2 <- mx.symbol.Convolution(data = pool1, kernel = c(5, 5), num_filter = 50)
  tanh2 <- mx.symbol.Activation(data = conv2, act_type = "tanh")
  pool2 <- mx.symbol.Pooling(data = tanh2, pool_type = "max", kernel = c(2, 2), stride = c(2, 2))
  # first fullc
  flatten <- mx.symbol.Flatten(data = pool2)
  fc1 <- mx.symbol.FullyConnected(data = flatten, num_hidden = 500)
  tanh3 <- mx.symbol.Activation(data = fc1, act_type = "tanh")
  # second fullc
  fc2 <- mx.symbol.FullyConnected(data = tanh3, num_hidden = num_classes)
  # loss
  lenet <- mx.symbol.SoftmaxOutput(data = fc2, name = 'softmax')
  return(lenet)
}
```

Configure custom network structure (MXNet syntax)

In [7]:
```
nclasses = h2o.nlevels(df[,response])
network <- get_symbol(nclasses)
cat(network$as.json(), file = "/tmp/symbol_lenet-R.json", sep = '')
```

Saving the custom network structure as a file

Train a Custom Network

```
model = h2o.deepwater(x=path, y=response, training_frame = df,
                      epochs=500, ## early stopping is on by default and might trigger before
                      network_definition_file="/tmp/symbol_lenet-R.json",  ## specify the model
                      image_shape=c(28,28),                                ## provide expected (or matchin
g) image size
                      channels=3)                                          ## 3 for color, 1 for monochrom
e
```

Point it to the custom network structure file

Model

```
Model Details:
==============

H2OMultinomialModel: deepwater
Model Key:  DeepWater_model_R_1477378862430_3
Status of Deep Learning Model: user, 1.6 MB, predicting C2, 3-class classifi
s, mini-batch size 32
  input_neurons     rate momentum
1          2352 0.004409 0.990000

H2OMultinomialMetrics: deepwater
** Reported on training data. **
** Metrics reported on full training frame **

Training Set Metrics:
=====================

Extract training frame with `h2o.getFrame("cat_dog_mouse.hex_sid_95f8_1")`
MSE: (Extract with `h2o.mse`) 0.03078524
RMSE: (Extract with `h2o.rmse`) 0.1754572
Logloss: (Extract with `h2o.logloss`) 0.1154222
Mean Per-Class Error: 0.03366487
Confusion Matrix: Extract with `h2o.confusionMatrix(<model>,train = TRUE)`)
=========================================================================
Confusion Matrix: vertical: actual; across: predicted
       cat dog mouse  Error      Rate
cat     88   2     0 0.0222 =  2 / 90
dog      2  82     1 0.0353 =  3 / 85
mouse    1   3    88 0.0435 =  4 / 92
Totals  91  87    89 0.0337 = 9 / 267
```

H2O.ai

Conclusions

Project “Deep Water”

- H_2O + TF + MXNet + Caffe
  - A powerful combination of widely used open source machine learning libraries.
- All Goodies from H_2O
  - Inherits all H_2O properties in scalability, ease of use and deployment.
- Unified Interface
  - Allows users to build, stack and deploy deep learning models from different libraries efficiently.

- Latest Nightly Build
  - https://s3.amazonaws.com/h2o-deepwater/public/nightly/latest/h2o.jar
- 100% Open Source
  - The party will get bigger!

Other H_2O Developments

- H_2O + xgboost [Link]
- Stacked Ensembles [Link]
- Automatic Machine Learning [Link]
- Time Series [Link]
- High Availability Mode in Sparkling Water [Link]
- Model Interpretation [Link]
- word2vec [Link]

- Previous Talks
  - https://github.com/h2oai/h2o-meetups/blob/master/2017_04_06_Amsterdam/2017_04_06_Latest_H2O_Developments.pdf

Thanks!

- Organizers & Sponsors
  - Poznan R Users Group (PAZUR)
  - H_2O.ai

- Code, Slides & Documents
  - bit.ly/h2o_meetups
  - docs.h2o.ai
- Contact
  - joe@h2o.ai
  - @matlabulous
  - github.com/woobe
- Please search/ask questions on **Stack Overflow**
  - Use the tag `**h2o**` (not H2 zero)