Intro to Machine Learning with H2O and AWS

Galvanize Seattle
May 2016

Navdeep Gill M.S.
H2O.ai

Introduction

- Hacker/Data Scientist @ H2O.ai
- Previous work:
  - Cisco (Software Development/Data Science)
  - FICO (Risk Management/Predictive Analytics)
  - Motista (Market Research)
  - Worked in Neuroscience Labs prior to industry (UC Berkeley, UCSF, and Smith Kettlewell Eye Research Institute) focusing on fMRI, EEG, and behavioral studies with a keen focus on attention, visual neuroscience, and aging
- Education:
  - M.S. Computational Statistics @ CSU East Bay
  - B.S./B.A. Statistics, Mathematics, and Psychology @ CSU East Bay

- What/who is H2O?
- H2O Platform
- H2O in Flow, R, Python, & Spark (Sparkling Water)
- H2O in AWS
  - H2O + EC2
  - H2O + EMR
  - H2O + AWS Lambda

H2O.ai

H2O Company

- Team: 50. Founded in 2012, Mountain View, CA
- Stanford Math & Systems Engineers

H2O Software

- Open Source Software
- Ease of Use via Web Interface
- R, Python, Scala, Spark & Hadoop Interfaces
- Distributed Algorithms Scale to Big Data


H2O.ai Founders

SriSatish Ambati

- CEO and Co-founder at H2O.ai
- Past: Platfora, Cassandra, DataStax, Azul Systems, UC Berkeley

Dr. Cliff Click

- CTO and Co-founder at H2O.ai
- Past: Azul Systems, Sun Microsystems
- Developed the Java HotSpot Server Compiler at Sun
- PhD in CS from Rice University

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*
- 108,404 citations (via Google Scholar)

Dr. Rob Tibshirani

- Professor of Statistics and Health Research and Policy, Stanford University
- PhD in Statistics, Stanford University
- COPPS Presidents' Award recipient
- 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. Stephen Boyd

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

H2O Overview

Speed Matters!

- Time is valuable
- In-memory is faster
- Distributed is faster
- High speed AND accuracy

No Sampling

- Scale to big data
- Access data links
- Use all data without sampling

Interactive UI

- Web-based modeling with H2O Flow
- Model comparison

Cutting-Edge Algorithms

- Suite of cutting-edge machine learning algorithms
- Deep Learning & Ensembles
- NanoFast Scoring Engine

Current Algorithm Overview

Statistical Analysis

- Linear Models (GLM)
- Naïve Bayes

Ensembles

- Random Forest
- Distributed Trees
- Gradient Boosting Machine
- R Package - Super Learner Ensembles

Deep Neural Networks

- Multi-layer Feed-Forward Neural Network
- Auto-encoder
- Anomaly Detection
- Deep Features

Clustering

- K-Means

Dimension Reduction

- Principal Component Analysis
- Generalized Low Rank Models

Solvers & Optimization

- Generalized ADMM Solver
- L-BFGS (Quasi Newton Method)
- Ordinary Least-Square Solver
- Stochastic Gradient Descent

Data Munging

- Scalable Data Frames
- Sort, Slice, Log Transform

H2O

GITTER JOIN CHAT →

H2O scales statistics, machine learning, and math over Big Data.

H2O uses familiar interfaces like R, Python, Scala, the Flow notebook graphical interface, Excel, & JSON so that Big Data enthusiasts & experts can explore, munge, model, and score datasets using a range of algorithms including advanced ones like Deep Learning. H2O is extensible so that developers can add data transformations and model algorithms of their choice and access them through all of those clients.

Data collection is easy. Decision making is hard. H2O makes it fast and easy to derive insights from your data through faster and better predictive modeling. H2O allows online scoring and modeling in a single platform.

https://github.com/h2oai/h2o-3

H2O Components

H2O Cluster

- Multi-node cluster with shared memory model.
- All computations in memory.
- Each node sees only some rows of the data.
- No limit on cluster size.

Distributed Key Value Store

- Objects in the H2O cluster such as data frames, models and results are all referenced by key.
- Any node in the cluster can access any object in the cluster by key.

H2O Frame

- Distributed data frames (collection of vectors).
- Columns are distributed (across nodes) arrays.
- Each node must be able to see the entire dataset (achieved using HDFS, S3, or multiple copies of the data if it is a CSV file).

