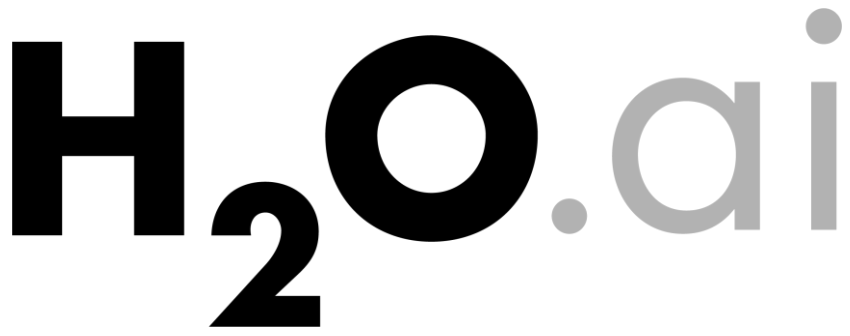
Introduction to Machine Learning with H_2O and Python



Jo-fai (Joe) Chow

Data Scientist

joe@h2o.ai

@matlabulous

PyData Amsterdam Tutorial at GoDataDriven

7th April, 2017

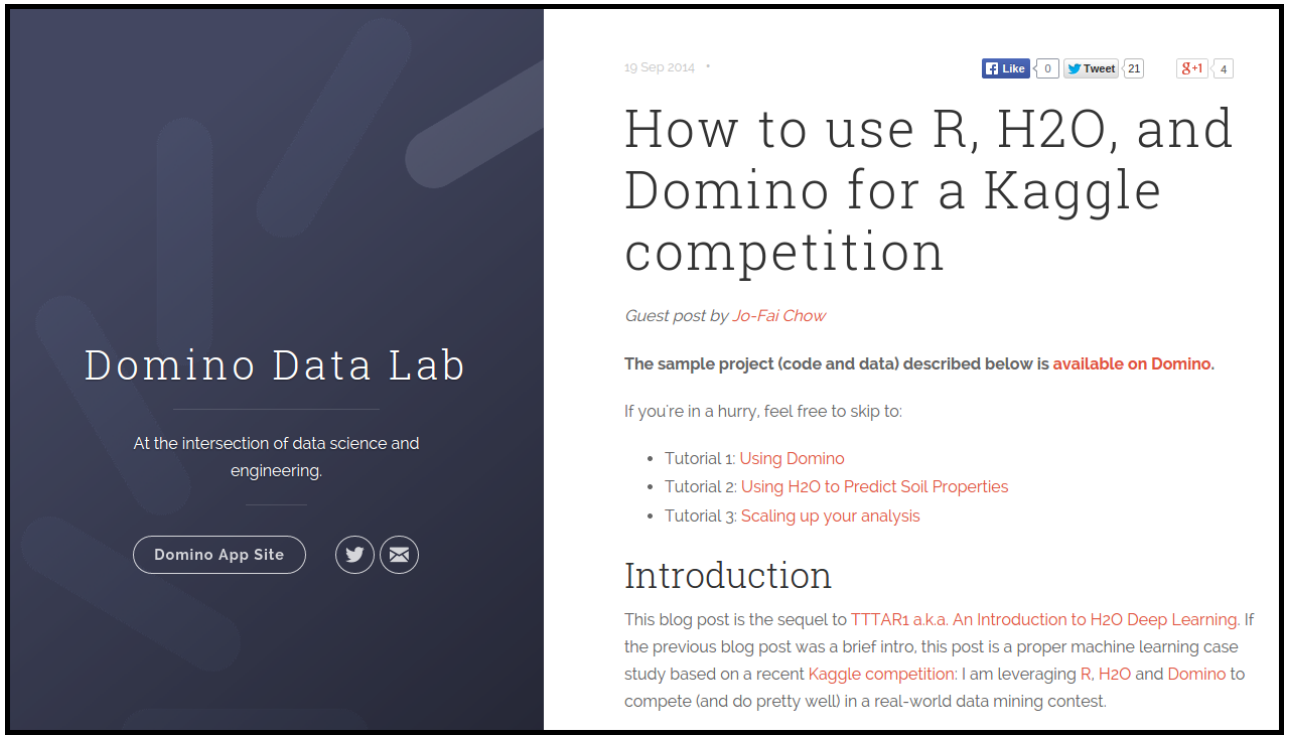
Slides and Code Examples:

bit.ly/joe_h2o_tutorials

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, US)
  - 2016 – Present
  - H_2O.ai (Silicon Valley, US)

About Me



R + H_2O + Domino for Kaggle

Guest Blog Post for Domino & H_2O (2014)

- The Long Story
  - bit.ly/joe_kaggle_story

H_2O.ai

Agenda

- About H_2O.ai
  - Company
  - Machine Learning Platform
- Tutorial
  - H_2O Python Module
    - Download & Install
  - Step-by-Step Examples:
    - Basic Data Import / Manipulation
    - Regression
    - Classification
    - Using H_2O in the Cloud

GoDataDriven
proudly part of Xebia Group

Booking.com

H2O.ai

About H_2O.ai

Company Overview

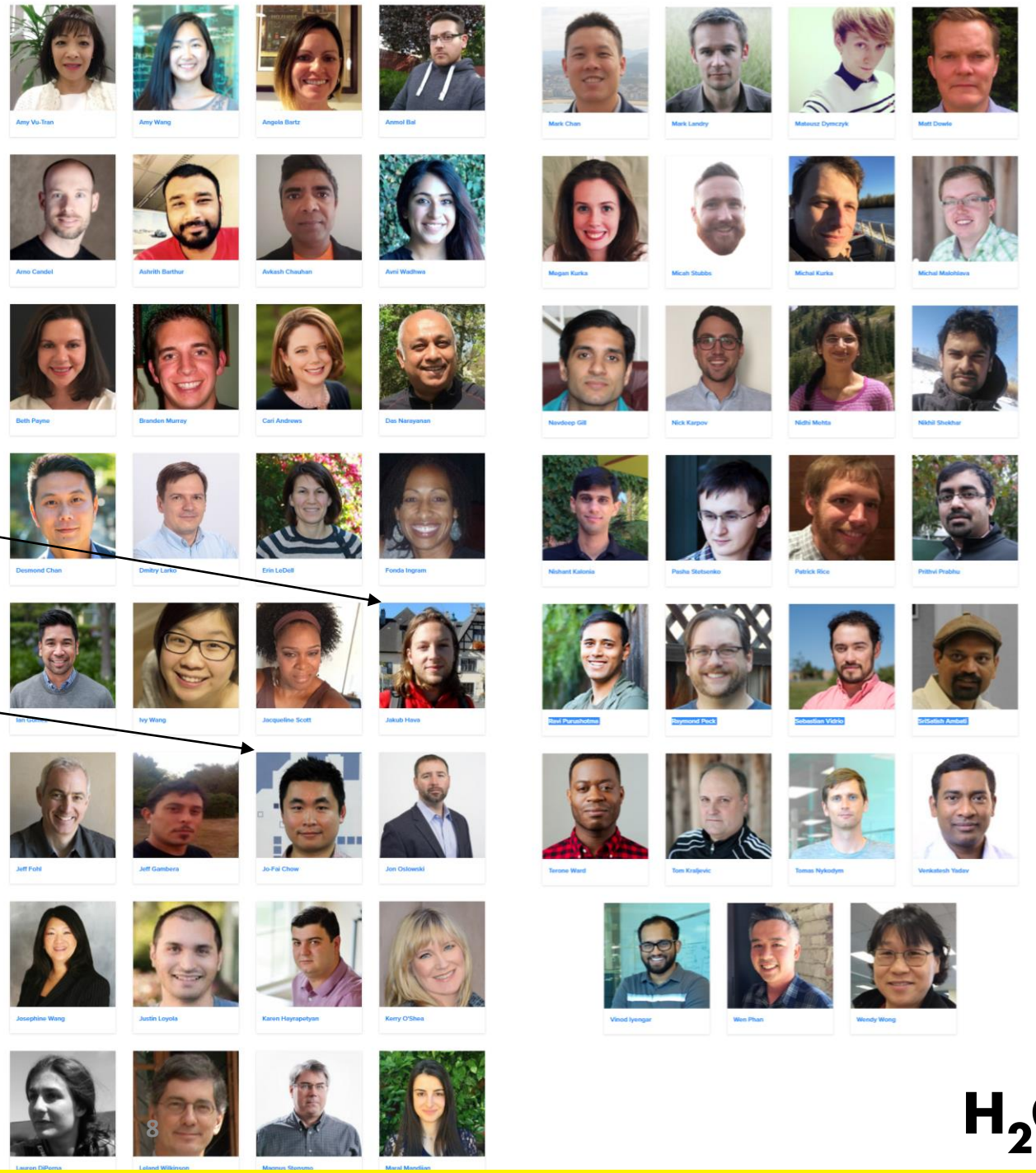
Founded	2011 Venture-backed, debuted in 2012
Products	• H_2O Open Source In-Memory AI Prediction Engine • Sparkling Water • Steam
Mission	Operationalize Data Science, and provide a platform for users to build beautiful data products
Team	70 employees • Distributed Systems Engineers doing Machine Learning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




Our Team

Kuba

Joe



H2O.ai

Scientific Advisory Council

Dr. Trevor Hastie

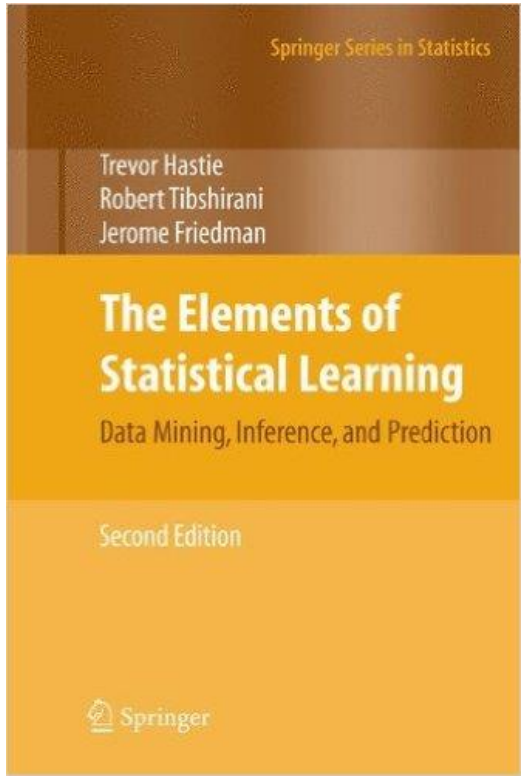
- John A. Overdeck Professor of Mathematics, Stanford University
- PhD in Statistics, Stanford University
- Co-author, *The Elements of Statistical Learning: Prediction, Inference and Data Mining*
- Co-author with John Chambers, *Statistical Models in S*
- Co-author, *Generalized Additive Models*

