Overview of H2O GBM implementation

Michal Kurka

Director of Engineering, H2O.ai

Don’t know H2O yet?

- Download H2O
  http://h2o-release.s3.amazonaws.com/h2o/latest_stable.html
  - Click on a tab corresponding to your environment (R, Py, K8S, Hadoop…)
- Start with H2O tutorials and/or training materials
  - https://github.com/h2oai/h2o-tutorials
  - https://github.com/h2oai/h2o-tutorials/tree/master/training/lending_club_exercise
- Have a question?
  - https://gitter.im/h2oai/h2o-3 - (the most) direct line to H2O developers
  - Also: https://stackoverflow.com/questions/tagged/h2o

What are we going to talk about?

1. Introduction to H2O
2. H2O Internals
3. GBM Overview
4. GBM Internals
5. Advanced Features
6. Productionizing

Introduction to H2O

What is H2O?

ML Platform

Open source in-memory ML framework

- Parallelized and distributed algorithms
- GLM, Random Forest, GBM, Deep Learning, etc.

Tech and API

Easy to use and adopt

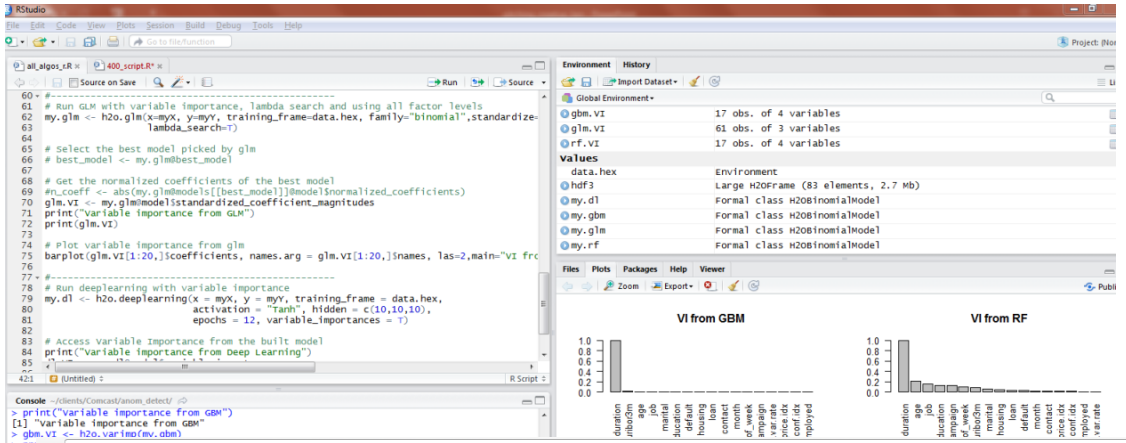
- Written in Java – integrates easily into big data environments (Spark, Hadoop, …)
- Installation is straightforward (PyPI, Conda, CRAN, “just one jar”)
- REST API (Java) – run H2O from Python, R, H2O Flow (web UI)

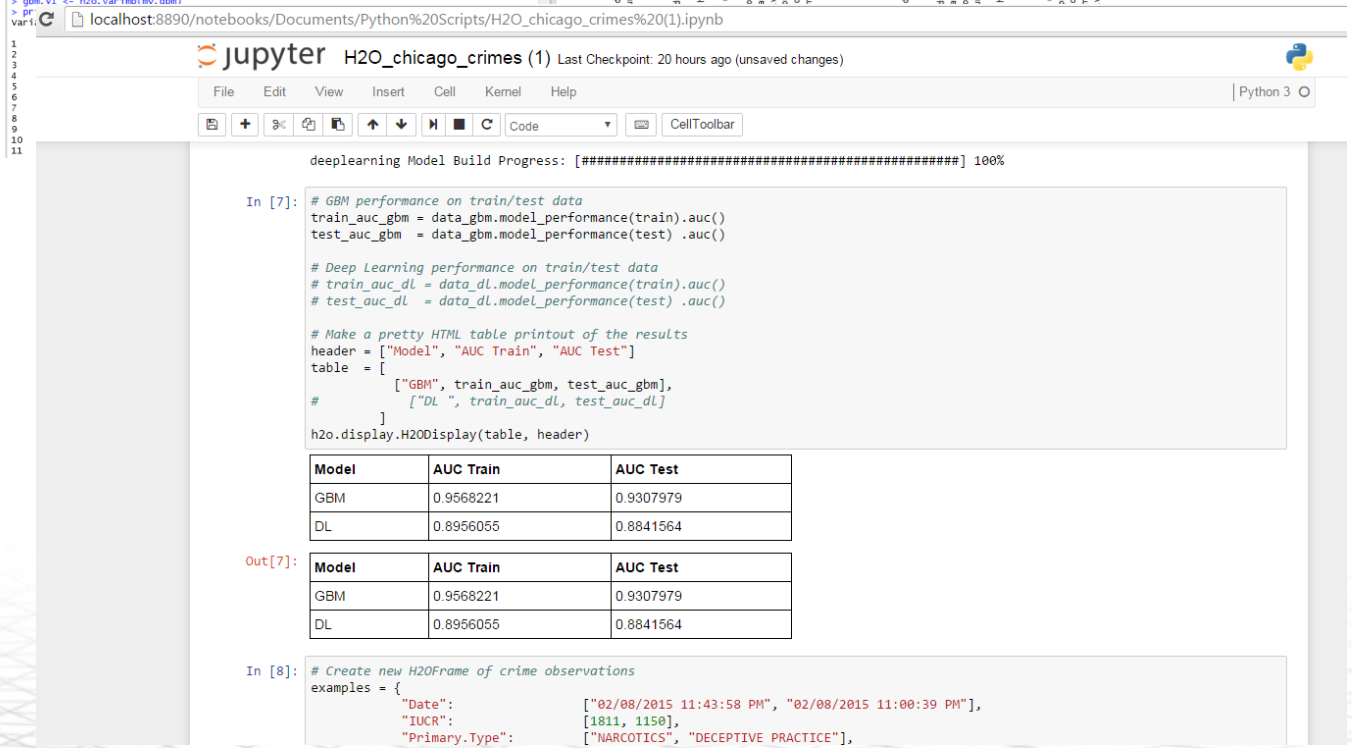
Big Data

More data? Or better models? BOTH

- Use all of your data – model without sampling
- More Data + Better Models = Better Predictions

R, Python and Flow





jupyter H2O_chicago_crimes (1) Last Checkpoint: 20 hours ago (unsaved changes)

File Edit View Insert Cell Kernel Help | Python 3

Code | CellToolbar

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

In [7]:
```
# GBM performance on train/test data
train_auc_gbm = data_gbm.model_performance(train).auc()
test_auc_gbm  = data_gbm.model_performance(test) .auc()

# Deep Learning performance on train/test data
# train_auc_dl = data_dl.model_performance(train).auc()
# test_auc_dl  = data_dl.model_performance(test) .auc()

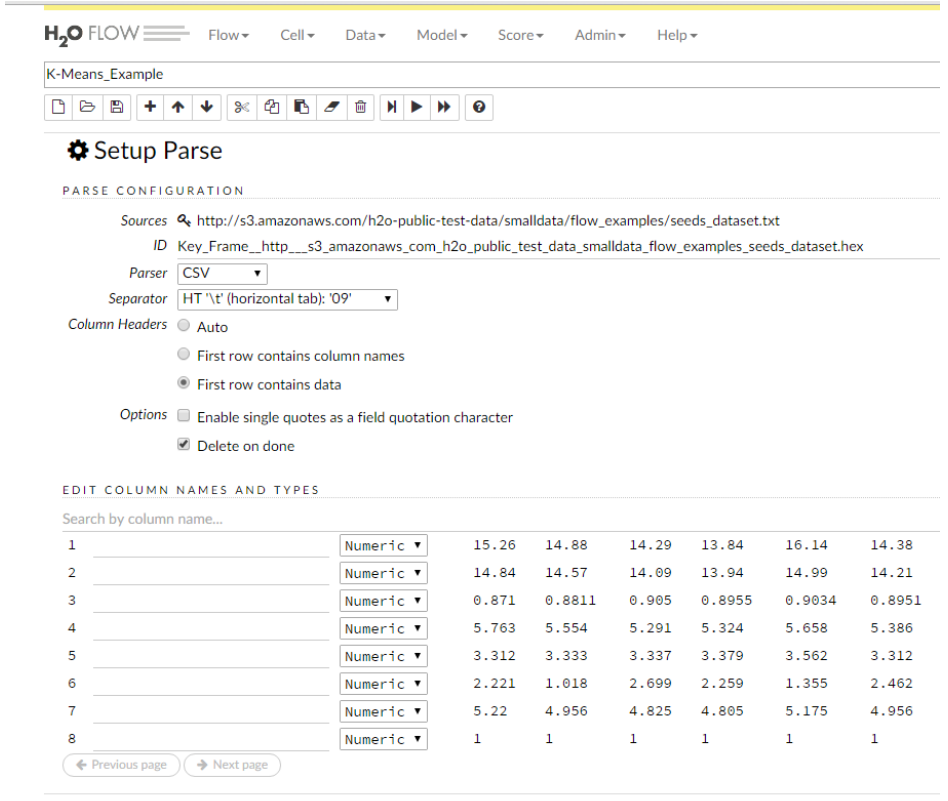
# Make a pretty HTML table printout of the results
header = ["Model", "AUC Train", "AUC Test"]
table  = [
            ["GBM", train_auc_gbm, test_auc_gbm],
#           ["DL ", train_auc_dl, test_auc_dl]
         ]
h2o.display.H2ODisplay(table, header)
```

Model	AUC Train	AUC Test
GBM	0.9568221	0.9307979
DL	0.8956055	0.8841564

Out[7]:

Model	AUC Train	AUC Test
GBM	0.9568221	0.9307979
DL	0.8956055	0.8841564

In [8]:
```
# Create new H2OFrame of crime observations
examples = {
            "Date":                 ["02/08/2015 11:43:58 PM", "02/08/2015 11:00:39 PM"],
            "IUCR":                 [1811, 1150],
            "Primary.Type":         ["NARCOTICS", "DECEPTIVE PRACTICE"],
```



H2O FLOW Flow Cell Data Model Score Admin Help

K-Means_Example

Setup Parse

PARSE CONFIGURATION

Sources http://s3.amazonaws.com/h2o-public-test-data/smalldata/flow_examples/seeds_dataset.txt

ID Key_Frame__http___s3_amazonaws_com_h2o_public_test_data_smalldata_flow_examples_seeds_dataset.hex

Parser CSV

Separator HT '\t' (horizontal tab): '09'

