H2O Automatic Machine Learning & Beyond
Using R & Python

How Many of these Task Can You Perform?

- H2O Automatic Machine Learning
- H2O Real-Time Scoring
- H2O Feature Engineering
  - Log Transformation
  - Winsorization
  - Numeric Binning
  - Categorical Binning
  - Mean Target Encoding
  - K-Means Clustering
  - Principal Components
  - Generalized Low-Rank Models

Online Resources

- H2O Tutorials and Training Material
  - https://github.com/h2oai/h2o-tutorials
  - https://github.com/h2oai/h2o-tutorials/tree/master/training/lending_club_exercise
  - https://github.com/h2oai/app-consumer-loan
- Additional Datasets
  - https://s3.amazonaws.com/h2o-public-test-data/bigdata/laptop

H2O Training

1. Introduction to ML and H2O
2. H2O Auto Machine Learning
3. Productionalizing Models
4. Machine Learning Concepts
5. H2O Feature Engineering
6. H2O Algorithms

Introduction:

INTRODUCTION TO MACHINE LEARNING

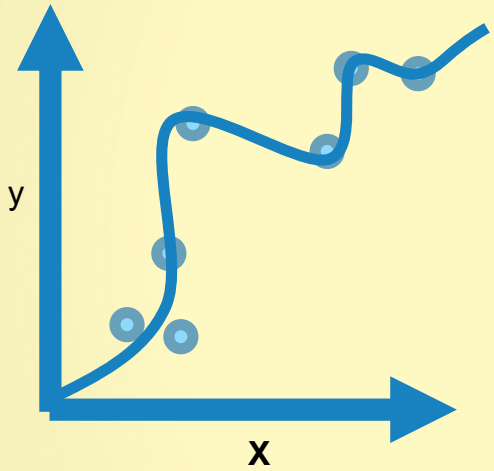
What is Machine Learning?

- Set of statistical and optimization tools to model data
  - Supervised Learning
  - Unsupervised Learning
- Focus on “production analytics”
  - Reducing need for
    - Sampling
    - Periodic revision of models
    - Subjective priors
    - Hand coding models in a production language

Supervised Learning

Regression:

How much will a customers spend?

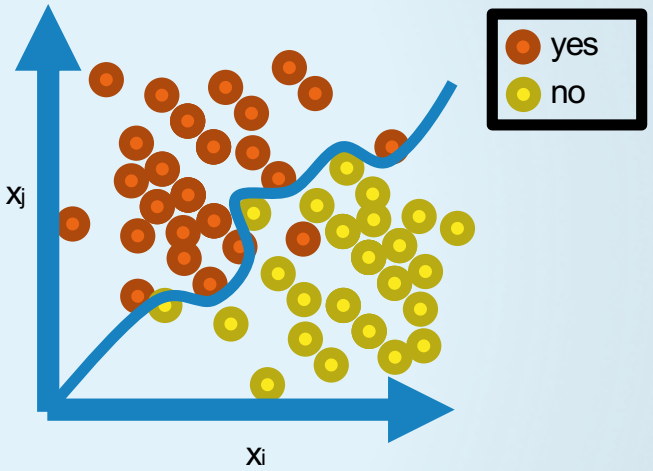


H2O algos:

Penalized Linear Models
Random Forest
Gradient Boosting
Neural Networks
Stacked Ensembles

Classification:

Will a customer make a purchase? Yes or No



H2O algos:

Penalized Linear Models
Naïve Bayes
Random Forest
Gradient Boosting
Neural Networks
Stacked Ensembles

Stacked Ensembles

$$\left[y \right] \approx \left[\hat{y}_l \right] = f_l \left(\left[x_1 \dots x_p \right] \right) \qquad \left[y \right] \approx g \left(\left[\hat{y}_1 \dots \hat{y}_L \right] \right)$$

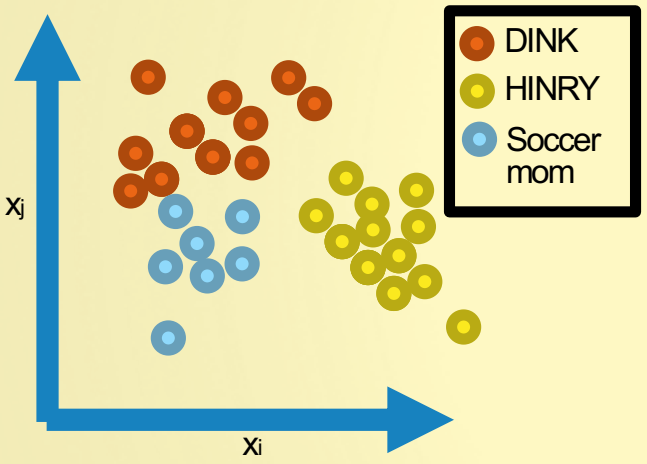
- Engineer original feature set
- Train L models using strong learners (GLM, RF, GBM, …)
- Control overfitting using k-fold CV

- Stack these predicted values to form new feature set
- Train metalearner on stacked predictions

Unsupervised Learning

Clustering:

Grouping rows – e.g. creating groups of similar customers

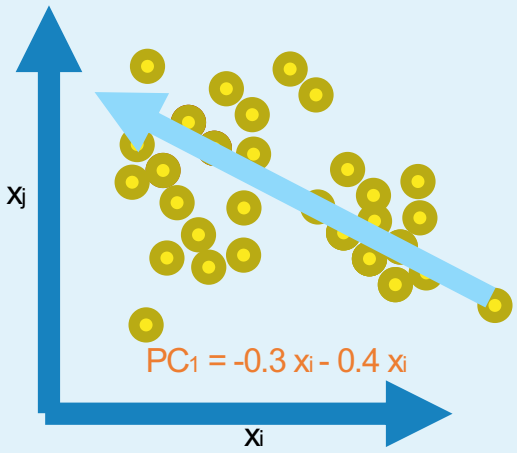


H2O algos:

k – means

Feature extraction:

Grouping columns – Create a small number of new representative dimensions

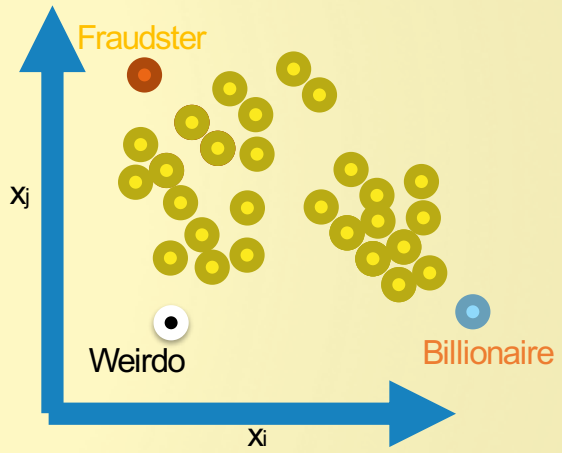


H2O algos:

Principal components
Generalized low rank models
Autoencoders
Word2Vec

Anomaly detection:

Detecting outlying rows - Finding high-value, fraudulent, or weird customers



H2O algos:

Principal components
Generalized low rank models
Autoencoders

Vocabulary

Concept	Statistics\Econometrics	Machine Learning
“Computation”	Fit\Estimate	Train
“Left-hand side”	Dependent variable	Target
“Right-hand side”	Regressor\Predictor\Class	Feature\Factor\Enum
“Goal”	Estimation\Explanation	Prediction

Machine Learning Methods

Supervised Learning Methods

- Regression (GLM)
  - Lasso
  - Ridge
  - E
- De
- Ra
- Gradient Boosted Models
- Support Vector Machine
- Neural Network
- Deep Learning

Know Y

Unsupervised Learning Methods

- Clustering
  - Kmeans
  - Hierarchical
- Princ
- Autc
- Non-
- Generalized Low Rank Models

Don’t know Y

When do I use what tool?

Introduction:

INTRODUCTION TO H2O

H2O Machine Learning Methods

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

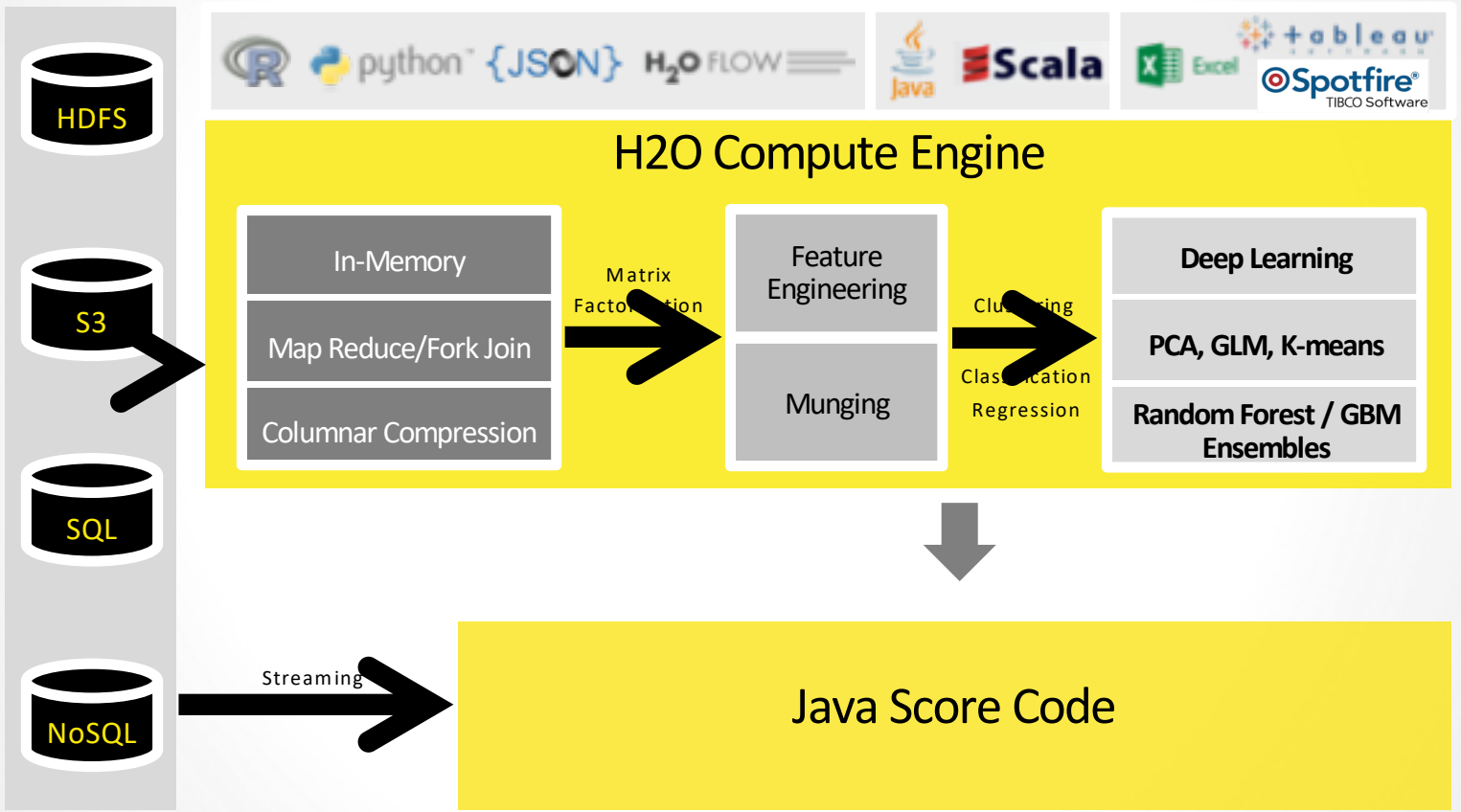
- **Deep neural networks**: Multi-layer feed-forward neural networks for standard data mining tasks
- **Convolutional neural networks:** Sophisticated architectures for pattern recognition in images, sound, and text

