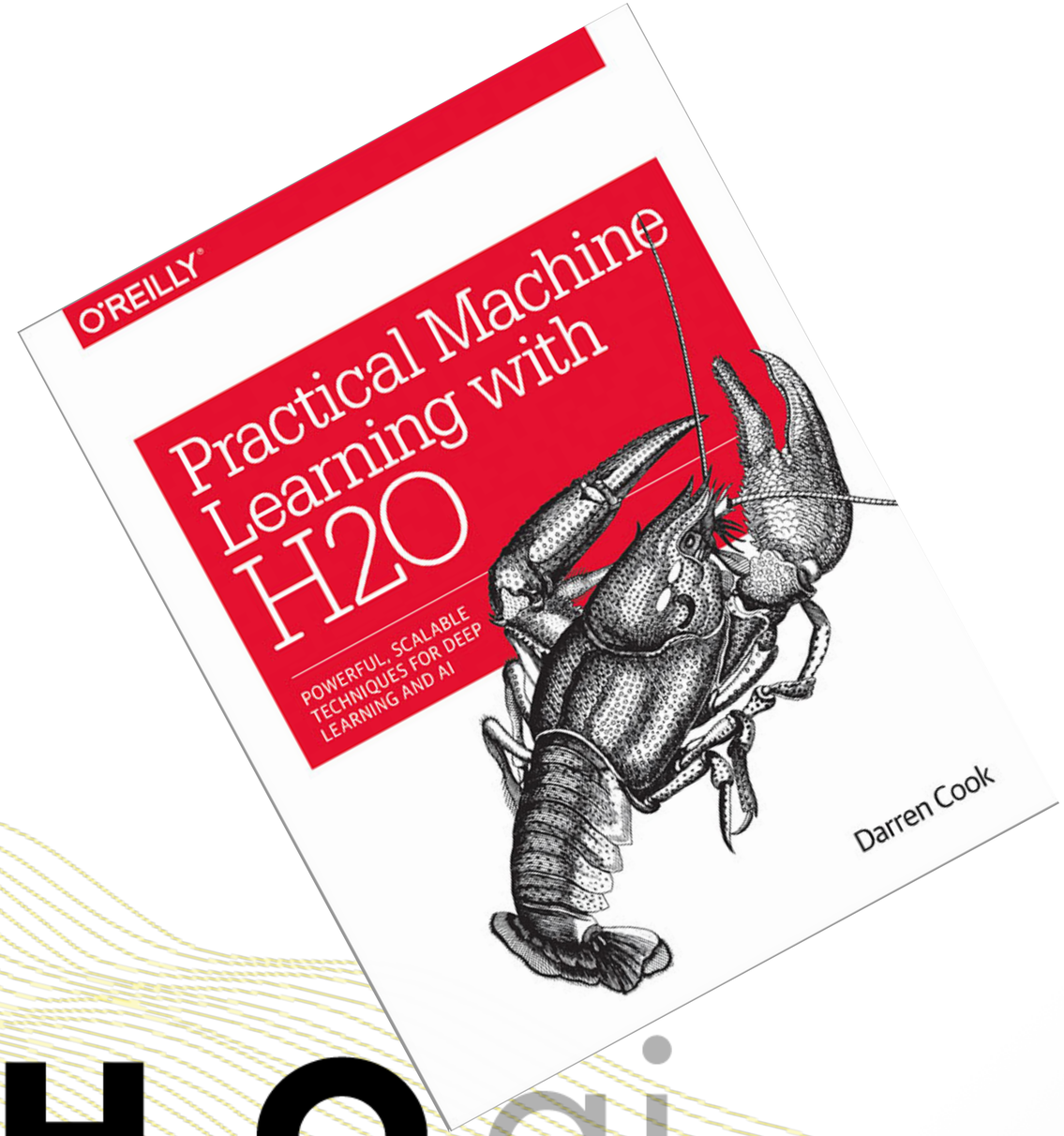


H2O.ai

Avkash Chauhan (avkash@h2o.ai)
VP, Enterprise Customers

Agenda

- H2O Intro
- Installation
- Using H2O from FLOW, R & Python
- Data munging in H2O with Python
- 2 examples of machine learning problems
  - GBM, GLM, DRF
  - Understanding Models, improvements,
- Machine learning production pipeline

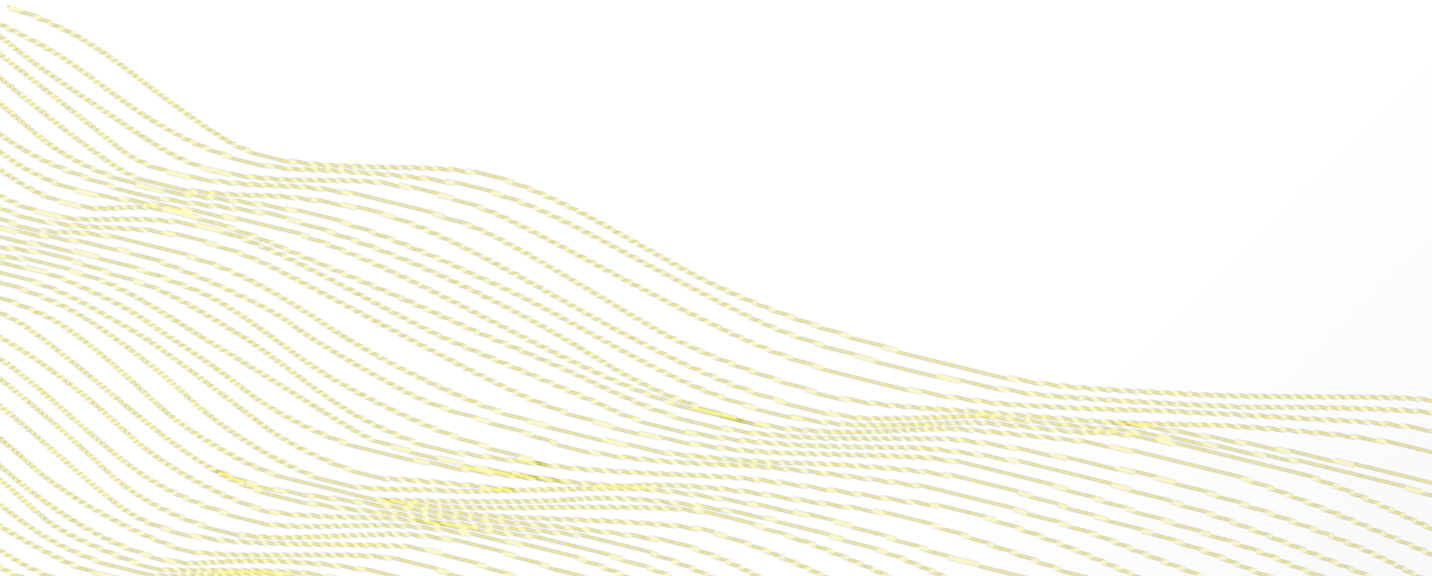
Figure 1. Magic Quadrant for Data Science Platforms

Source: Gartner (February 2017)

H2O.ai is a Visionary in the Gartner Magic Quadrant for Data Science Platforms

READ THE REPORT

Introduction & Overview



H2O.ai Company Overview

Founded	2011 Venture-backed, debuted in 2012
Products	• H2O Open Source In-Memory AI Prediction Engine • Sparkling Water • STEAM • DEEP WATER
Mission	Operationalize Data Science, and provide a platform for users to build beautiful data products
Team	60+ employees worldwide • CA, NY, UT, Japan, UK • Distributed Systems Engineers doing Machine Learning • World-class visualization designers
Headquarters	Mountain View, CA