Column Headers Auto / First row contains column names / First row contains data

Options Enable single quotes as a field quotation character / Delete on done

EDIT COLUMN NAMES AND TYPES

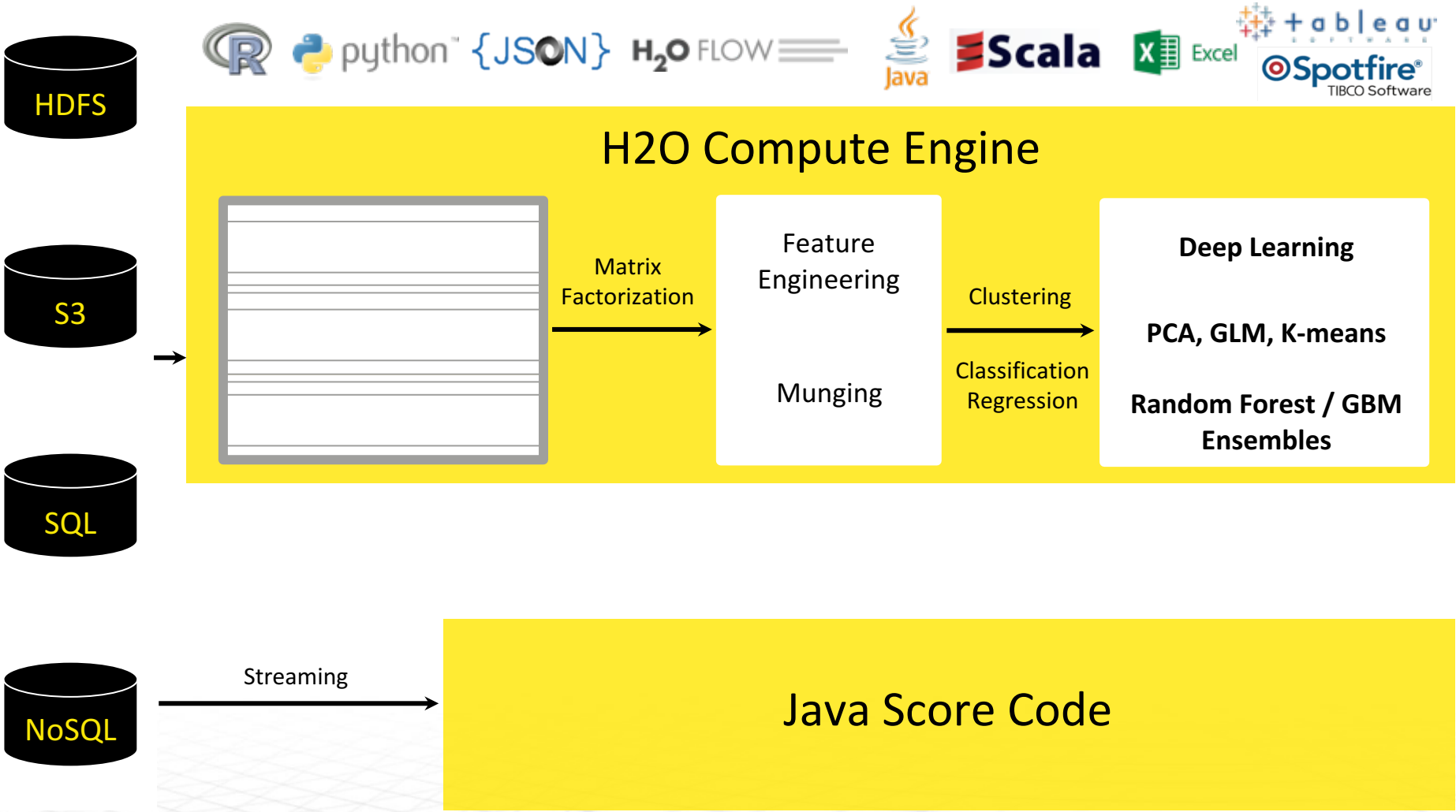
Search by column name...

1		Numeric	15.26	14.88	14.29	13.84	16.14	14.38
2		Numeric	14.84	14.57	14.09	13.94	14.99	14.21
3		Numeric	0.871	0.8811	0.905	0.8955	0.9034	0.8951
4		Numeric	5.763	5.554	5.291	5.324	5.658	5.386
5		Numeric	3.312	3.333	3.337	3.379	3.562	3.312
6		Numeric	2.221	1.018	2.699	2.259	1.355	2.462
7		Numeric	5.22	4.956	4.825	4.805	5.175	4.956
8		Numeric	1	1	1	1	1	1

Previous page Next page

Ready

Data and Client Agnostic



H2O Machine Learning Methods

H2O.ai

Supervised Learning

Statistical Analysis

- **Penalized Linear Models**: Super-fast, super-scalable, and interpretable
- **Naïve Bayes:** Straightforward linear classifier

Decision Tree Ensembles

- **Distributed Random Forest**: Easy-to-use tree-bagging ensembles
- **Gradient Boosting Machine**: Highly tunable tree-boosting ensembles
- **eXtreme Gradient Boosting**: Popular XGBoost algorithm in H2O

Stacking

- **Stacked Ensemble**: Combine multiple types of models for better predictions

AutoML

- **Automatic Machine Learning**: Automated exploration of supervised learning approaches

Unsupervised Learning

Clustering

- **K-means**: Partitions observations into similar groups; automatically detects number of groups

Dimensionality Reduction

- **Principal Component Analysis**: Transforms correlated variables to independent components
- **Generalized Low Rank Models:** Extends the idea of PCA to handle arbitrary data consisting of numerical, Boolean, categorical, and missing data

Aggregator

- **Aggregator:** Efficient, advanced sampling that creates smaller data sets from larger data sets

Neural Networks

Multilayer Perceptron

Deep Learning

- **Deep neural networks**: Multi-layer feed-forward neural networks for standard data mining tasks
- **Convolutional neural networks:** Sophisticated architectures for pattern recognition in images, sound, and text

Anomaly Detection

Term Embeddings

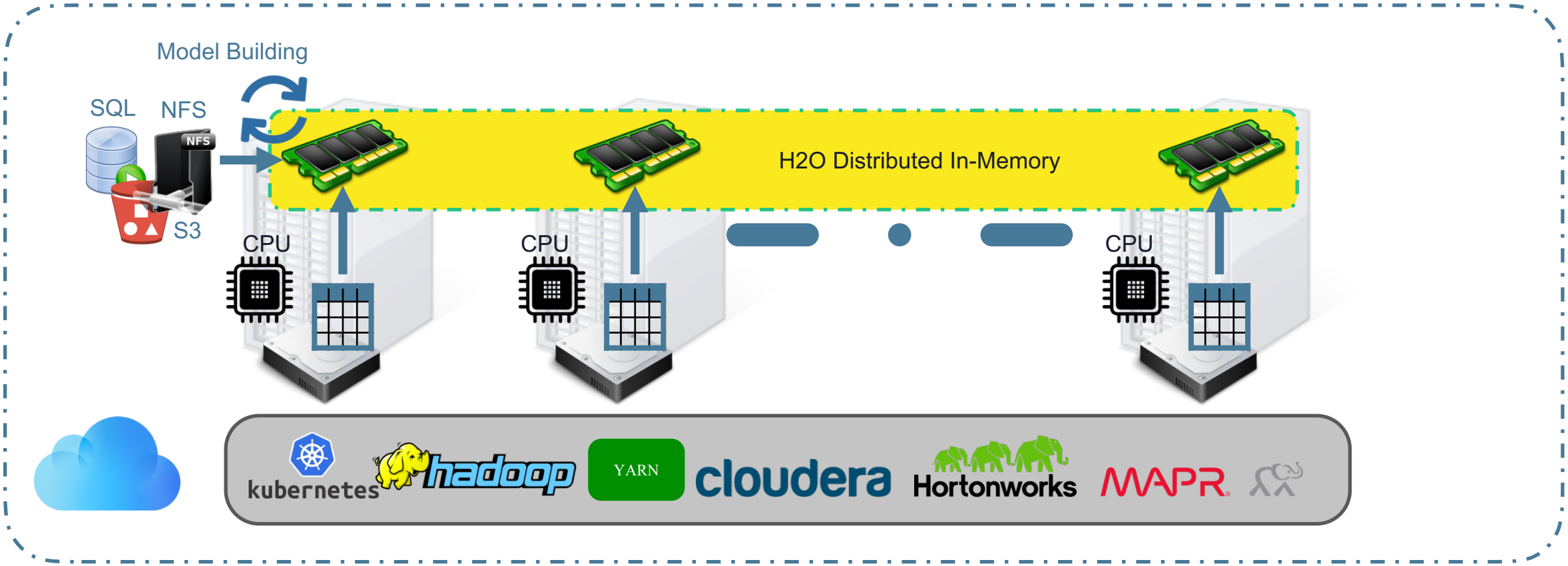
- **Autoencoders**: Find outliers using a nonlinear dimensionality reduction technique
- **Word2vec:** Generate context-sensitive numerical representations of a large text corpus