Anomaly Detection

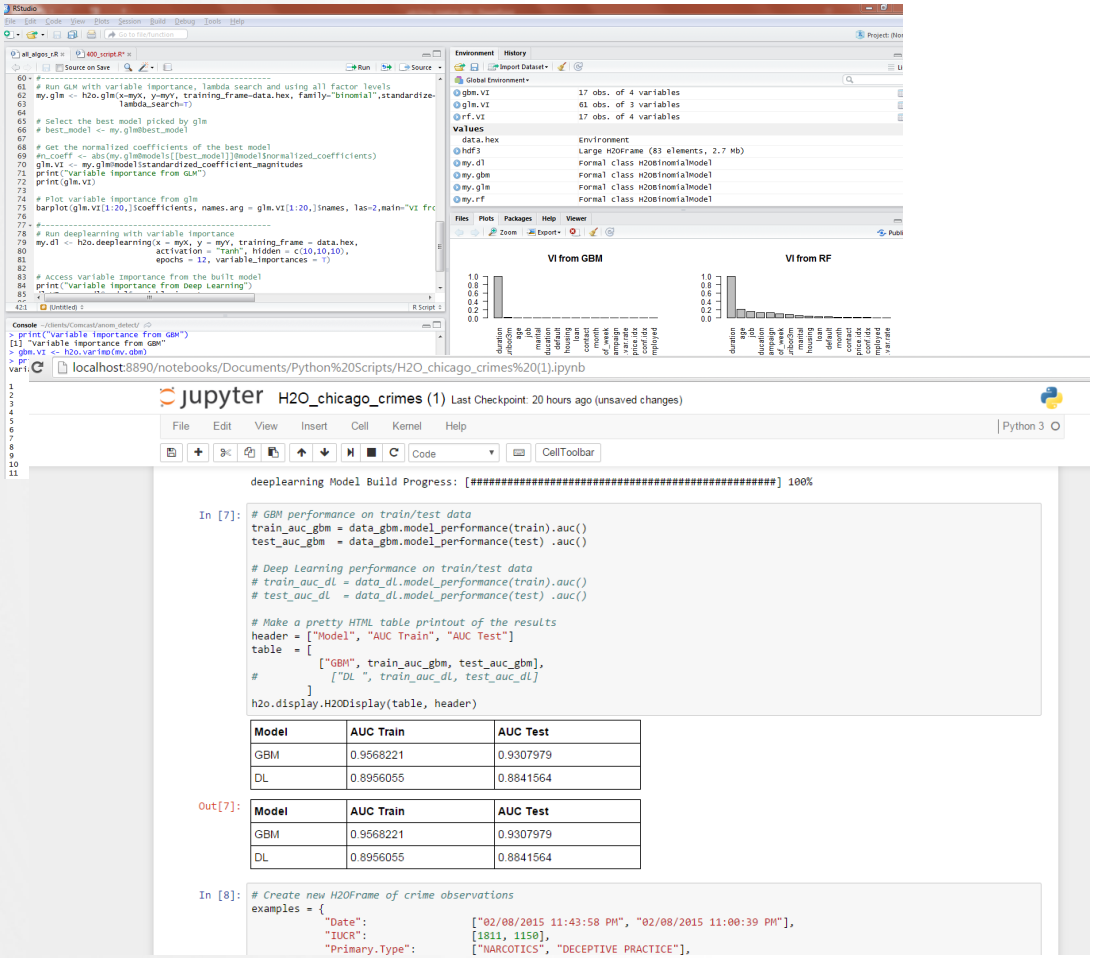
Term Embeddings

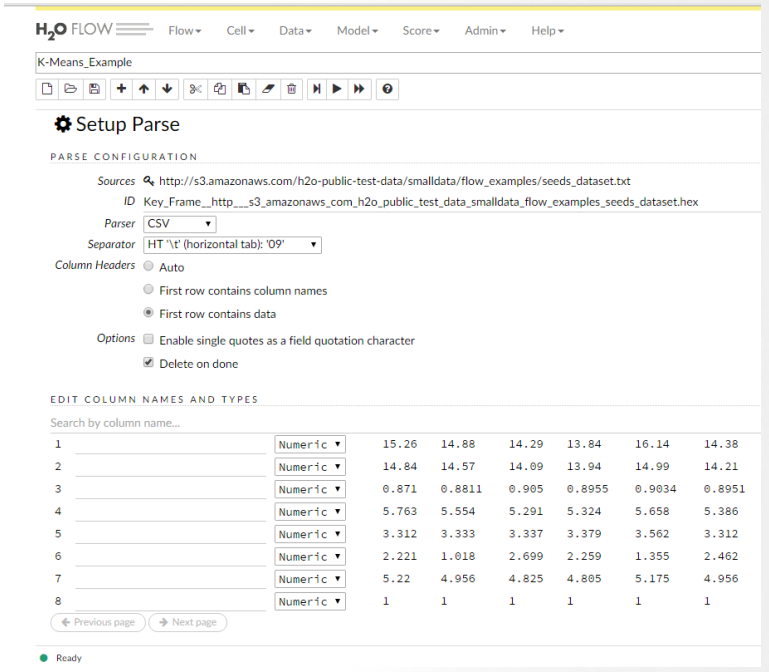
- **Autoencoders**: Find outliers using a nonlinear dimensionality reduction technique
- **Word2vec:** Generate context-sensitive numerical representations of a large text corpus

Data and Client Agnostic



R, Python and Flow





R Interface Overview

Action	R	H2O
Reading data	`read_csv(data_path)`	`h2o.importFile(data_path)`
Summarizing data	`summary(data_frame)`	`h2o.summary(h2o_frame)`
Summary statistics	`mean(data_frame[["x"]])`	`h2o.mean(h2o_frame)`
Combining rows	`rbind(data_frame1, data_frame2)`	`h2o.rbind(h2o_frame1, h2o_frame2)`
Combining columns	`cbind(data_frame1, data_frame2)`	`h2o.cbind(h2o_frame1, h2o_frame2)`
Data selection	`data_frame[,]`	`h2o_frame[,]`
Transforming columns	`log(data_frame[,"x"])` `sqrt(data_frame[,"x"])`	`log(h2o_frame[,"x"])` `sqrt(h2o_frame[,"x"])`
Building Random Forest	`model = randomForest(y ~ x, data_frame)`	`model = h2o.randomForest(x, y, train_frame)`
Model Prediction	`predict(model, data_frame)`	`h2o.predict(model, h2o_frame)`
Model Metrics	`performance(model)` `auc(model)`	`metrics = model.model_performance(frame)` `h2o.auc(model)`