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

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/

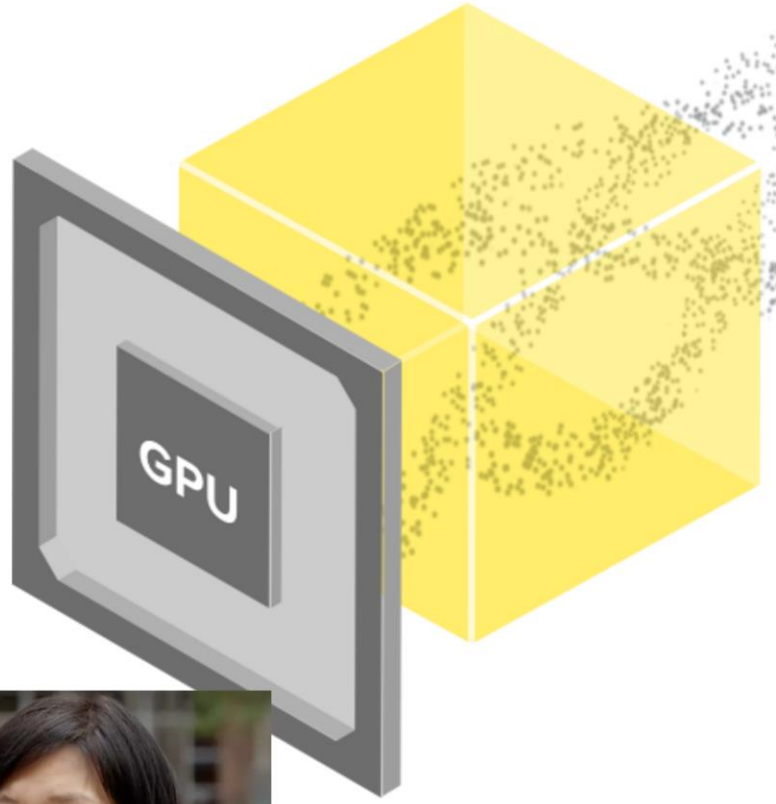
H2O.ai

Check out our website h2o.ai

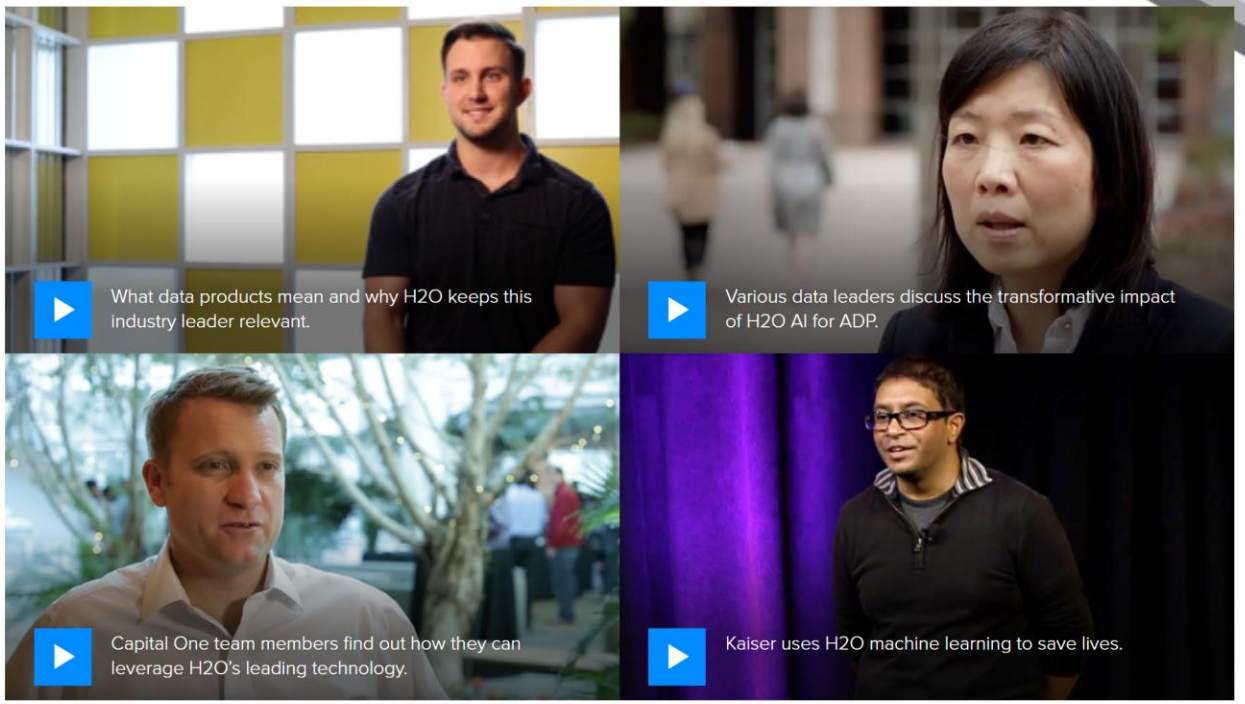
World Record Performance for AI

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

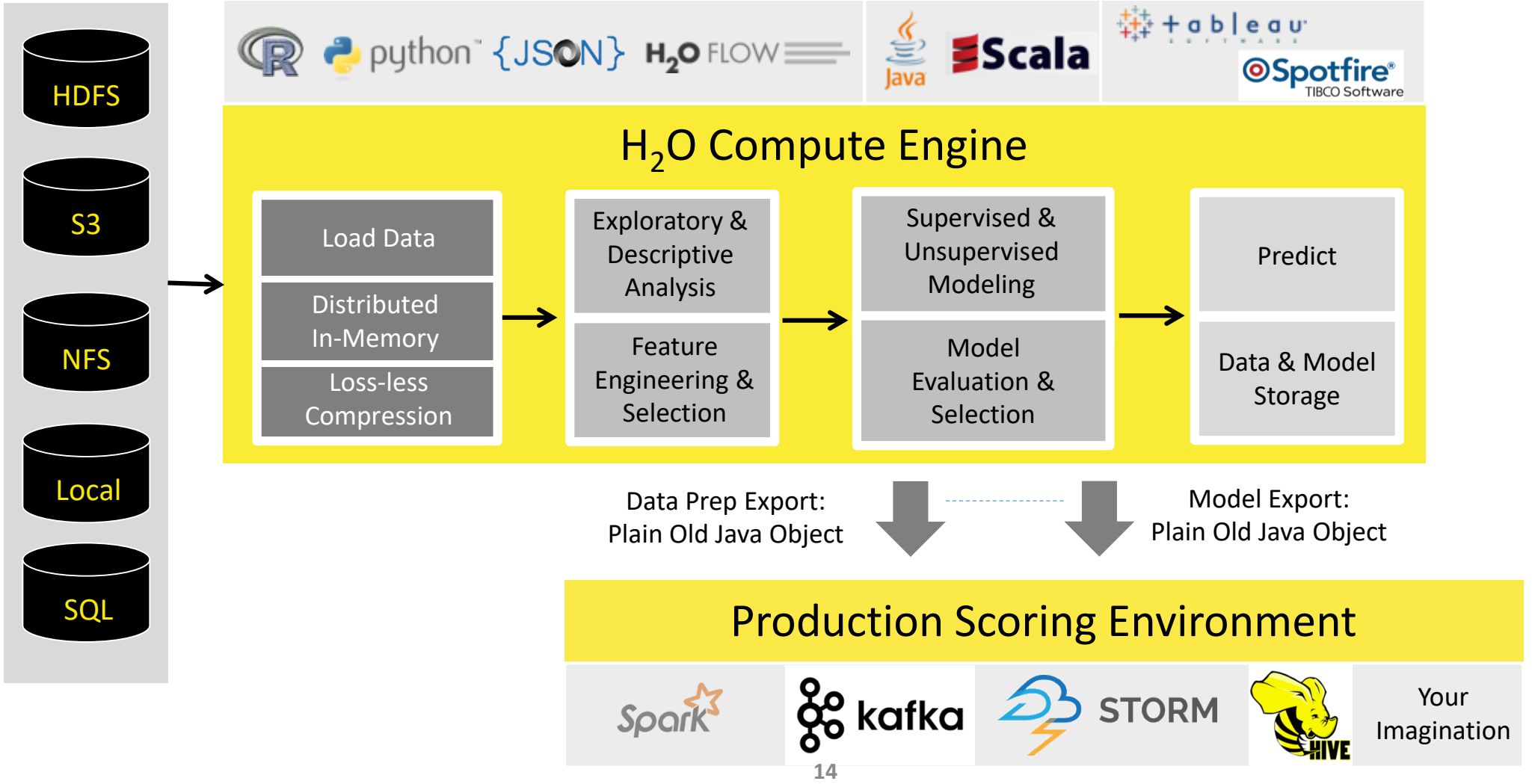


H2O in Action



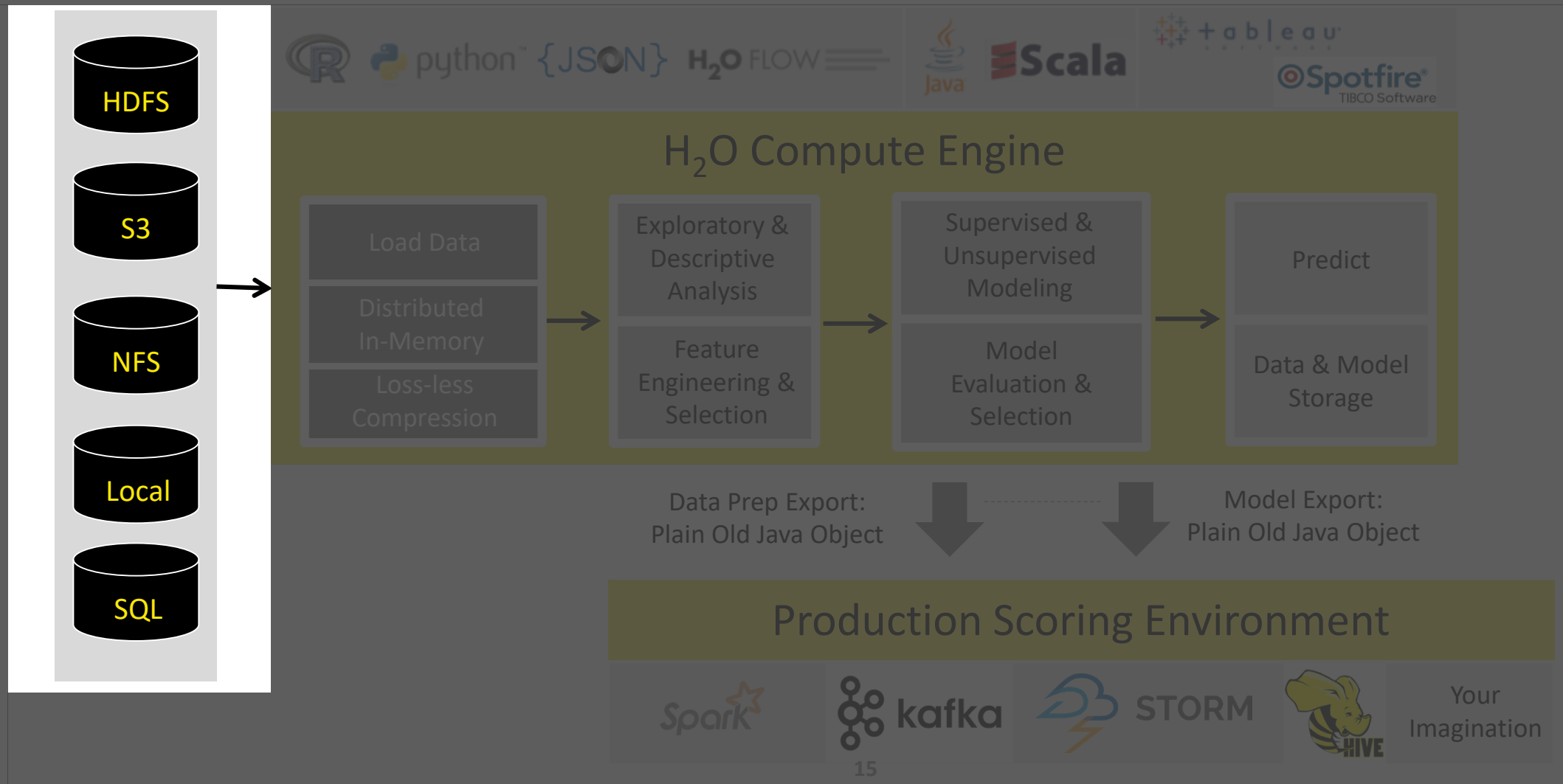