Distributed H2O Frame

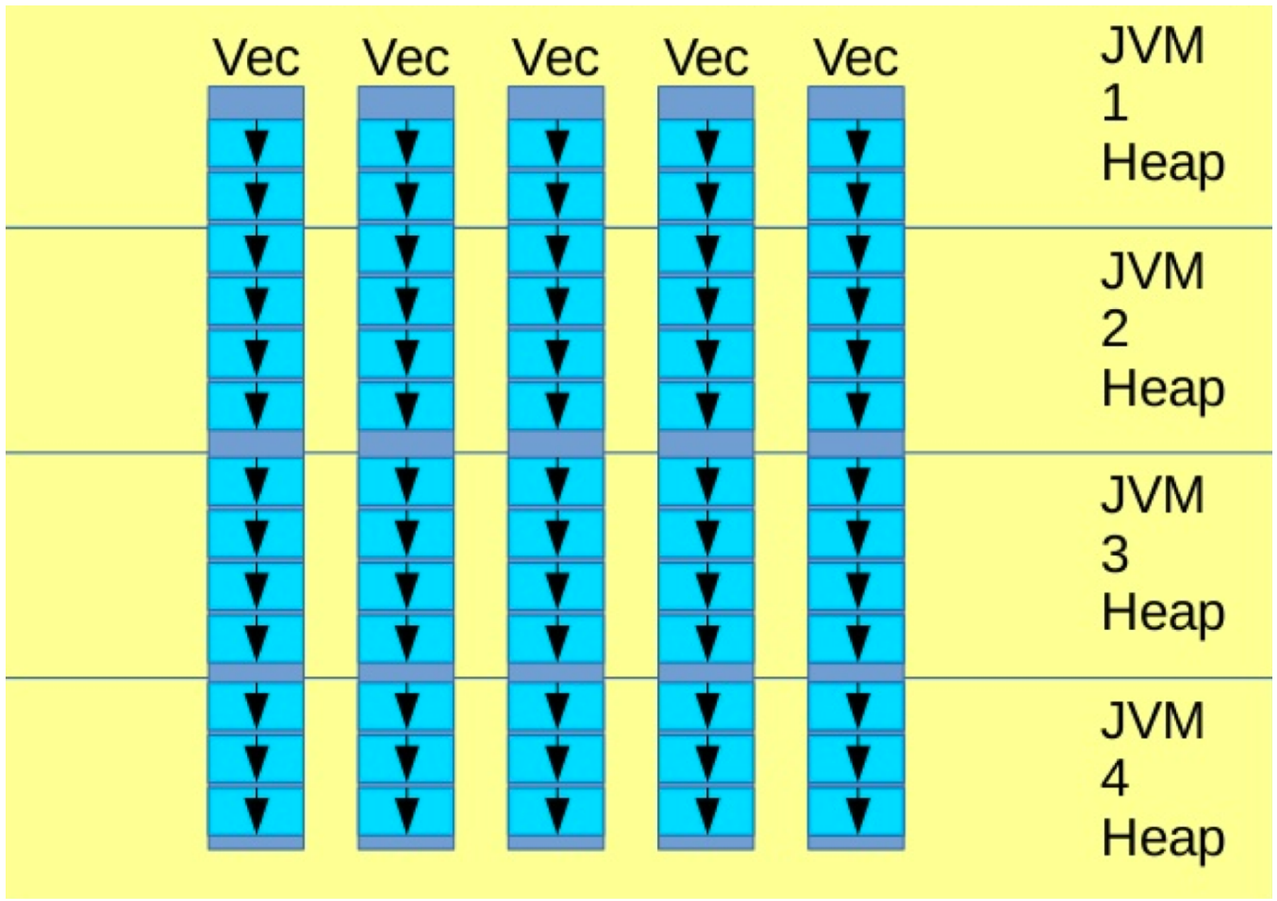
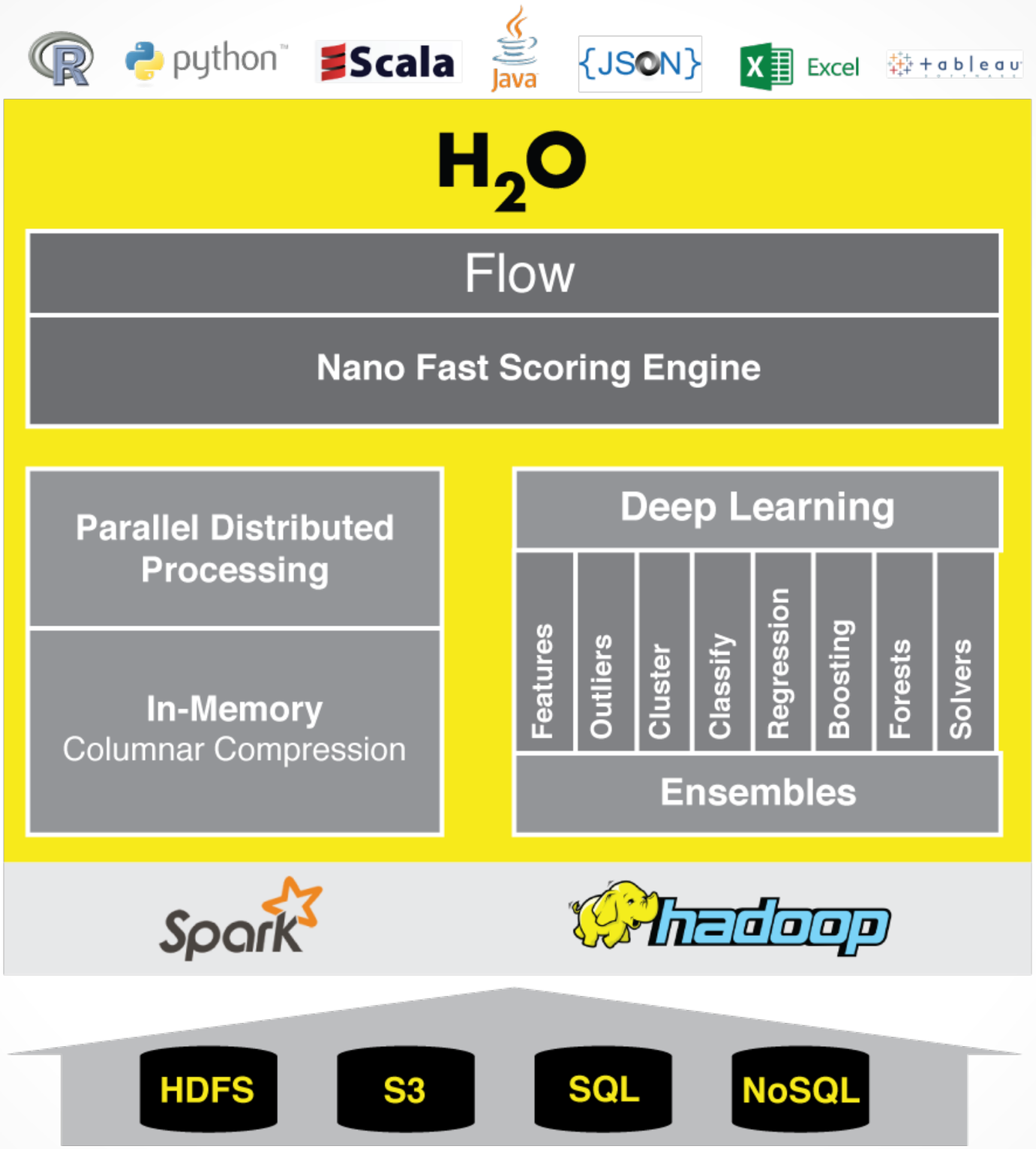


Diagram of distributed arrays. An “H2O Frame” is a collection of distributed arrays.

H2O Software

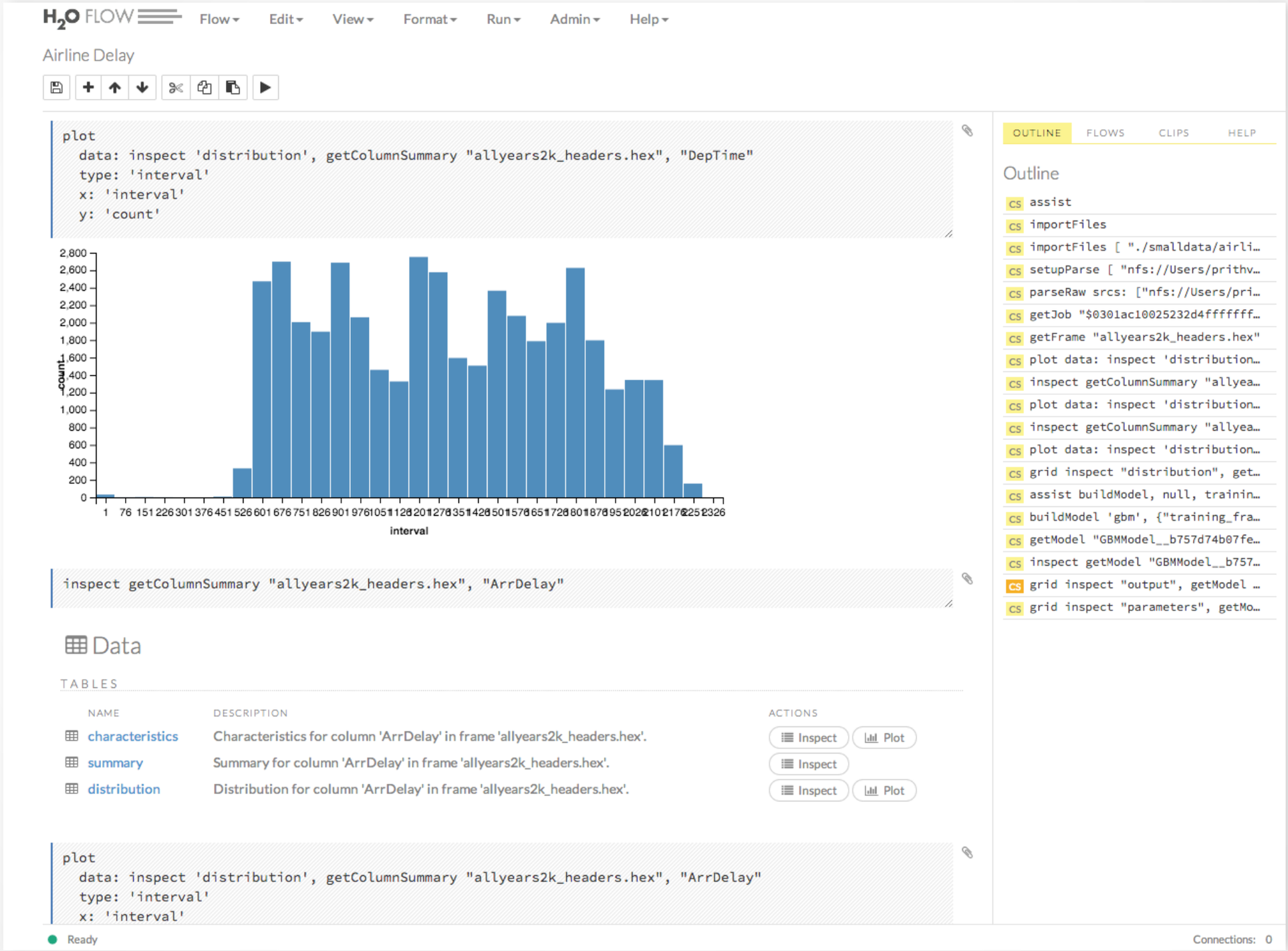
H2O is an open source, distributed, Java machine learning library.