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()`

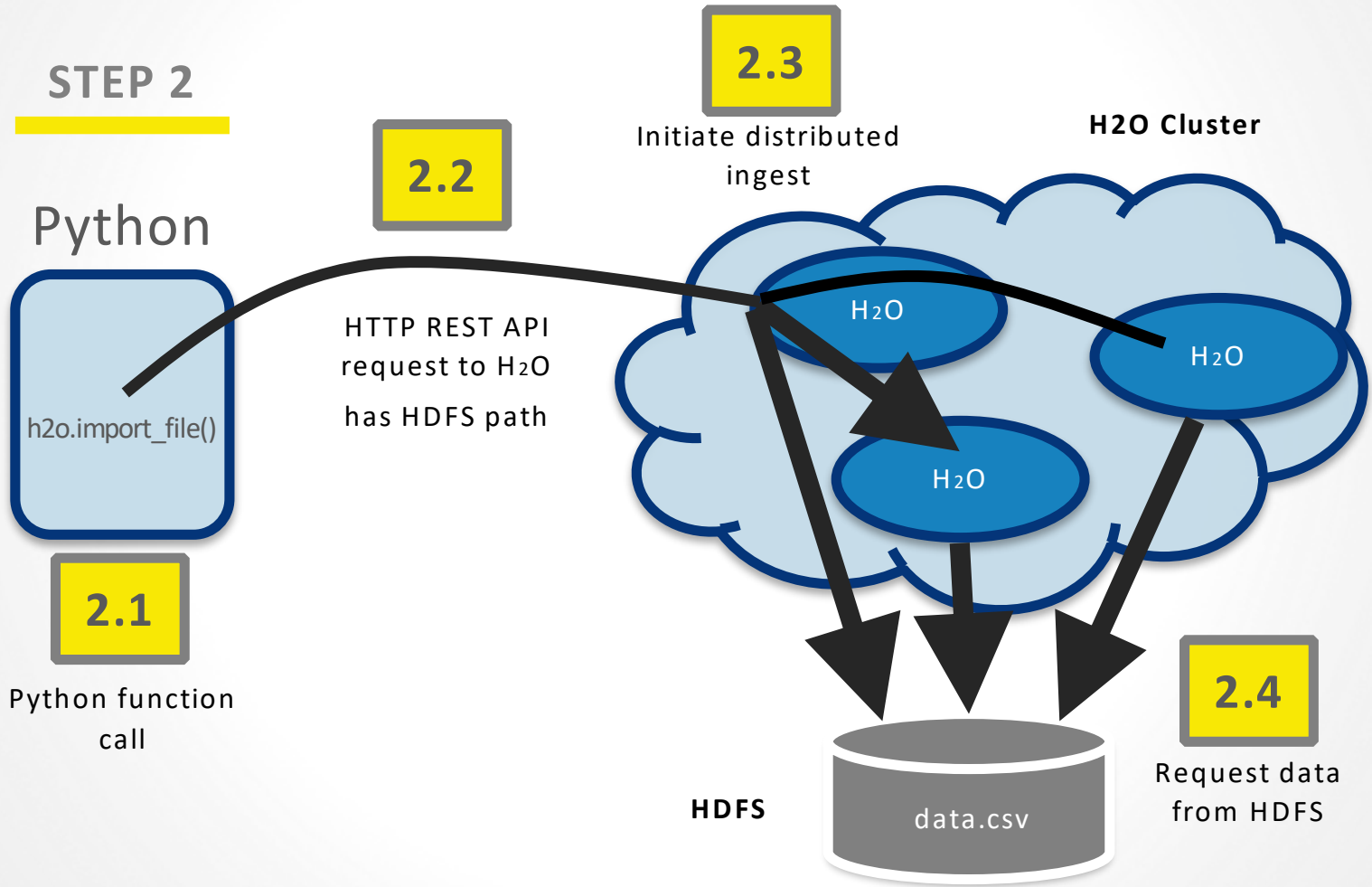
Reading Data into H2O with Python

STEP 1

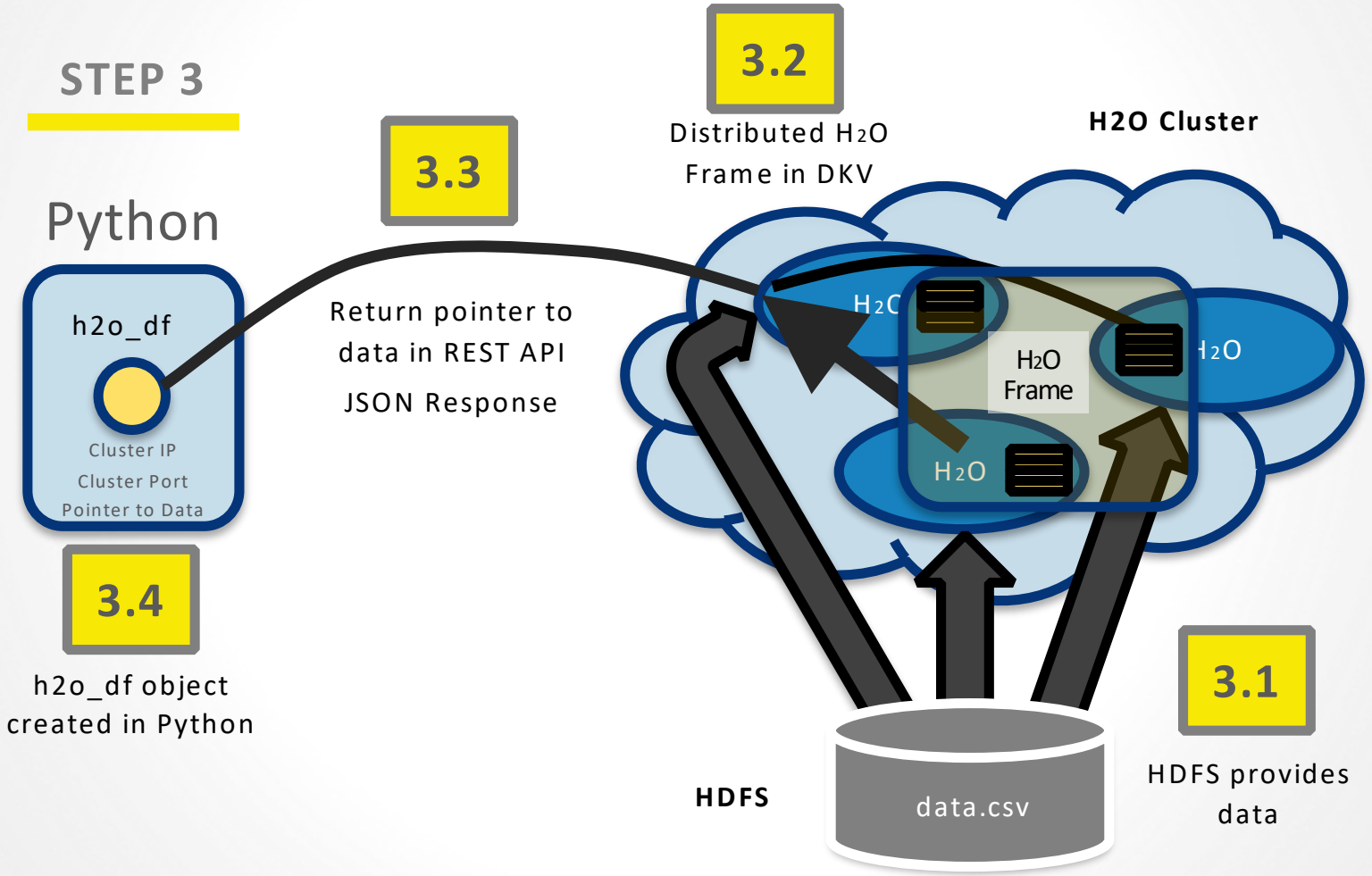
Python user

```
h2o_df = h2o.import_file("../data/allyears2k.csv")
```