H_2O Machine Learning Platform

High Level Architecture



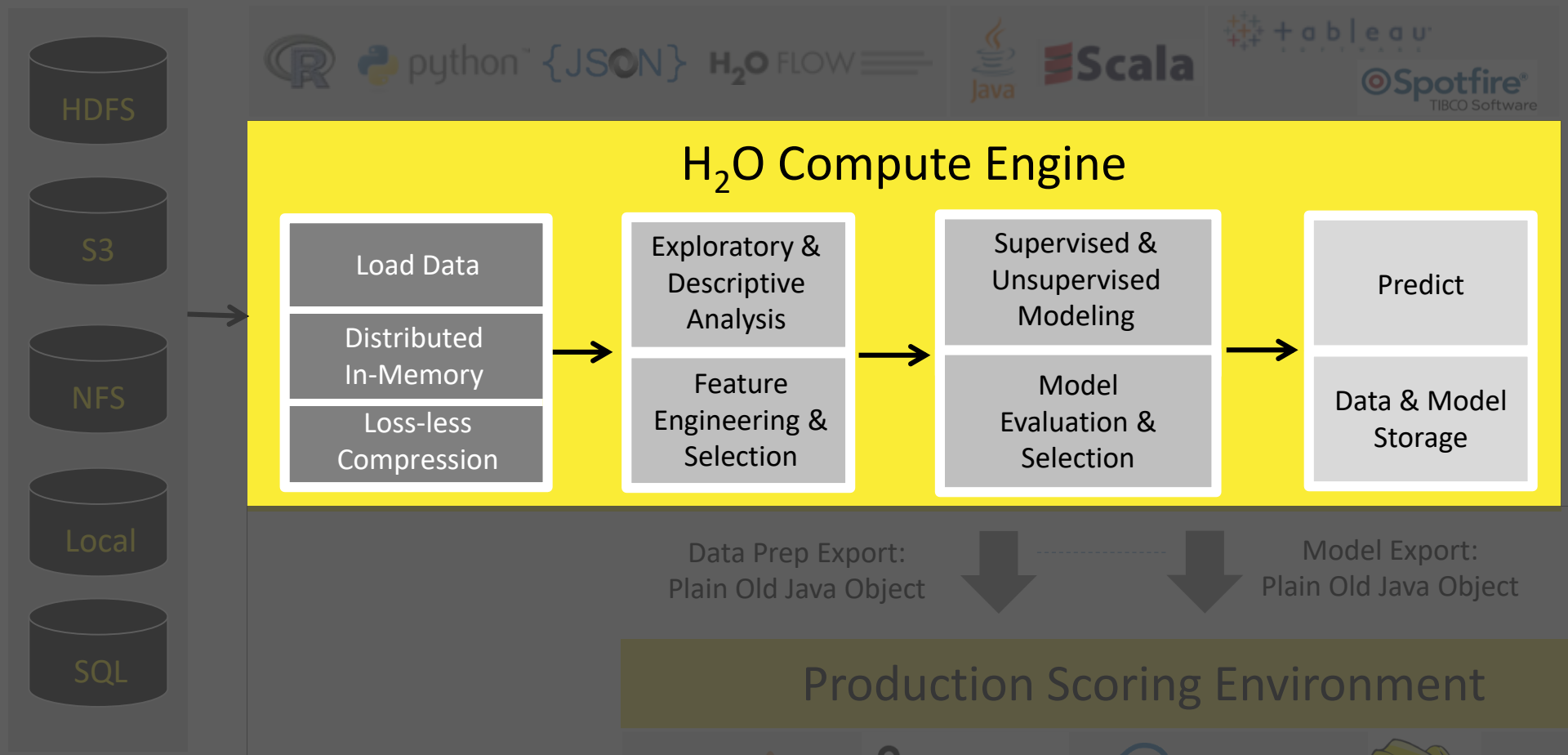
High Level Architecture

Import Data from Multiple Sources



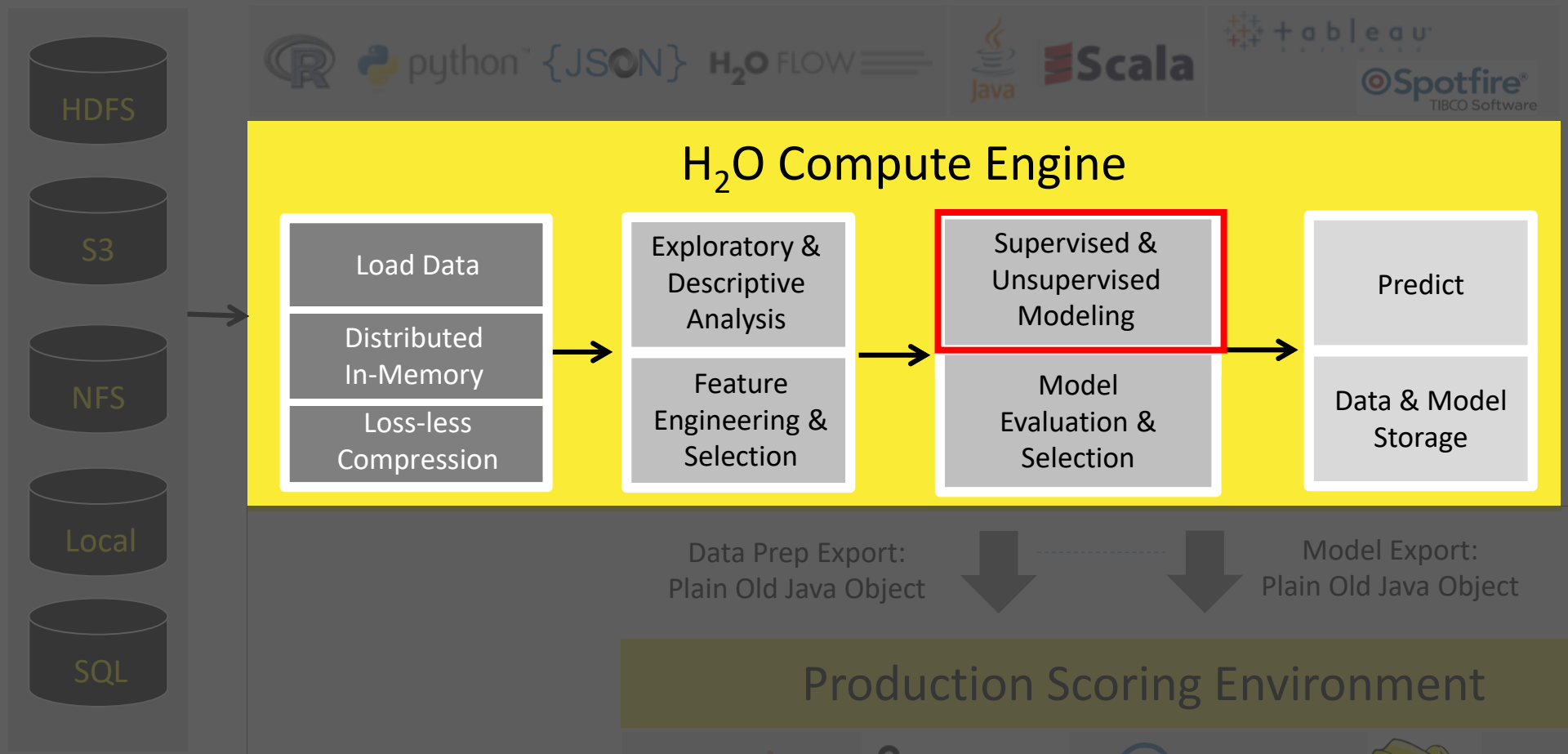
High Level Architecture

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%

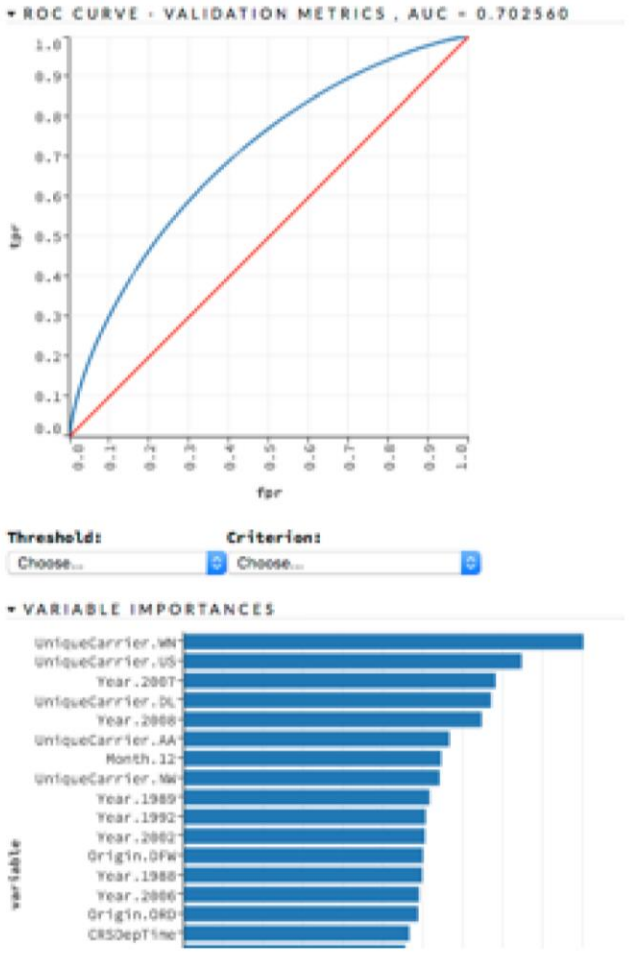
Iterations: 12. Epochs: 0.628821. Speed: 2,243,735 samples/sec. Estimated time left: 21.849 sec