Please visit: http://www.h2o.ai/customers/#

H2O Users List: http://www.h2o.ai/user-list/

Customers and Use Cases

Financial

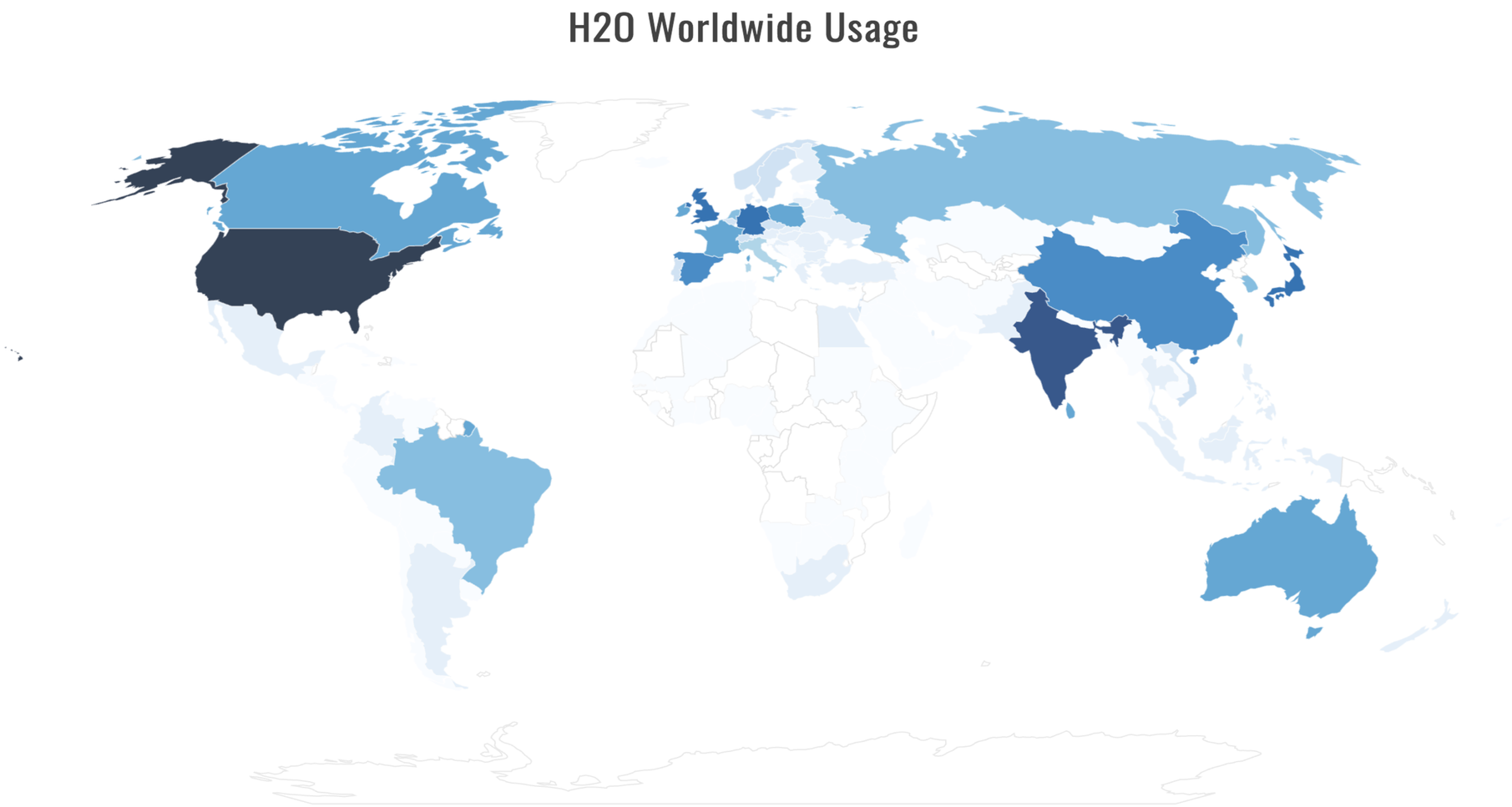
Telecom

Insurance

Healthcare

Marketing

Open Source Users & Community



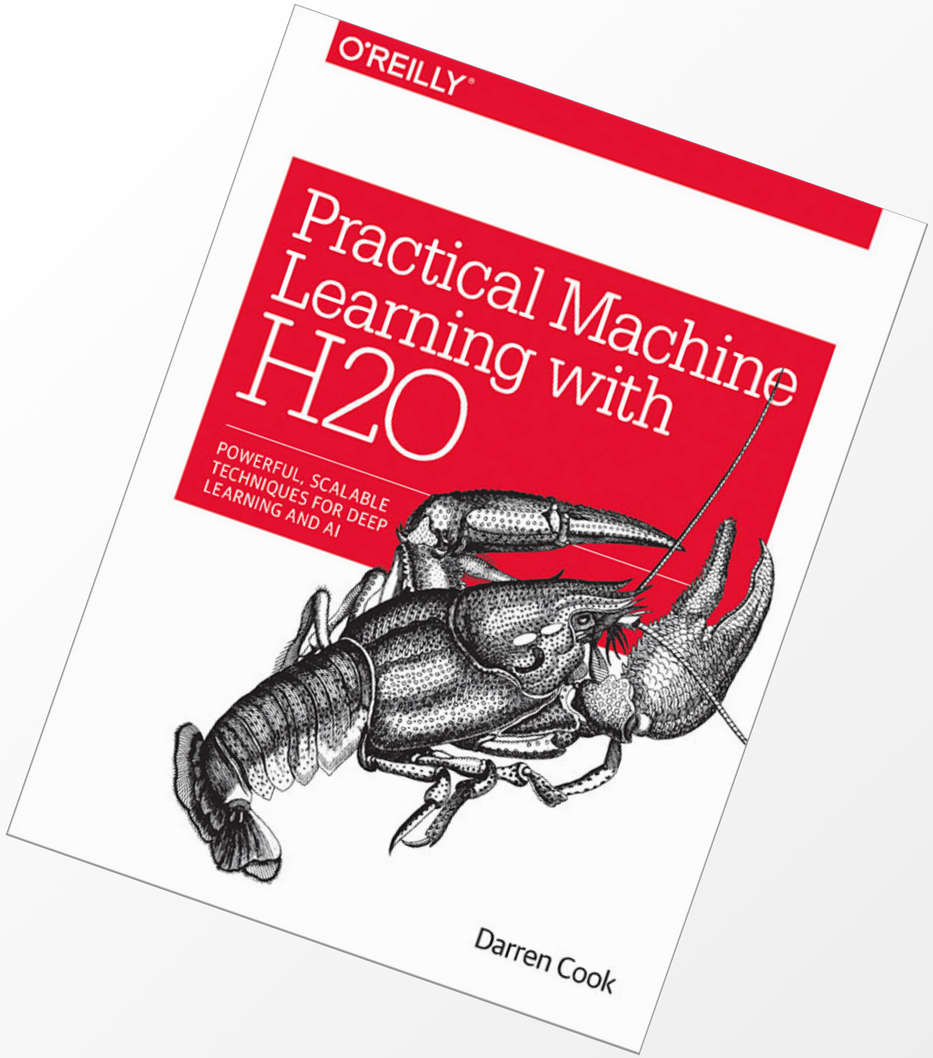
70,000 users, 8,000 organizations

H2O Users List: http://www.h2o.ai/user-list/

H2O.ai

About Myself

- VP – Enterprise products & customers
  - Handling paid enterprise customer's requirements
  - building product(s)
  - Helping community
- LinkedIn: https://www.linkedin.com/in/avkashchauhan/
- Blog: https://aichamp.wordpress.com/
- Community: https://community.h2o.ai/index.html
- Twitter: @avkashchauhan



Products and Features

H2O Platform(s)

H2O.ai

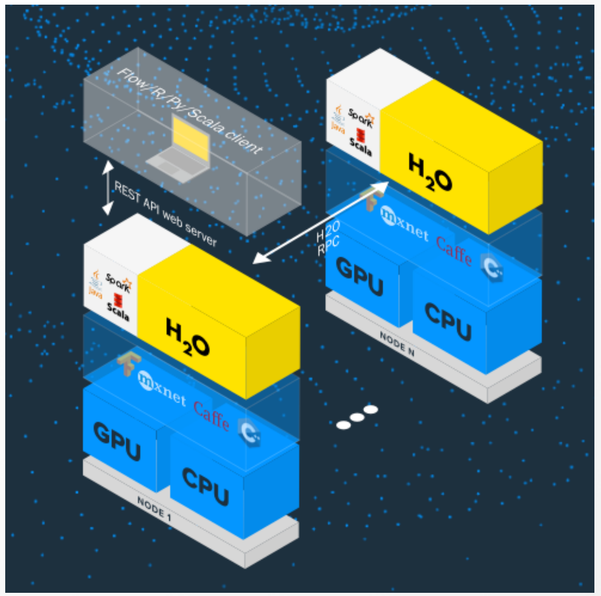
In-Memory, Distributed Machine Learning Algorithms with H_2O Flow GUI

Spark + H2O

SPARKLING WATER

H2O AI Open Source Engine Integration with Spark

DEEP WATER



Key features

- Open Source (Apache 2.0)
- All supported ML algorithms are coded by our engineers
- Designed for speed, scalability and for super large data-sets
- Same distribution for open source community & enterprise
- Very active production, every other week release
- Vibrant open source community
  - https://community.h2o.ai
- Enterprise Support portal
  - https://support.h2o.ai
- We have 70,000 users, 8,000 organizations and growing daily