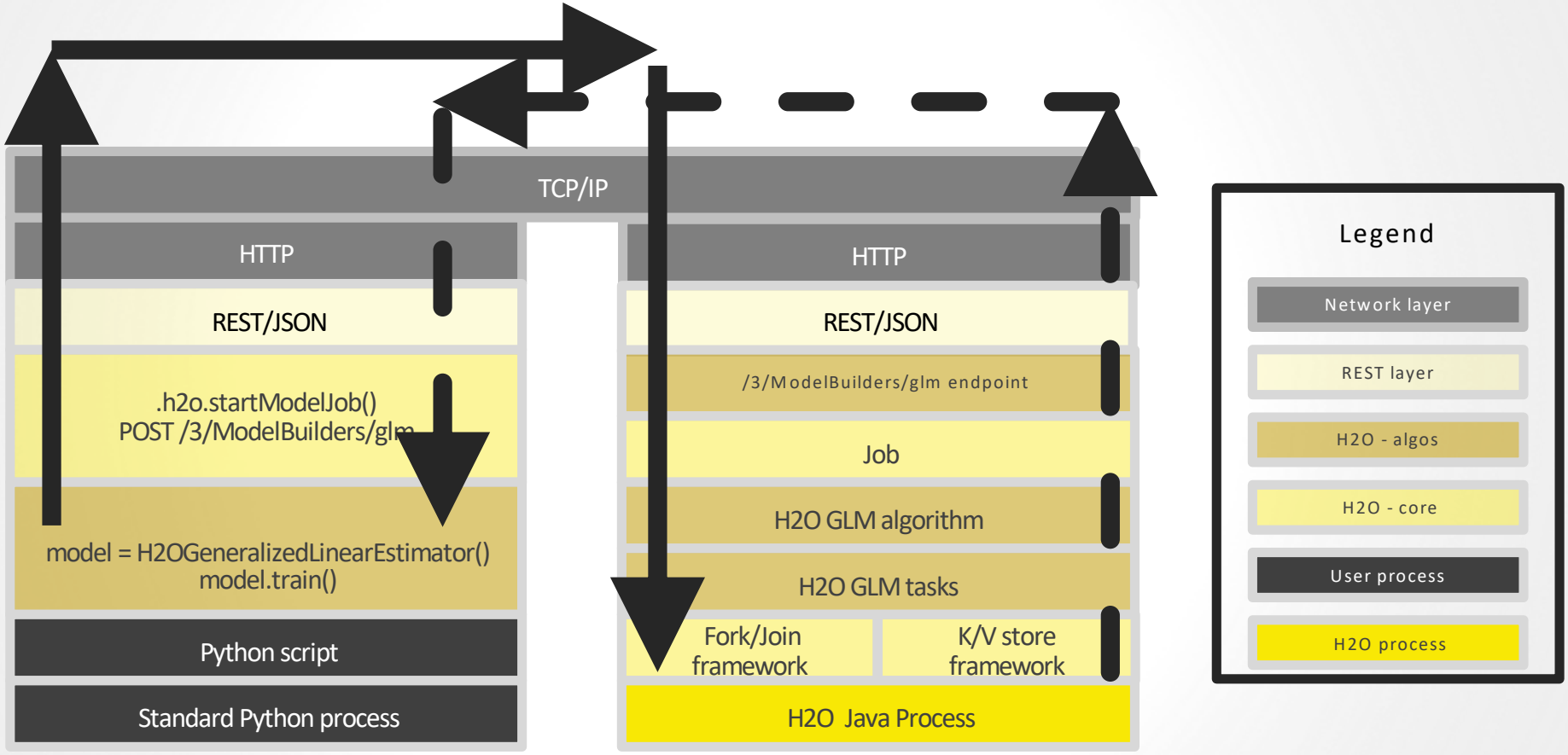
Reading Data into H2O with Python



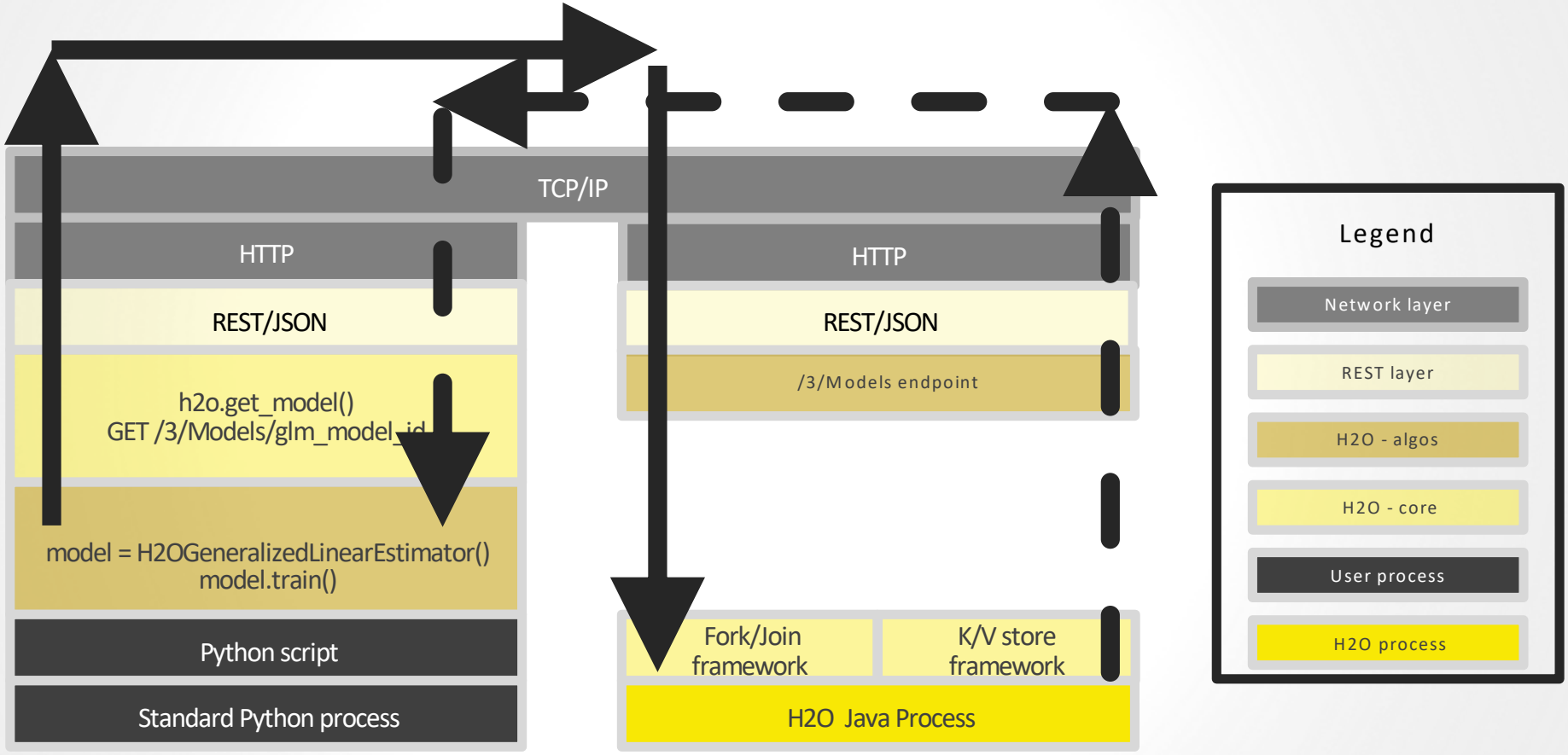
Reading Data into H2O with Python



Training GLM from Python



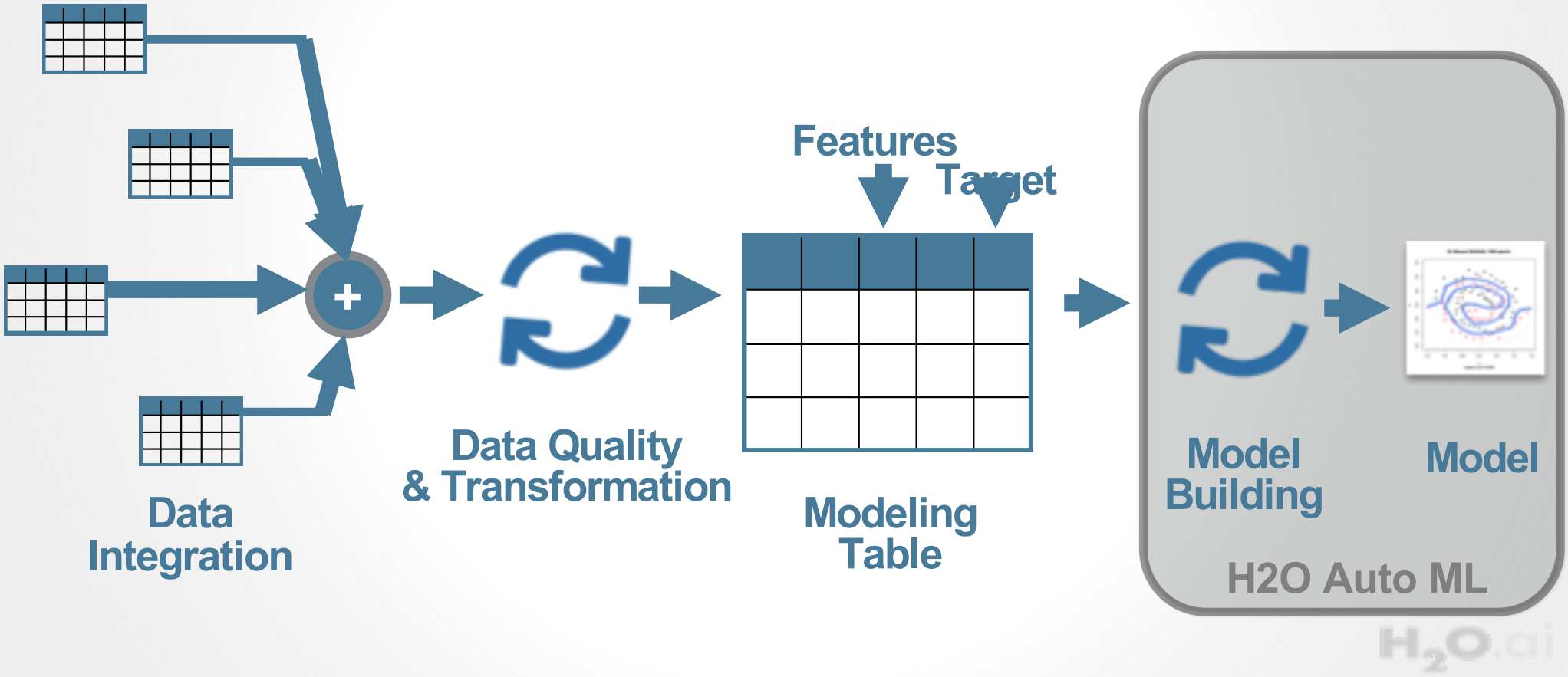
Retrieving GLM Result from Python



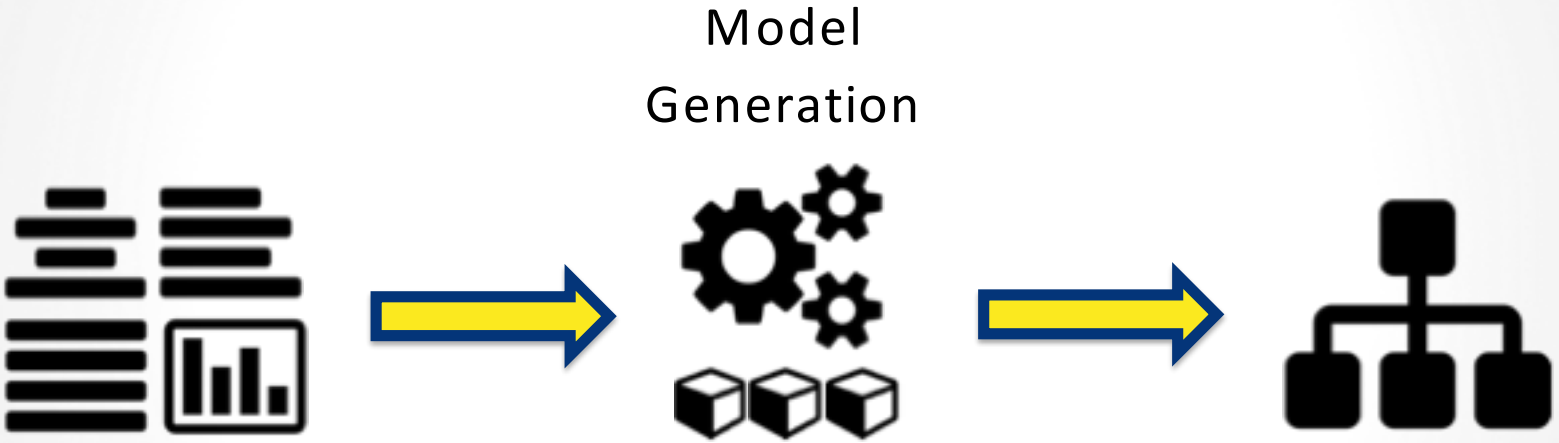
H2O Training

1. Introduction to ML and H2O
2. H2O Auto Machine Learning
3. Productionalizing Models
4. Machine Learning Concepts
5. H2O Feature Engineering
6. H2O Algorithms

H2O Auto ML in Typical Enterprise Workflow



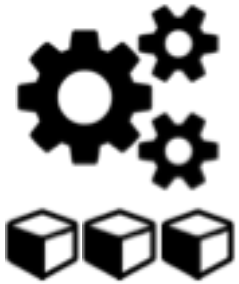
H2O Auto ML Pipeline



H2O AutoML Light Data Preparation

- Impute missing data
- Standardize numeric features
- One-hot encode categorical features

H2O AutoML Model Generation



- Explore multiple algorithms
  - GLMs, Random Forests, GBMs, Deep Neural Networks
- Perform random grid search over hyper-parameter space
- Use early stopping rules for models and grids

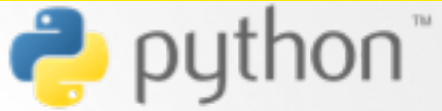
H2O AutoML Stacked Ensemble Creation

- Stacked ensembles of all models
- Stacked ensembles of best model for each algorithm

H2OAutoML R Syntax

```
h2o.automl(x, y,
           training_frame,
           validation_frame = NULL,
           leaderboard_frame = NULL,
           nfolds = 5,
           fold_column = NULL,
           weights_column = NULL,
           max_runtime_secs = 3600,
           max_models = NULL,
           stopping_metric = c("AUTO", "deviance", "logloss", "MSE",
             "RMSE", "MAE", "RMSLE", "AUC", "lift_top_group",
             "misclassification", "mean_per_class_error"),
           stopping_tolerance = NULL,
           stopping_rounds = 3,
           seed = NULL,
           project_name = NULL,
           exclude_algos = NULL)
```

H2OAutoML Python Syntax




```
# Set up automatic machine learning experiment
aml = H2OAutoML(nfolds = 5, max_runtime_secs = 3600, max_models = None,
                stopping_metric = 'AUTO', stopping_tolerance = None,
                stopping_rounds = 3, seed = None, project_name = None,
                exclude_algos = None)
```

```
# Train models
aml.train(x = None, y = None, training_frame = None, fold_column = None,
          weights_column = None, validation_frame = None,
          leaderboard_frame = None)
```

Case Study: Lending Club Dataset

- Loan data from 2007 up until 2015 including rejected applications and accepted applications.
- Of the 500k accepted applicants about 160k loans have either been completely paid off or defaulted.
- There are about 4 million applicants in the rejected loans dataset.
- **Use Case 1**: Predict the likelihood of a user defaulting based on the information supplied when applying for a loan.
- **Use Case 2**: Determine the interest rate Lending Club would have offered the user based on the information supplied when applying for a loan.
- Full Data: https://www.kaggle.com/wendykan/lending-club-loan-data
- H2O Subset: https://s3.amazonaws.com/h2o-public-test-data/bigdata/laptop/lending-club/loan.csv

Case Study: Lending Club Dataset

	Column Name	Description	Unit	Role
1	loan_amnt	Requested loan amount	US dollars	Predictor
2	term	Longest term length	Months	Predictor
	int_term	**Recommended interest rate**	**Rate**	**Response**
4	emp_length	Employment length	Years	Predictor
5	home_ownership	Housing status	Categorical	Predictor
6	annual_inc	Annual income	US dollars	Predictor
7	purpose	Purpose for the loan	Categorical	Predictor
8	addr_state	State of residence	Categorical	Predictor
9	dti	Debt to income ratio	Percent	Predictor
10	delinq_2yrs	Number of delinquencies in the past 2 years	Count	Predictor
11	revol_util	Revolving credit line utilized	Percent	Predictor
12	total_acc	Number of active accounts	Count	Predictor
13	**bad_loan**	**Bad loan indicator**	**Boolean**	**Response**
14	longest_credit_length	Age of oldest active account	Years	Predictor
15	verification_status	Income verification status	Boolean	Predictor