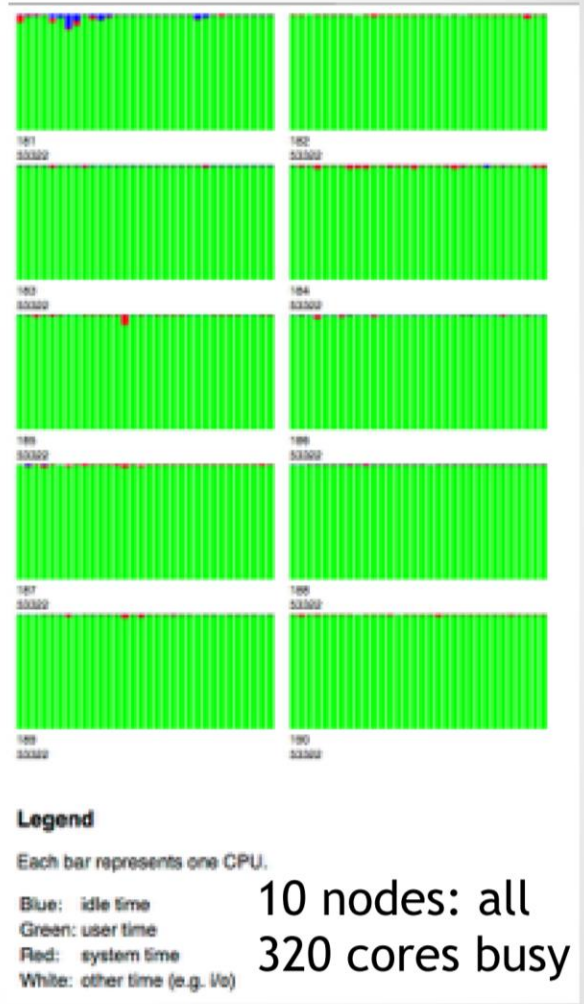
Actions: View | Cancel Job

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





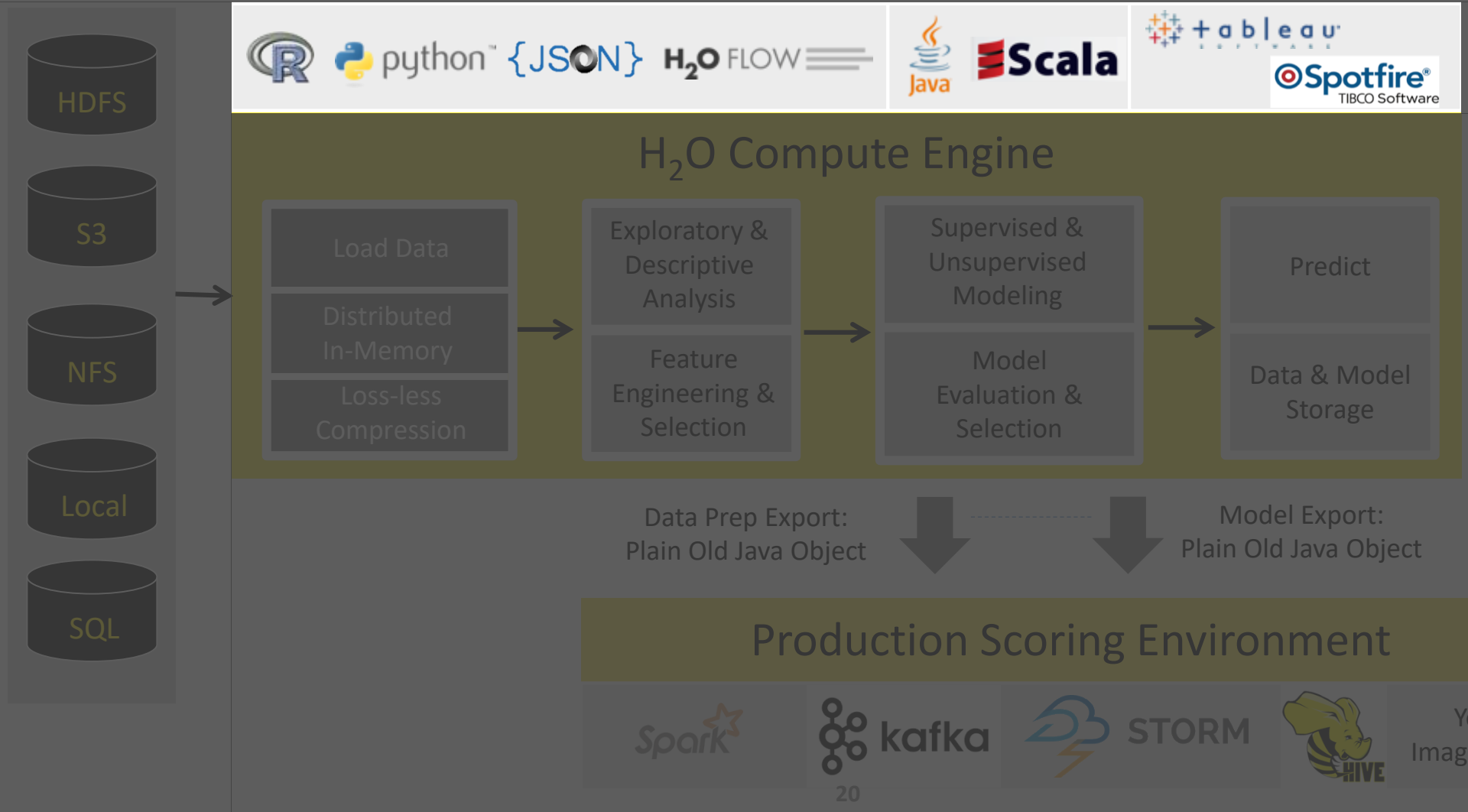
10 nodes: all
320 cores busy

real-time, interactive
model inspection in Flow

H_2O.ai **Deep Learning Model**

High Level Architecture

Multiple Interfaces



H_2O + Python

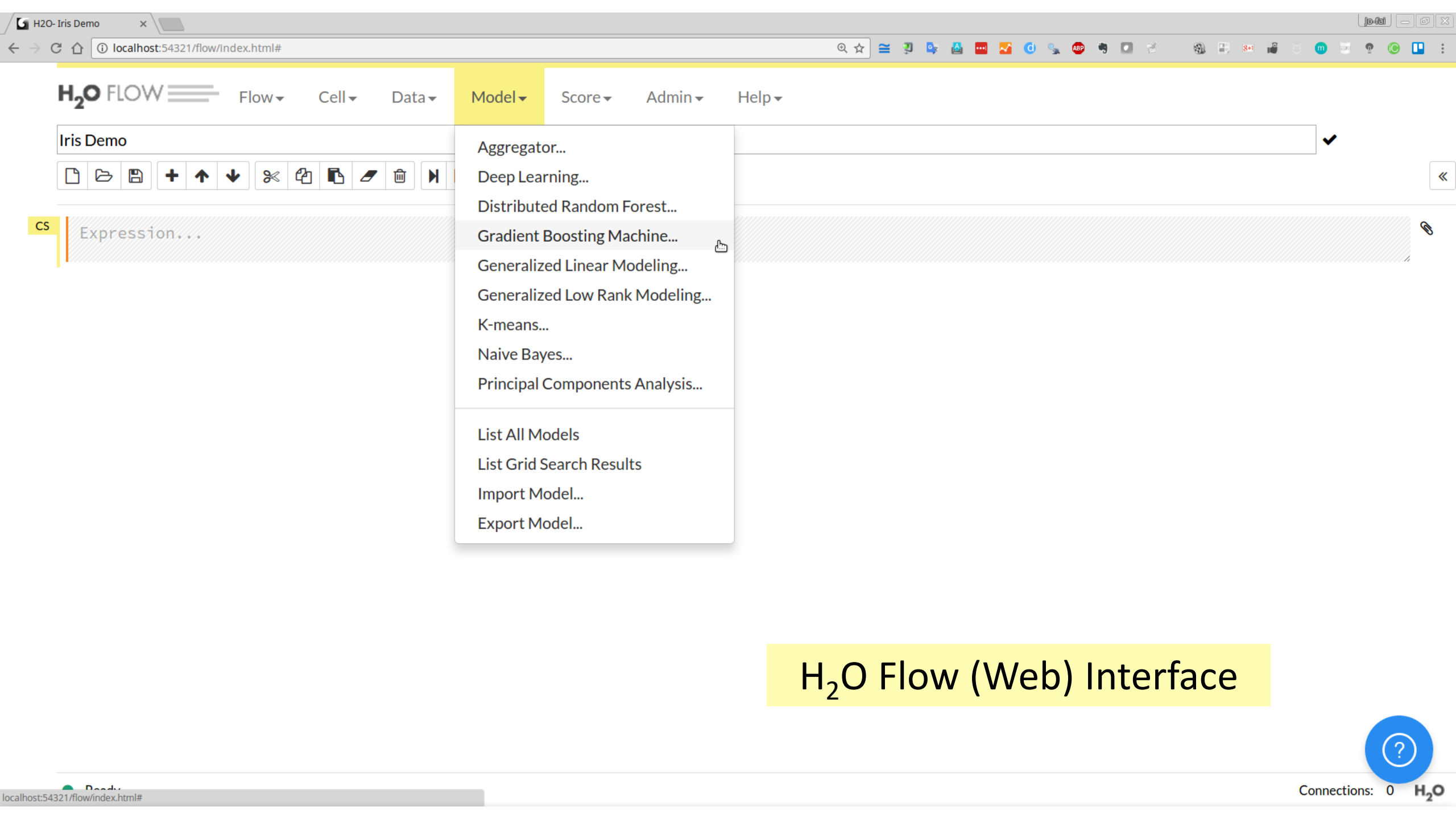
Gradient Boosting Machines

```
# 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 + R

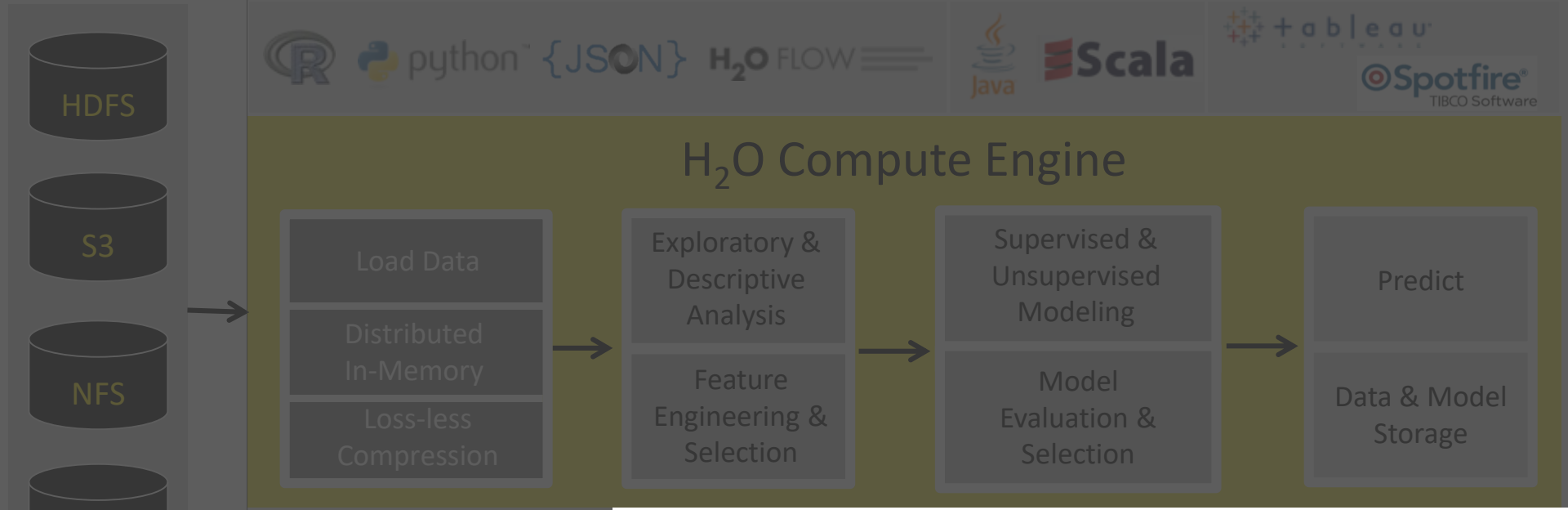
```
# ------------------------------------------------------------
# Train a H2O Model
# ------------------------------------------------------------

# 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 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 + Python Tutorial

Learning Objectives

- Start and connect to a local H_2O cluster from Python.
- Import data from Python data frames, local files or web.
- Perform basic data transformation and exploration.
- Train regression and classification models using various H_2O machine learning algorithms.
- Evaluate models and make predictions.
- Improve performance by tuning and stacking.

woobe / h2o_tutorials

Branch: master ▾ h2o_tutorials / introduction_to_machine_learning /

Create new file | Upload files | Find file | History

woobe Data for code examples — Latest commit aff9cf3 2 hours from now

..		
kaggle_titanic.csv	Data for code examples	just now
py_01_data_in_h2o.ipynb	Intro to ML with H2O and Python code examples	just now
py_02_data_manipulation.ipynb	Intro to ML with H2O and Python code examples	just now
py_03a_regression_basics.ipynb	Intro to ML with H2O and Python code examples	just now
py_03b_regression_grid_search.ipynb	Intro to ML with H2O and Python code examples	just now
py_03c_regression_ensembles.ipynb	Intro to ML with H2O and Python code examples	just now
py_04a_classification_basics.ipynb	Intro to ML with H2O and Python code examples	just now
py_04b_classification_ensembles.ipynb	Intro to ML with H2O and Python code examples	just now
py_05_h2o_in_the_cloud.ipynb	Intro to ML with H2O and Python code examples	just now
winequality-white.csv	Data for code examples	just now

Install H_2O

h2o.ai -> Download -> Install in Python

H_2O

Version 3.10.4.3

Fast Scalable Machine Learning API
For Smarter Applications

DOWNLOAD AND RUN | INSTALL IN R | INSTALL IN PYTHON | INSTALL ON HADOOP | USE FROM MAVEN

Use H_2O directly from Python

1. Prerequisite: Python 2.7

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

```
pip install requests
pip install tabulate
pip install scikit-learn
```

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

```
# The following command removes the H2O module for Python.
pip uninstall h2o

# Next, use pip to install this version of the H2O Python module.
pip install http://h2o-release.s3.amazonaws.com/h2o/rel-ueno/3/Python/h2o-3.10.4.3-py2.py3-none-any.whl
```

Start and Connect to a Local H_2O Cluster

py_01_data_in_h2o.ipynb

Local H_2O Cluster

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

Import H2O module

Start a local H2O cluster
nthreads = -1 means
using ALL CPU resources

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

```
Checking whether there is an H2O instance running at http://localhost:54321..... not found.
Attempting to start a local H2O server...
  Java Version: java version "1.8.0_121"; Java(TM) SE Runtime Environment (build 1.8.0_121-b13); Java HotSp
ot(TM) 64-Bit Server VM (build 25.121-b13, mixed mode)
  Starting server from /home/joe/anaconda3/lib/python3.5/site-packages/h2o/backend/bin/h2o.jar
  Ice root: /tmp/tmpxfli3vwa
  JVM stdout: /tmp/tmpxfli3vwa/h2o_joe_started_from_python.out
  JVM stderr: /tmp/tmpxfli3vwa/h2o_joe_started_from_python.err
  Server is running at http://127.0.0.1:54321
Connecting to H2O server at http://127.0.0.1:54321... successful.
```

H2O cluster uptime:	02 secs
H2O cluster version:	3.10.4.3
H2O cluster version age:	5 days
H2O cluster name:	H2O_from_python_joe_vr14r9
H2O cluster total nodes:	1
H2O cluster free memory:	5.210 Gb
H2O cluster total cores:	8
H2O cluster allowed cores:	8
H2O cluster status:	accepting new members, healthy
H2O connection url:	http://127.0.0.1:54321
H2O connection proxy:	None
H2O internal security:	False
Python version:	3.5.2 final

Importing Data into H_2O

py_01_data_in_h2o.ipynb

In [2]:
```
# Method 1 - Import data from a local CSV file
data_from_csv = h2o.import_file("winequality-white.csv")
data_from_csv.head(5)
```

Parse progress: |██| 100%

Out[2]:

fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
7	0.27	0.36	20.7	0.045	45	170	1.001	3	0.45	8.8	6
6.3	0.3	0.34	1.6	0.049	14	132	0.994	3.3	0.49	9.5	6
8.1	0.28	0.4	6.9	0.05	30	97	0.9951	3.26	0.44	10.1	6
7.2	0.23	0.32	8.5	0.058	47	186	0.9956	3.19	0.4	9.9	6
7.2	0.23	0.32	8.5	0.058	47	186	0.9956	3.19	0.4	9.9	6

In [3]:
```
# Method 2 - Import data from the web
data_from_web = h2o.import_file("https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequality-white.csv")
data_from_web.head(5)
```

Parse progress: |██| 100%

fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
7	0.27	0.36	20.7	0.045	45	170	1.001	3	0.45	8.8	6
6.3	0.3	0.34	1.6	0.049	14	132	0.994	3.3	0.49	9.5	6
8.1	0.28	0.4	6.9	0.05	30	97	0.9951	3.26	0.44	10.1	6
7.2	0.23	0.32	8.5	0.058	47	186	0.9956	3.19	0.4	9.9	6
7.2	0.23	0.32	8.5	0.058	47	186	0.9956	3.19	0.4	9.9	6

Out[3]:

In [4]:
```
# Method 3 - Convert Python data frame into H2O data frame

## Import Wine Quality data using Pandas
import pandas as pd
wine_df = pd.read_csv('winequality-white.csv', sep = ';')
wine_df.head(5)
```

Out[4]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.0	0.27	0.36	20.7	0.045	45.0	170.0	1.0010	3.00	0.45	8.8	6
1	6.3	0.30	0.34	1.6	0.049	14.0	132.0	0.9940	3.30	0.49	9.5	6
2	8.1	0.28	0.40	6.9	0.050	30.0	97.0	0.9951	3.26	0.44	10.1	6
3	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0.40	9.9	6
4	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0.40	9.9	6

In [5]:
```
## Convert Pandas data frame into H2O data frame
data_from_df = h2o.H2OFrame(wine_df)
data_from_df.head(5)
```

Parse progress: |██| 100%

fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
7	0.27	0.36	20.7	0.045	45	170	1.001	3	0.45	8.8	6
6.3	0.3	0.34	1.6	0.049	14	132	0.994	3.3	0.49	9.5	6
8.1	0.28	0.4	6.9	0.05	30	97	0.9951	3.26	0.44	10.1	6
7.2	0.23	0.32	8.5	0.058	47	186	0.9956	3.19	0.4	9.9	6
7.2	0.23	0.32	8.5	0.058	47	186	0.9956	3.19	0.4	9.9	6

Basic Data Transformation & Exploration

py_02_data_manipulation.ipynb

(see notebooks)

In [2]:
```
# Import Titanic data (local CSV)
titanic = h2o.import_file("kaggle_titanic.csv")
```

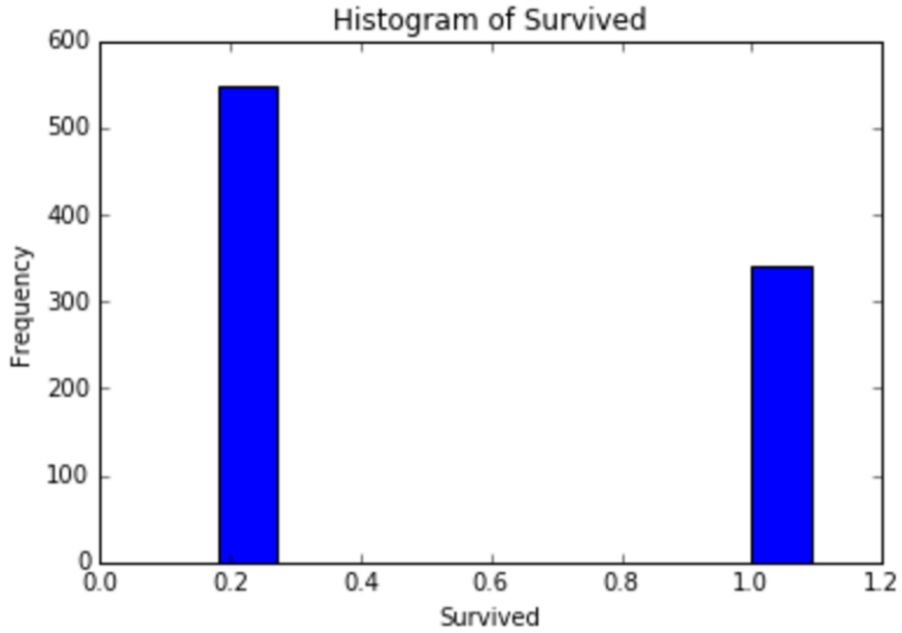
```
Parse progress: |████████████████████████████████████████████████████████| 100%
```