APIs are available for:
R, Python, Scala & JSON



H2O.ai
Machine Intelligence

H2O Flow Interface



H2O.ai

Machine Intelligence

H2O in R & Python

h2o R package

Requirements

- The only requirement to run the “h2o” R package is R >=3.1.0 and Java 7 or later.
- Linux, OS X and Windows.

Installation

- The easiest way to install the “h2o” R package is to install directly from CRAN.
- Latest version: `http://h2o.ai/download`

Design

- No computation is ever performed in R.
- All computations are performed (in highly optimized Java code) in the H2O cluster and initiated by REST calls from R.

Download

Use H_2O directly from R

Copy and paste these commands into R one line at a time:

```
# The following two commands remove any previously installed H2O packages for R.
if ("package:h2o" %in% search()) { detach("package:h2o", unload=TRUE) }
if ("h2o" %in% rownames(installed.packages())) { remove.packages("h2o") }

# Next, we download packages that H2O depends on.
pkgs <- c("methods","statmod","stats","graphics","RCurl","jsonlite","tools","utils")
for (pkg in pkgs) {
    if (! (pkg %in% rownames(installed.packages()))) { install.packages(pkg) }
}

# Now we download, install and initialize the H2O package for R.
install.packages("h2o", type="source", repos=(c("http://h2o-release.s3.amazonaws.com/h2o/rel-tibshirani/8/R")))
library(h2o)
localH2O = h2o.init()

# Finally, let's run a demo to see H2O at work.
demo(h2o.kmeans)
```

http://h2o.ai/download/h2o/r

h2o Python module

Requirements

- Java 7 or later.
- Python 2 or 3.
- A few Python module dependencies.
- Linux, OS X or Windows.

Installation

- The easiest way to install the “h2o” Python module is PyPI (pip install).
- Latest version: `http://h2o.ai/download`

Design

- No computation is ever performed in Python.
- All computations are performed in highly optimized Java code in the H2O cluster and initiated by REST calls from Python.

Download

Use H2O directly from Python

1. Prerequisite: Python 2.7

2. Install dependencies (prepending with `sudo` if needed):

```
[sudo] pip install requests
[sudo] pip install tabulate
```

At the command line, copy and paste these commands one line at a time:

```
# The following command removes the H2O module for Python.
[sudo] pip uninstall h2o
# Next, use pip to install this version of the H2O Python module.
[sudo] pip install http://h2o-release.s3.amazonaws.com/h2o/rel-tibshirani/8/Python/h2o-3.6.0.8-py2.py3-none-any.whl
```

http://h2o.ai/download/h2o/python

H2O.ai
Machine Intelligence

Start H2O Cluster from R

```
> library(h2o)
> localH2O <- h2o.init(nthreads = -1, max_mem_size = "8G")

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

Note:  In case of errors look at the following log files:
    /var/folders/2j/jg4sl53d5q53tc2_nzm9fz5h0000gn/T//RtmpAXY9gj/h2o_me_started_from_r.out
    /var/folders/2j/jg4sl53d5q53tc2_nzm9fz5h0000gn/T//RtmpAXY9gj/h2o_me_started_from_r.err

java version "1.8.0_45"
Java(TM) SE Runtime Environment (build 1.8.0_45-b14)
Java HotSpot(TM) 64-Bit Server VM (build 25.45-b02, mixed mode)

.Successfully connected to http://127.0.0.1:54321/

R is connected to the H2O cluster:
    H2O cluster uptime:         1 seconds 96 milliseconds
    H2O cluster version:        3.3.0.99999
    H2O cluster name:           H2O_started_from_R_me_kfo618
    H2O cluster total nodes:    1
    H2O cluster total memory:   7.11 GB
    H2O cluster total cores:    8
    H2O cluster allowed cores:  8
    H2O cluster healthy:        TRUE

>
```

H2O in R: Load Data

Example

```
library(h2o)  # First install from CRAN
localH2O <- h2o.init()  # Initialize the H2O cluster

# Data directly into H2O cluster (avoids R)
train <- h2o.importFile(path = "train.csv")

# Data into H2O from R data.frame
train <- as.h2o(my_df)
```