Case Study: Auto ML of Lending Club Dataset

```r
# Load package and connect to cluster
library(h2o)
h2o.init(max_mem_size = "6g")

# Import data and manage data types
train_path <- "https://raw.githubusercontent.com/h2oai/app-consumer-loan/master/data/loan.csv"
train <- h2o.importFile(train_path, destination_frame = "loan_train")
train["bad_loan"] = h2o.asfactor(train["bad_loan"])

# Set target and predictor variables
y <- "bad_loan"
x <- h2o.colnames(train)
x <- setdiff(x, c(y, "int_rate"))

# Use Auto ML to train models
aml <- h2o.automl(x = x, y = y, training_frame = train, max_runtime_secs = 300)
```

Case Study: Auto ML of Lending Club Dataset

```python
# Load package and connect to cluster
import h2o
h2o.init(max_mem_size = "6g")
```

```python
# Import data and manage data types
train_path = "https://raw.githubusercontent.com/h2oai/app-consumer-loan/master/data/loan.csv"
train = h2o.import_file(train_path, destination_frame = "loan_train")
train["bad_loan"] = train["bad_loan"].asfactor()
```

```python
# Set target and predictor variables
y = "bad_loan"
x = train.col_names
x.remove(y)
x.remove("int_rate")
```

```python
# Use Auto ML to train models
from h2o.automl import H2OAutoML
aml = H2OAutoML(max_runtime_secs = 300)
aml.train(x = x, y = y, training_frame = train)
```

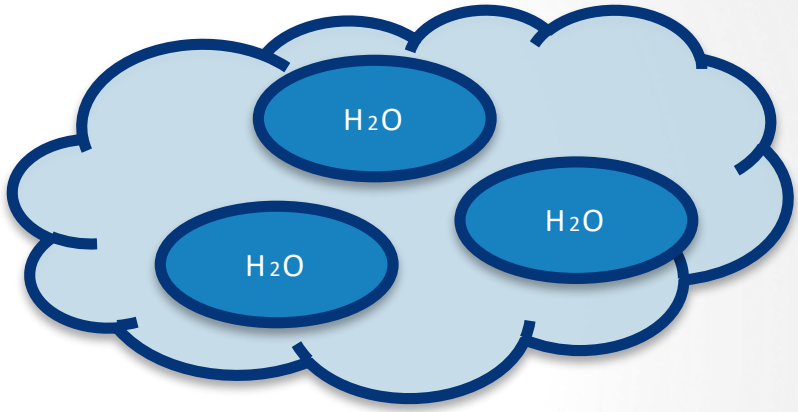
H2O Training

1. Introduction to ML and H2O
2. H2O Auto Machine Learning
3. Productionalizing Models
4. Machine Learning Concepts
5. H2O Feature Engineering
6. H2O Algorithms

Operationalizing Machine Learning:

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*

R Syntax H2O Scoring

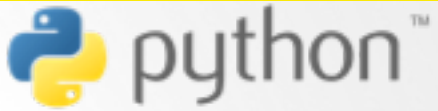
```
# Save Real-Time Scoring Code
h2o.download_pojo(model, path = NULL, get_jar = TRUE, jar_name = "")

h2o.download_mojo(model, path = getwd(), get_genmodel_jar = FALSE,
                  genmodel_name = "")
```

```
# Save / Load Batch Scoring Model
h2o.saveModel(object, path = "", force = FALSE)
model <- h2o.loadModel(path)

# Batch Scoring
h2o.predict(model, newdata, ...)
```

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)
```

Java Syntax H2O Real-Time Scoring

```
import java.io.*;
import hex.genmodel.easy.RowData;
import hex.genmodel.easy.EasyPredictModelWrapper;
import hex.genmodel.easy.prediction.*;