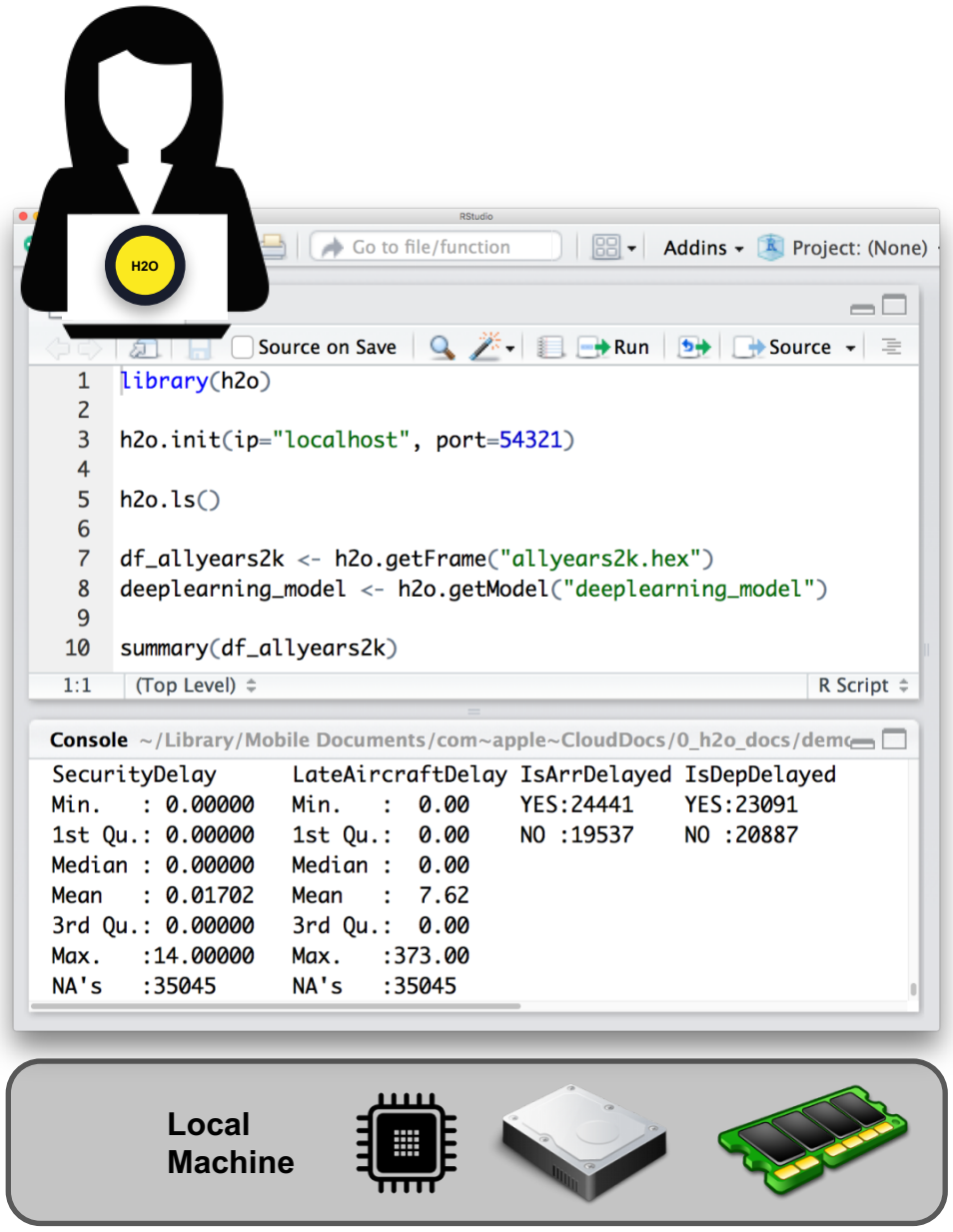
Python Interface Overview

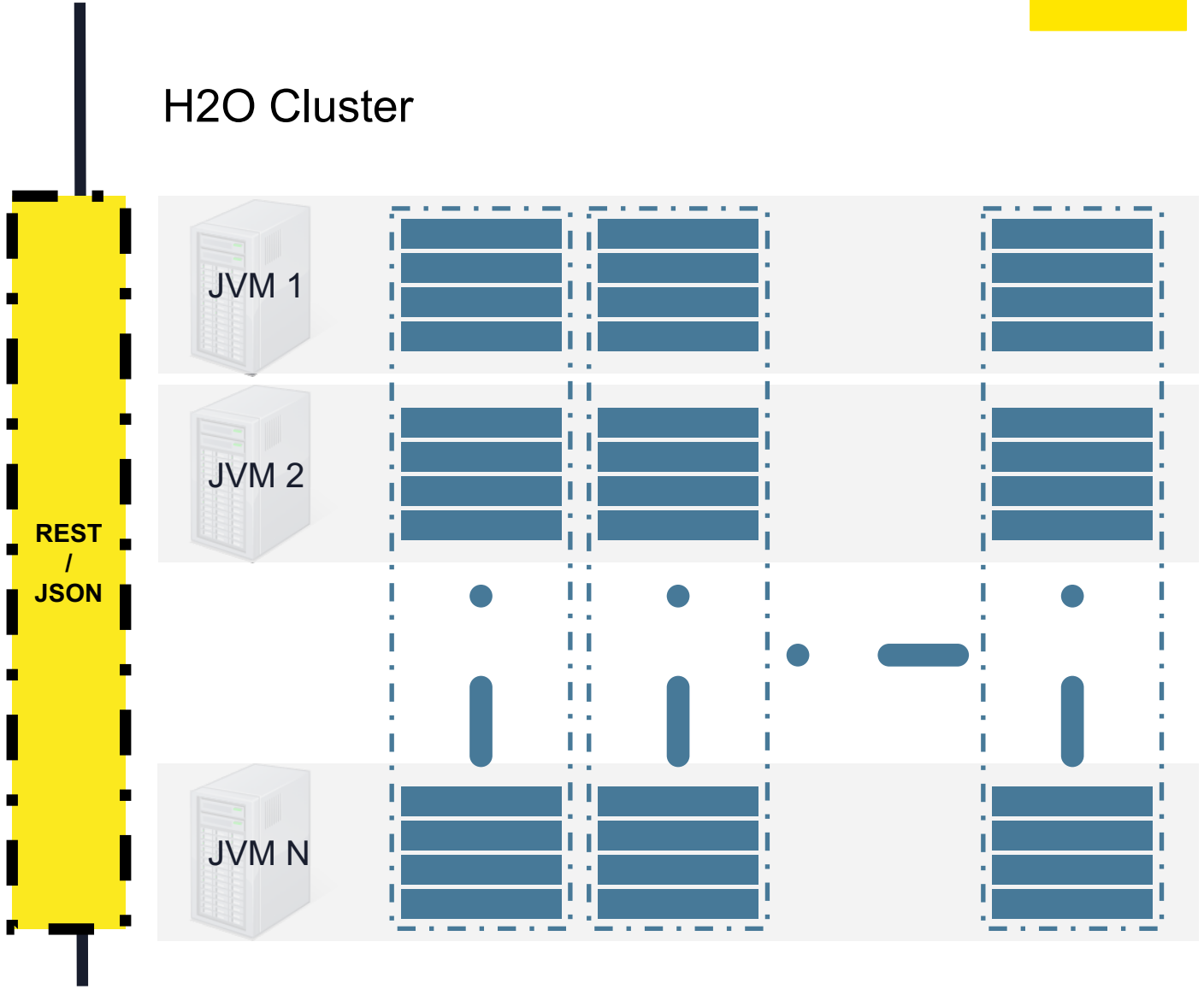
Action	Pandas or scikit-learn	H2O
Reading data	pandas.read_csv(data_path)	h2o.import_file(data_path)
Summarizing data	pandas_frame.describe()	h2o_frame.describe()
Summary statistics	pandas_frame.mean()	h2o_frame.mean()
Combining rows	pandas.concat(list[frame1,frame2])	h2o_frame.rbind(h2o_frame2)
Combining columns	pandas.concat(list[frame1,frame2],axis = 1)	h2o_frame.cbind(h2o_frame2)
Data selection	pandas_frame[:, :]	h2o_frame[:, :]
Transforming columns	np.log(pandas_frame[x]) np.sqrt(pandas_frame[x])	h2o_frame[x].log() h2o_frame[x].sqrt()
Building Random Forest	model = RandomForestClassifier(n_estimators = 100) model = model.fit(x_frame, y_frame)	model = H2ORandomForestClassifier(n_trees = 100) model = model.train(x, y, train_frame)
Model Prediction	model.predict	model.predict
Model Metrics	metrics.auc	metrics = model.model_performance(frame) metrics.auc()

H2O Internals

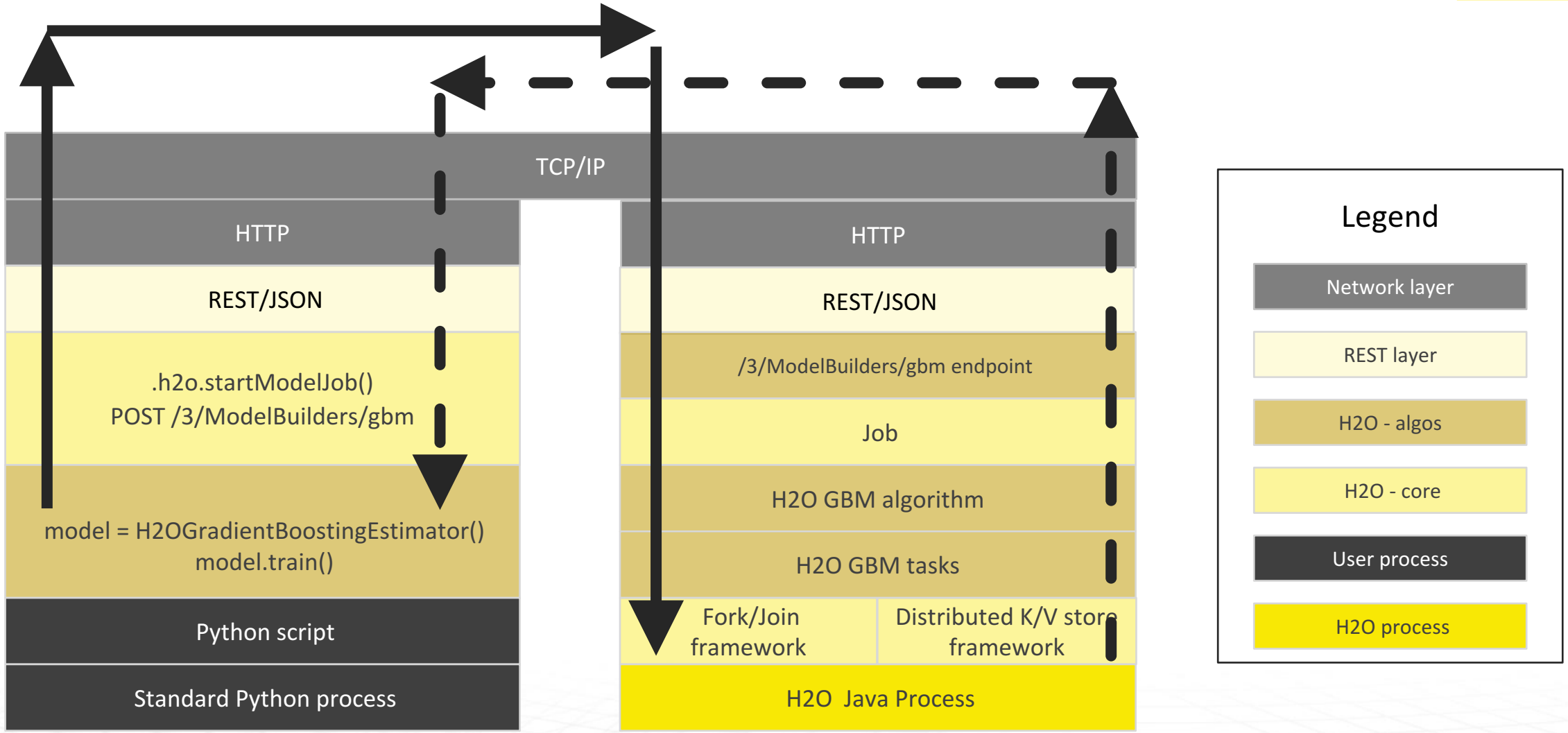
H2O Cluster


Model Building
SQL
NFS
NFS
S3
H2O Distributed In-Memory
CPU
CPU
CPU
kubernetes
hadoop
YARN
cloudera
Hortonworks
MAPR