R code example: Load data

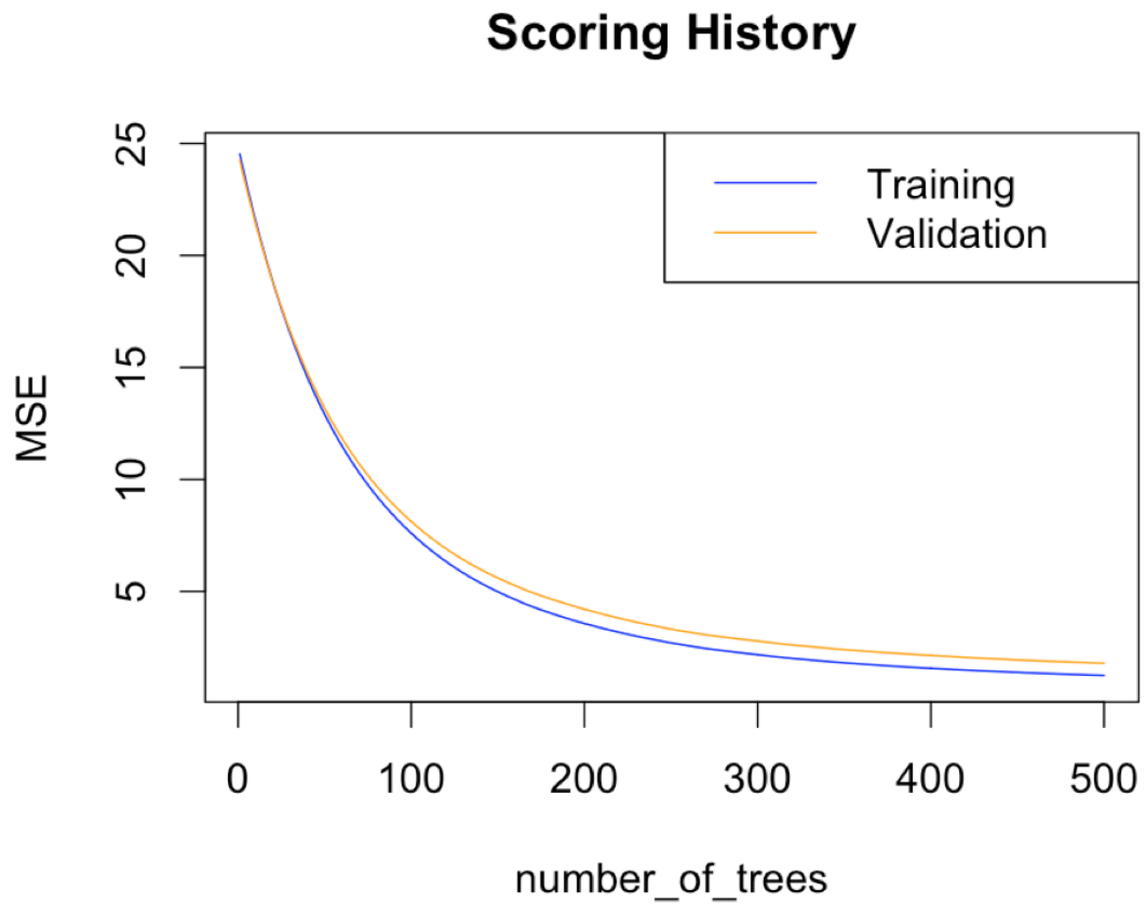
H2O in R: Train & Test

Example

```
y <- "Class"
x <- setdiff(names(train), y)

fit <- h2o.gbm(x = x, y = y, training_frame = train)


pred <- h2o.predict(fit, test)
```

R code example: Train and Test a GBM

H2O in R: Plotting



`plot(fit)` plots scoring history over time.

H2O in R: Grid Search

Example

```
hidden_opt <- list(c(200,200), c(100,300,100), c(500,500))
l1_opt <- c(1e-5,1e-7)
hyper_params <- list(hidden = hidden_opt, l1 = l1_opt)

model_grid <- h2o.grid("deeplearning",
        hyper_params = hyper_params,
        x = x, y = y,
        training_frame = train,
        validation_frame = test)
```

R code example: Execute a DL Grid Search

Start H2O Cluster from Python

In [1]:
```
import h2o

# Start an H2O Cluster on your local machine
h2o.init()
```

```
No instance found at ip and port: localhost:54321. Trying to start local jar...

JVM stdout: /var/folders/2j/jg4sl53d5q53tc2_nzm9fz5h0000gn/T/tmpsfXfv2/h2o_me_started_from_py
thon.out
JVM stderr: /var/folders/2j/jg4sl53d5q53tc2_nzm9fz5h0000gn/T/tmpnySTva/h2o_me_started_from_py
thon.err
Using ice_root: /var/folders/2j/jg4sl53d5q53tc2_nzm9fz5h0000gn/T/tmpUU8e_0


Java Version: java version "1.8.0_45"
Java(TM) SE Runtime Environment (build 1.8.0_45-b14)
Java HotSpot(TM) 64-Bit Server VM (build 25.45-b02, mixed mode)


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

Start H2O Cluster from Python

In [2]:
```
import h2o

# Start an H2O Cluster on your local machine
h2o.init()
```

H2O cluster uptime:	12 minutes 16 seconds 831 milliseconds
H2O cluster version:	3.6.0.3
H2O cluster name:	H2O_started_from_python
H2O cluster total nodes:	1
H2O cluster total memory:	3.56 GB
H2O cluster total cores:	8
H2O cluster allowed cores:	8
H2O cluster healthy:	True
H2O Connection ip:	127.0.0.1
H2O Connection port:	54321

Train a model (e.g. GBM)

In [23]:
```
# Import H2O GBM:
from h2o.estimators.gbm import H2OGradientBoostingEstimator
```

We first create a `model` object of class, `"H2OGradientBoostingEstimator"`. This does not actually do any training, it just sets the model up for training by specifying model parameters.

In [24]:
```
model = H2OGradientBoostingEstimator(distribution='bernoulli',
                                     ntrees=100,
                                     max_depth=4,
                                     learn_rate=0.1)
```

In [28]:
```
model.train(x=x, y=y, training_frame=train, validation_frame=valid)
```

```
gbm Model Build Progress: [##################################################] 100%
```

Inspect Model Performance

In [27]:
```
print(model)
```

```
Model Details
=============
H2OGradientBoostingEstimator :  Gradient Boosting Machine
Model Key:  GBM_model_python_1448559565749_9080

Model Summary:
```

	number_of_trees	model_size_in_bytes	min_depth	max_depth	mean_depth	min_leaves	max_leaves	mean_leaves
	100.0	23614.0	4.0	4.0	4.0	10.0	16.0	14.9

```
ModelMetricsBinomial: gbm
** Reported on train data. **

MSE: 0.114026790434
R^2: 0.539835211
LogLoss: 0.376005292812
AUC: 0.936370388939
Gini: 0.872740777878