public class main {
  private static String modelClassName = "model_pojo";
  public static void main(String[] args) throws Exception {
    hex.genmodel.GenModel rawModel;
    rawModel = (hex.genmodel.GenModel)
                  Class.forName(modelClassName).newInstance();
    EasyPredictModelWrapper model = new EasyPredictModelWrapper(rawModel);
    RowData row = new RowData();
    BinomialModelPrediction p = model.predictBinomial(row);
  }
}
```

H2O Generated POJO Model WebApp Example

https://github.com/h2oai/app-consumer-loan

H2O generated POJO model WebApp Example

This example shows a generated Java POJO being called using a REST API from a JavaScript Web application.

The application simulates the experience of a consumer applying for a loan. The consumer provides some information about themselves and is either offered a loan or denied.

H2O World 2015 Presentation

The "Building a Smarter Application" presentation given at H2O World 2015 references this repo.

- https://github.com/h2oai/h2o-world-2015-training/tree/master/tutorials/building-a-smarter-application

Pieces at work

Processes

(Front-end)

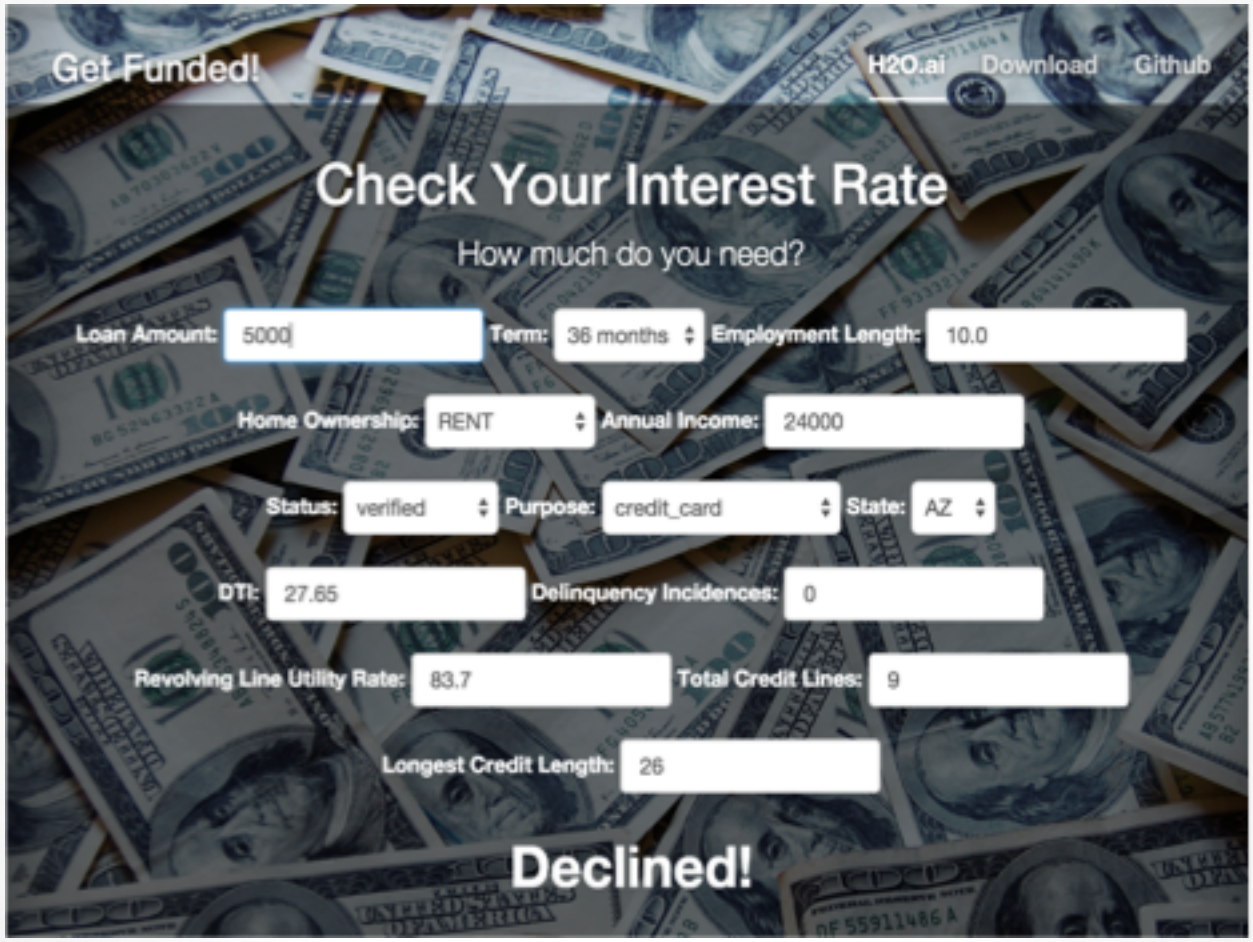
1. Web browser

(Back-end)

1. Jetty servlet container

> Note: Not to be confused with the H2O embedded web port (default 54321) which is also powered by Jetty.

Predict Loan Defaults



Example Model Output

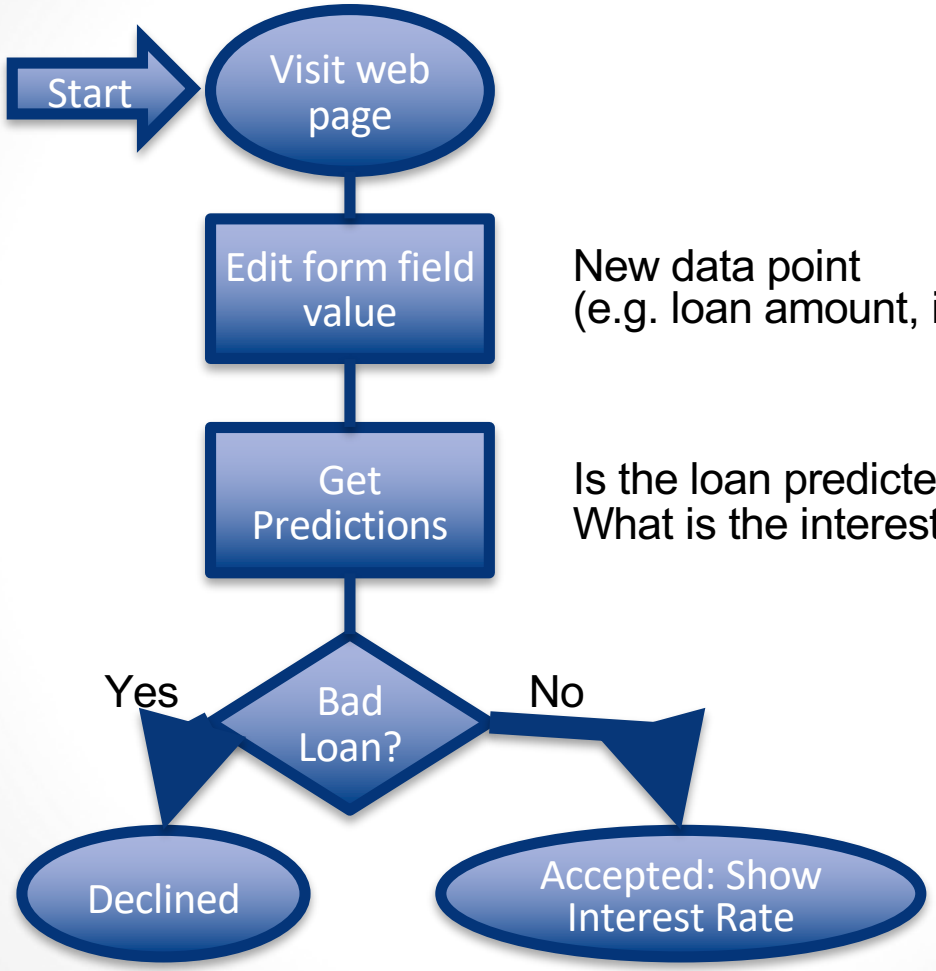
Bad Loan Model

Algorithm:	GBM
Model category:	Binary
	Classification
ntrees:	100
max_depth:	5
learn_rate:	0.05
AUC on valid:	.685
max F1:	0.202

Interest Rate Model

Algorithm:	GBM
Model category:	Regression
ntrees:	100
max_depth:	5
learn_rate:	0.05
MSE:	11.1
R2:	0.424

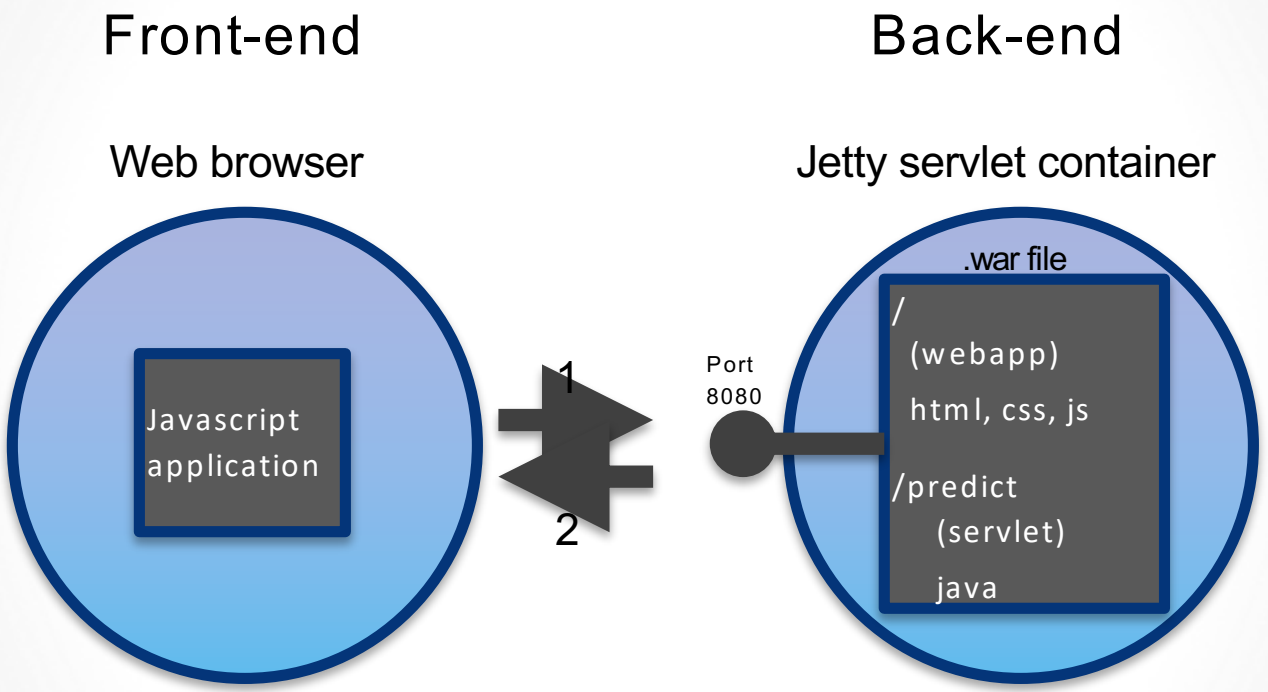
Workflow For This App



New data point
(e.g. loan amount, income)

Is the loan predicted to be bad?
What is the interest rate?

App Architecture Diagram



1. HTTP GET with query parameters (loan_amt, annual_inc, etc.)
2. JSON response with predictions

Software Pieces

- Offline
  - R/Python + H2O (model building)
- Online
  - Front-end
    - Web browser
    - JavaScript application (run in the browser)
  - Back-end
    - Jetty servlet container
    - H2O-generated model POJO (hosted by servlet container)

H2O Training

1. Introduction to ML and H2O
2. H2O Auto Machine Learning
3. Productionalizing Models
4. Machine Learning Concepts
5. H2O Feature Engineering
6. H2O Algorithms

Machine Learning Concepts

1. Out of Sample Data
2. Data Leakage
3. Target Class Imbalance
4. Extreme Values & Outliers
5. Low Frequency Categories
6. Redundant Data
7. Irrelevant Data
8. Missing Data
9. Model Scoring Properties
10. Model Interpretability

Training and Test Data Sets

Training Set vs. Test Set

- Partition the original data (randomly or stratified) into a **training** set and a **test** set. (e.g. 70/30)

Training Error vs. Test Error

- It can be useful to evaluate the training error, but you should not look at training error alone.
- Training error is not an estimate of **generalization error** (on a test set or cross-validated), which is what you should care more about.
- Training error vs test error over time is an useful thing to calculate. It can tell you when you start to overfit your model, so it is a useful metric in supervised machine learning.

Performance Metrics

- Regression: R^2, MSE, RMSE
- Classification: Accuracy, F1, H-measure, Log-loss
- Ranking (Binary Outcome): AUC, Partial AUC

Training and Test Data Sets

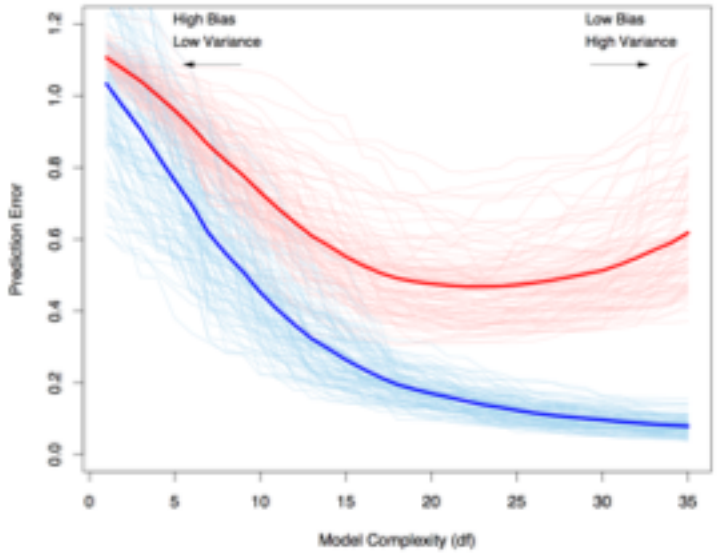


FIGURE 7.1. *Behavior of test sample and training sample error as the model complexity is varied. The light blue curves show the training error* $\overline{\text{err}}$*, while the light red curves show the conditional test error* $\text{Err}_{\mathcal{T}}$ *for 100 training sets of size 50 each, as the model complexity is increased. The solid curves show the expected test error* Err *and the expected training error* $\text{E}[\overline{\text{err}}]$.

Source: Elements of Statistical Learning

Option 1: Separate Validation Set

Training Set vs. Validation Set vs. Test Set

- If you have "enough" data and plan to do some model tuning, you should really partition your data into three parts — Training, Validation and Test sets.
- There is **no general rule** for how you should partition the data and it will depend on how strong the signal in your data is, but an example could be: 50% Train, 25% Validation and 25% Test.

Validation is for Model Tuning

- The validation set is used **strictly for model tuning** (via validation of models with different parameters) and the test set is used to make a final estimate of the generalization error.

Option 2: Cross-Validation

Training Set vs. Test Set

- Partition the original data (randomly) into a training set and a test set. (e.g. 70/30)
- Train a model using the training set and evaluate performance (a single time) on the test set.

K-fold Cross-validation

- Train and test K models using separate folds.
- Average the model performance over the K test sets.
- Report cross-validated metrics.

Validation & Cross-Validation H2O Parameters

Choose one of two approaches:

1. Validation
   - **`validation_frame`**: id of the validation data frame.
2. Cross-Validation
   - **`nfolds`**: number of folds for k-fold cross-validation
   - **`fold_column`**: column with cross-validation fold index assignment per observation.
   - **`fold_assignment`**: cross-validation fold assignment scheme, if fold column is not specified.
   - **`keep_cross_validation_fold_assignment`**: keep the cross-validation fold assignment.
   - **`keep_cross_validation_predictions`**: keep the predictions of the cross-validation models.

Data Leakage

What Is It

- Leakage is allowing your model to use information that will not be available in a production setting.
- Example: using the Dow Jones daily gain/loss as part of a model that predicts stock performance.

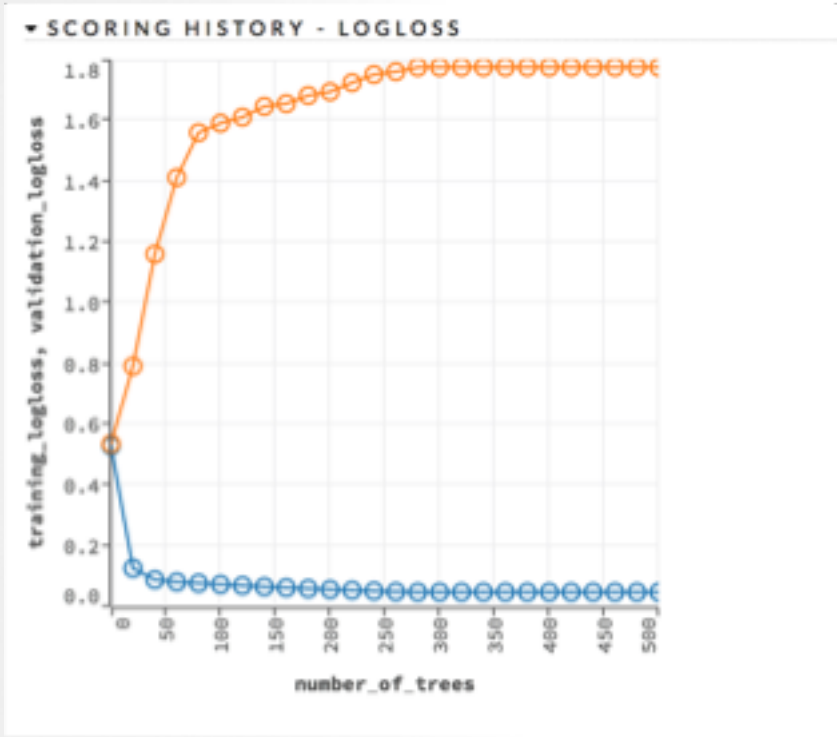
What Happens

- Model is overfit.
- Predictions will be inconsistent with those scored during model training (even with a validation set).
- Insights derived from the model will be incorrect.

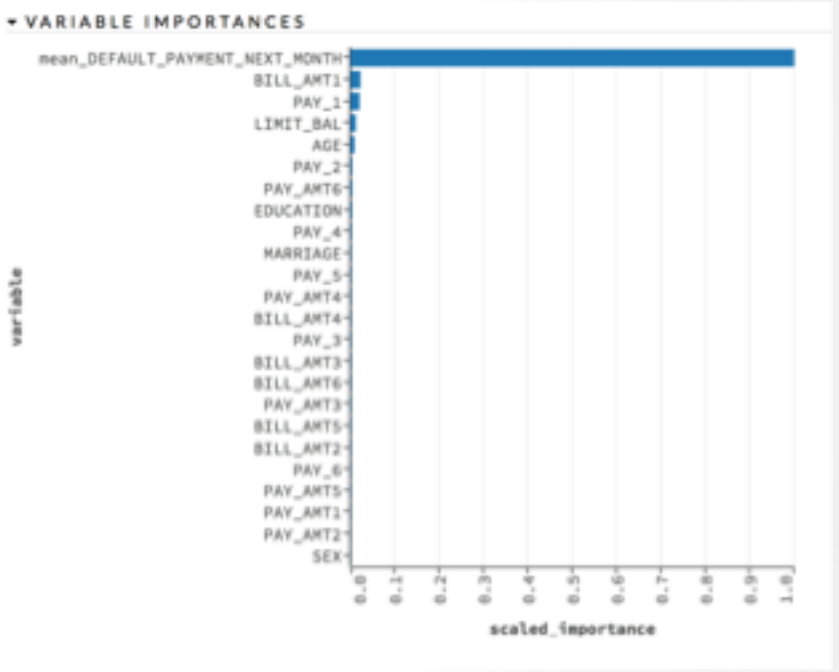
What to Do

- Understand the nature of your problem and data.
- Scrutinize model feedback, such as relative influence or linear coefficient.

Data Leakage



Scoring History: Training vs Testing



Data Leakage Feature is the only important feature

Target Class Imbalance

Imbalanced Response Variable

- A dataset is said to be **imbalanced** when categorical responses occur at widely varying rates.
- Standard optimizations by machine learning algorithms may favor majority classes.
- Rule of thumb for binary response: If the minority class makes < 10% of the data, this can cause issues.

Common Examples Across Industries

- Advertising — Probability that someone clicks on ad is very low… very very low.
- Healthcare & Medicine — Certain diseases or adverse medical conditions are rare.
- Fraud Detection — Insurance or credit fraud is rare.

Target Class Imbalance

Artificial Balance

- You can **balance** the training set using sampling.

Potential Pitfalls

- Don't balance the test set! The test set should represent the true data distribution.
- The same goes for a hold-out validation set and cross-validation sets.
- Cross-validation will probably require custom coding.

Solutions

- H2O has a `balance_classes` argument that can be used to do this properly & automatically.
- You can manually **up-sample** (or **down-sample**) your minority (or majority) class(es) set either by duplicating (or sub-sampling) rows, or by using row weights.

Target Class Imbalance H2O Parameters

- **`balance_classes`**: balance training data class counts via over/under-sampling.
- **`class_sampling_factors`**: desired over/under-sampling ratios per class (in lexicographic order). If not specified, sampling factors will be automatically.
- **`max_after_balance_size`**: maximum relative size of the training data after balancing class counts.
- **`sample_rate_per_class`**: variable row sampling rate per class.

Extreme Values & Outliers

Types of Outliers

- Extreme values can exist in response or predictors
- Valid: rare, extreme events
- Invalid: erroneous measurements

What Can Happen

- Extreme values can have a disproportionate effect.
- MSE will focus on handling extreme observations more to reduce squared error.
- Boosting will spend considerable modeling effort fitting these observations.

What to Do

- Remove observations that represent outliers.
- Apply a transformation to reduce impact: e.g. log skewed data, create categorical bins, impose cap on low/high values (winsorize).
- Choose a more robust loss function: e.g. MAE vs MSE.
- Ask questions: Understand whether the values are valid or invalid, to make the most appropriate choice.

H2O Extreme Value Handling