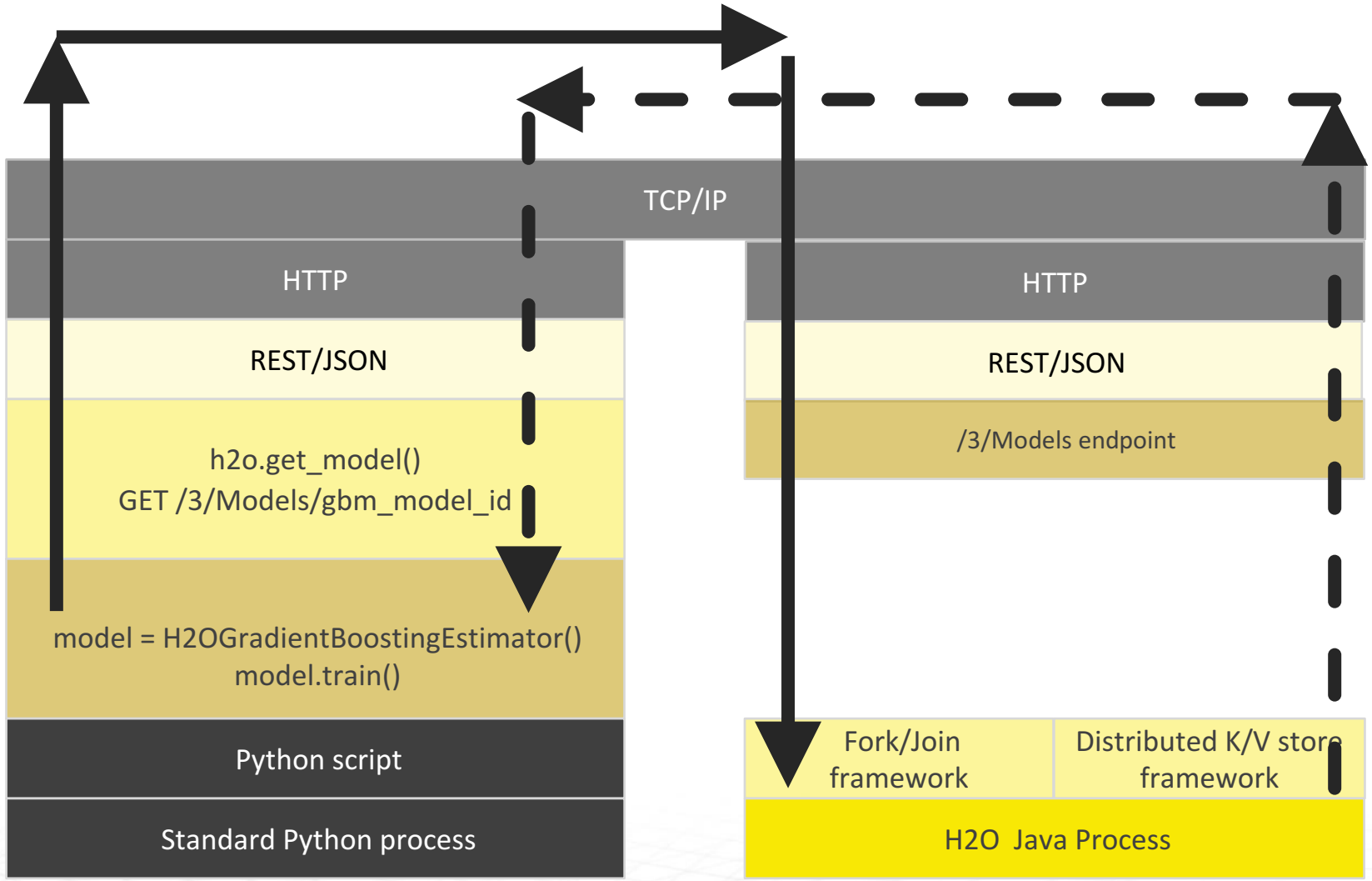

H2O
RStudio
Go to file/function
Addins
Project: (None)
Source on Save
Run
Source
1 library(h2o)
2
3 h2o.init(ip="localhost", port=54321)
4
5 h2o.ls()
6
7 df_allyears2k <- h2o.getFrame("allyears2k.hex")
8 deeplearning_model <- h2o.getModel("deeplearning_model")
9
10 summary(df_allyears2k)
1:1 (Top Level)
R Script
Console ~/Library/Mobile Documents/com~apple~CloudDocs/0_h2o_docs/dem
SecurityDelay LateAircraftDelay IsArrDelayed IsDepDelayed
Min. : 0.00000 Min. : 0.00 YES:24441 YES:23091
1st Qu.: 0.00000 1st Qu.: 0.00 NO :19537 NO :20887
Median : 0.00000 Median : 0.00
Mean : 0.01702 Mean : 7.62
3rd Qu.: 0.00000 3rd Qu.: 0.00
Max. :14.00000 Max. :373.00
NA's :35045 NA's :35045
Local Machine

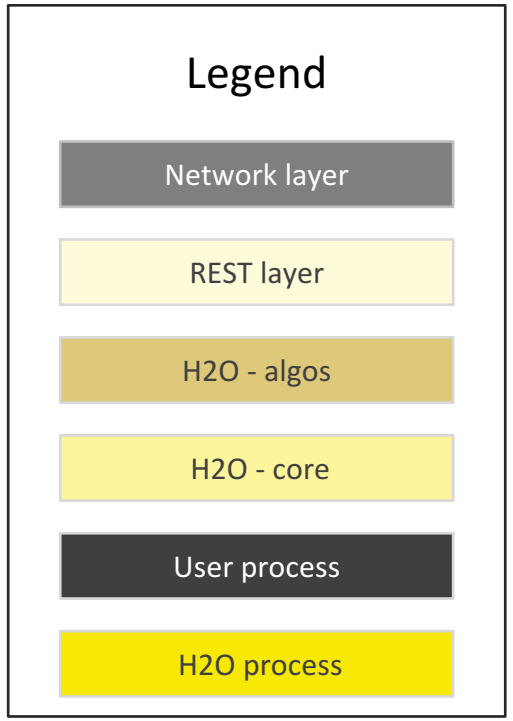

H2O Cluster
REST / JSON
JVM 1
JVM 2
JVM N

Training GBM from Python



Retrieving GBM Result from Python





H2O GBM Overview

H2O Supervised Algos – Common features

- supervised learning for regression and classification tasks
- automatic handling of categorical values
- distributed and parallelized computation on either a single node or a multi-node cluster
- built-in cross-validation support
- automatic early stopping based on convergence of user-specified metrics to user-specified relative tolerance
  - supports stopping on user defined metric
- grid search for hyperparameter optimization and model selection
  - ```
    gs = H2OGridSearch(H2OGradientBoostingEstimator(nfolds=5, seed=42),
                       hyper_params=hyper_parameters)
    ```
  - ```
    gs.train(x=list(range(4)), y=4, training_frame=train)
    ```
- parallel model training on sub-populations (“segments”) of the training dataset
  - ```
    gbm = H2OGradientBoostingEstimator(**params)
    models = gbm.train_segments(y="survived", training_frame=titanic,
                                segments=["pclass", "sex"])
    ```

- saving and re-loading a binary model
- model checkpointing and resuming training from a checkpoint (with possibly new data)
- model export in plain Java code (POJO) or model package (MOJO) for deployment in production environments
    ```

GBM – Summary of features

- support for exponential families (Poisson, Gamma, Tweedie) and loss functions in addition to binomial (Bernoulli), Gaussian and multinomial distributions, such as Quantile regression (including Laplace), support for Quasibinomial distribution
- automatically handles categorical values without need to do any encoding
- ability to define custom loss function
- handling of imbalanced problems
- column and row sampling (per split and per tree) for better generalization
- supports Platt scaling for model calibration
- allows to specify monotone constraints
- learning rate and learning rate annealing
- built-in consistency checks

- After training (all included in MOJO)
  - TreeSHAP
  - leaf node assignment, observation path traversal
  - tree API – inspect trees in Python/R API
  - tree visualization (dot – graphviz, png/jpeg – built-in, json)
  - analysis of feature interactions

- MOJO, POJO, ONNX,.., C++/C# runtime

GBM – Basic use in Python

```
import h2o
from h2o.estimators.gbm import H2OGradientBoostingEstimator

h2o.init()