Confusion Matrix (Act/Pred) for max f1 @ threshold = 0.43076103173:
```

	0	1	Error	Rate
0	4102.0	814.0	0.1656	(814.0/4916.0)
1	534.0	3538.0	0.1311	(534.0/4072.0)
Total	4636.0	4352.0	0.15	(1348.0/8988.0)

H2O in Spark

Spark + H_2O

SPARKLING WATER

Apache Spark and SparkR

Apache Spark

- Apache Spark is an open source in-memory processing engine built around speed.
- It was originally developed at UC Berkeley in 2009.

Spark for HPC

- Spark is commonly used on commodity clusters (such as Amazon EC2).
- CRAY has been working with Spark community to optimize Spark for CRAY supercomputers.

SparkR

- Spark is written in Scala, but APIs exist for Python and R.
- “SparkR” is the R API and has been part of Spark since Spark 1.4 (June, 2015).

H2O vs SparkR

Architecture	• Major difference is that SparkR creates a collection of slave R instances. • H2O uses a single R session and communicates to the H2O Java cluster via REST calls.
Machine Learning Algorithms	• In Spark 1.5 (latest release), only GLM is accessible in R via SparkR. • All H2O algorithms are available via R.
Distributed Frames	• Both H2O and Spark use distributed data frames. • SparkR is most useful for data processing on distributed data frames.

H2O Sparkling Water

Spark Integration

- Sparkling Water is transparent integration of H2O into the Spark ecosystem.
- H2O runs inside the Spark Executor JVM.

Benefits

- Provides advanced machine learning algorithms to Spark workflows.
- Sophisticated alternative to the default MLlib library in Spark.

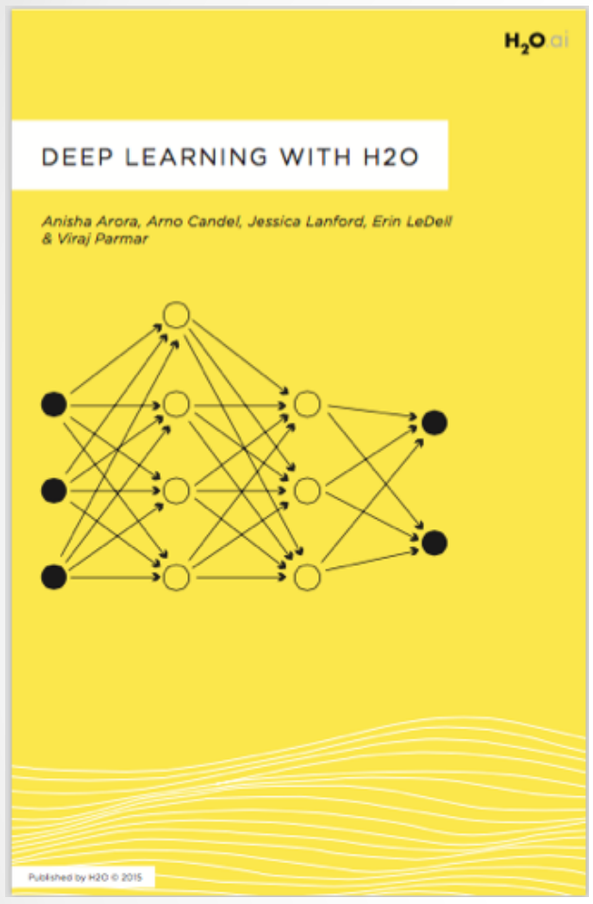
Sparkling Shell

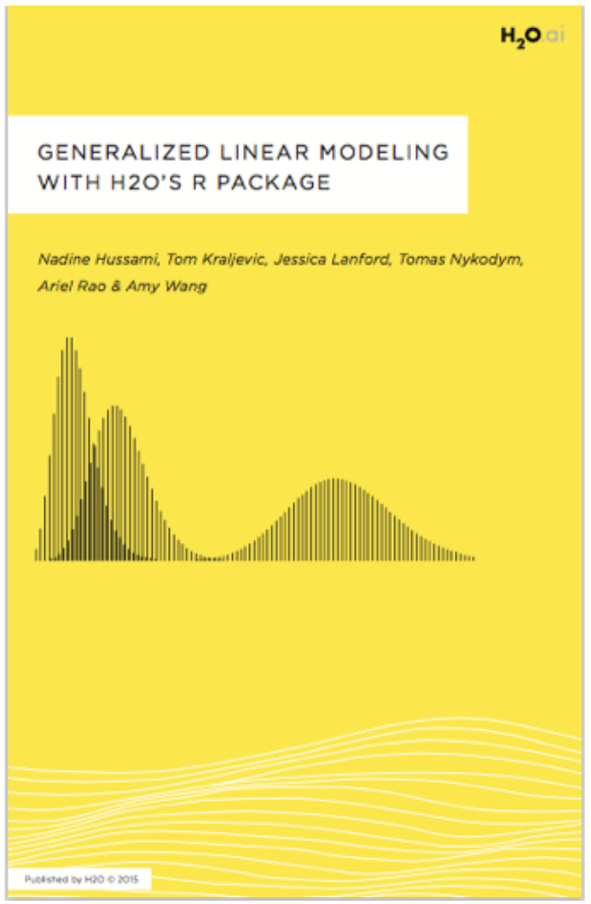
- Sparkling Shell is just a standard Spark shell with additional Sparkling Water classes
- `export MASTER="local-cluster[3,2,1024]"`
- `spark-shell –jars sparkling-water.jar`

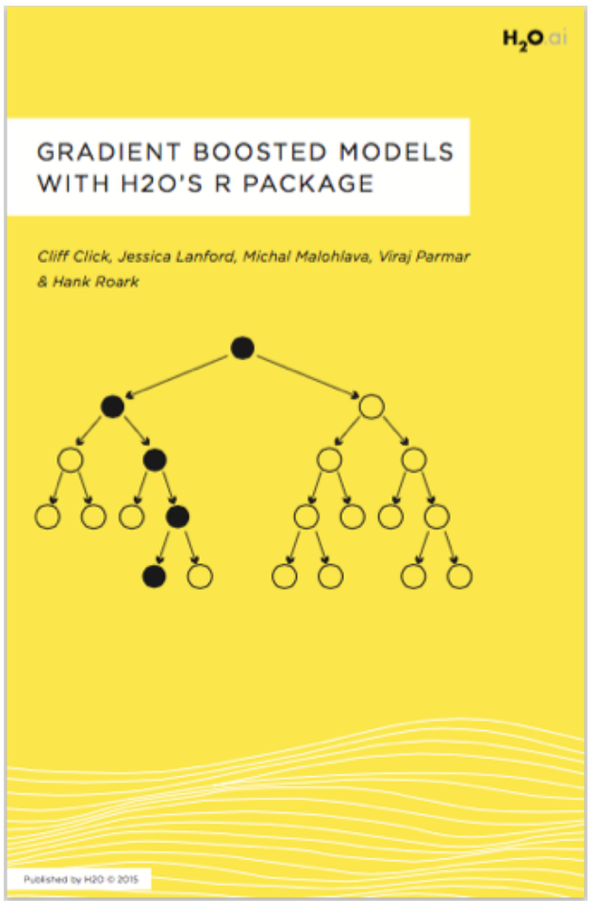
Where to learn more?

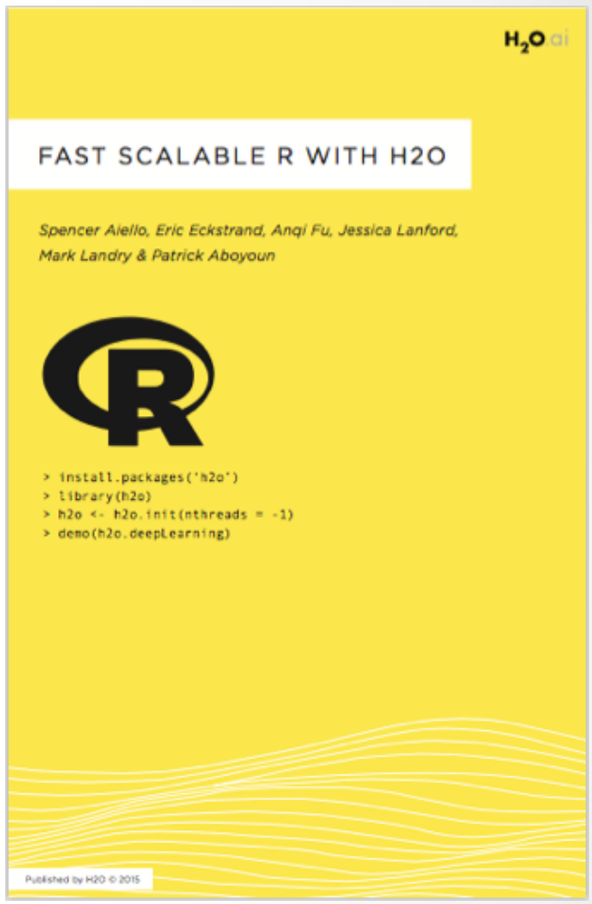
- H2O Online Training (free): http://learn.h2o.ai
- H2O Slidedecks: http://www.slideshare.net/0xdata
- H2O Video Presentations: https://www.youtube.com/user/0xdata
- H2O Community Events & Meetups: http://h2o.ai/events
- Machine Learning & Data Science courses: http://coursebuffet.com

H2O Booklets









https://github.com/h2oai/h2o-3/tree/master/h2o-docs/src/booklets/v2_2015/PDFs/online

H2O in AWS

H2O in AWS

- Setting the stage…
  - Information is the new gold and information comes from data!
  - Data is increasing everyday and at a rapid pace
    - IoT
    - Transactional Data
      - Fraud, Risk & Marketing Analytics, Consumer behavior, Churn, etc.
  - Current ML algorithms must scale to data to make meaningful predictions in a small time window
    - ML practitioners want answers and they want them now!
  - ML is great, but scalable ML is even better.

H2O in AWS

- H2O can achieve the previous with the following:
  - Highly optimized Java code(in-house)
  - Distributed in memory KV store and map/reduce computation framework
  - Efficient data parser(HDFS,S3, NFS, HTTP, local drives, etc)
  - Read/write access to distributed data frames
  - ML algos that are distributed in a cluster

H2O in AWS

- AWS can help the previous cause as follows:
  - Cloud computing on demand
  - Scalable
    - Using AWS tools, Auto Scaling, and Elastic Load Balancing, your application can scale up or down based on demand.
  - Ability to access different compute resources
    - Different OS's and compute power
  - Reliable framework
  - Cost effective