```r
h2o_frame["log_x"] <- h2o.log1p(h2o_frame["x"])

h2o_frame["cat_x"] <- h2o.cut(h2o_frame["x"], breaks)

h2o_frame["winz_x"] <- h2o.ifelse(h2o_frame["x"] < low, low, h2o_frame["x"])
h2o_frame["winz_x"] <- h2o.ifelse(h2o_frame["winz_x"] > high, high,
                                  h2o_frame["winz_x"])
```

```python
h2o_frame["log_x"] = h2o_frame["x"].log1p()

h2o_frame["cat_x"] = h2o_frame["x"].cut(breaks)

h2o_frame["winz_x"] = h2o.H2OFrame.ifelse(h2o_frame["x"] < low,
                                          low, h2o_frame["x"])
h2o_frame["winz_x"] = h2o.H2OFrame.ifelse(h2o_frame["winz_x"] > high,
                                          high, h2o_frame["winz_x"])
```

Low Frequency Categories

Real Data

- Most real world datasets contain categorical data.

Too Many Categories

- Problems can arise if you have **too many categories**:
  - Computational complexity during estimation
  - Infrequent categories can lead to overfitting

Solutions

- Use knowledge about hierarchical data to collapse categories.
- Use Cross-Validated Mean Target Encoding.
- Use Cross-Validated Weight of Evidence Encoding when modeling binary outcome.
- Use H2O's `categorical_encoding` and `nbins_cat` decision tree tuning arguments

Target Mean Encoding

What?

Replace categorical variables with the mean of the response

Why?

Categorical variables increase the number of features (dummy encoding) and can cause us to overfit

H2O Target Mean Encoding

```r
h2o.target_encode_apply(data, x, y, target_encode_map, holdout_type,
                        fold_column = NULL, blended_avg = TRUE,
                        noise_level = NULL, seed = -1)
```

```python
def mean_target_encoding(data, x, y, fold_column):
    grouped_data = data[[x, fold_column, y]].group_by([x, fold_column])
    grouped_data.sum(na = "ignore").count(na = "ignore")
    df = grouped_data.get_frame().as_data_frame()
    df_list = []
    nfold = int(data[fold_column].max()) + 1
    for j in range(0, nfold):
        te_x = "te_{}".format(x)
        sum_y = "sum_{}".format(y)
        oof = df.loc[df[fold_column] != j, [x, sum_y, "nrow"]]
        stats = oof.groupby([x]).sum()
        stats[x] = stats.index
        stats[fold_column] = j
        stats[te_x] = stats[sum_y] / stats["nrow"]
        df_list.append(stats[[x, fold_column, te_x]])
    return h2o.H2OFrame(pd.concat(df_list))
```

Target Mean Encoding

Pay 1	Default Payment
Up To Date	0
Up To Date	0
Up To Date	0
Missed 1 Mo	1
Missed 1 Mo	0
Missed 1 Mo	0
Missed 5 Mo	1

Target Mean Encoding

Pay 1	Default Payment	Mean Target Encoding
Up To Date	0	0
Up To Date	0	0
Up To Date	0	0
Missed 1 Mo	1	0.33
Missed 1 Mo	0	0.33
Missed 1 Mo	0	0.33
Missed 5 Mo	1	1

Target Mean Encoding

- Mean Target Encoding is based on the response column of the rows
- The lower the number of rows in the group, the more it reveals the response column value

Pay 1	Default Payment	Mean Target Encoding
Missed 5 Mo	1	1

Worst Case Scenario: Response Column = Mean Target Encoding

Cross Validation Target Encoding

Fold	Pay 1	Default Payment
1	Up To Date	0
2	Up To Date	0
3	Up To Date	0
2	Missed 1 Mo	1
1	Missed 1 Mo	0
3	Missed 1 Mo	0
1	Missed 5 Mo	1

Cross Validation Target Encoding

Fold	Pay 1	Default Payment
2	Up To Date	0
3	Up To Date	0
2	Missed 1 Mo	1
3	Missed 1 Mo	0

Fold	Pay 1	Default Payment
1	Up To Date	0
1	Missed 1 Mo	0
1	Missed 5 Mo	1

Cross Validation Target Encoding

Fold	Pay 1	Default Payment	Mean Target Encoding
2	Up To Date	0	0
3	Up To Date	0	0
2	Missed 1 Mo	1	0.5
3	Missed 1 Mo	0	0.5

Fold	Pay 1	Default Payment
1	Up To Date	0
1	Missed 1 Mo	0
1	Missed 5 Mo	1

Cross Validation Target Encoding

Fold	Pay 1	Default Payment	Mean Target Encoding
2	Up To Date	0	0
3	Up To Date	0	0
2	Missed 1 Mo	1	0.5
3	Missed 1 Mo	0	0.5

Fold	Pay 1	Default Payment	CV Target Encoding
1	Up To Date	0	0
1	Missed 1 Mo	0	
1	Missed 5 Mo	1	

Cross Validation Target Encoding

Fold	Pay 1	Default Payment	Mean Target Encoding
2	Up To Date	0	0
3	Up To Date	0	0
2	Missed 1 Mo	1	0.5
3	Missed 1 Mo	0	0.5

Fold	Pay 1	Default Payment	CV Target Encoding
1	Up To Date	0	0
1	Missed 1 Mo	0	0.5
1	Missed 5 Mo	1	

Cross Validation Target Encoding

Fold	Pay 1	Default Payment	Mean Target Encoding
2	Up To Date	0	0
3	Up To Date	0	0
2	Missed 1 Mo	1	0.5
3	Missed 1 Mo	0	0.5

Fold	Pay 1	Default Payment	CV Target Encoding
1	Up To Date	0	0
1	Missed 1 Mo	0	0.5
1	Missed 5 Mo	1	NA

Weight Of Evidence Encoding

What?

In binary classification, replace categorical variables with

$$WOE_{ja} = \ln \frac{P(X_j = a|Y = 1)}{P(X_j = a|Y = 0)}$$

Why?

Leverage rich history in information theory and Bayesian statistics to manage overfitting of high cardinality variables

Weight Of Evidence Encoding

Pay 1	Default Payment	% 0s	% 1s	(% 1s) / (% 0s)	WOE
Up To Date	0	60 %	0 %	0	- Inf
Up To Date	0	60 %	0 %	0	- Inf
Up To Date	0	60 %	0 %	0	- Inf
Missed 1 Mo	1	40 %	50 %	1.25	0.223
Missed 1 Mo	0	40 %	50 %	1.25	0.223
Missed 1 Mo	0	40 %	50 %	1.25	0.223
Missed 5 Mo	1	0 %	50 %	Inf	Inf

Redundant Data

Real Data

- Features can be highly interrelated.
- Not all features are related to the target.

Too much of the same is not a good thing

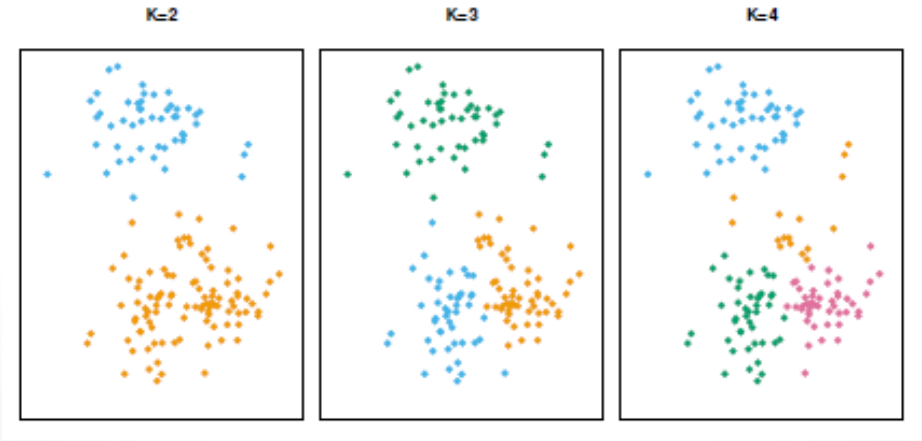
- Leads to numeric instability in machine learning algorithms (GLM)
- Overweights importance of redundant features (RF, GBM)

Solutions

- Select representatives from clustered / grouped features
- Use dimensionality reduction techniques (PCA, GLRM, MCA, MDS)

K-Means Clustering

- K-Means clustering groups observations based on numeric features
  - Assumes clusters are roughly the same sized hyperspheres
  - Minimize Euclidean distance between observations and cluster centers
- Number of methods for choosing the number of clusters, k
  - Choose several and evaluate performance
  - Use business rules



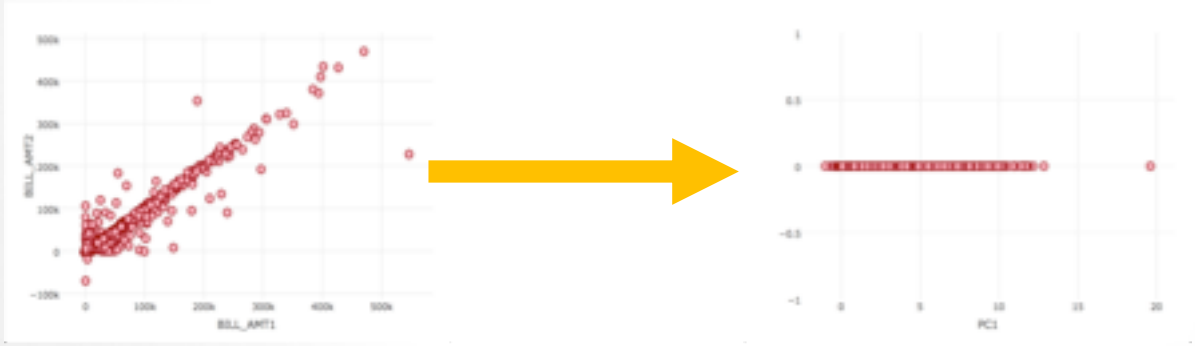
H2O K-Means Clustering