Usage: Simple Solution

- Single Deployable compiled Java code (jar)
- Ready to use point and click FLOW Interface
- Connection from R and Python after specific packages are installed
- Use Java, Scala natively and any other language through RESTful API
- Deployable models - Binary & Java (POJO & MOJO)
- One click prediction/scoring engine

Usage: Complex Solution

- Multi-node Deployment
- Spark and Hadoop distributed environment
  - Sparkling Water (Spark + H2O)
- Data ingested from various inputs
  - S3, HDFS, NFS, JDBC, Object store etc.
  - Streaming support in Spark (through Sparking Water)
- Distributed machine learning for every algorithm in platform
- Prediction service deployment on several machines

Current Algorithm Overview

Statistical Analysis
- Linear Models (GLM)
- Naïve Bayes

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

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

Clustering
- K-Means

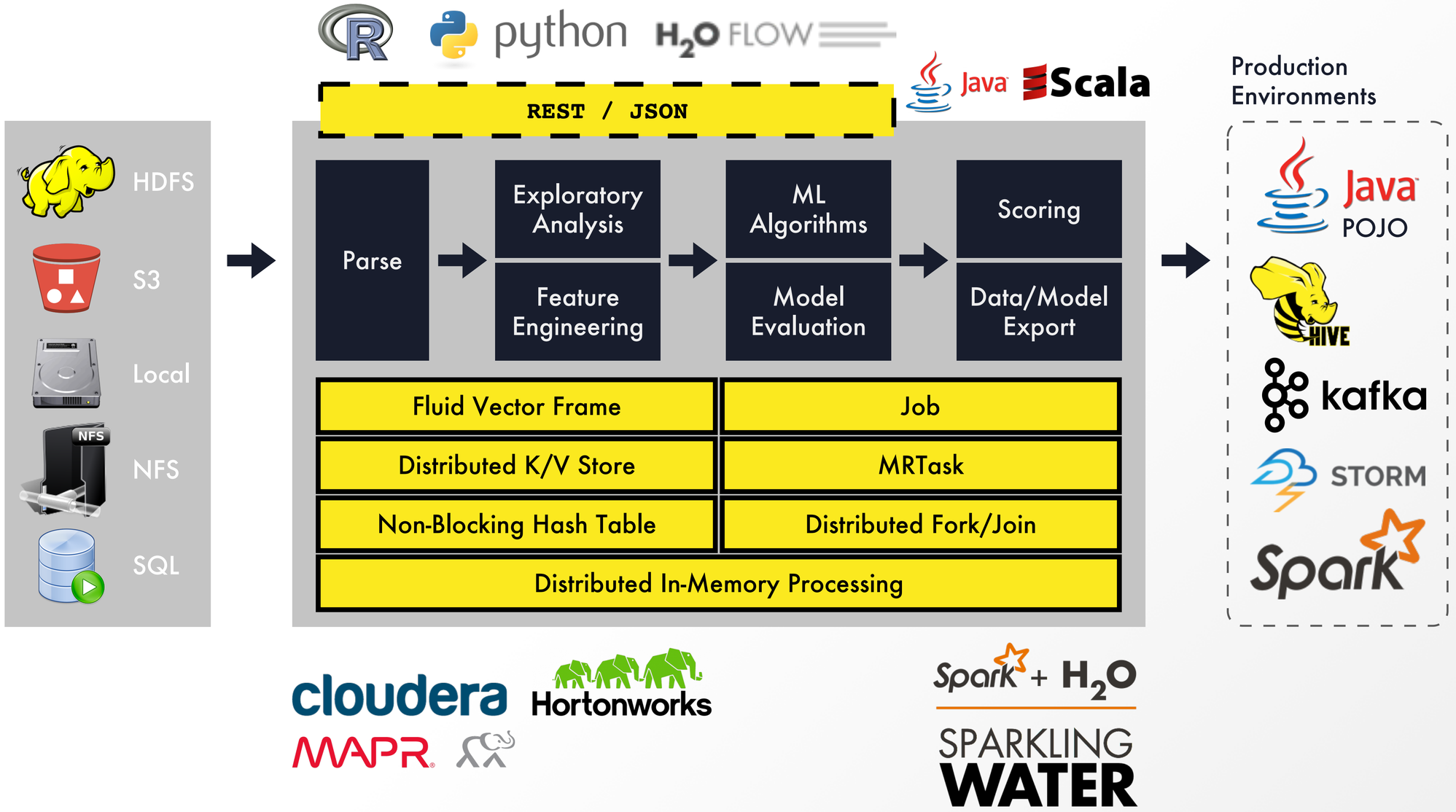
Dimension Reduction
- Principal Component Analysis
- Generalized Low Rank Models

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

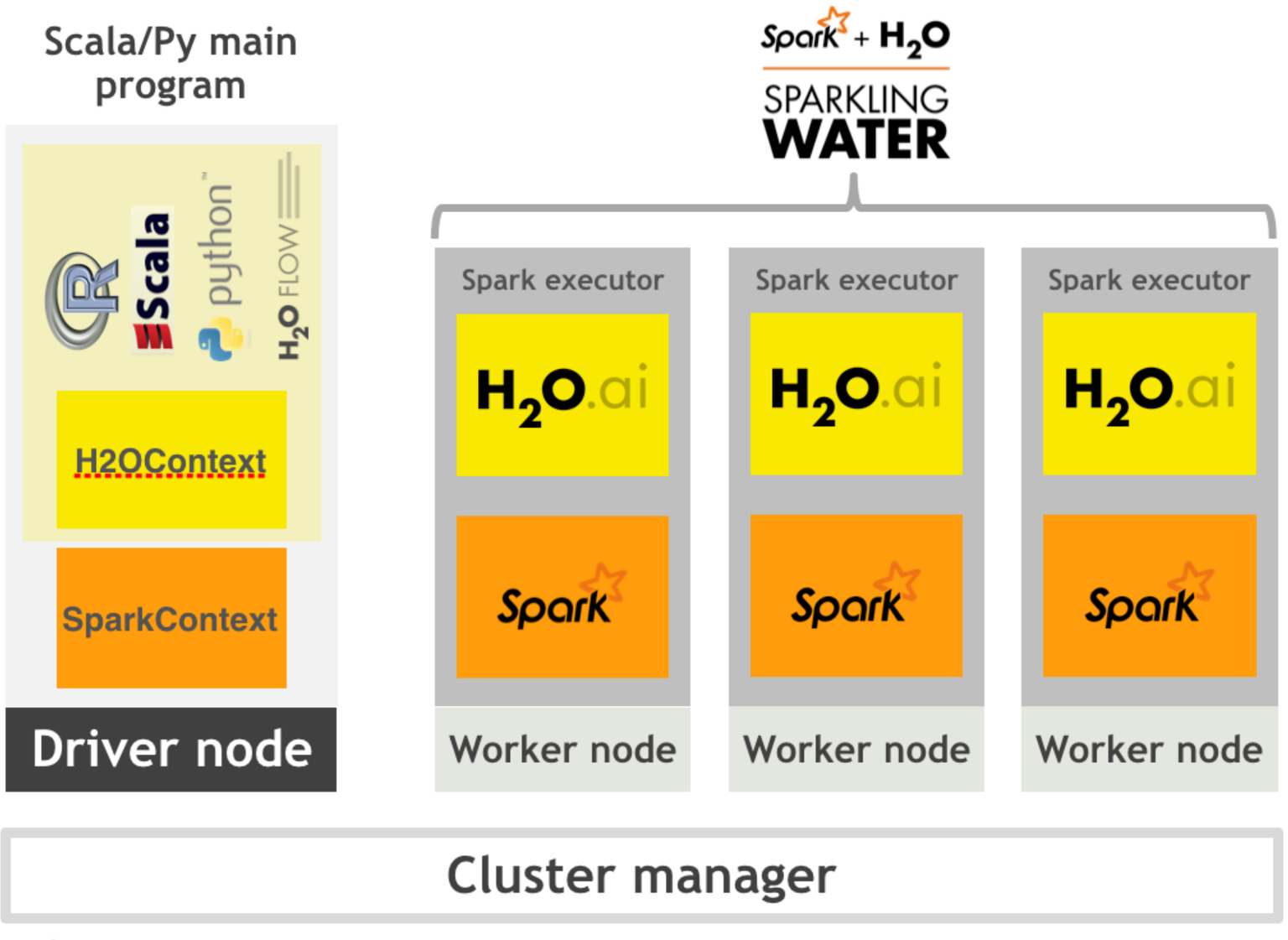
Data Munging
- Scalable Data Frames
- Sort, Slice, Log Transform
- Data.table (1B rows groupBy record)