H2O in AWS

- H2O in AWS = Distributed Machine Learning/Computing at your fingertips!

H2O on AWS EC2

Amazon EC2

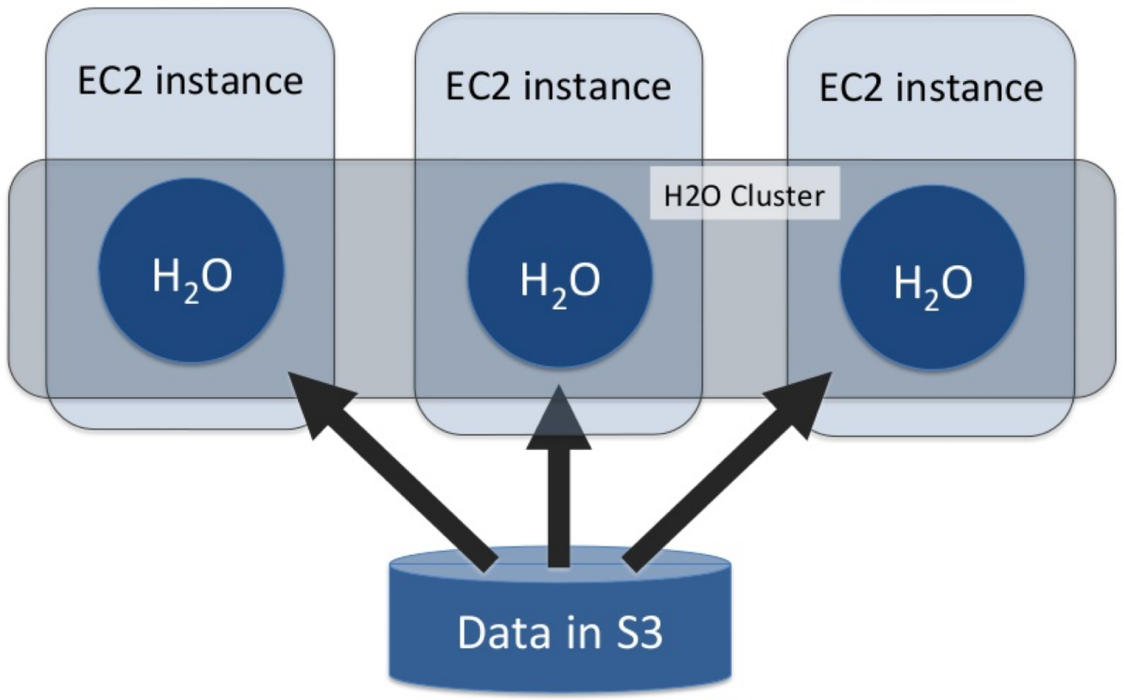
What is Amazon EC2?

- Amazon Elastic Compute Cloud (Amazon EC2) is a web service that provides resizable compute capacity in the cloud. It is designed to make web-scale cloud computing easier for developers.
- Amazon EC2's simple web service interface allows you to obtain and configure capacity with minimal friction.
- It provides you with complete control of your computing resources and lets you run on Amazon's proven computing environment.
- Amazon EC2 reduces the time required to obtain and boot new server instances to minutes, allowing you to quickly scale capacity, both up and down, as your computing requirements change.
- Amazon EC2 changes the economics of computing by allowing you to pay only for capacity that you actually use.
- Amazon EC2 provides developers the tools to build failure resilient applications and isolate themselves from common failure scenarios.

Benefits of Amazon EC2

- Elastic Web Scale Computing
  - Increase/decrease capacity within minutes.
  - Commission one, hundreds, or thousands of server instances simultaneously.
- Completely Controlled
  - Root access to each instance (Interact with them as you would with any machine).
  - Stop instance while retaining the data on your boot partition.
- Flexible Cloud Hosting Services
  - Multiple instance types and OS.
- Designed for use with other AWS services (S3, RDS, SQS)
- Reliable
- Secure
  - AWS EC2 works in conjunction with Amazon VPC to provide security and robust networking functionality.
  - Decide which instances are exposed to the internet and which are private.
- Inexpensive

H2O on Amazon EC2



H2O can easily be deployed on an Amazon EC2 cluster. The GitHub repository contains example scripts that help to automate the cluster deployment.

DEMO!

H2O Sparkling Water on AWS EMR

Spark + H_2O

SPARKLING WATER

AMAZON EMR

amazon web services

H_2O.ai

Machine Intelligence

What is AWS EMR?

- Amazon Elastic MapReduce (Amazon EMR) is a web service that makes it easy to quickly and cost-effectively process vast amounts of data.
- Amazon EMR simplifies big data processing, providing a managed Hadoop framework that makes it easy, fast, and cost-effective for you to distribute and process vast amounts of your data across dynamically scalable Amazon EC2 instances.
- You can also run other popular distributed frameworks such as Apache Spark and Presto in Amazon EMR, and interact with data in other AWS data stores such as Amazon S3 and Amazon DynamoDB.
- Amazon EMR securely and reliably handles your big data use cases, including log analysis, web indexing, data warehousing, machine learning, financial analysis, scientific simulation, and bioinformatics.

Benefits of Amazon EMR

- Easy to use
  - You can launch an Amazon EMR cluster in minutes. No need to worry about provisioning, cluster setup, Hadoop config, or cluster tuning.
- Low Cost
  - You pay and hourly rate for every instance hour you use.
  - Can launch 10 node hadoop cluster for as little as $0.15 an hour.
- Reliable
  - Spend less time tuning and monitoring your cluster.
  - EMR has tuned Hadoop for the cloud; also monitors your cluster — retrying failed tasks and automatically replacing poorly performing instances
- Secure