In [3]:
```
# Explore the dataset using various functions
titanic.head(10)
```

PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
1	0	3	Braund, Mr. Owen Harris	male	22	1	0	nan	7.25		S
2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Thayer)	female	38	1	0	nan	71.2833	C85	C
3	1	3	Heikkinen, Miss. Laina	female	26	0	0	nan	7.925		S
4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35	1	0	113803	53.1	C123	S
5	0	3	Allen, Mr. William Henry	male	35	0	0	373450	8.05		S
6	0	3	Moran, Mr. James	male	nan	0	0	330877	8.4583		Q
7	0	1	McCarthy, Mr. Timothy J	male	54	0	0	17463	51.8625	E46	S
8	0	3	Palsson, Master. Gosta Leonard	male	2	3	1	349909	21.075		S
9	1	3	Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)	female	27	0	2	347742	11.1333		S
10	1	2	Nasser, Mrs. Nicholas (Adele Achem)	female	14	1	0	237736	30.0708		C

Out[3]:

In [5]:
```
# Use hist() to create a histogram
titanic['Survived'].hist()
```



In [6]:
```
# Use table() to summarize 0s and 1s
titanic['Survived'].table()
```

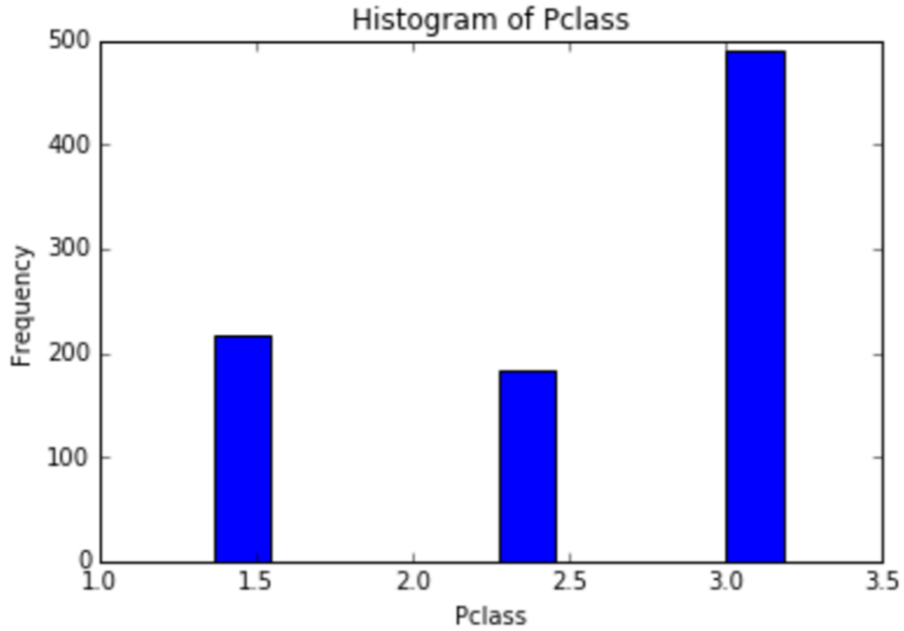
Survived	Count
0	549
1	342

In [7]:
```
# Convert 'Survived' to categorical variable
titanic['Survived'] = titanic['Survived'].asfactor()
```

In [8]:
```
# Look at the summary of 'Survived' again
# The feature is now an 'enum' (enum is the name of categorical variable in Java)
titanic['Survived'].summary()
```

	Survived
type	enum
mins	
mean	
maxs	
sigma	
zeros	
missing	0

In [10]:
```
# Use hist() to create a histogram
titanic['Pclass'].hist()
```



In [11]:
```
# Use table() to summarize 1s, 2s and 3s
titanic['Pclass'].table()
```

Pclass	Count
1	216
2	184
3	491

Regression Models (Basics)

py_03a_regression_basics.ipynb

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

docs.h2o.ai

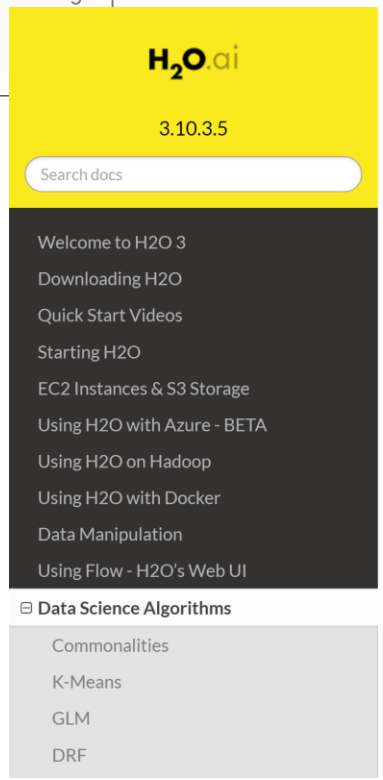
Data Science Algorithms

Supervised Learning

Generalized Linear Modeling (GLM)	Tutorial	Booklet	Reference	Tuning
Gradient Boosting Machine (GBM)	Tutorial	Booklet	Reference	Tuning
Deep Learning	Tutorial	Booklet	Reference	Tuning
Distributed Random Forest	Tutorial	Booklet	Reference	Tuning
Naive Bayes	Tutorial	Booklet	Reference	Tuning
Ensembles (Stacking)	Tutorial	Booklet	Reference	

Unsupervised Learning

Generalized Low Rank Models (GLRM)	Tutorial	Reference
K-Means Clustering	Tutorial	Reference
Principal Components Analysis (PCA)	Tutorial	Reference



Docs » Data Science Algorithms » GBM View page source

GBM

Introduction

Gradient Boosting Machine (for Regression and Classification) is a forward learning ensemble method. The guiding heuristic is that good predictive results can be obtained through increasingly refined approximations. H2O's GBM sequentially builds regression trees on all the features of the dataset in a fully distributed way - each tree is built in parallel.

The current version of GBM is fundamentally the same as in previous versions of H2O (same algorithmic steps, same histogramming techniques), with the exception of the following changes:

- Improved ability to train on categorical variables (using the `nbins_cats` parameter)
- Minor changes in histogramming logic for some corner cases

There was some code cleanup and refactoring to support the following features:

- Per-row observation weights
- Per-row offsets
- N-fold cross-validation
- Support for more distribution functions (such as Gamma, Poisson, and Tweedie)

H2O.ai

In [2]:
```
# Import wine quality data from a local CSV file
wine = h2o.import_file("winequality-white.csv")
wine.head(5)
```

Parse progress: |██| 100%

fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
7	0.27	0.36	20.7	0.045	45	170	1.001	3	0.45	8.8	6
6.3	0.3	0.34	1.6	0.049	14	132	0.994	3.3	0.49	9.5	6
8.1	0.28	0.4	6.9	0.05	30	97	0.9951	3.26	0.44	10.1	6
7.2	0.23	0.32	8.5	0.058	47	186	0.9956	3.19	0.4	9.9	6
7.2	0.23	0.32	8.5	0.058	47	186	0.9956	3.19	0.4	9.9	6

In [3]:
```
# Define features (or predictors)
features = list(wine.columns) # we want to use all the information
features.remove('quality')    # we need to exclude the target 'quality' (otherwise there is nothing to pre
dict)
features
```

Out[3]:
```
['fixed acidity',
 'volatile acidity',
 'citric acid',
 'residual sugar',
 'chlorides',
 'free sulfur dioxide',
 'total sulfur dioxide',
 'density',
 'pH',
 'sulphates',
 'alcohol']
```

In [4]:
```
# Split the H2O data frame into training/test sets
# so we can evaluate out-of-bag performance
wine_split = wine.split_frame(ratios = [0.8], seed = 1234)

wine_train = wine_split[0] # using 80% for training
wine_test = wine_split[1]  # using the rest 20% for out-of-bag evaluation
```

In [5]:
```
wine_train.shape
```

Out[5]: (3932, 12)

In [6]:
```
wine_test.shape
```

Out[6]: (966, 12)

Generalized Linear Model

In [7]:
```
# Build a Generalized Linear Model (GLM) with default settings

# Import the function for GLM
from h2o.estimators.glm import H2OGeneralizedLinearEstimator

# Set up GLM for regression
glm_default = H2OGeneralizedLinearEstimator(family = 'gaussian', model_id = 'glm_default')

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

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

Regression Performance – MSE

If $\hat{Y}$ is a vector of n predictions, and Y is the vector of observed values corresponding to the inputs to the function which generated the predictions, then the MSE of the predictor can be estimated by

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(\hat{Y}_i - Y_i)^2$$

I.e., the MSE is the *mean* $\left(\frac{1}{n}\sum_{i=1}^{n}\right)$ of the *square of the errors* ($(\hat{Y}_i - Y_i)^2$). This is an easily computable quantity for a particular sample (and hence is sample-dependent).

In [8]:
```
# Check the model performance on training dataset
glm_default
```

```
Model Details
=============
H2OGeneralizedLinearEstimator :  Generalized Linear Modeling
Model Key:  glm_default


ModelMetricsRegressionGLM: glm
** Reported on train data. **

MSE: 0.5663260600439028
RMSE: 0.7525463839816805
MAE: 0.5855739117180464
RMSLE: 0.11135798916908822
R^2: 0.28516909778801736
Mean Residual Deviance: 0.5663260600439028
Null degrees of freedom: 3931
Residual degrees of freedom: 3920
Null deviance: 3115.1340284842345
Residual deviance: 2226.794068092626
AIC: 8948.855269434132
```

In [9]:
```
# Check the model performance on test dataset
glm_default.model_performance(wine_test)
```

```
ModelMetricsRegressionGLM: glm
** Reported on test data. **

MSE: 0.5546397919709444
RMSE: 0.7447414262486977
MAE: 0.5795791157106437
RMSLE: 0.11079661971451717
R^2: 0.26184927981796147
Mean Residual Deviance: 0.5546397919709444
Null degrees of freedom: 965
Residual degrees of freedom: 954
Null deviance: 725.858730540242
Residual deviance: 535.7820390439323
AIC: 2197.9936843132646
```

Making Predictions

In [19]:
```
# Use GLM model to make predictions
yhat_test_glm = glm_default.predict(wine_test)
yhat_test_glm.head(5)
```

```
glm prediction progress: |████████████████████████████████████████████████| 100%
```

predict
5.76109
5.76721
5.64325
5.85764
5.77967

Out[19]:

Distributed Random Forest

```
# Build a Distributed Random Forest (DRF) model with default settings

# Import the function for DRF
from h2o.estimators.random_forest import H2ORandomForestEstimator

# Set up DRF for regression
# Add a seed for reproducibility
drf_default = H2ORandomForestEstimator(model_id = 'drf_default', seed = 1234)

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

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)
```

H2O Deep Learning

```
# Build a Deep Learning (Deep Neural Networks, DNN) model with default settings

# Import the function for DNN
from h2o.estimators.deeplearning import H2ODeepLearningEstimator

# Set up DNN for regression
dnn_default = H2ODeepLearningEstimator(model_id = 'dnn_default')

# (not run) Change 'reproducible' to True if you want to reproduce the results
# The model will be built using a single thread (could be very slow)
# dnn_default = H2ODeepLearningEstimator(model_id = 'dnn_default', reproducible = True)

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

Classification Models (Basics)

py_04_classification_basics.ipynb

In [2]:
```
# Import Titanic data (local CSV)
titanic = h2o.import_file("kaggle_titanic.csv")
titanic.head(5)
```

```
Parse progress: |████████████████████████████████████████████████████████| 100%
```

PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
1	0	3	Braund, Mr. Owen Harris	male	22	1	0	nan	7.25		S
2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Thayer)	female	38	1	0	nan	71.2833	C85	C
3	1	3	Heikkinen, Miss. Laina	female	26	0	0	nan	7.925		S
4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35	1	0	113803	53.1	C123	S
5	0	3	Allen, Mr. William Henry	male	35	0	0	373450	8.05		S

Out[2]:

In [3]:
```
# Convert 'Survived' and 'Pclass' to categorical values
titanic['Survived'] = titanic['Survived'].asfactor()
titanic['Pclass'] = titanic['Pclass'].asfactor()
```

In [4]:
```
titanic['Survived'].table()
```

Out[4]:

Survived	Count
0	549
1	342

In [5]:
```
titanic['Pclass'].table()
```

Out[5]:

Pclass	Count
1	216
2	184
3	491

In [12]:
```
# Define features (or predictors) manually
features = ['Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Fare', 'Embarked']
```

In [13]:
```
# Split the H2O data frame into training/test sets
# so we can evaluate out-of-bag performance
titanic_split = titanic.split_frame(ratios = [0.8], seed = 1234)

titanic_train = titanic_split[0] # using 80% for training
titanic_test = titanic_split[1]  # using the rest 20% for out-of-bag evaluation
```

In [14]:
```
titanic_train.shape
```

Out[14]: (712, 12)