Technical Architecture

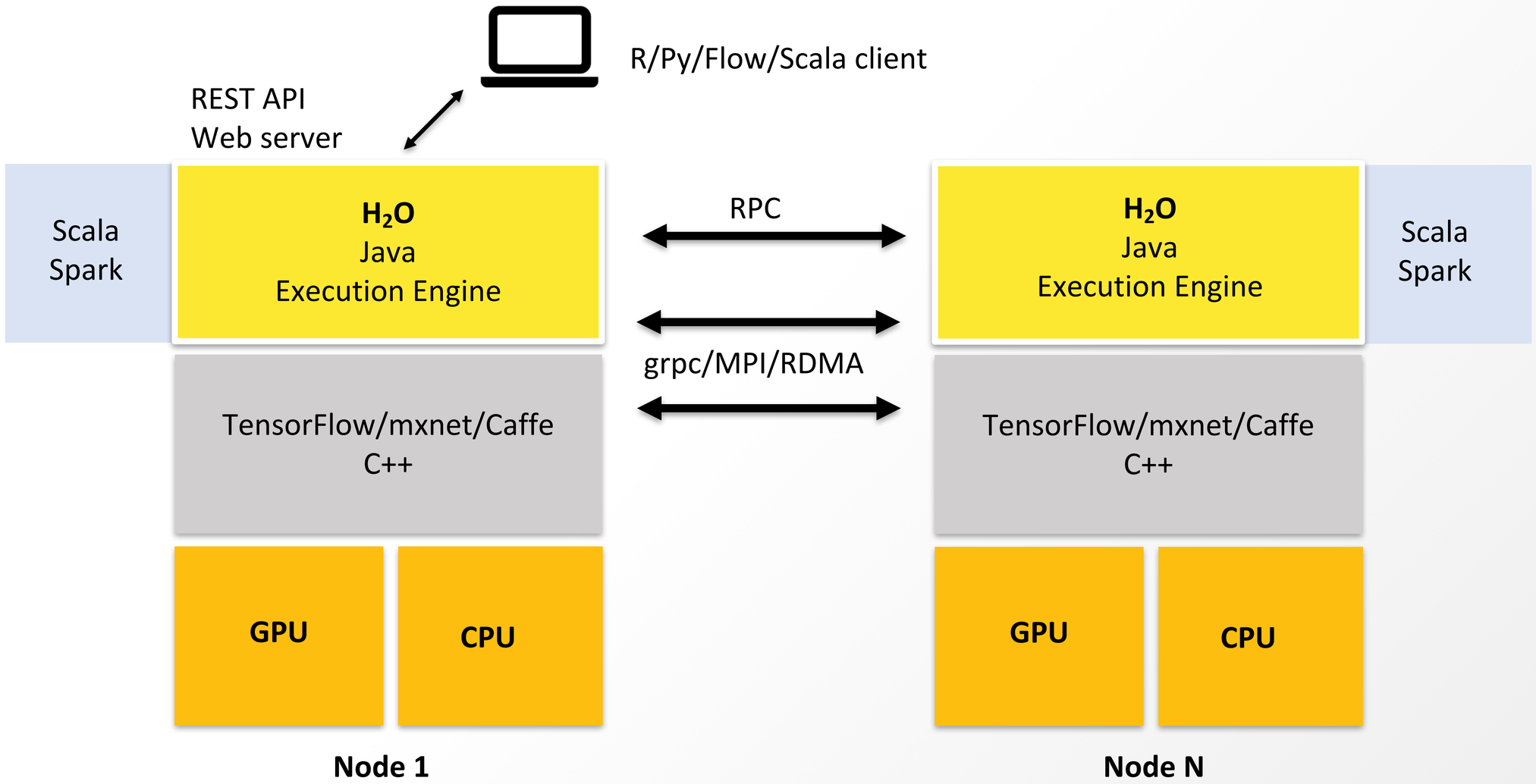
Core H2O: Architecture



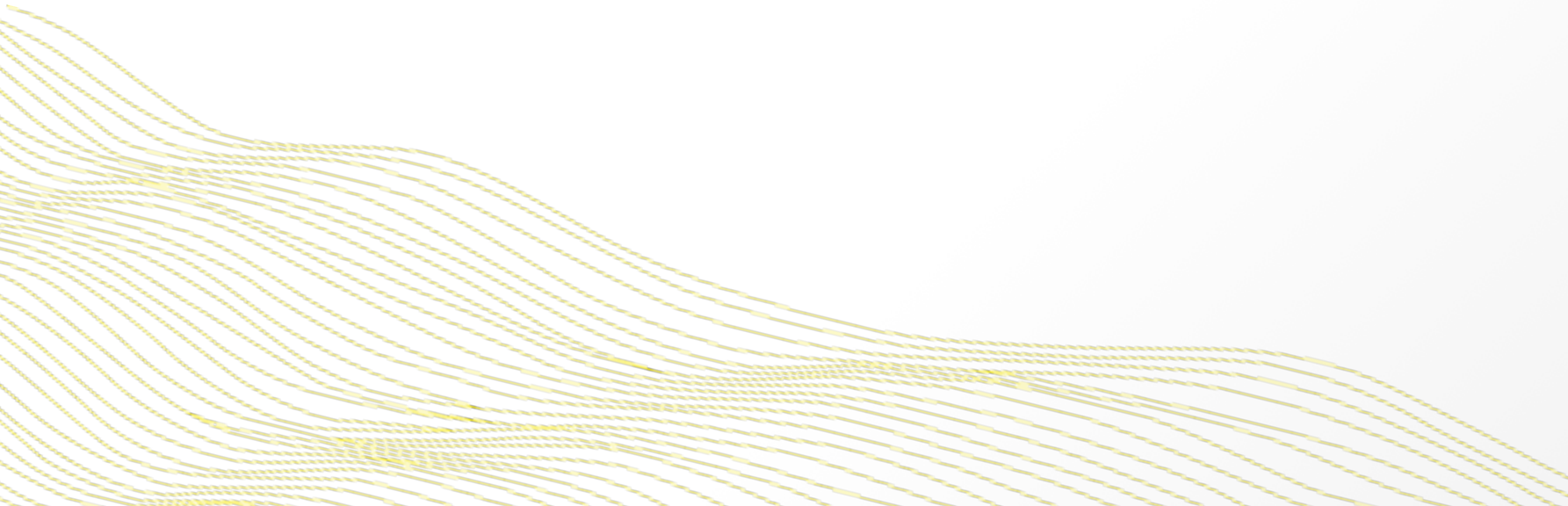
Sparkling Water - High Level Architecture



Deep Water : Architecture



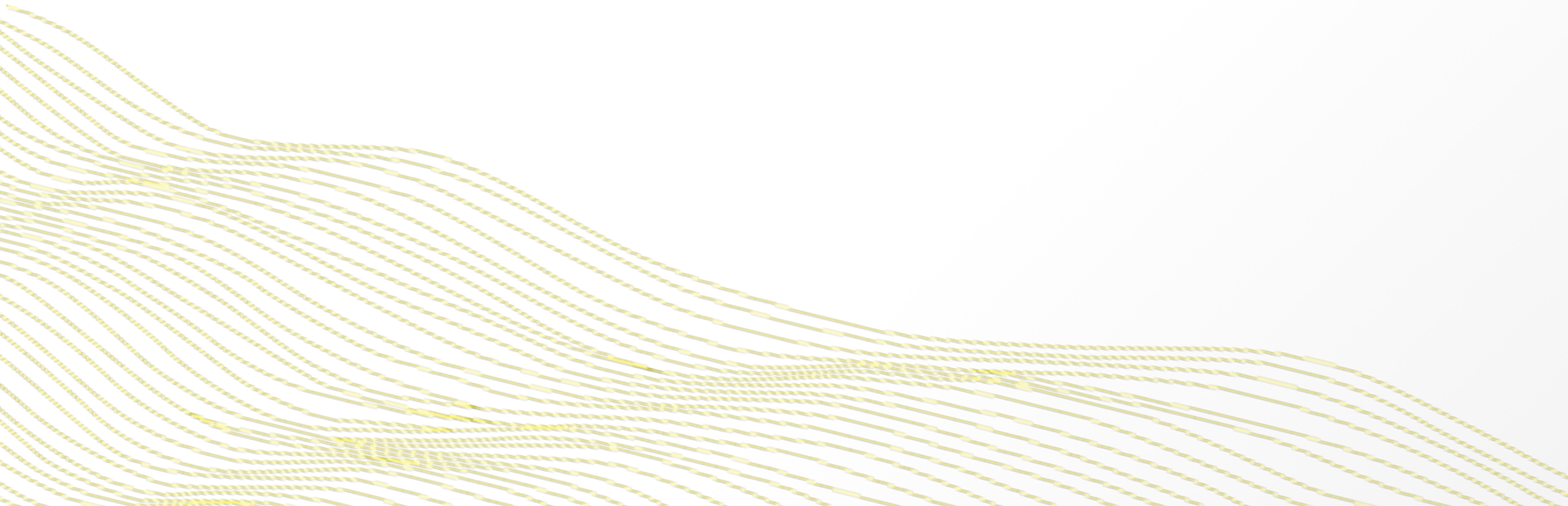
H2O Installation



What we covered with H2O Installation

- Installation, using H2O with R & Python
  - Installation H2O
    - Help – help(h2o.init), h2o.cluster_status()
  - H2O Github repo
    - Source code - glance
  - R package installation
  - Python Package Installation
  - Connecting H2O from R
  - Connecting H2O from Python