  - Amazon EMR automatically configures Amazon EC2 firewall settings that control network access to instances, and you can launch clusters in an Amazon Virtual Private Cloud (VPC), a logically isolated network you define.
  - For objects stored in Amazon S3, you can use Amazon S3 server-side encryption or Amazon S3 client-side encryption with EMRFS, with AWS Key Management Service or customer-managed keys.
- Flexible just like Amazon EC2

DEMO!

H2O on AWS Lambda

What is AWS Lambda?

- AWS Lambda lets you run code without provisioning or managing servers.
- You pay only for the compute time you consume - there is no charge when your code is not running.
- With Lambda, you can run code for virtually any type of application or backend service - all with zero administration.
- Just upload your code and Lambda takes care of everything required to run and scale your code with high availability.
- You can set up your code to automatically trigger from other AWS services or call it directly from any web or mobile app.
- It simplifies the process of running code in the cloud by managing compute resources automatically.
- Offloads DevOps tasks related to VMs:
  - Server and operating system maintenance
  - Capacity provisioning
  - Scaling
  - Code monitoring and logging
  - Security patches

Benefits of AWS Lambda

- No Servers to Manage
  - AWS Lambda automatically runs your code without requiring you to provision or manage servers. Just write the code and upload it to Lambda.
- Continous Scaling
  - AWS Lambda automatically scales your application by running code in response to each trigger. Your code runs in parallel and processes each trigger individually, scaling precisely with the size of the workload.
- Sub second Monitoring
  - With AWS Lambda, you are charged for every 100ms your code executes and the number of times your code is triggered. You don't pay anything when your code isn't running.

H2O + AWS Lambda = Machine Learning Applications

Majors Steps for H2O + Lambda

Step 1: Identify problem to solve
Step 2: Train model on data
Step 3: Export the model as a POJO
Step 4: Write code for Lambda handler
Step 5: Build deployment package (.zip file) and upload to Lambda
Step 6: Map API endpoint to Lambda function
Step 7: Embed endpoint in application

Use Case: Domain Name Classification

Malicious domains
Carry out malicious activity - botnets, phishing, malware hosting, etc
Names are generated by algorithms to defeat security systems

Goal: Classify domains as legitimate vs. malicious

Legitimate	Malicious
h2o	zyxgifnjobqhzptuodmzov
zen-cart	c3p4j7zdxexg1f2tuzk117wyzn
fedoraforum	batdtrbtrikw

Features

- String length
- Shannon Entropy
  - Measure of uncertainty in a random variable

$$H(X) = -\sum_{i=1}^{n} p(x_i) \log_b p(x_i)$$

- Number of substrings that are English words
- Proportion of vowels

Data

- Domains and whether they are malicious
  - http://datadrivensecurity.info/blog/data/2014/10/legit-dga_domains.csv.zip
  - 133,927 rows
- English words
  - https://raw.githubusercontent.com/dwyl/english-words/master/words.txt
  - 354,985 rows

Model Information

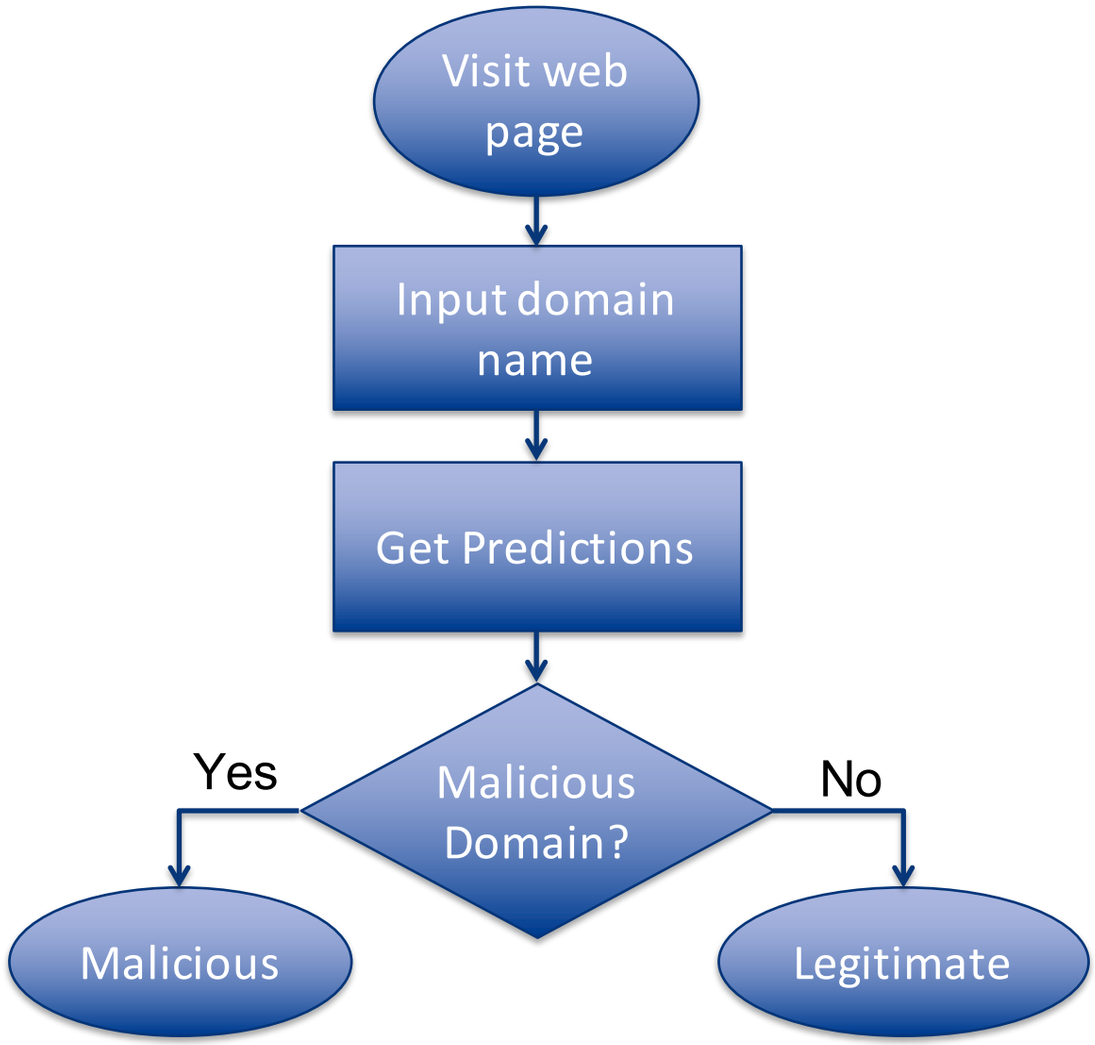
Malicious Domain Model

Algorithm:	GLM
Model family:	Binomial
Regularization:	Ridge
Threshold (max F1):	0.4935

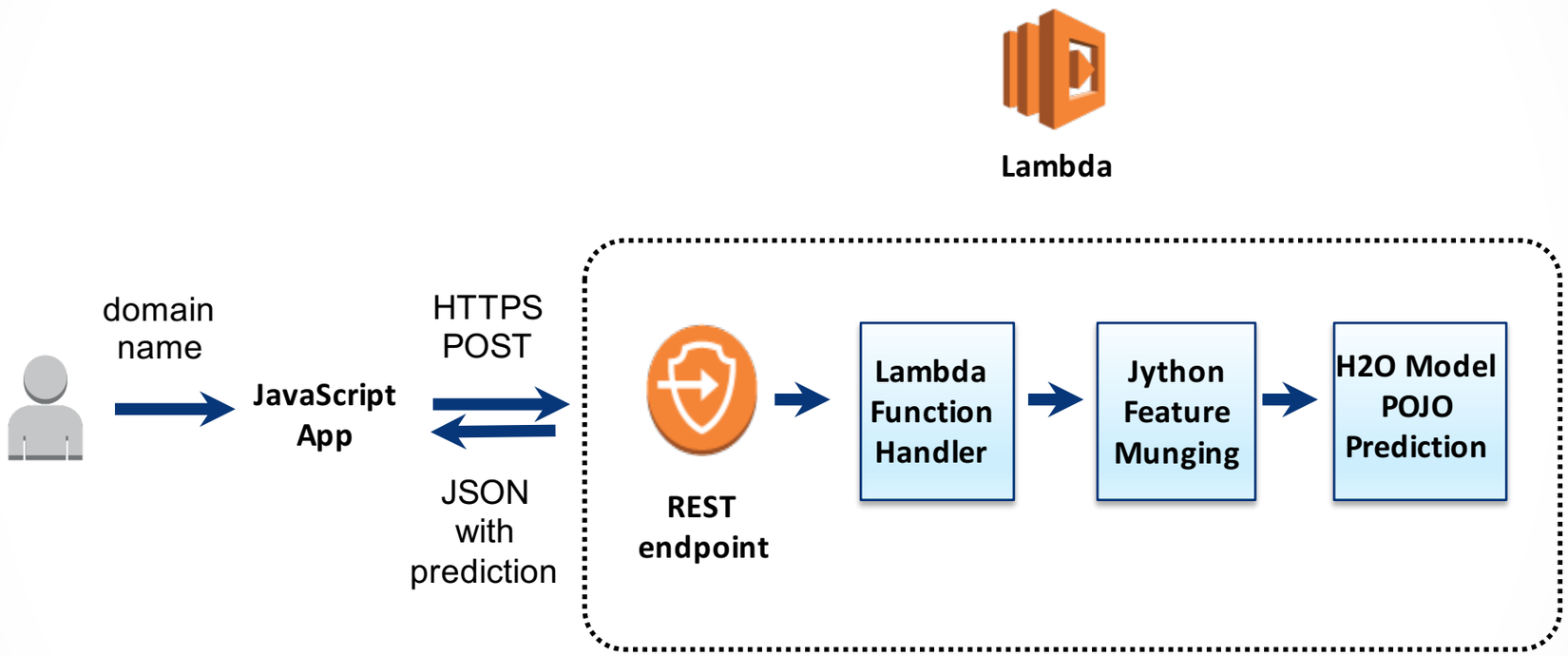
Confusion matrix on validation data

Actual \ Predicted	0	1	Error
0	15889	315	FPR 0.0194
1	346	10043	FNR 0.0333

Workflow For This App



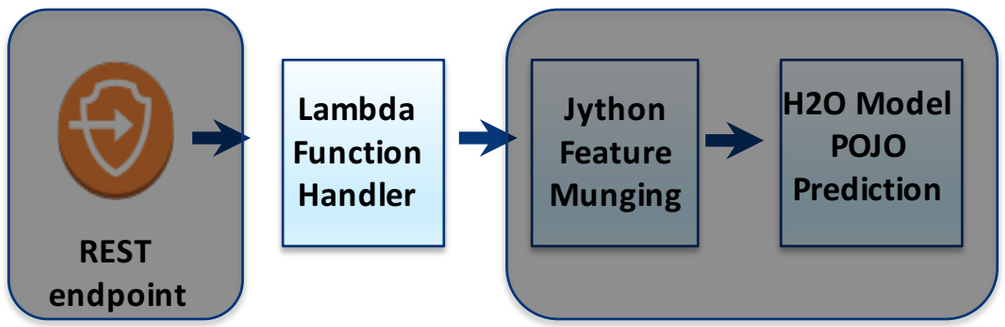
APP Arch Diagram



DEMO!

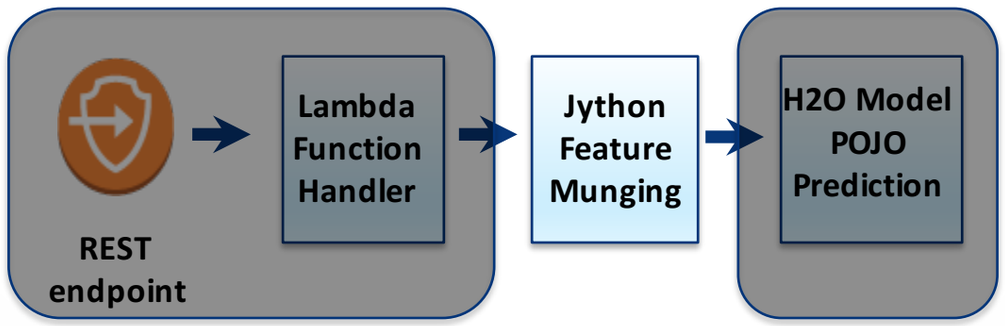
Lambda Function Handler