cars = h2o.import_file("https://s3.amazonaws.com/h2o-public-test-
data/smalldata/junit/cars_20mpg.csv")
response = "economy_20mpg"
cars[response] = cars["economy_20mpg"].asfactor()
predictors = ["displacement","power","weight","acceleration","year"]
train, valid = cars.split_frame(ratios=[.8], seed=1234)
cars_gbm = H2OGradientBoostingEstimator(seed=1234)
cars_gbm.train(x=predictors,
...               y=response,
...               training_frame=train,
...               validation_frame=valid)
cars_gbm.auc(valid=True)
```

Note: No need to to handle categorical features – GBM handles them natively (no encoding)

GBM Internals

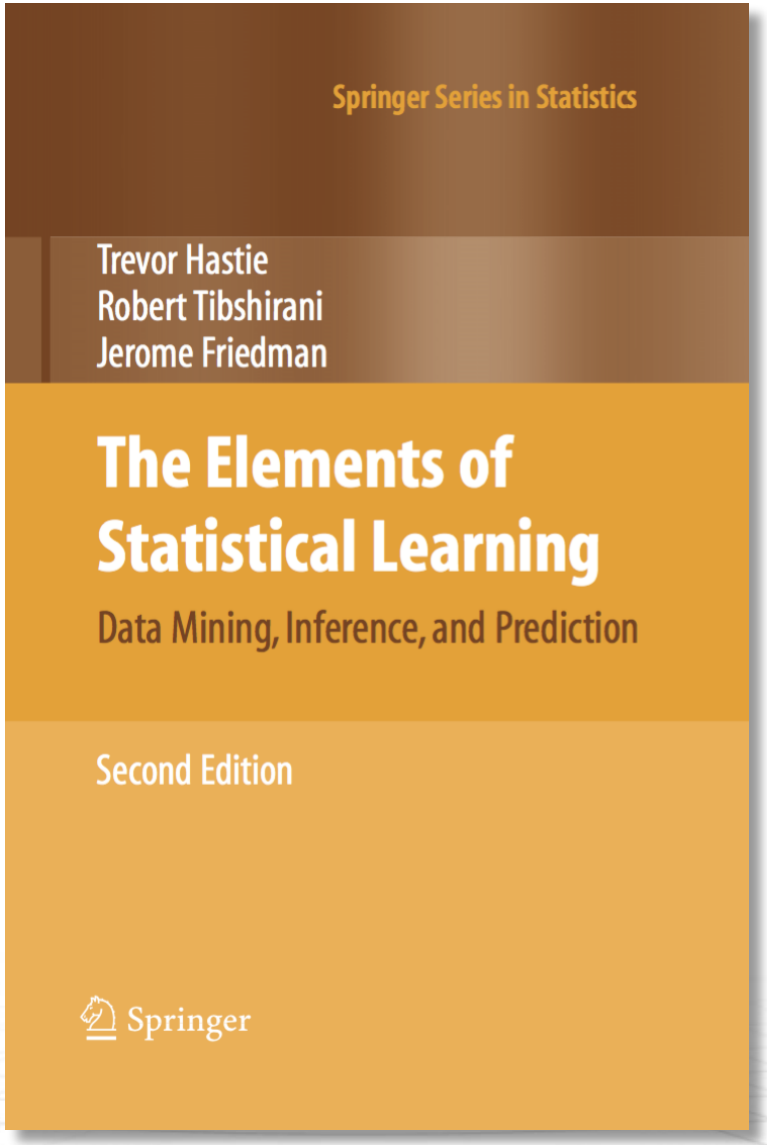
H2O GBM – Main Concepts

- Implementation follows the algorithm outlined in *Elements of Statistical Learning II, Trevor Hastie, Robert Tibshirani, and Jerome Friedman on page 387* (shown on next slide)
- Fully parallelized and distributed, no limits in terms of #rows it can process
- Uses feature binning – based on building histograms (multiple histogram types supported)
- By default categorical features don't have an explicit encoding – categorical splits try to separate the levels in the best way possible for each split
- Outputs compressed representation of trees optimized for fast scoring with low GC requirements
- Constraints are self-checked – it is better to output no model than a model that violates user-given constraints
- Model reproducibility is always guaranteed as long as reproducibility requirements are met

GBM Algorithm

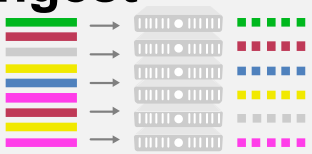
Algorithm 10.3 *Gradient Tree Boosting Algorithm.*

1. Initialize $f_0(x) = \arg\min_\gamma \sum_{i=1}^{N} L(y_i, \gamma)$.
2. For $m = 1$ to M:
   (a) For $i = 1, 2, \ldots, N$ compute
   $$r_{im} = -\left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f=f_{m-1}}.$$
   (b) Fit a regression tree to the targets r_{im} giving terminal regions R_{jm}, $j = 1, 2, \ldots, J_m$.
   (c) For $j = 1, 2, \ldots, J_m$ compute
   $$\gamma_{jm} = \arg\min_\gamma \sum_{x_i \in R_{jm}} L\left(y_i, f_{m-1}(x_i) + \gamma\right).$$
   (d) Update $f_m(x) = f_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$.
3. Output $\hat{f}(x) = f_M(x)$.



GMB Implementation - Overview

1 Parallel Data Ingest



Data is stored in-memory on all cluster compute nodes

- **Rows are evenly distributed across the cluster**
- **Columns are stored separately and compressed**

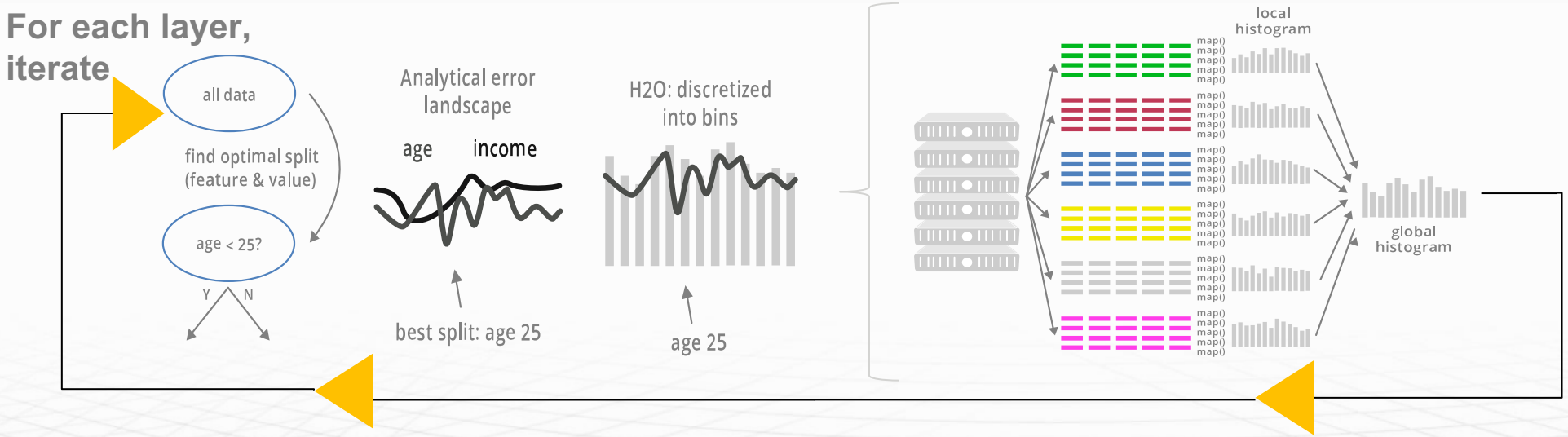
Basis for fine-grain Map/Reduce for histogram calculation

2 Distributed Tree Building via Fine-Grain Map/Reduce to find optimal split points of data layer by layer

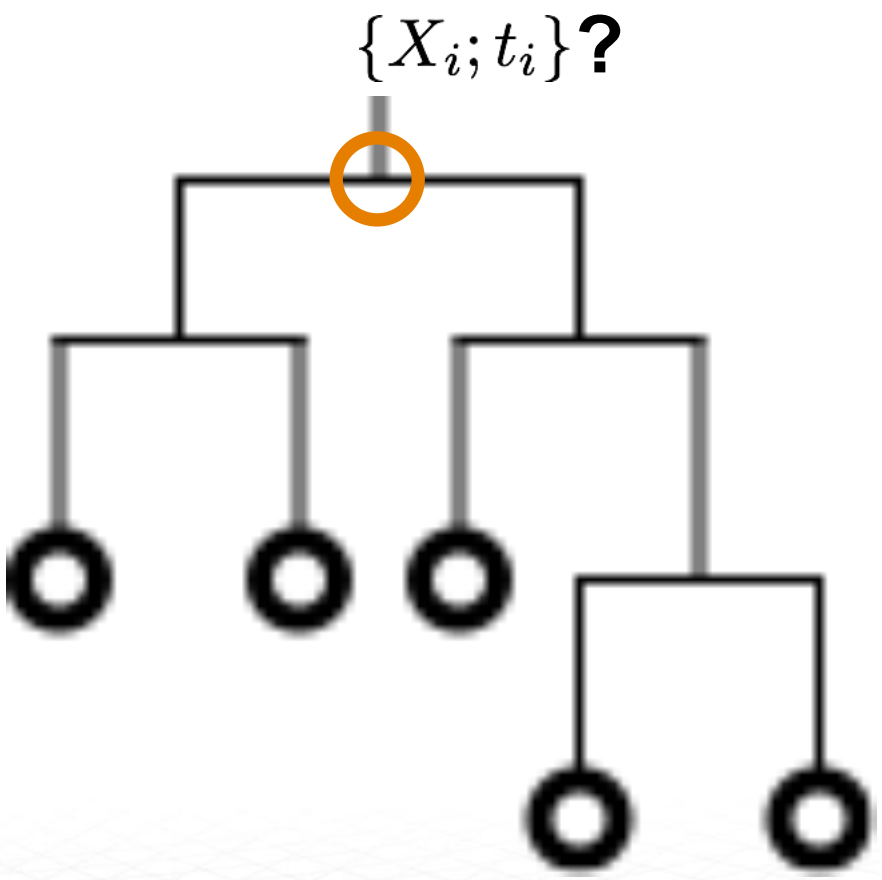
Start with root node and build layers of tree nodes

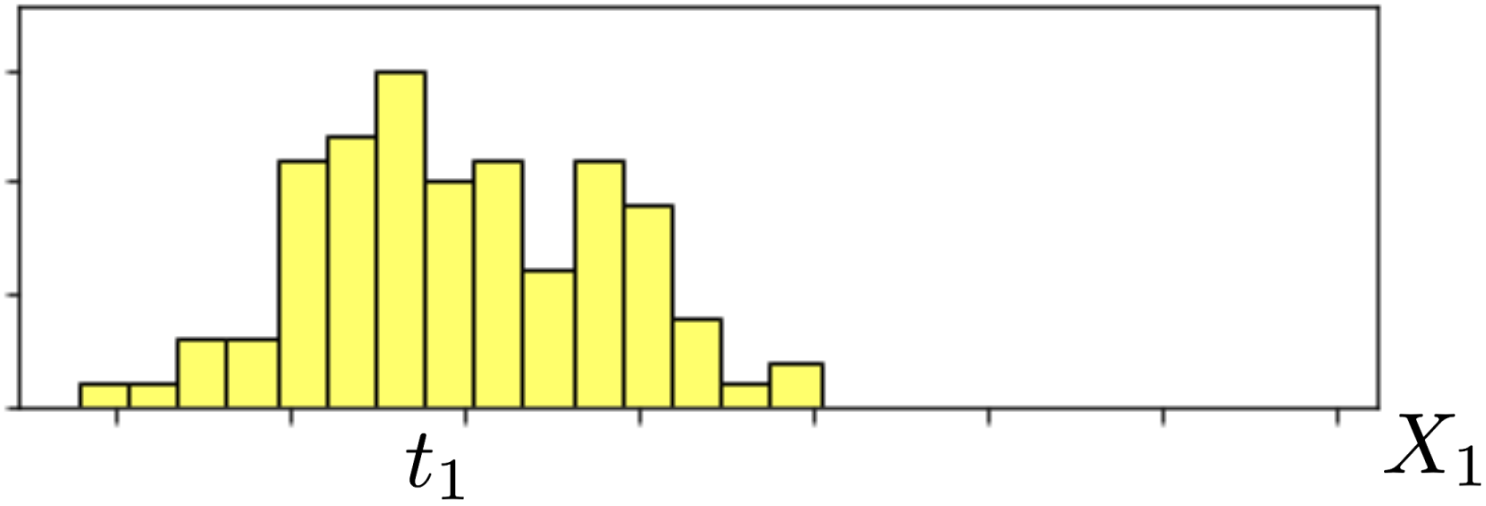