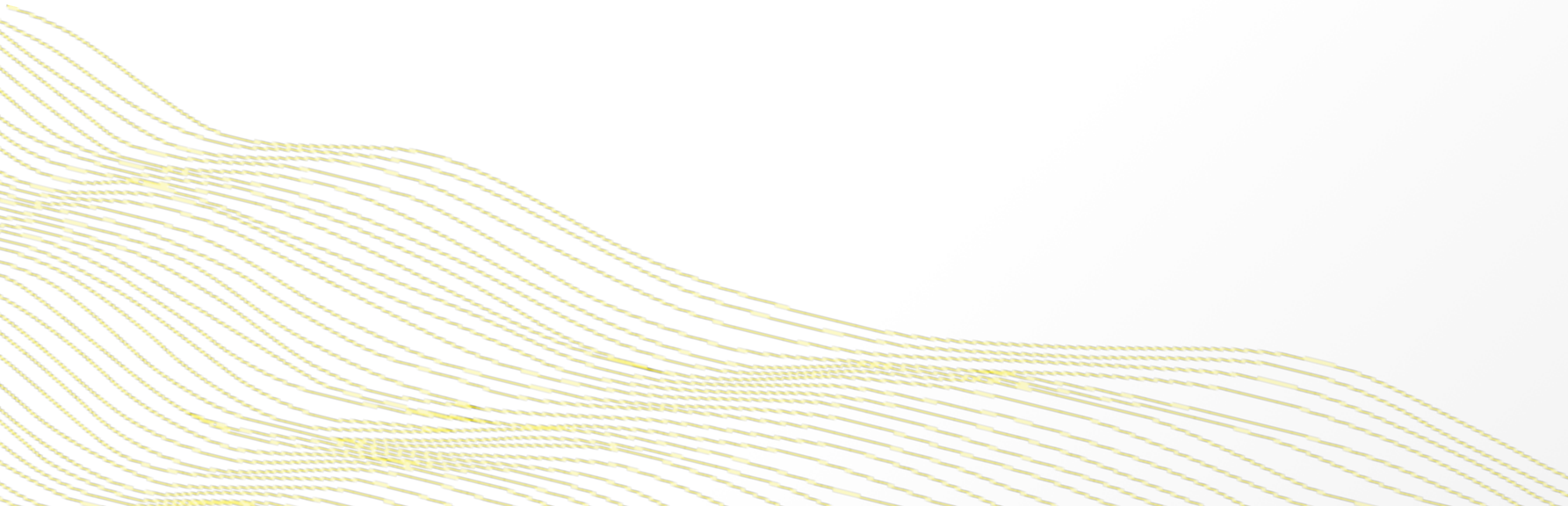
H2O FLOW DEMO



What we covered in FLOW DEMO

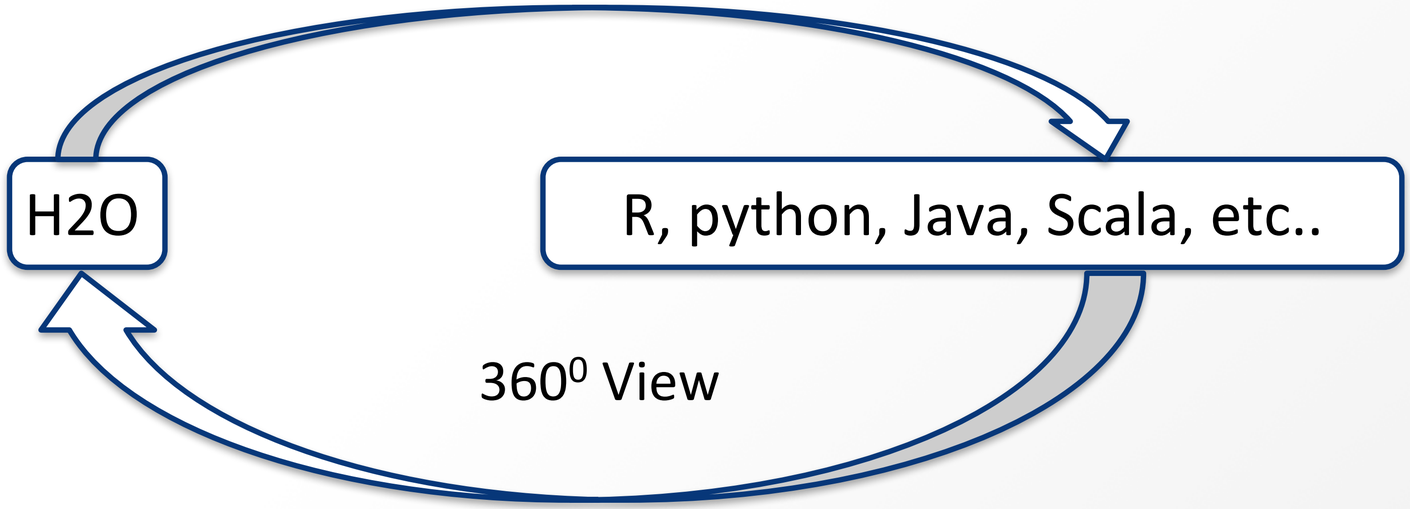
- FLOW Intro
- Running Examples
- Generating Data
- Working with UI, Cell, Running FLOW Script
- Importing Data
  - Chunk Distribution
  - Feature analysis
- Building models from imported data
- Understanding models
  - Binary Model, POJO, MOJO
- Listing all Jobs
- Using HELP
- Understanding RESTful Interface
- Reading Logs, Water Meter (CPU analysis), Stack Trace etc.

Data manipulation between H2O, R & python



Data manipulation between H2O, R, python

- Import data in between H2O, python, R and others..