```
public static ResponseClass myHandler(RequestClass
request, Context context) throws PyException {

  PyModule module = new PyModule();

  //Prediction code is in pymodule.py
  double[]predictions=module.predict(request.domain);
  return new ResponseClass(predictions);
}
```



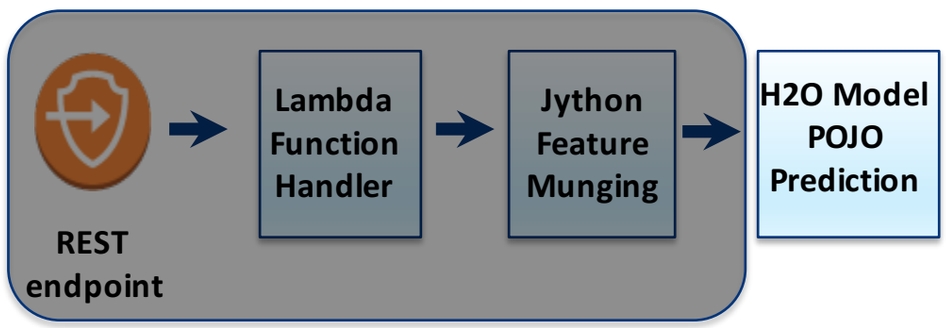
Jython Feature Munging

```
def predict(domain):
  domain = domain.split('.')[0]
  row = RowData()
  functions = [len, entropy, p_vowels, num_valid_substrings]
  eval_features = [f(domain) for f in functions]
  names = NamesHolder_MaliciousDomainModel().VALUES
  beta = MaliciousDomainModel().BETA().VALUES
  feature_coef_product = [beta[len(beta) - 1]]
  for i in range(len(names)):
    row.put(names[i], float(eval_features[i]))
    feature_coef_product.append(eval_features[i] * beta[i])

#prediction
model = EasyPredictModelWrapper(MaliciousDomainModel())
p = model.predictBinomial(row)
```



H2O Model POJO

```
• static final class BETA_0 implements java.io.Serializable {
    static final void fill(double[] sa) {
      sa[0] = 1.49207826021648;
      sa[1] = 2.850271697856019 4;
      sa[2] = -8.839804567200542;
      sa[3] = -0.7977065034624655;
      sa[4] = -14.94132841574946;
    }
  }
```



Thank you!

@Navdeep_Gill_ on Twitter
navdeep-G on GitHub
navdeep@h2o.ai

Slides available at: https://github.com/h2oai/h2o-meetups/tree/master/2016_05_25_AWS
H2O + EC2 Tutorial: https://github.com/h2oai/h2o-3/tree/master/ec2
H2O +EMR Tutorial: https://github.com/navdeep-G/sparkling-water-emr
H2O + AWS Lambda Tutorial: https://github.com/h2oai/app-malicious-domains