In [15]:
```
titanic_test.shape
```

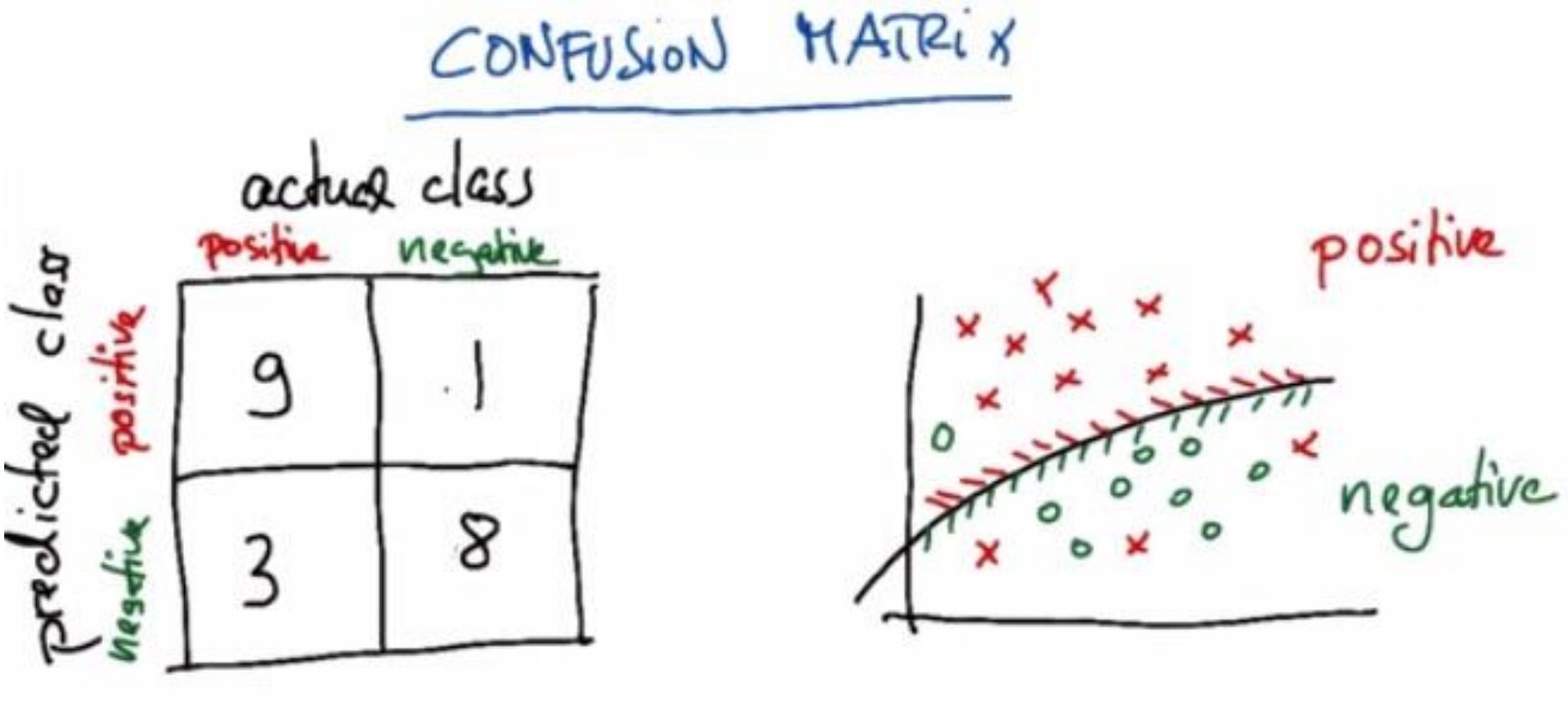
Out[15]: (179, 12)

Generalized Linear Model

In [16]:
```
# Build a Generalized Linear Model (GLM) with default settings

# Import the function for GLM
from h2o.estimators.glm import H2OGeneralizedLinearEstimator

# Set up GLM for binary classification
glm_default = H2OGeneralizedLinearEstimator(family = 'binomial', model_id = 'glm_default')

# Use .train() to build the model
glm_default.train(x = features,
                  y = 'Survived',
                  training_frame = titanic_train)
```

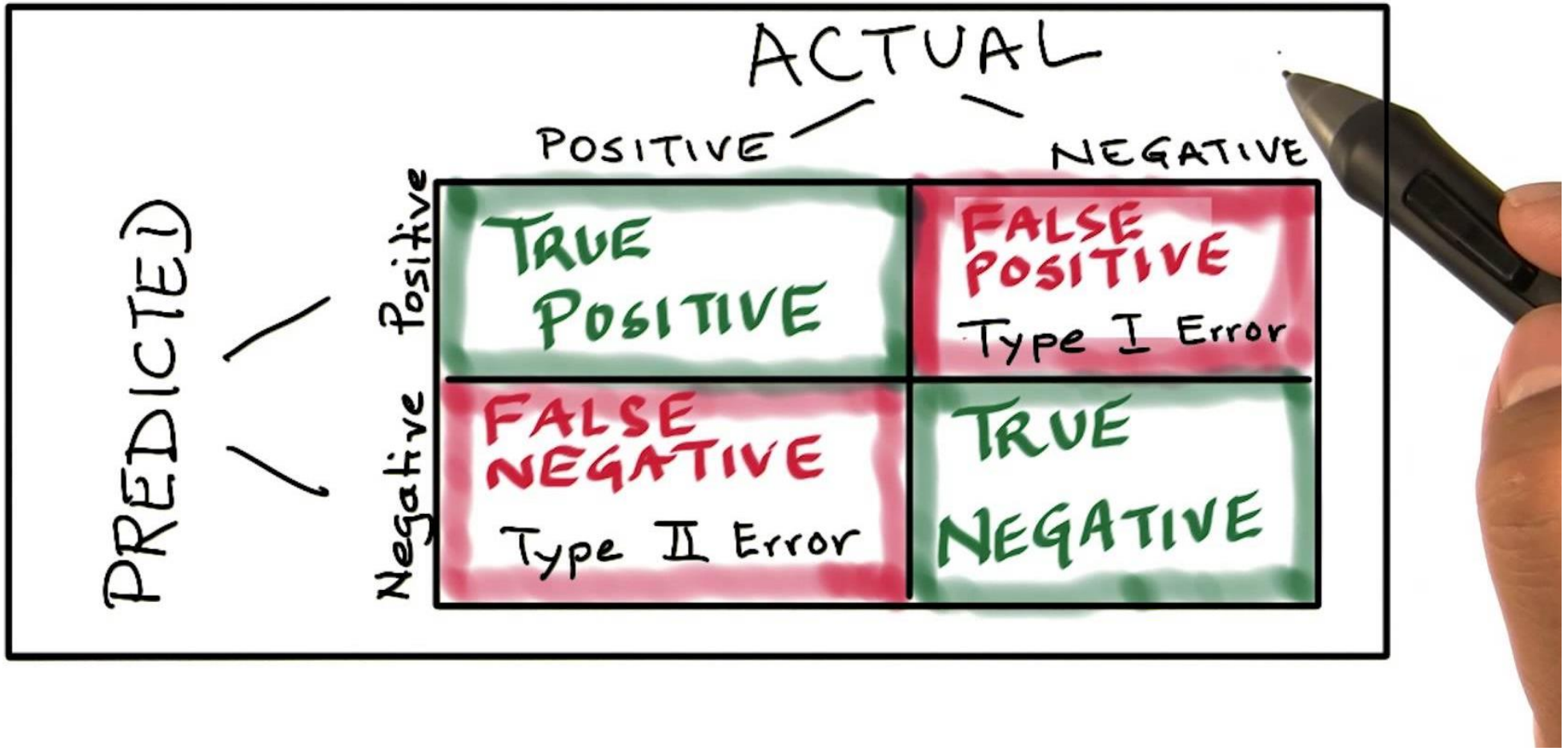
```
glm Model Build progress: |████████████████████████████████████████████████| 100%
```

Classification Performance – Confusion Matrix



H2O.ai

Confusion Matrix



H2O.ai

In [17]:
```
# Check the model performance on training dataset
glm_default
```

```
Model Details
=============
H2OGeneralizedLinearEstimator :  Generalized Linear Modeling
Model Key:  glm_default


ModelMetricsBinomialGLM: glm
** Reported on train data. **

MSE: 0.1382288206898215
RMSE: 0.37179136715343664
LogLoss: 0.4365971276608109
Null degrees of freedom: 711
Residual degrees of freedom: 700
Null deviance: 939.9987544934203
Residual deviance: 621.7143097889948
AIC: 645.7143097889948
AUC: 0.8541387024608501
Gini: 0.7082774049217002
Confusion Matrix (Act/Pred) for max f1 @ threshold = 0.326408717958219:
```

	0	**1**	**Error**	**Rate**
0	354.0	93.0	0.2081	(93.0/447.0)
1	54.0	211.0	0.2038	(54.0/265.0)
Total	408.0	304.0	0.2065	(147.0/712.0)

In [18]:
```
# Check the model performance on test dataset
glm_default.model_performance(titanic_test)
```

```
ModelMetricsBinomialGLM: glm
** Reported on test data. **

MSE: 0.14604424271085578
RMSE: 0.38215735333872064
LogLoss: 0.46125694368411685
Null degrees of freedom: 178
Residual degrees of freedom: 167
Null deviance: 247.17154578244123
Residual deviance: 165.12998583891383
AIC: 189.12998583891383
AUC: 0.8584797555385791
Gini: 0.7169595110771583
Confusion Matrix (Act/Pred) for max f1 @ threshold = 0.6037543251670895:
```

	0	**1**	**Error**	**Rate**
0	96.0	6.0	0.0588	(6.0/102.0)
1	24.0	53.0	0.3117	(24.0/77.0)
Total	120.0	59.0	0.1676	(30.0/179.0)

Making Predictions

In [28]:
```
# Use GLM model to make predictions
yhat_test_glm = glm_default.predict(titanic_test)
yhat_test_glm.head(5)
```

```
glm prediction progress: |████████████████████████████████████████████████| 100%
```

predict	p0	p1
1	0.099139	0.900861
1	0.110799	0.889201
1	0.262639	0.737361
0	0.789631	0.210369
1	0.482829	0.517171

Out[28]:

Distributed Random Forest

```
# Build a Distributed Random Forest (DRF) model with default settings

# Import the function for DRF
from h2o.estimators.random_forest import H2ORandomForestEstimator

# Set up DRF for regression
# Add a seed for reproducibility
drf_default = H2ORandomForestEstimator(model_id = 'drf_default', seed = 1234)

# Use .train() to build the model
drf_default.train(x = features,
                  y = 'Survived',
                  training_frame = titanic_train)
```

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 = 'Survived',
                  training_frame = titanic_train)
```

H2O Deep Learning

```
# Build a Deep Learning (Deep Neural Networks, DNN) model with default settings

# Import the function for DNN
from h2o.estimators.deeplearning import H2ODeepLearningEstimator

# Set up DNN for regression
dnn_default = H2ODeepLearningEstimator(model_id = 'dnn_default')

# (not run) Change 'reproducible' to True if you want to reproduce the results
# The model will be built using a single thread (could be very slow)
# dnn_default = H2ODeepLearningEstimator(model_id = 'dnn_default', reproducible = True)

# Use .train() to build the model
dnn_default.train(x = features,
                  y = 'Survived',
                  training_frame = titanic_train)
```

Regression Models (Tuning)

py_03b_regression_grid_search.ipynb

Improving Model Performance (Step-by-Step)

Model Settings	MSE (CV)	MSE (Test)
GBM with default settings	N/A	0.4551
GBM with manual settings	N/A	0.4433
Manual settings + cross-validation	0.4502	0.4433
Manual + CV + early stopping	0.4429	0.4287
CV + early stopping + full grid search	0.4378	0.4196
CV + early stopping + random grid search	0.4227	0.4047
Stacking best two from random grid search	N/A	0.3969

In [2]:
```
# Import wine quality data from a local CSV file
wine = h2o.import_file("winequality-white.csv")
wine.head(5)
```

Parse progress: |██| 100%

fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
7	0.27	0.36	20.7	0.045	45	170	1.001	3	0.45	8.8	6
6.3	0.3	0.34	1.6	0.049	14	132	0.994	3.3	0.49	9.5	6
8.1	0.28	0.4	6.9	0.05	30	97	0.9951	3.26	0.44	10.1	6
7.2	0.23	0.32	8.5	0.058	47	186	0.9956	3.19	0.4	9.9	6
7.2	0.23	0.32	8.5	0.058	47	186	0.9956	3.19	0.4	9.9	6

Out[2]:

In [3]:
```
# Define features (or predictors)
features = list(wine.columns) # we want to use all the information
features.remove('quality')    # we need to exclude the target 'quality' (otherwise there is nothing to pre
dict)
features
```

Out[3]:
```
['fixed acidity',
 'volatile acidity',
 'citric acid',
 'residual sugar',
 'chlorides',
 'free sulfur dioxide',
 'total sulfur dioxide',
 'density',
 'pH',
 'sulphates',
 'alcohol']
```

Step 1 - Gradient Boosting Machines (GBM) with Default Settings

In [7]:
```
# 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%
```

In [8]:
```
# Check the model performance on test dataset
gbm_default.model_performance(wine_test)
```

```
ModelMetricsRegression: gbm
** Reported on test data. **

MSE: 0.45511211588709155
RMSE: 0.6746199788674299
MAE: 0.5219768028633305
RMSLE: 0.10013755931021842
Mean Residual Deviance: 0.45511211588709155
```

Out[8]:

Improving Model Performance (Step-by-Step)

Model Settings	MSE (CV)	MSE (Test)
GBM with default settings	N/A	0.4551
GBM with manual settings	N/A	0.4433
Manual settings + cross-validation	0.4502	0.4433
Manual + CV + early stopping	0.4429	0.4287
CV + early stopping + full grid search	0.4378	0.4196
CV + early stopping + random grid search	0.4227	0.4047
Stacking best two from random grid search	N/A	0.3969

Step 2 - GBM with Manual Settings

In [9]:
```
# Build a GBM with manual settings

# Set up GBM for regression
# Add a seed for reproducibility
gbm_manual = H2OGradientBoostingEstimator(model_id = 'gbm_manual',
                                          seed = 1234,
                                          ntrees = 100,
                                          sample_rate = 0.9,
                                          col_sample_rate = 0.9)