Data manipulation between H2O, R, python

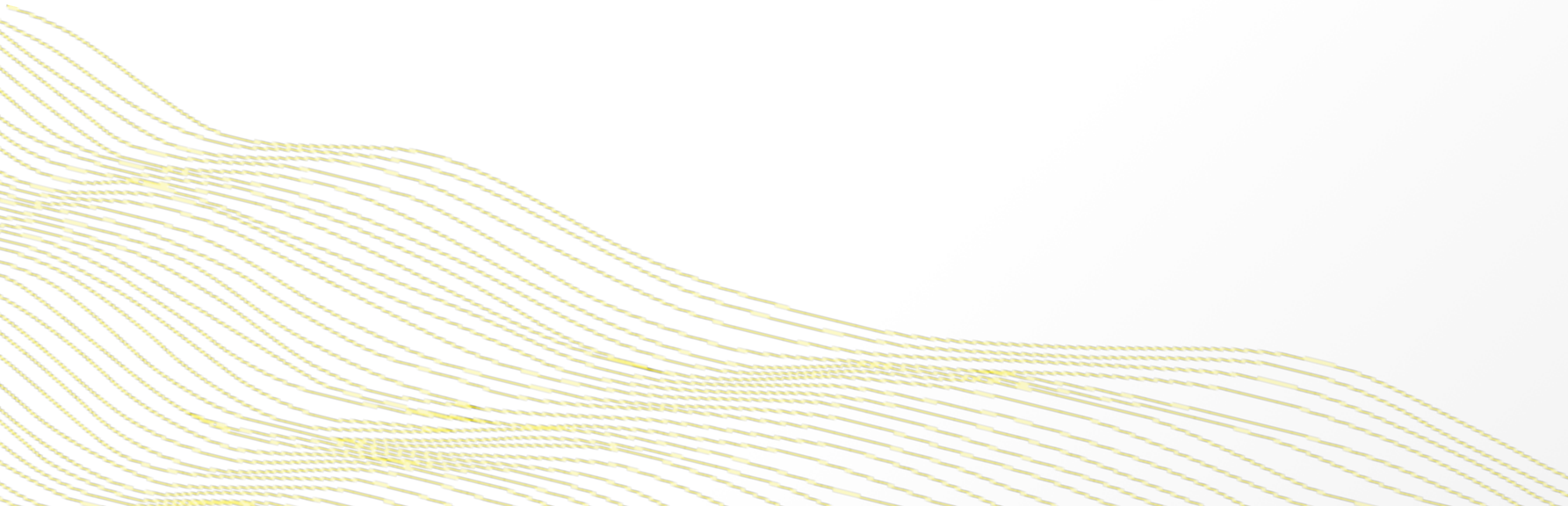
- Import data in python
  - import pandas as pd
    - datasetpath = "/Users/avkashchauhan/tools/datasets/kaggle-imdb-word2vec"
    - train_data = pd.read_csv(datasetpath + "/labeledTrainData.tsv", header=0, delimiter="\t", quoting=3)
  - h2o.ls()
  - train_data.describe()
  - train_data_h2o = h2o.H2OFrame(train_data, destination_frame = "train_data_h2o")
  - h2o.ls()
  - train_data_h2o.describe()
  - Now Look the same frame in the FLOW - **getFrames**
- In R
- > h2o.ls()
- > rdf = h2o.getFrame("train_data_h2o")
- > rdf
- > summary(rdf)

Reference: http://localhost:8888/notebooks/H2O-start-test-and-data-switch-python-r.ipynb

Data manipulation between H2O, R, python

- Import data in R
  - > iris
  - > mydf = as.h2o(iris)
    - This frame will be imported as iris original frame name
  - > summary(mydf)
  - > summary(iris)
  - > h2o.ls()
    - You will see the **iris** entry as h2o frames list
    - Check FLOW as well and you will see iris there too
  - > mydf = as.h2o(iris, destination_frame = “mydf”)
  - > h2o.ls()
  - In Python
    - h2o.ls()
    - my_python_df= h2o.get_frame("mydf")
    - my_python_df
    - h2o.ls()

Data munging in H2O with Python

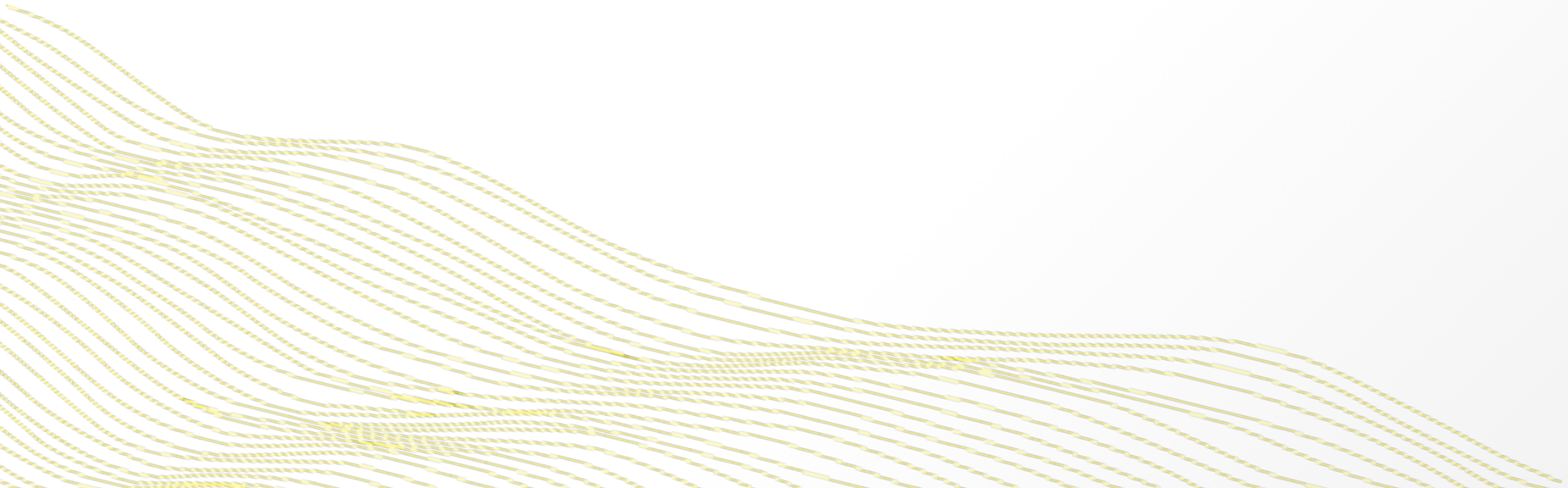


What we covered Data munging in H2O with python

- H2O and Python
- Jupyter notebook Demo
- Data import
- Row, column, data frames, slicing, binding, exporting, factoring
- Using functions

Reference:
[1] http://localhost:8888/notebooks/H2O%20Data%20Ingest%20Demo.ipynb
[2] http://localhost:8888/notebooks/H2O%20Frame%20manipulation.ipynb

Price Prediction using GLM, GBM, DRF



Problem Description

- Kaggle
  - https://www.kaggle.com/harlfoxem/housesalesprediction
- Local Datasets
  - /Users/avkashchauhan/learn/seattle-workshop/kc_house_data.csv
  - /Users/avkashchauhan/learn/seattle-workshop/kc_house_orig.csv
- Documentation
  - http://docs.h2o.ai/h2o/latest-stable/h2o-docs/index.html

What we covered

- Data Import
- Understood data, frame distribution, chunk, compression etc.
- Understood all features, through histograms, enums, etc.
- Split data frames for train and test
- Converting features to Factors/Enum
- Imputation of values
- Training
  - with Training frame only
  - With cross-validation
  - with validation frame
- Understanding Model details
- Documentation
  - http://docs.h2o.ai/h2o/latest-stable/h2o-docs/index.html

What we covered

- Response Column - Price
- buildModel 'glm', {"model_id":"glm-2fcdf58a-da59-4b3f-8a68-1177bce9531c","training_frame":"kc90_house_data.hex","nfolds":"10","seed":-1,"response_column":"price","ignored_columns":["id"],"ignore_const_cols":true,"family":"gaussian","solver":"AUTO","alpha":[],"lambda":[],"lambda_search":false,"standardize":true,"non_negative":false,"fold_assignment":"AUTO","score_each_iteration":false,"compute_p_values":false,"remove_collinear_columns":false,"max_iterations":-1,"link":"family_default","max_runtime_secs":0,"keep_cross_validation_predictions":false,"keep_cross_validation_fold_assignment":false,"missing_values_handling":"MeanImputation","intercept":true,"objective_epsilon":-1,"beta_epsilon":0.0001,"gradient_epsilon":-1,"prior":-1,"max_active_predictors":-1}
  - r2 0.017224
- buildModel 'glm', {"model_id":"glm-2fcdf58a-da59-4b3f-8a68-1177bce9531c","training_frame":"kc90_house_data.hex","nfolds":"10","seed":-1,"response_column":"price","ignored_columns":["id"],"ignore_const_cols":true,"family":"gaussian","solver":"AUTO","alpha":[0.001],"lambda":[0.1],"lambda_search":false,"standardize":false,"non_negative":false,"fold_assignment":"AUTO","score_each_iteration":false,"compute_p_values":false,"remove_collinear_columns":false,"max_iterations":-1,"link":"family_default","max_runtime_secs":0,"keep_cross_validation_predictions":false,"keep_cross_validation_fold_assignment":false,"missing_values_handling":"MeanImputation","intercept":true,"objective_epsilon":-1,"beta_epsilon":0.0001,"gradient_epsilon":-1,"prior":-1,"max_active_predictors":-1}
  - r2 0.619503

GLM - Regularization

- L1 Lasso and L2 Ridge:
  - Regularization is used solve problems with overfitting in GLM.
  - Penalties are introduced to avoid overfitting,
  - To reduce variance of the prediction error
  - To handle correlated predictors.
- H2O - Elastic Net
  - Alpha (0 – 1)/ Lambda(0 – 1, into very small fraction i.e. 0.0001)
  - Alpha = 0 –> Ridge
  - Alpha = 1 -> LASSO
  - Lambda – 0.0 > No regularization
- Lambda Search
  - Enable Lambda_Search
  - lambda_min_ratio
  - Nlambdas
  - max_active_predictors
- Look for Intercept, and p-values in the docs too
- Documentation: http://docs.h2o.ai/h2o/latest-stable/h2o-docs/data-science/glm.html#regularization