For each layer, repeat the following:

- **For a set of features, find a split point for every possible feature**
- **Find the split that minimizes total squared error in produced child nodes**
- **Use discretization to limit the number of potential splits**
  - **To find the split, local histograms are calculated on each node and then aggregated into a global histogram**
  - **From the global histogram, the best split column is chosen**



Numerical Binning





- **`nbins_top_level`**: number of bins to use at the top node, then halve at each ensuing level
- **`nbins`**: minimum number of bins in histogram
- **`histogram_type`**: method for binning:
  - Uniform Adaptive,
  - Random,
  - QuantilesGlobal,
  - RoundRobin

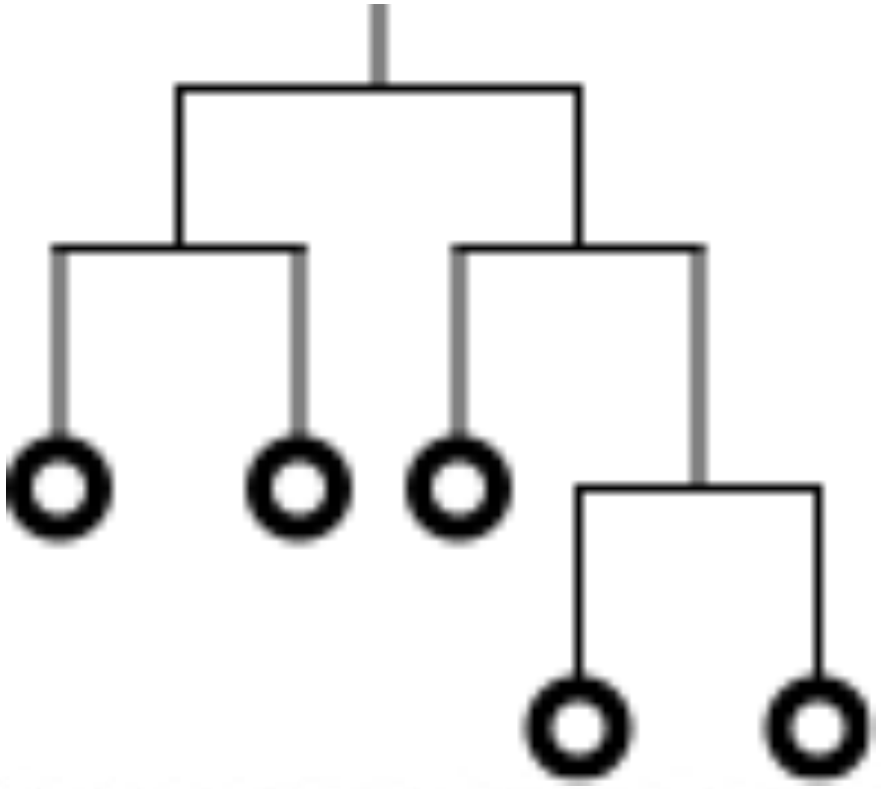
Categorical Binning

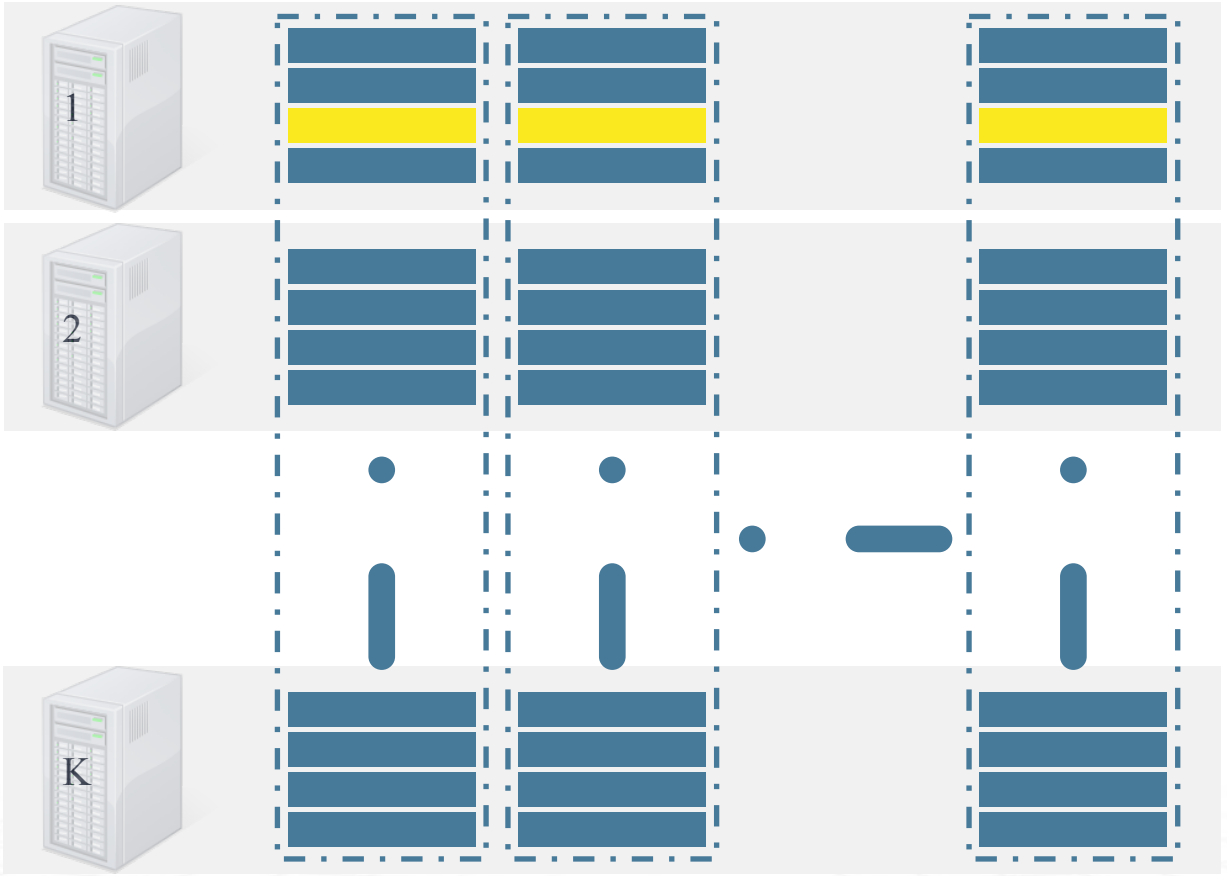
- Each categorical level is assigned an index
- Domain of size N has indices 0..N-1
- Levels are by default ordered lexicographically and index is assigned sequentially
  - Users are able to re-order the domain and thus modify the level indices

- Example: {Red, Blue, Yellow, Orange, Purple, Green}
  - Lexicographical order: {Blue, Green, Orange, Purple, Red, Yellow}
  - `nbin_cats = 2`: {Blue, Green, Orange}, {Purple, Red, Yellow}
  - `nbin_cats = 3`: {Blue, Green}, {Orange, Purple}, {Red, Yellow}
  - `nbin_cats >= 6`: {Blue}, {Green}, {Orange}, {Purple}, {Red}, {Yellow}

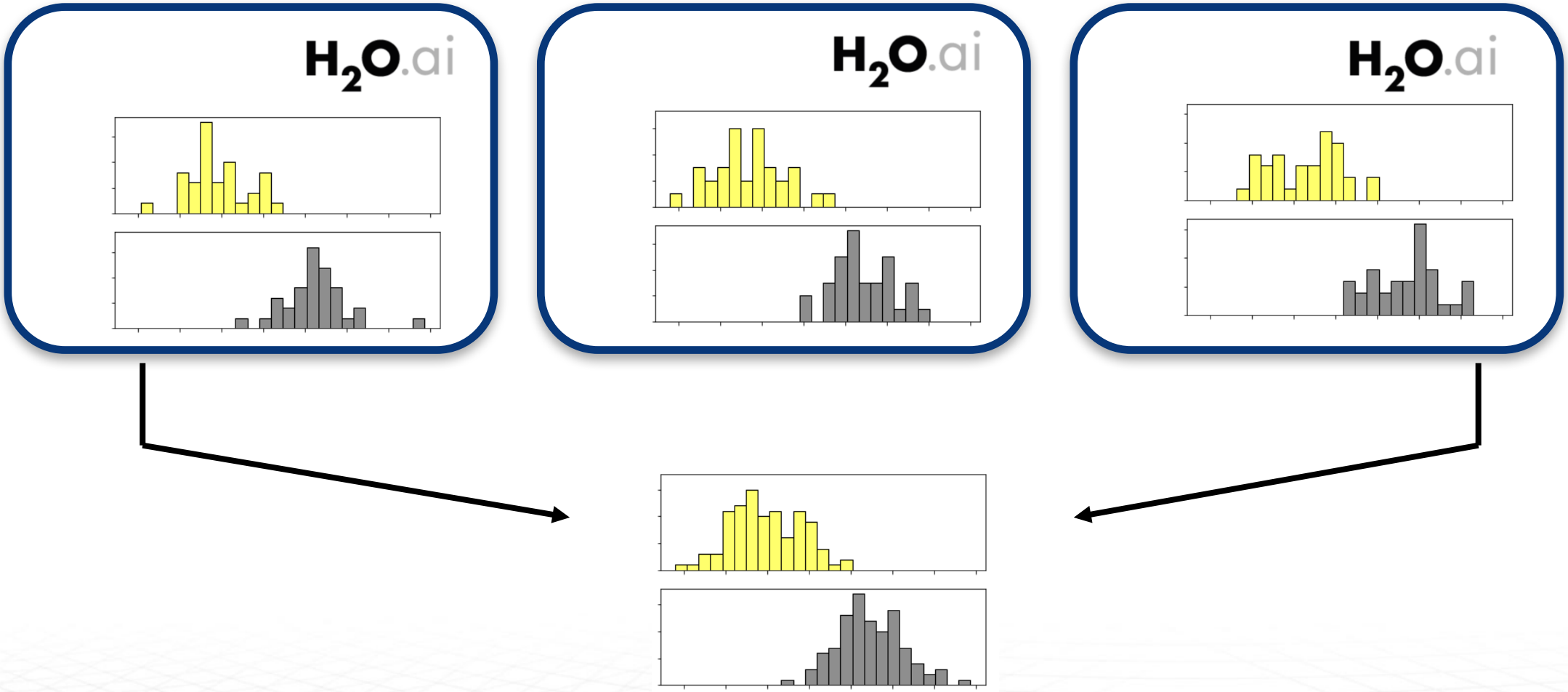
- The value of `nbin_cats` has greater impact than `nbins` has for numerical features
  - For columns with many factors, a value can add randomness to the split decisions (since the factor levels get grouped together somewhat arbitrarily), while large values can lead to perfect splits, resulting in overfitting.

Tree Growth with Data Parallelism

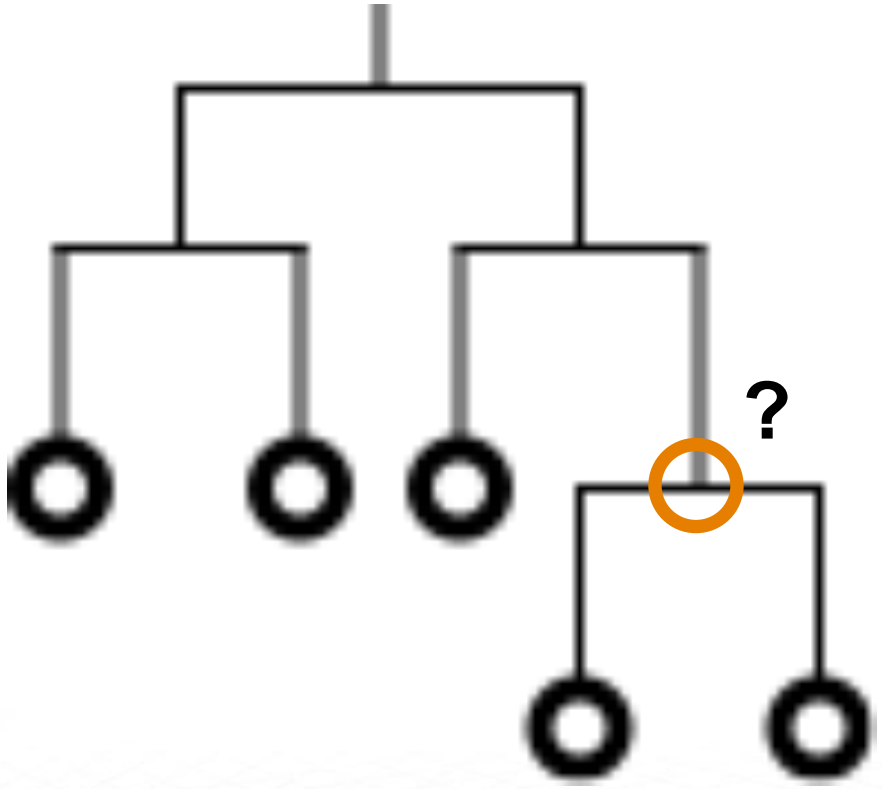




Splitting with Distributed Data



Finding the best split point



- In Java `DTree#findBestSplitPoint`
  - very complex and hard to read function
- Input: Histogram for the given feature based on observation that are marked for the given node
  - each bin has *w*, *wY*, *wYY*
- For categorical features: sort the bins by the average response to establish ordering on the bins
- Consider splitting NAs away from other observations
- Compute mean/var for cumulative bins
- Evaluate each bin boundary as a split point
  - objective is to minimize total Squared Error in child nodes
  - consider sending NAs left or right
  - test whether given split would meet split requirements (minimum number of observations in child nodes, minimum relative SE improvement)
- For categorical features:
  - find a simple (numerical <) representation of the split
  - find a BitSet representation
- Return a Split candidate (with calculated SE)