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

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

In [10]:
```
# Check the model performance on test dataset
gbm_manual.model_performance(wine_test)
```

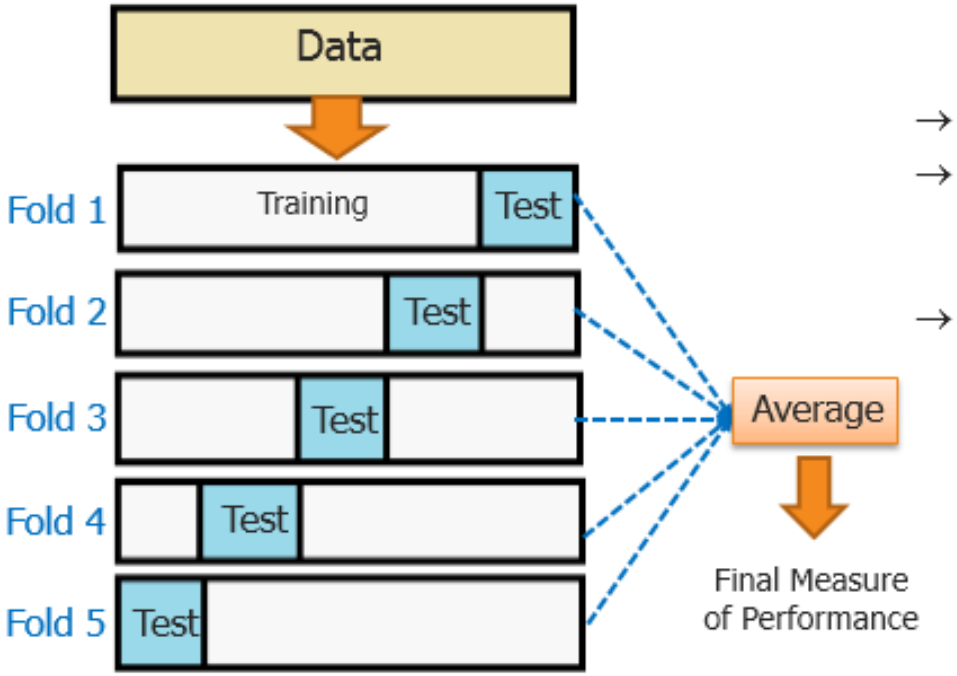
```
ModelMetricsRegression: gbm
** Reported on test data. **

MSE: 0.44325665649714924
RMSE: 0.6657752297113112
MAE: 0.5114358481376113
RMSLE: 0.09895809708429235
Mean Residual Deviance: 0.44325665649714924
```

Out[10]:

Improving Model Performance (Step-by-Step)

Model Settings	MSE (CV)	MSE (Test)
GBM with default settings	N/A	0.4551
GBM with manual settings	N/A	0.4433
Manual settings + cross-validation	0.4502	0.4433
Manual + CV + early stopping	0.4429	0.4287
CV + early stopping + full grid search	0.4378	0.4196
CV + early stopping + random grid search	0.4227	0.4047
Stacking best two from random grid search	N/A	0.3969

Cross-Validation



→ Technique to validate models/classifiers

→ Method to estimate how accurately the model generalizes to unseen data i.e., how well it performs/predicts

→ K-fold CV

- » Most popular
- » k is typically set to 10
- » Every sample/record is used both in training and test sets

Step 3 - GBM with Manual Settings & Cross-Validation (CV)

In [11]:
```
# Build a GBM with manual settings & cross-validation

# Set up GBM for regression
# Add a seed for reproducibility
gbm_manual_cv = H2OGradientBoostingEstimator(model_id = 'gbm_manual_cv',
                                             seed = 1234,
                                             ntrees = 100,
                                             sample_rate = 0.9,
                                             col_sample_rate = 0.9,
                                             nfolds = 5)

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

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

In [12]:
```
# Check the cross-validation model performance
gbm_manual_cv
```

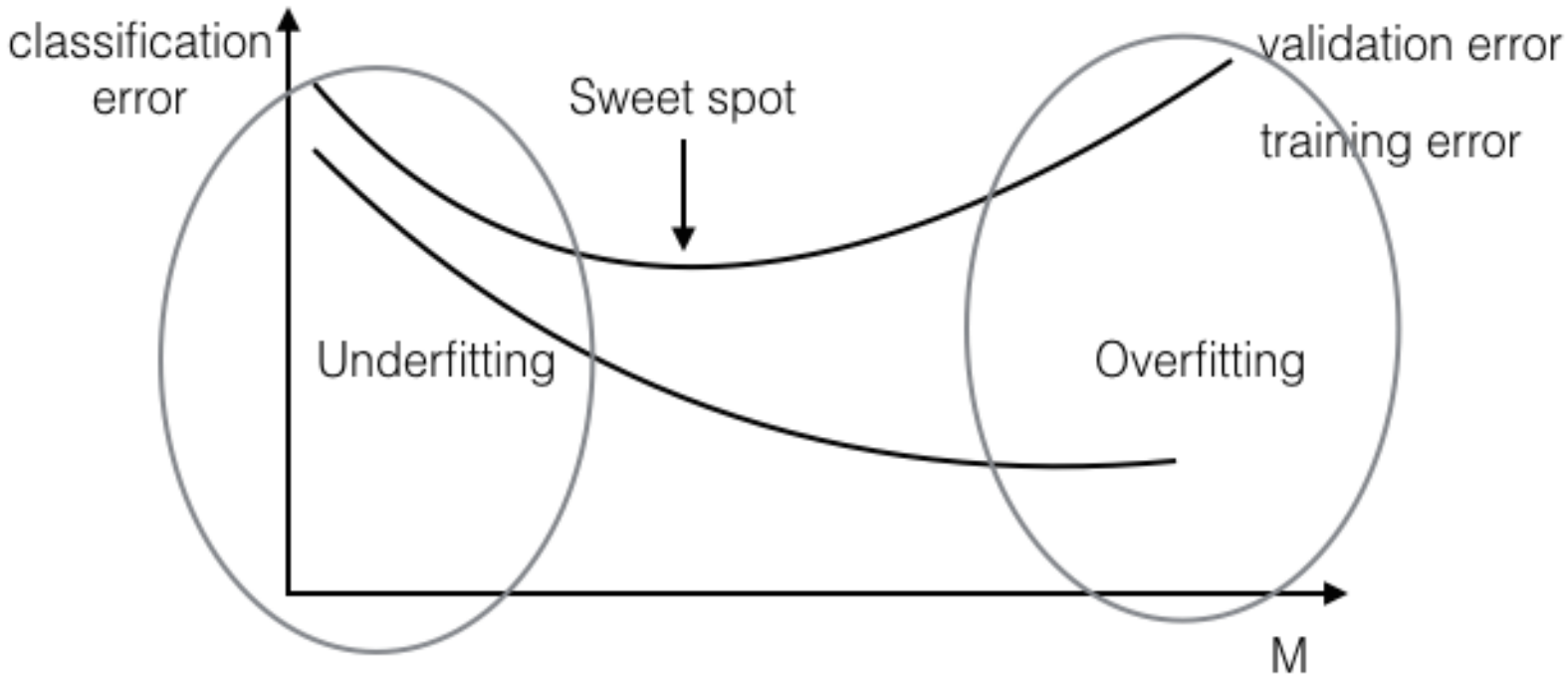
```
Model Details
=============
H2OGradientBoostingEstimator :  Gradient Boosting Machine
Model Key:  gbm_manual_cv


ModelMetricsRegression: gbm
** Reported on train data. **

MSE: 0.27438346229216
RMSE: 0.5238162485950202
MAE: 0.4075920913493524
RMSLE: 0.0774835431572533
Mean Residual Deviance: 0.27438346229216

ModelMetricsRegression: gbm
** Reported on cross-validation data. **

MSE: 0.45021820302163834
RMSE: 0.6709830124687497
MAE: 0.5185163944803867
RMSLE: 0.10007842575662584
Mean Residual Deviance: 0.45021820302163834
Cross-Validation Metrics Summary:
```

	mean	sd	cv_1_valid	cv_2_valid	cv_3_valid	cv_4_valid	cv_5_valid
mae	0.5183839	0.0057996	0.5173063	0.5328950	0.507699	0.5151827	0.5188366
mse	0.4501753	0.0090064	0.4428255	0.4701306	0.4537360	0.4317105	0.4524740
r2	0.4309830	0.0176490	0.4365418	0.3850305	0.4380041	0.4612002	0.4341383
residual_deviance	0.4501753	0.0090064	0.4428255	0.4701306	0.4537360	0.4317105	0.4524740

Improving Model Performance (Step-by-Step)

Model Settings	MSE (CV)	MSE (Test)
GBM with default settings	N/A	0.4551
GBM with manual settings	N/A	0.4433
Manual settings + cross-validation	0.4502	0.4433
Manual + CV + early stopping	0.4429	0.4287
CV + early stopping + full grid search	0.4378	0.4196
CV + early stopping + random grid search	0.4227	0.4047
Stacking best two from random grid search	N/A	0.3969

Early Stopping



Step 4 - GBM with Manual Settings, CV and Early Stopping

In [14]:
```
# Build a GBM with manual settings, CV and early stopping

# Set up GBM for regression
# Add a seed for reproducibility
gbm_manual_cv_es = H2OGradientBoostingEstimator(model_id = 'gbm_manual_cv_es',
                                                seed = 1234,
                                                ntrees = 10000,   # increase the number of trees
                                                sample_rate = 0.9,
                                                col_sample_rate = 0.9,
                                                nfolds = 5,
                                                stopping_metric = 'mse', # let early stopping feature dete
rmine
                                                stopping_rounds = 15,     # the optimal number of trees
                                                score_tree_interval = 1) # by looking at the MSE metric
# Use .train() to build the model
gbm_manual_cv_es.train(x = features,
                       y = 'quality',
                       training_frame = wine_train)
```

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

In [15]:
```
# Check the model summary
gbm_manual_cv_es.summary()
```

```
Model Summary:
```

	number_of_trees	number_of_internal_trees	model_size_in_bytes	min_depth	max_depth	mean_depth	min_leaves	ma
	155.0	155.0	49777.0	5.0	5.0	5.0	7.0	32.

Out[15]:

Improving Model Performance (Step-by-Step)

Model Settings	MSE (CV)	MSE (Test)
GBM with default settings	N/A	0.4551
GBM with manual settings	N/A	0.4433
Manual settings + cross-validation	0.4502	0.4433
Manual + CV + early stopping	0.4429	0.4287
CV + early stopping + full grid search	0.4378	0.4196
CV + early stopping + random grid search	0.4227	0.4047
Stacking best two from random grid search	N/A	0.3969

Grid Search

```
# define the range of hyper-parameters for grid search
hyper_params = {'sample_rate': [0.7, 0.8, 0.9],
                'col_sample_rate': [0.7, 0.8, 0.9]}
```

Combination	Parameter 1	Parameter 2
1	0.7	0.7
2	0.7	0.8
3	0.7	0.9
4	0.8	0.7
5	0.8	0.8
6	0.8	0.9
7	0.9	0.7
8	0.9	0.8
9	0.9	0.9

Step 5 - GBM with CV, Early Stopping and Full Grid Search

In [18]:
```
# import Grid Search
from h2o.grid.grid_search import H2OGridSearch
```

In [19]:
```
# define the criteria for full grid search
search_criteria = {'strategy': "Cartesian"}
```

In [20]:
```
# define the range of hyper-parameters for grid search
hyper_params = {'sample_rate': [0.7, 0.8, 0.9],
                'col_sample_rate': [0.7, 0.8, 0.9]}
```

In [21]:
```
# Set up GBM grid search
# Add a seed for reproducibility
gbm_full_grid = H2OGridSearch(
                    H2OGradientBoostingEstimator(
                        model_id = 'gbm_full_grid',
                        seed = 1234,
                        ntrees = 10000,
                        nfolds = 5,
                        stopping_metric = 'mse',
                        stopping_rounds = 15,
                        score_tree_interval = 1),
                    search_criteria = search_criteria, # full grid search
                    hyper_params = hyper_params)
```

In [22]:
```
# Use .train() to start the grid search
gbm_full_grid.train(x = features,
                    y = 'quality',
                    training_frame = wine_train)
```

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

In [23]:
```
# Sort and show the grid search results
gbm_full_grid_sorted = gbm_full_grid.get_grid(sort_by='mse', decreasing=False)
print(gbm_full_grid_sorted)
```

```
    col_sample_rate sample_rate  \
0               0.8         0.9
1               0.7         0.9
2               0.8         0.8
3               0.9         0.9
4               0.9         0.8
5               0.9         0.7
6               0.7         0.8
7               0.7         0.7
8               0.8         0.7

                                                 model_ids  \
0  Grid_GBM_py_4_sid_aee8_model_python_1491484901484_1_model_7
1  Grid_GBM_py_4_sid_aee8_model_python_1491484901484_1_model_6
2  Grid_GBM_py_4_sid_aee8_model_python_1491484901484_1_model_4
3  Grid_GBM_py_4_sid_aee8_model_python_1491484901484_1_model_8
4  Grid_GBM_py_4_sid_aee8_model_python_1491484901484_1_model_5
5  Grid_GBM_py_4_sid_aee8_model_python_1491484901484_1_model_2
6  Grid_GBM_py_4_sid_aee8_model_python_1491484901484_1_model_3
7  Grid_GBM_py_4_sid_aee8_model_python_1491484901484_1_model_0
8  Grid_GBM_py_4_sid_aee8_model_python_1491484901484_1_model_1

                   mse
0  0.43780785687779805
1   0.44060532786277523
2   0.44096100224896634
3   0.44288792056243054
4   0.44475412455519636
5    0.4457317997358452
6     0.448140619501795
7    0.4528872144586896
8    0.4529771807006373
```

In [24]:
```
# Extract the best model from full grid search
best_model_id = gbm_full_grid_sorted.model_ids[0]
best_gbm_from_full_grid = h2o.get_model(best_model_id)
best_gbm_from_full_grid.summary()
```

Model Summary:

	number_of_trees	number_of_internal_trees	model_size_in_bytes	min_depth	max_depth	mean_depth	min_leaves	ma
	187.0	187.0	57180.0	5.0	5.0	5.0	7.0	31.

Out[24]:

In [25]:
```
# Check the model performance on test dataset
best_gbm_from_full_grid.model_performance(wine_test)
```