Linear Regression - Prediction

- Import data to predict
- Feature Engineering
  - Proper encoding for numerical data
  - Hide – ID and other features
- Experimentation with Alpha and Lambda
  - 0.001 & 0.1 – 0.61 – Did Prediction
  - 0.0001 & 0.1 - 0.61
  - 0.0001 & 0.5 – 0.58
  - 0.001 & 0.01 - 0.67 – Did Prediction
  - 0.004 & 0.01 – 0.70 – Did prediction
- Understanding Training and validation – r2 values

Price Prediction - GBM

- Feature Engineering
  - Proper encoding for numerical data
  - Hide – ID and other features
- All default
  - Training r2 = 0.95 & Validation r2 = 0.76 – Try prediction
  - Learning Rate – 0.5 + all default = 0.98/0.71
  - Learning Rate – 0.01 + all default = 0.50/0.40
- Setting – stopping rounds

Understanding GLM, GBM, DRF

Note: Using 3 FLOW interface build – GLM, GBM, DRF Model

- Validation and Cross Validation
- Scoring History, Validation History (Validation Frame/CV)
- Training and Validation Metrics
- Using Stopping metrics into Tree based algorithms – DRF/GBM
- How adding tree depth changes the results?
- Variable Importance
- Gains and Lift Chart
- Confusion Matrix (TPR/FPR)
- Solvers

Improving Overall Results

- Feature Engineering
  - Adding proper categories
    - Year, Waterfront, view, condition, grade, zipcode – Factors
- How r2 values helps better prediction
- GBM Improvements with CV
    - 0.3 Learning, all default – 0.795 >> Perform Prediction
    - Ntree=60, depth=5,l-rate=0.21,row-sr=0.8, col-sr=0.8> 0.80
  - GBM with Validation frame
    - Ntree=60, depth=5,l-rate=0.21,row-sr=0.8, col-sr=0.8> 0.82
  - DRF with CV and default settings > 0.80
  - Finally now – Ignore Date column
    - DRF - 0.86
    - GBM – 0.86
  - With Update Feature Engineering
    - GLM – Aplha:0.004, lambda:0.01 > r2 = 0.77

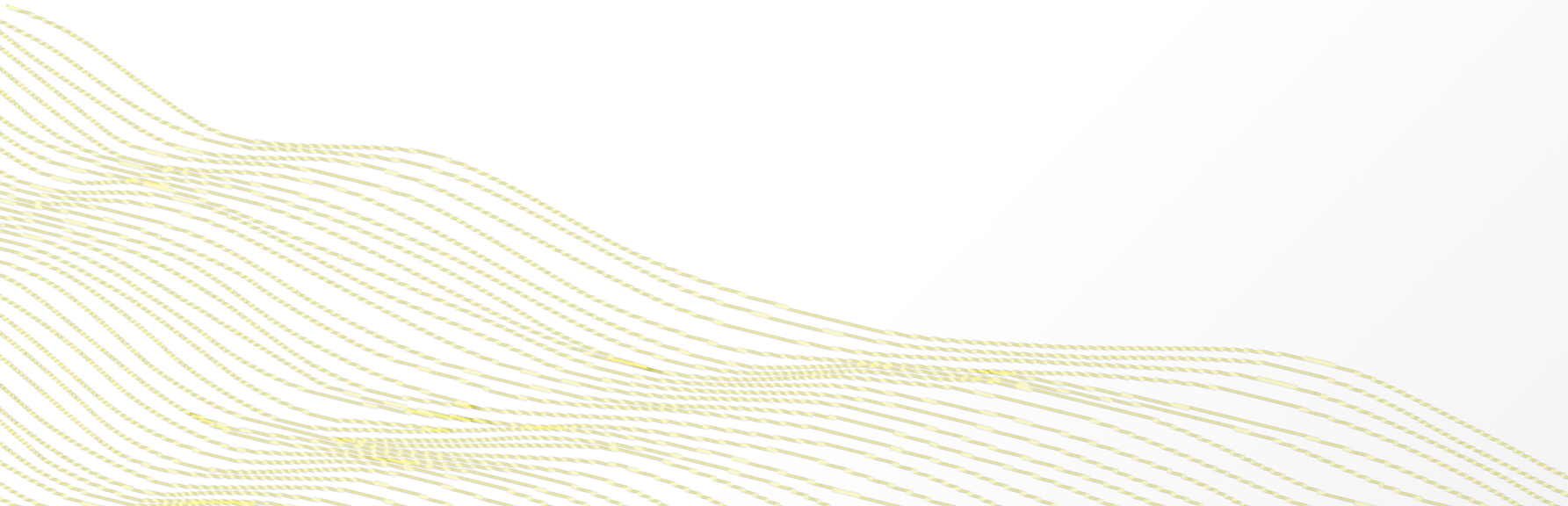
R - Demo

- Start R
- > h2o.init()
- > h2o.clusterStatus()
- ?h2o.gbm > Use the sample
- > h2o.varimp(gbm)
- > h2o.varimp_plot(gbm)
- Performance
  - > perf <- h2o.performance(gbm, australia.hex)
  - > h2o.mse(perf)
  - > h2o.r2(perf)
- Add CV – Rebuild the model with nfolds = 5
  - > h2o.cross_validation_models(gbm)
- Kmeans - ?h2o.kmeans
  - Run the example
  - Want to eval the K?
    - kmodel = h2o.kmeans(training_frame = prostate.hex, k = 10, x = c("AGE", "RACE", "VOL", "GLEASON"))
    - kmodel = h2o.kmeans(training_frame = prostate.hex, k = 100, x = c("AGE", "RACE", "VOL", "GLEASON"), estimate_k = T)

Supplement Information

- https://www.kaggle.com/harlfoxem/d/harlfoxem/housesalesprediction/house-price-prediction-part-1/discussion
- https://www.kaggle.com/auygur/d/harlfoxem/housesalesprediction/step-by-step-house-price-prediction-r-2-0-77/code
- https://rpubs.com/MagicSea/property_price

Solving a Binomial Classification problem



Binomial Classification Problem Description

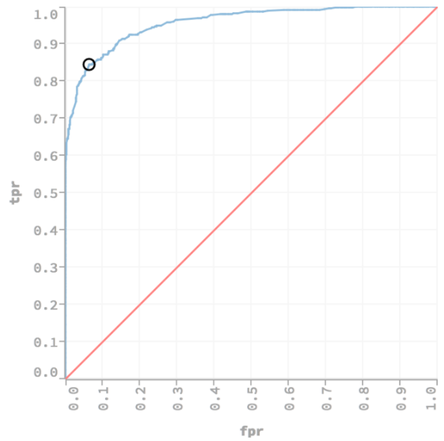
- Titanic Survivors list
  - Passengers on board: 1317, others were crew of total 2224
- Kaggle
  - https://www.kaggle.com/c/titanic
- Download Dataset:
  - http://biostat.mc.vanderbilt.edu/wiki/pub/Main/DataSets/titanic3.xls
- Local Datasets
  - /Users/avkashchauhan/learn/seattle-workshop/titanic_list.csv
- Documentation
  - http://docs.h2o.ai/h2o/latest-stable/h2o-docs/index.html

Predicting Titanic Survival Rate

- Reference:
  - Using scikit-learn
    - http://localhost:8888/notebooks/Titanic%20Survival%20Demo%20in%20Python.ipynb
  - Using H2O python estimators
    - http://localhost:8888/notebooks/Titanic%20Survival%20Demo%20in%20H2O%20and%20Python.ipynb

What we learn- ROC Curve

▾ ROC CURVE - TRAINING METRICS , AUC = 0.957211



Threshold: 0.4351133219432349 **Criterion:** max f1

Selected mark(s):