```
// We're making an unbiased estimator, so that MSE==Var.
// Then Squared Error = MSE*N = Var*N
//                    = (wYY/N - wY^2)*N
//                    = wYY - N*wY^2
//                    = wYY - N*(wY/N)(wY/N)
//                    = wYY - wY^2/N
```

Advanced Features

Inspecting trees

- Users can get insights from the build model by inspecting the individual trees
- Trees can be rendered into *json*, *dot* and *png* formats
  - can be run offline using MOJO
  - *json* format is the most suitable for converting H2O internal model representation into a different format (ONNX)
  - *dot* format is the best way to visualize trees (use *graphviz* to convert to image)
  - *png* format is a direct way to render trees into images in H2O (for users who cannot install graphviz)
- Trees can also be inspected programmatically by using “Tree API”
  - requires running H2O cluster

```
prostate["CAPSULE"] = prostate["CAPSULE"].asfactor()
gbm_h2o = H2OGradientBoostingEstimator(ntrees = 3, max_depth = 3, min_rows = 10)
gbm_h2o.train(x = list(range(1,prostate.ncol)), y = "CAPSULE", training_frame = prostate)
mojo_path = gbm_h2o.download_mojo()
png_dir = h2o.print_mojo(mojo_path, format="png")
```

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

```
from h2o.tree import H2OTree
tree = H2OTree(model = gbm_h2o, tree_number = 0, tree_class = None)
```

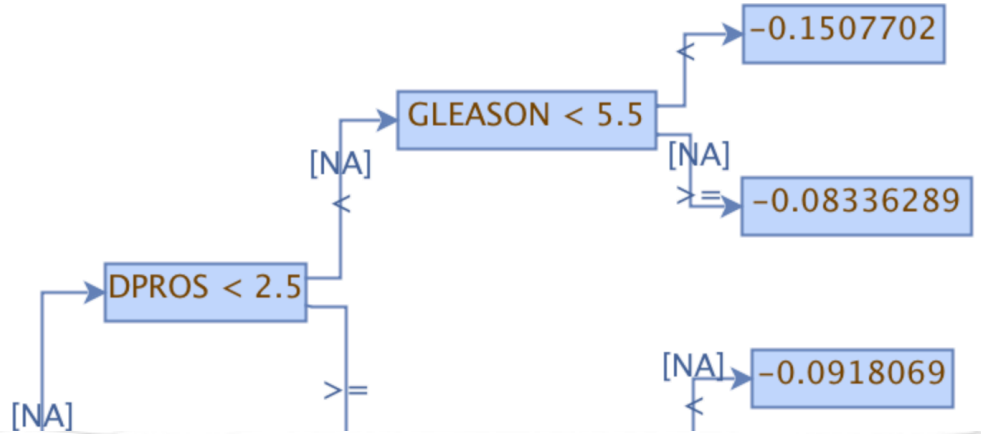
```
tree.root_node.show()
```

```
Node ID 0
Left child node ID = 1
Right child node ID = 2

Splits on column GLEASON
Split threshold < 6.5 to the left node, >= 6.5 to the right node

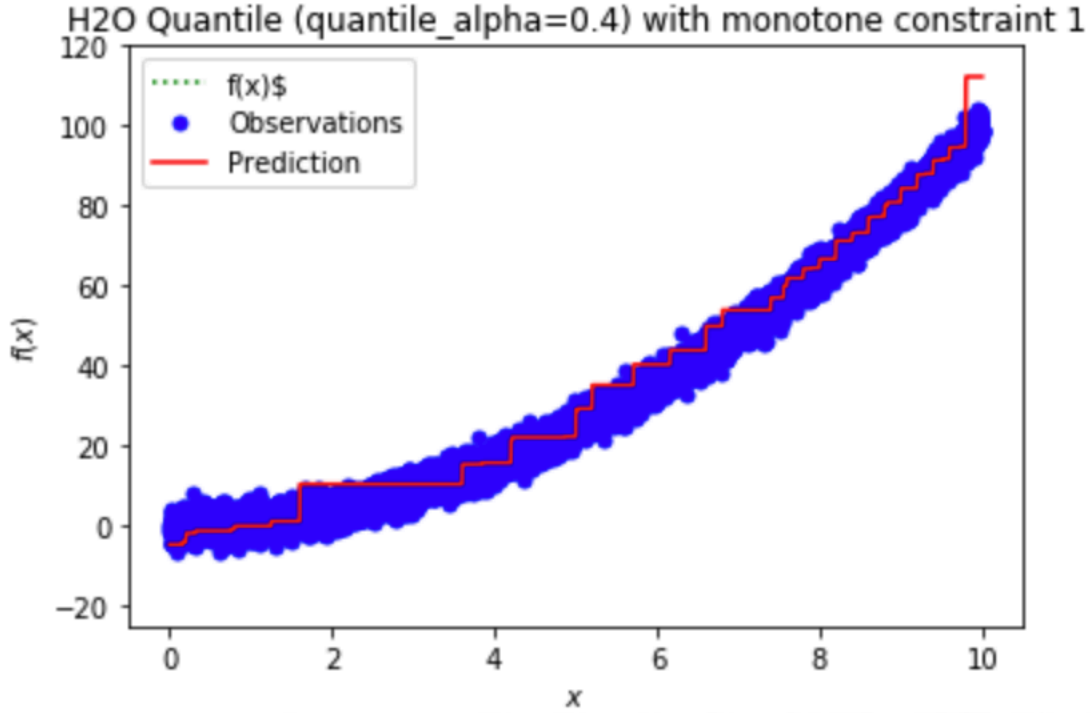
NA values go to the LEFT
```

```
from IPython.display import Image
Image(os.path.join(png_dir, "Tree0_Class0.png"))
```



Monotonic Constraints

- Ensure monotonic relationship of a given predictor with the response
  - might be mandated by business considerations
  - can result in better generalization and thus better model performance
- Supported for *gaussian*, *bernoulli*, *tweedie* and *quantile* distributions
- More computationally intensive
  - requires calculating additional information during tree splitting
  - we actually need to know how would the prediction look like for each tree node (including inner nodes)
  - trivial for gaussian, hard for quantile
- Constraints are validated on final trees
  - be careful when using other libraries
- Straightforward API




```
feature_names = ['MedInc', 'AveOccup', 'HouseAge']
monotone_constraints = {"MedInc": 1, "AveOccup": -1, "HouseAge": 1}

gbm_mono = H2OGradientBoostingEstimator(monotone_constraints=monotone_constraints, seed=42)
gbm_mono.train(x=feature_names, y="target", training_frame=train, validation_frame=test)
```

Discovering Feature Interactions

- built-in function `gbm_model.feature_interaction()`
  - based on https://github.com/Far0n/xgbfi
- provides insights into higher-order interactions between features in trees
- outputs leaf statistics and split value histograms
- H2O RuleFit can internally use GBM models

Interaction Depth 0:

		interaction	gain	fscore	wfscore	average_wfscore	average_gain
0		CAPSULE	82.175049	2.0	2.000000	1.000000	41.087524
1		DPROS	29.176030	5.0	1.052632	0.210526	5.835206
2		PSA	122.923945	15.0	6.647368	0.443158	8.194930

Interaction Depth 1:

		interaction	gain	fscore	wfscore	average_wfscore	average_gain
0		PSA|PSA	131.999369	9.0	3.371053	0.374561	14.666597
1		DPROS|PSA	82.044136	8.0	1.328947	0.166118	10.255517

PSA Split Value Histogram:

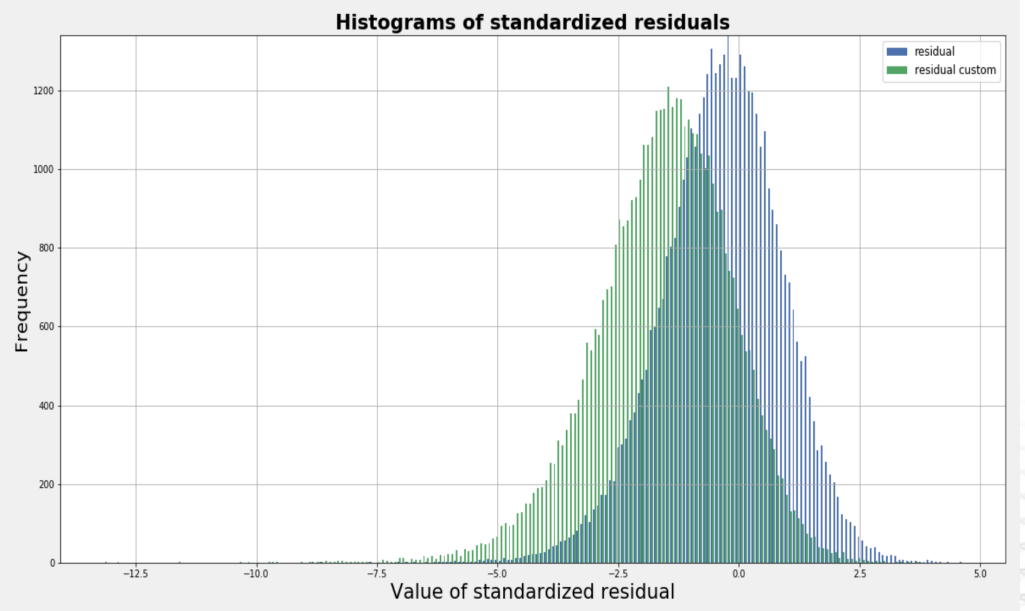
		split_value	count
0		0.500000	1
1		0.700000	1
2		0.800000	2
3		2.700000	1
4		10.600000	1
5		10.800000	1

Custom Evaluation and Loss Function

- User defined custom evaluation function is not specific to GBM and can be used for early stopping
- GBM also supports defining a custom loss function
  - defined as a custom distribution
- Different use cases
  - eg.: when risk/penalty of getting the prediction wrong is asymmetrical
- Only available in Python
  - Evaluated in Jython in the H2O Java backend
- Users need to implement interface defined by CustomDistributionGeneric
  - Convenience classes are provided to customize behavior of common distributions
    - CustomDistributionGaussian, CustomDistributionBernoulli, CustomDistributionMultinomial