```
ModelMetricsRegression: gbm
** Reported on test data. **

MSE: 0.4196124030489544
RMSE: 0.6477749632773363
MAE: 0.48965435078727043
RMSLE: 0.09630232810628427
Mean Residual Deviance: 0.4196124030489544
```

Out[25]:

Improving Model Performance (Step-by-Step)

Model Settings	MSE (CV)	MSE (Test)
GBM with default settings	N/A	0.4551
GBM with manual settings	N/A	0.4433
Manual settings + cross-validation	0.4502	0.4433
Manual + CV + early stopping	0.4429	0.4287
CV + early stopping + full grid search	0.4378	0.4196
CV + early stopping + random grid search	0.4227	0.4047
Stacking best two from random grid search	N/A	0.3969

GBM with CV, Early Stopping and Random Grid Search

In [26]:
```
# define the criteria for random grid search
search_criteria = {'strategy': "RandomDiscrete",
                   'max_models': 9,
                   'seed': 1234}
```

In [27]:
```
# define the range of hyper-parameters for grid search
# 27 combinations in total
hyper_params = {'sample_rate': [0.7, 0.8, 0.9],
                'col_sample_rate': [0.7, 0.8, 0.9],
                'max_depth': [3, 5, 7]}
```

In [28]:
```
# Set up GBM grid search
# Add a seed for reproducibility
gbm_rand_grid = H2OGridSearch(
                    H2OGradientBoostingEstimator(
                        model_id = 'gbm_rand_grid',
                        seed = 1234,
                        ntrees = 10000,
                        nfolds = 5,
                        stopping_metric = 'mse',
                        stopping_rounds = 15,
                        score_tree_interval = 1),
                    search_criteria = search_criteria, # full grid search
                    hyper_params = hyper_params)
```

In [29]:
```
# Use .train() to start the grid search
gbm_rand_grid.train(x = features,
                    y = 'quality',
                    training_frame = wine_train)
```

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

In [30]:
```
# Sort and show the grid search results
gbm_rand_grid_sorted = gbm_rand_grid.get_grid(sort_by='mse', decreasing=False)
print(gbm_rand_grid_sorted)
```

```
    col_sample_rate max_depth sample_rate  \
0               0.9         7         0.9
1               0.7         7         0.7
2               0.9         7         0.7
3               0.8         7         0.7
4               0.7         5         0.8
5               0.8         3         0.9
6               0.9         3         0.9
7               0.8         3         0.8
8               0.7         3         0.7

                                                model_ids  \
0  Grid_GBM_py_4_sid_aee8_model_python_1491484901484_2_model_5
1  Grid_GBM_py_4_sid_aee8_model_python_1491484901484_2_model_1
2  Grid_GBM_py_4_sid_aee8_model_python_1491484901484_2_model_6
3  Grid_GBM_py_4_sid_aee8_model_python_1491484901484_2_model_4
4  Grid_GBM_py_4_sid_aee8_model_python_1491484901484_2_model_0
5  Grid_GBM_py_4_sid_aee8_model_python_1491484901484_2_model_7
6  Grid_GBM_py_4_sid_aee8_model_python_1491484901484_2_model_2
7  Grid_GBM_py_4_sid_aee8_model_python_1491484901484_2_model_3
8  Grid_GBM_py_4_sid_aee8_model_python_1491484901484_2_model_8

                   mse
0   0.4227388012308513
1   0.4327748309201154
2   0.4369533108701783
3   0.4397321318633594
4    0.448140619501795
5   0.4647039373596571
6   0.4690321721360509
7  0.47384072192391513
8  0.47745552186979223
```

Improving Model Performance (Step-by-Step)

Model Settings	MSE (CV)	MSE (Test)
GBM with default settings	N/A	0.4551
GBM with manual settings	N/A	0.4433
Manual settings + cross-validation	0.4502	0.4433
Manual + CV + early stopping	0.4429	0.4287
CV + early stopping + full grid search	0.4378	0.4196
CV + early stopping + random grid search	0.4227	0.4047
Stacking best two from random grid search	N/A	0.3969

Regression Models (Ensembles)

py_03c_regression_ensembles.ipynb

Stacked Ensembles in H2O

https://github.com/h2oai/h2o-meetups/blob/master/2017_02_23_Metis_SF_Sacked_Ensembles_Deep_Water/stacked_ensembles_in_h2o_feb2017.pdf

February 2017

Erin LeDell Ph.D.
Machine Learning Scientist

H2O.ai

Step 1: Build GBM Models using Random Grid Search and Extract the Best Model

In [8]:
```
# define the range of hyper-parameters for GBM grid search
# 27 combinations in total
hyper_params = {'sample_rate': [0.7, 0.8, 0.9],
                'col_sample_rate': [0.7, 0.8, 0.9],
                'max_depth': [3, 5, 7]}
```

In [9]:
```
# Set up GBM grid search
# Add a seed for reproducibility
gbm_rand_grid = H2OGridSearch(
                    H2OGradientBoostingEstimator(
                        model_id = 'gbm_rand_grid',
                        seed = 1234,
                        ntrees = 10000,
                        nfolds = 5,
                        fold_assignment = "Modulo",               # needed for stacked ensembles
                        keep_cross_validation_predictions = True, # needed for stacked ensembles
                        stopping_metric = 'mse',
                        stopping_rounds = 15,
                        score_tree_interval = 1),
                    search_criteria = search_criteria,
                    hyper_params = hyper_params)
```

In [10]:
```
# Use .train() to start the grid search
gbm_rand_grid.train(x = features,
                    y = 'quality',
                    training_frame = wine_train)
```

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

Step 2: Build DRF Models using Random Grid Search and Extract the Best Model

In [13]:
```
# define the range of hyper-parameters for DRF grid search
# 27 combinations in total
hyper_params = {'sample_rate': [0.5, 0.6, 0.7],
                'col_sample_rate_per_tree': [0.7, 0.8, 0.9],
                'max_depth': [3, 5, 7]}
```

In [14]:
```
# Set up DRF grid search
# Add a seed for reproducibility
drf_rand_grid = H2OGridSearch(
                    H2ORandomForestEstimator(
                        model_id = 'drf_rand_grid',
                        seed = 1234,
                        ntrees = 200,
                        nfolds = 5,
                        fold_assignment = "Modulo",                 # needed for stacked ensembles
                        keep_cross_validation_predictions = True),  # needed for stacked ensembles
                    search_criteria = search_criteria,
                    hyper_params = hyper_params)
```

In [15]:
```
# Use .train() to start the grid search
drf_rand_grid.train(x = features,
                    y = 'quality',
                    training_frame = wine_train)
```

```
drf Grid Build progress: |████████████████████████████████████████████████| 100%
```

In [16]:
```
# Sort and show the grid search results
drf_rand_grid_sorted = drf_rand_grid.get_grid(sort_by='mse', decreasing=False)
print(drf_rand_grid_sorted)
```

Step 3: Build DNN Models using Random Grid Search and Extract the Best Model

In [18]:
```
# define the range of hyper-parameters for DNN grid search
# 81 combinations in total
hyper_params = {'activation': ['tanh', 'rectifier', 'maxout'],
                'hidden': [[50], [50,50], [50,50,50]],
                'l1': [0, 1e-3, 1e-5],
                'l2': [0, 1e-3, 1e-5]}
```

In [19]:
```
# Set up DNN grid search
# Add a seed for reproducibility
dnn_rand_grid = H2OGridSearch(
                    H2ODeepLearningEstimator(
                        model_id = 'dnn_rand_grid',
                        seed = 1234,
                        epochs = 20,
                        nfolds = 5,
                        fold_assignment = "Modulo",                # needed for stacked ensembles
                        keep_cross_validation_predictions = True), # needed for stacked ensembles
                    search_criteria = search_criteria,
                    hyper_params = hyper_params)
```

In [20]:
```
# Use .train() to start the grid search
dnn_rand_grid.train(x = features,
                    y = 'quality',
                    training_frame = wine_train)
```

```
deeplearning Grid Build progress: |████████████████████████████████████████| 100%
```

In [21]:
```
# Sort and show the grid search results
dnn_rand_grid_sorted = dnn_rand_grid.get_grid(sort_by='mse', decreasing=False)
print(dnn_rand_grid_sorted)
```

Model Stacking

In [23]:
```
# Define a list of models to be stacked
# i.e. best model from each grid
all_ids = [best_gbm_model_id, best_drf_model_id, best_dnn_model_id]
```

In [24]:
```
# Set up Stacked Ensemble
ensemble = H2OStackedEnsembleEstimator(model_id = "my_ensemble",
                                       base_models = all_ids)
```

In [25]:
```
# use .train to start model stacking
# GLM as the default metalearner
ensemble.train(x = features,
               y = 'quality',
               training_frame = wine_train)
```

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

Comparison of Model Performance on Test Data

In [26]:
```
print('Best GBM model from Grid (MSE) : ', best_gbm_from_rand_grid.model_performance(wine_test).mse())
print('Best DRF model from Grid (MSE) : ', best_drf_from_rand_grid.model_performance(wine_test).mse())
print('Best DNN model from Grid (MSE) : ', best_dnn_from_rand_grid.model_performance(wine_test).mse())
print('Stacked Ensembles        (MSE) : ', ensemble.model_performance(wine_test).mse())
```

```
Best GBM model from Grid (MSE) :  0.4013942890547201
Best DRF model from Grid (MSE) :  0.4781568285687009
Best DNN model from Grid (MSE) :  0.4995303971383784
Stacked Ensembles        (MSE) :  0.39692749423528306
```

Lowest MSE =
Best Performance

Improving Model Performance (Step-by-Step)

Model Settings	MSE (CV)	MSE (Test)
GBM with default settings	N/A	0.4551
GBM with manual settings	N/A	0.4433
Manual settings + cross-validation	0.4502	0.4433
Manual + CV + early stopping	0.4429	0.4287
CV + early stopping + full grid search	0.4378	0.4196
CV + early stopping + random grid search	0.4227	0.4047
Stacking best two from random grid search	N/A	0.3969

Classification Models (Ensembles)

py_04_classification_ensembles.ipynb

Model Stacking

In [19]:
```
# Define a list of models to be stacked
# i.e. best model from each grid
all_ids = [best_gbm_model_id, best_drf_model_id]
```

In [20]:
```
# Set up Stacked Ensemble
ensemble = H2OStackedEnsembleEstimator(model_id = "my_ensemble",
                                       base_models = all_ids)
```

In [21]:
```
# use .train to start model stacking
# GLM as the default metalearner
ensemble.train(x = features,
               y = 'Survived',
               training_frame = titanic_train)
```

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

Comparison of Model Performance on Test Data

In [22]:
```
print('Best GBM model from Grid (AUC) : ', best_gbm_from_rand_grid.model_performance(titanic_test).auc())
print('Best DRF model from Grid (AUC) : ', best_drf_from_rand_grid.model_performance(titanic_test).auc())
print('Stacked Ensembles        (AUC) : ', ensemble.model_performance(titanic_test).auc())
```

```
Best GBM model from Grid (AUC) :  0.8892284186401833
Best DRF model from Grid (AUC) :  0.8903106697224344
Stacked Ensembles        (AUC) :  0.8918385536032595
```

Highest AUC =
Best Performance

H_2O in the Cloud

py_05_h2o_in_the_cloud.ipynb

Step 1: Create a H2O cluster in the Cloud

Follow the instructions from http://h2o-release.s3.amazonaws.com/h2o/latest_stable.html

Step 2: Import H2O module

```
# Import module
import h2o
```

Step 3: Connect to H2O cluster with IP address

```
# In order to connect to a H2O cluster in the cloud, you need to specify the IP address
h2o.connect(ip = "xxx.xxx.xxx.xxx") # fill in the real IP
```

That is all you need to do

All other tasks (importing data, training models, making predictions etc) are exactly the same as shown in previous code examples.

DOWNLOAD AND RUN | INSTALL IN R | INSTALL IN PYTHON | INSTALL ON HADOOP | USE FROM MAVEN

DOWNLOAD H_2O

Get started with H_2O in 3 easy steps

1. Download H_2O. This is a zip file that contains everything you need to get started.

2. From your terminal, run:

```
cd ~/Downloads
unzip h2o-3.10.4.3.zip
cd h2o-3.10.4.3
java -jar h2o.jar
```

3. Point your browser to http://localhost:54321

User Documentation

Recap

Learning Objectives

- Start and connect to a local H_2O cluster from Python.
- Import data from Python data frames, local files or web.
- Perform basic data transformation and exploration.
- Train regression and classification models using various H_2O machine learning algorithms.
- Evaluate models and make predictions.
- Improve performance by tuning and stacking.

Thanks!

- Organizers & Sponsors

GoDataDriven
proudly part of Xebia Group

Booking.com

- Find us at PyData Conference
  - Live Demos

H2O.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)