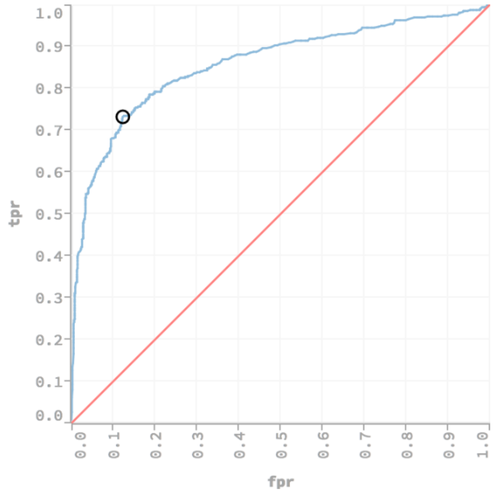
threshold	0.4351
f1	0.8669
f2	0.8535
f0point5	0.8807
accuracy	0.9018
precision	0.8902
recall	0.8448
specificity	0.9365
absolute_mcc	0.7898
min_per_class_accuracy	0.8448
mean_per_class_accuracy	0.8906
tns	693
fns	70
fps	47
tps	381
tnr	0.9365
fnr	0.1552
fpr	0.0635
tpr	0.8448
idx	187

CM

Actual/Predicted	0	1	Error	Rate
0	693	47	0.0635	47 / 740
1	70	381	0.1552	70 / 451
Total	763	428	0.0982	117 / 1191

AUC of 0.5 is random and 1 is perfect

▾ ROC CURVE - CROSS VALIDATION METRICS , AUC = 0.858262



Threshold: 0.43778509928697557 **Criterion:** max f1

Selected mark(s):

threshold	0.4378
f1	0.7569
f2	0.7416
f0point5	0.7728
accuracy	0.8220
precision	0.7838
recall	0.7317
specificity	0.8770
absolute_mcc	0.6177
min_per_class_accuracy	0.7317
mean_per_class_accuracy	0.8044
tns	649
fns	121
fps	91
tps	330
tnr	0.8770
fnr	0.2683
fpr	0.1230
tpr	0.7317
idx	188

CM

Actual/Predicted	0	1	Error	Rate
0	649	91	0.1230	91 / 740
1	121	330	0.2683	121 / 451
Total	770	421	0.1780	212 / 1191

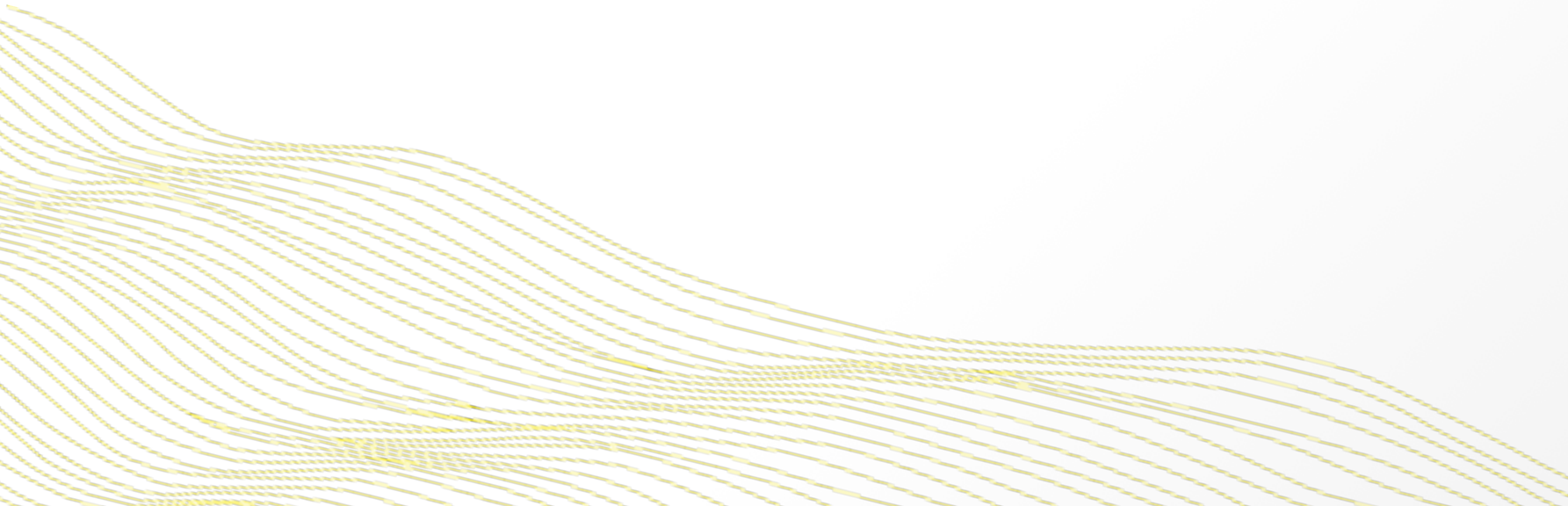
What we did

- GLM/GBM/RF in Python + scikit-learn
- GLM in H2O from Python
- Grid Search
- Working in FLOW
  - Ingest, split frame
  - GLM in H2O from FLOW
    - Response must be number
      - Understanding r2
  - GBM in FLOW
    - Response numeric– Regression
      - r2
    - Response enum – classification
      - AUC (Area under the curve)
      - Confusion matrix
      - ROC Curve – NEXT PAGE
- AUC
  - AUC of 0.5 is random and 1 is perfect
  - Improve AUC

What we covered

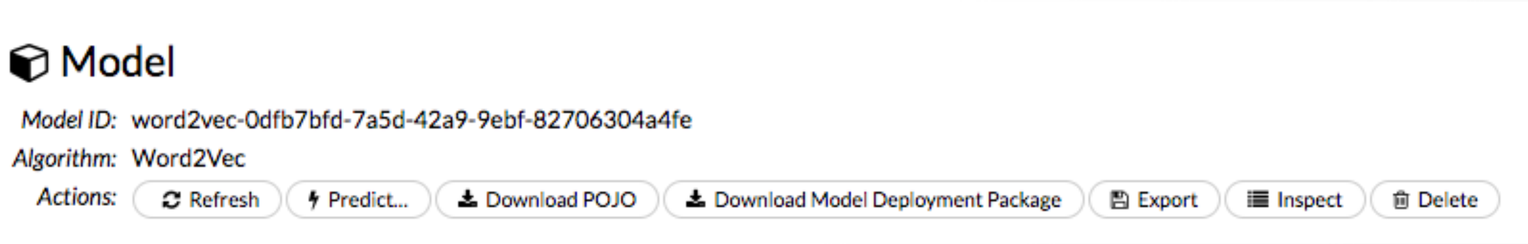
- Data Import
- Understood data, frame distribution, chunk, compression etc.
- Understood all features, through histograms, enums, etc.
- Split data frames for train and test
- Converting features to Factors/Enum
- Imputation of values

Model Deployment



Supported Model Types in H2O

- Binary
- POJO
- MOJO



exportModel "word2vec-0dfb7bfd-7a5d-42a9-9ebf-82706304a4fe", "/Users/avkashchauhan/Downloads/word2vec-0dfb7bfd-7a5d-42a9-9ebf-82706304a4fe", overwrite: true

POJO Demo

- Run “GBM_Airlines_Classification” demo from FLOW examples
- Download POJO from FLOW
- Create a temp folder/Be in temp folder
  - Move gbm_pojo_test.java
    - Get Model from RESTful API
      - $ curl -X GET "http://localhost:54321/3/**Models.java**/gbm_pojo_test" >> gbm_pojo_test_1.java
  - Get H2O gen-model
    - curl http://localhost:54321/3/h2o-genmodel.jar > h2o-genmodel.jar
  - Create main.java
  - Compile
    - $ javac -cp h2o-genmodel.jar -J-Xmx2g -J-XX:MaxPermSize=128m gbm_pojo_test.java main.java
    - Verify with compiled class files
  - Run
    - $ java -cp .:h2o-genmodel.jar main
    - See Results
- Docs: https://github.com/h2oai/h2o-3/blob/master/h2o-docs/src/product/howto/POJO_QuickStart.md

MOJO Demo

- Run “GBM_Airlines_Classification” demo from FLOW examples
- Download POJO from FLOW
- Create a temp folder/Be in temp folder
  - Move gbm_pojo_test.zip
    - Get Model from RESTful API
      - $ curl -X GET "http://localhost:54321/3/**Models.java**/gbm_pojo_test.zip" >> gbm_pojo_test.zip
  - Get H2O gen-model
    - curl http://localhost:54321/3/h2o-genmodel.jar > h2o-genmodel.jar
  - Create main_mojo.java
  - Compile
    - $ javac -cp h2o-genmodel.jar -J-Xmx2g -J-XX:MaxPermSize=128m gbm_pojo_test.java main_mojo.java
      - Make sure main method name is main_mojo
    - Verify with compiled class files
  - Run
    - $ java -cp .:h2o-genmodel.jar main
    - See Results
- Docs: https://github.com/h2oai/h2o-3/blob/master/h2o-docs/src/product/howto/MOJO_QuickStart.md

Thank you so much