```
class AsymmetricLossDistribution(CustomDistributionGaussian):

    def gradient(self, y, f):
        error = y - f
        # smaller predicted sales value is better error than the bigger one
        return 10 * error if error < 0 else 0.5 * error
```

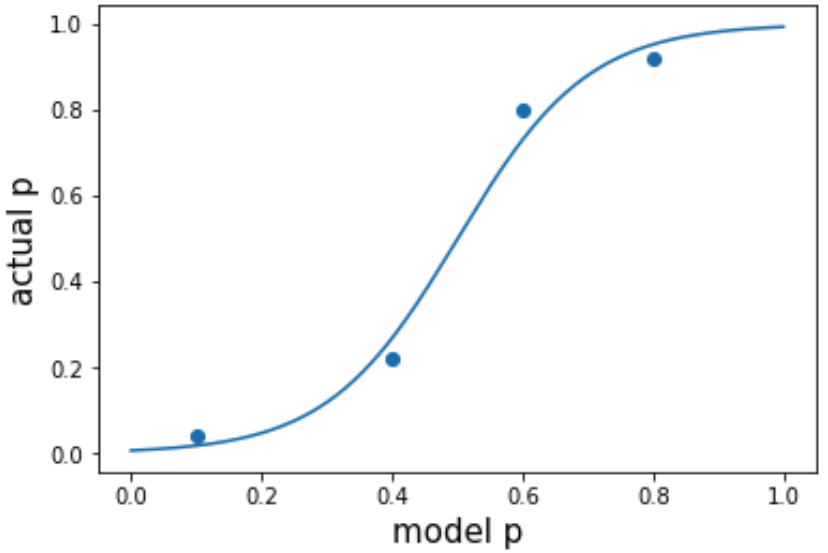


Model Calibration - Platt Scaling

- Many ML algorithms introduce biases when it comes to class probability

SVMs, GBMs { underpredict high-prob classes; overpredict low-prob classes

Naïve Bayes { overpredict high-prob classes; underpredict low-prob classes



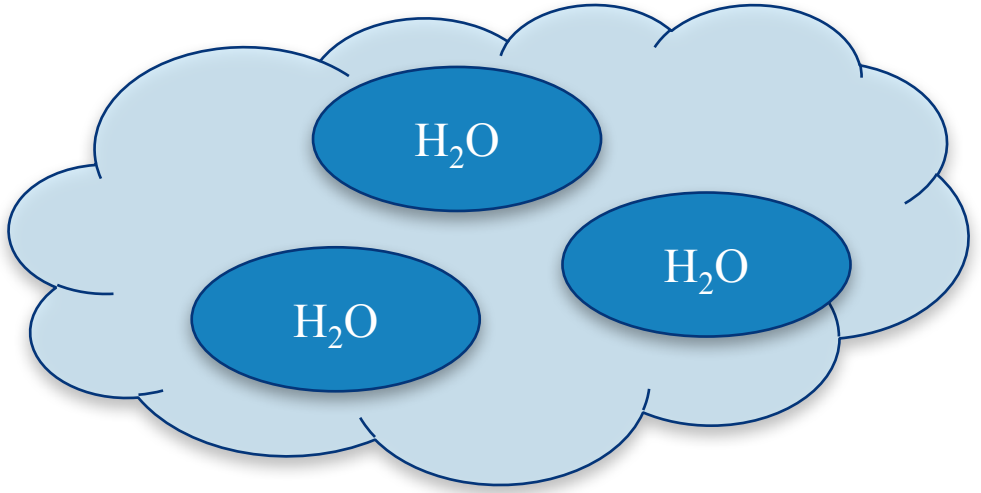
- Correct for this by fitting a sigmoid to the model output:

$$P(\mathbf{x}) = \frac{1}{1 + \exp(Af(\mathbf{x}) + B)}$$

- Split data into two frames:
  1) Training dataframe: used to generate f(x)
  2) Platt calibration frame: used to fit A and B

Productionalizing Models

Productionalizing Models



Immediate Action

- H2O Real-Time (RT) Scoring
  - POJO/MOJO + JRE
  - *Feature engineering not included*

Later Action

- H2O Off-Line (Batch) Scoring
  - H2O Cluster
  - *Feature engineering supported*

What is a H2O MOJO?

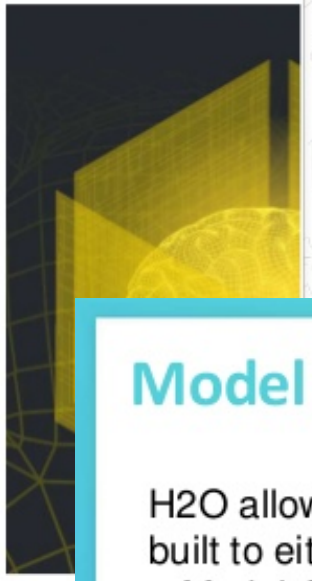
H2O MOJO

At large scale, new models are roughly **20-25 times smaller** in disk space

2-3 times faster during "hot" scoring, and 10-40 times faster in "cold" scoring compared to POJOs.

The efficiency gains are larger the bigger the size of the model.

iyzico



Model Deployment

H2O allows you to convert the models you have built to either a Plain Old Java Object (**POJO**) or a Model Object, Optimized (**MOJO**).

H2O-generated MOJO and POJO models are intended to be easily **embeddable in any Java environment**

iyzico

Deployment Ready Artifact - H2O-3

Packaged
- Model
- Binary Representation

Fast
- Low Latency Scoring

Portable
- Environment (Cloud or On-Prem)
- Runtime (Java)

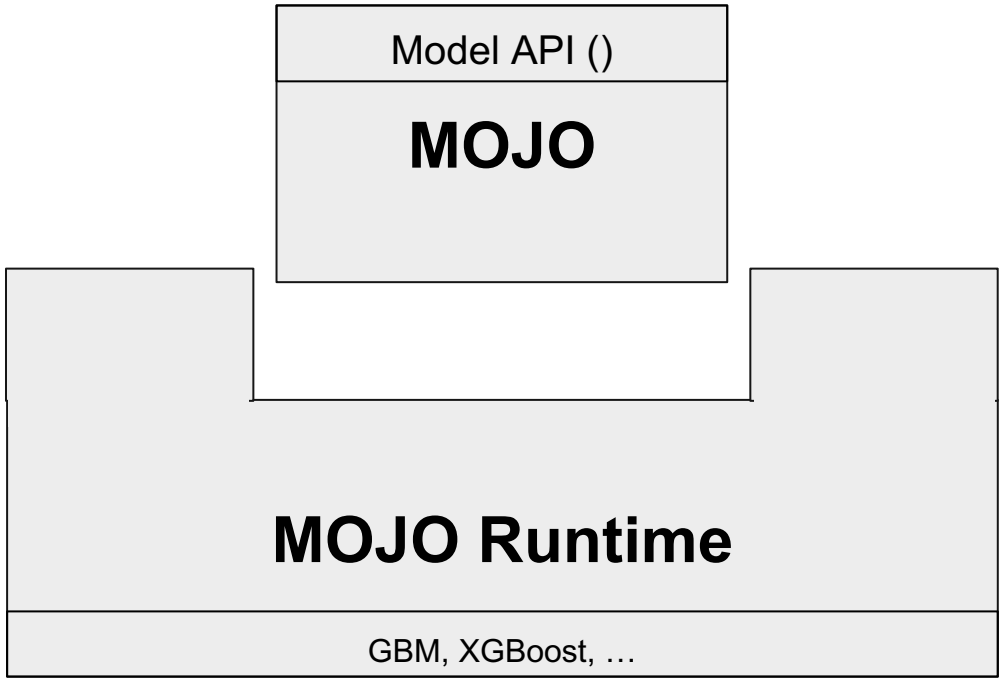
Flexible and Embeddable
- Batch (Application, Database, …)
- Realtime (REST, Streaming, …)

GBM Specifics

- C++ runtime, C++ and C# API
- ONNX convertor

H2O

Model **Ob**J**ect** **O**ptimized (MOJO)



Train Once Run Anywhere

POJO vs MOJO

H2O.ai

	Java POJO	MOJO / Java Runtime	MOJO / C++ Runtime
LANGUAGES	• Java	• Java	• C++, R, Python, .NET, other
ALGORITHMS	• All but GLRM	• GLM, DRF, GBM, GLRM (but eventually all)	• GBM (can add more if needed)
COMPILING	• Compiled Large POJO Compilation Errors	• Interpreted No compile errors	• Same as Java
CLASS LOADING	• Java class loading issues for many models — must tune MaxPermSize	• No Java class loading issues (very few fixed number of classes)	• (Does not apply)
ALLOCATION	• Models use PermGen storage	• Models stored in the Java heap	• Models stored in the C heap
PERFORMANCE	• Faster for small trees (about 1.5x-2x)	• Faster for large trees (about 20x)	• Same as Java
BACKWARD COMPATIBILITY	• POJO genmodel backward compatibility unsupported • Difficult to run old and new versions side-by-side (requires multiple library versions and classloading tricks)	• Strong MOJO genmodel backward compatibility Easy to run old and new versions side-by-side Same prediction results with future genmodels for a saved MOJO	• Same as Java
SIZE	• ~10x larger than MOJO, so ~10x larger than .zip file	• Size is 1/10th of a POJO	• Same as Java
EASY WRAPPER	• Works with Java EasyPredict API	• Works with Java EasyPredict API	• Works with C++ EasyPredict API
DECISION VISIBILITY	• Decisions are visible in generated code	• Opaque (but open source) binary format Visualization tools provided (e.g. tree viewer)	• Same as Java

Python Syntax H2O Scoring


```
# Save Real-Time Scoring Code
model.download_pojo(path = u'', get_genmodel_jar = False, genmodel_name = u'')

model.download_mojo(path = u'', get_genmodel_jar = False, genmodel_name = u'')
```

```
# Save / Load Batch Scoring Model
h2o.save_model(model, path = u'', force = False)
model = h2o.load_model(path)

# Batch Scoring
model.predict(test_data, custom_metric = None, custom_metric_func = None)
```