```
h2o.kmeans(training_frame, x, model_id = NULL, validation_frame = NULL,
           nfolds = 0, keep_cross_validation_predictions = FALSE,
           keep_cross_validation_fold_assignment = FALSE,
           fold_assignment = c("AUTO", "Random", "Modulo", "Stratified"),
           fold_column = NULL, ignore_const_cols = TRUE,
           score_each_iteration = FALSE, k = 1, estimate_k = FALSE,
           user_points = NULL, max_iterations = 10, standardize = TRUE,
           seed = -1, init = c("Random", "PlusPlus", "Furthest", "User"),
           max_runtime_secs = 0, categorical_encoding = c("AUTO", "Enum",
           "OneHotInternal", "OneHotExplicit", "Binary", "Eigen",
           "LabelEncoder", "SortByResponse", "EnumLimited"))
```

```
from h2o.estimators.kmeans import H2OKMeansEstimator
clusters = H2OKMeansEstimator(…)
clusters.train(x = x, training_frame = data)
```

Principal Components Analysis

- Orthogonal rotation of covariance or correlation matrix that orders derived measures from highest to lowest variation
- Useful for dimensionality reduction / removing collinearities



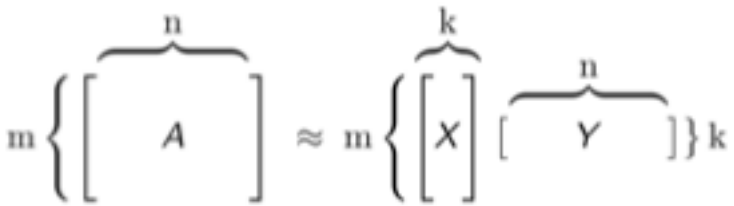
H2O Principal Components Analysis

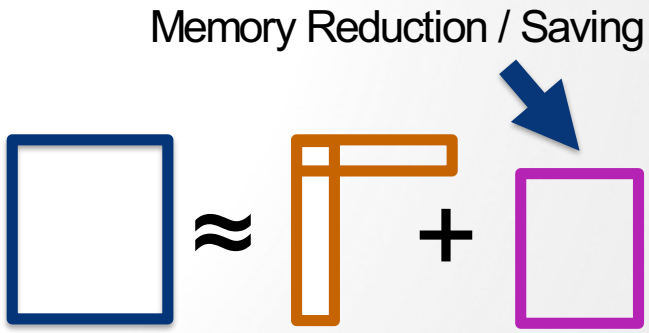
```
h2o.prcomp(training_frame, x, model_id = NULL, validation_frame = NULL,
           ignore_const_cols = TRUE, score_each_iteration = FALSE,
           transform = c("NONE", "STANDARDIZE", "NORMALIZE", "DEMEAN",
           "DESCALE"), pca_method = c("GramSVD", "Power", "Randomized",
           "GLRM"), k = 1, max_iterations = 1000,
           use_all_factor_levels = FALSE, compute_metrics = TRUE,
           impute_missing = FALSE, seed = -1, max_runtime_secs = 0)
```

```
from h2o.estimators.pca import H2OPrincipalComponentAnalysisEstimator
pca = H2OPrincipalComponentAnalysisEstimator(…)
pca.train(x = x, training_frame = data)
```

Generalized Low-Rank Models

- GLRM is an extension of well-known matrix factorization methods such as Principal Component Analysis (PCA).
- Unlike PCA which is limited to numerical data, GLRM can also handle categorical, ordinal and Boolean data.
- **Given**: Data table *A* with *m* rows and *n* columns
- **Find**: Compressed representation as numeric tables *X* and *Y* where *k* is a small user-specified number





- *Y* = archetypal features created from columns of *A*
- *X* = row of A in reduced feature space
- GLRM can approximately reconstruct *A* from product *XY*

H2O Generalized Low-Rank Models

```
h2o.glrm(training_frame, cols = NULL, model_id = NULL, validation_frame = NULL,
        ignore_const_cols = TRUE, score_each_iteration = FALSE, loading_name = NULL,
        transform = c("NONE", "STANDARDIZE", "NORMALIZE", "DEMEAN", "DESCALE"), k = 1,
        loss = c("Quadratic", "Absolute", "Huber", "Poisson", "Hinge", "Logistic", "Periodic"),
        loss_by_col = c("Quadratic", "Absolute", "Huber", "Poisson", "Hinge", "Logistic",
        "Periodic", "Categorical", "Ordinal"), loss_by_col_idx = NULL,
        multi_loss = c("Categorical", "Ordinal"), period = 1,
        regularization_x = c("None", "Quadratic", "L2", "L1", "NonNegative", "OneSparse",
        "UnitOneSparse", "Simplex"), regularization_y = c("None", "Quadratic", "L2", "L1",
        "NonNegative", "OneSparse", "UnitOneSparse", "Simplex"), gamma_x = 0, gamma_y = 0,
        max_iterations = 1000, max_updates = 2000, init_step_size = 1,
        min_step_size = 1e-04, seed = -1, init = c("Random", "SVD", "PlusPlus", "User"),
        svd_method = c("GramSVD", "Power", "Randomized"), user_y = NULL, user_x = NULL,
        expand_user_y = TRUE, impute_original = FALSE, recover_svd = FALSE, max_runtime_secs = 0)
```

```
from h2o.estimators.glrm import H2OGeneralizedLowRankEstimator
model = H2OGeneralizedLowRankEstimator(…)
model.train(x = x, training_frame = data)
```

Irrelevant Data

Real Data	• Not all features are related to the target.
Not all data have value	• Noise can be mistaken as signal by machine learning algorithms.
Solution	• Once identified, remove from the analysis. Do not rely on algorithms to remove irrelevant features. Have doubts? Simulate random numeric and categorical features and find how many of them appear to be important.

Missing Data

Types of Missing Data

- Unavailable: Valid for the observation, but not available in the data set.
- Removed: Observation quality threshold may have not been reached, and data removed.
- Not applicable: measurement does not apply to the particular observation (e.g. number of tires on a boat observation)

What to Do

- Ignore entire observation.
- Create an binary variable for each predictor to indicate whether the data was missing or not.
- Segment model based on data availability.
- Estimate missing values (Generalized Low Rank Models)
- Use alternative algorithm: decision trees accept missing values; linear models typically do not.

H2O Simple Missing Data Imputation

```r
h2o.impute(data, column = 0, method = c("mean", "median", "mode"),
           combine_method = c("interpolate", "average", "lo", "hi"),
           by = NULL, groupByFrame = NULL, values = NULL)
```

```python
h2o_frame.impute(column = -1, method = u'mean',
                 combine_method = u'interpolate',
                 by = None, group_by_frame = None, values = None)
```

Model Scoring Properties

Implication of Model Complexity

- Complex models tend to have higher predictive power than simple models.
- Complex models tend to require more compute instructions to generate predictions than simple models.

Solutions

- Real-Time Scoring
  - Use fast scoring infrastructure (e.g. POJO / MOJO)
  - Simplify models that take too long to score
- Offline Scoring
  - More flexible in choice of scoring infrastructure
  - Better able to use most accurate model created

Model Interpretability

Implication of Model Complexity

- Model complexity tends to reduce interpretability.

Solutions

- Create parsimonious models
- When fitting complex models use:
  - Variable importance measures
  - Leave One Covariate Out (LOCO)
  - Surrogate generalized linear models and decision trees with model predictions as target
  - Local Interpretable Model-Agnostic Explanations (LIME)

H2O Training

1. Introduction to ML and H2O
2. H2O Auto Machine Learning
3. Productionalizing Models
4. Machine Learning Concepts
5. H2O Feature Engineering
6. H2O Algorithms

H2O Data Munging

- Data Transfer
- H2OFrame Attributes
- Data Column Types
- Row & Columns Selection
- Categorical Data Operations
- Filters & Logical Operations
- Missing Data Handling
- Summary & Aggregation
- Numeric Data Summaries
- Numeric Data Transformations
- Date/Time Manipulations
- String Munging
- Joins Between Two H2OFrames
- Histogram of Numeric Columns

Data Transfer between Python and H2O

h2o.H2OFrame

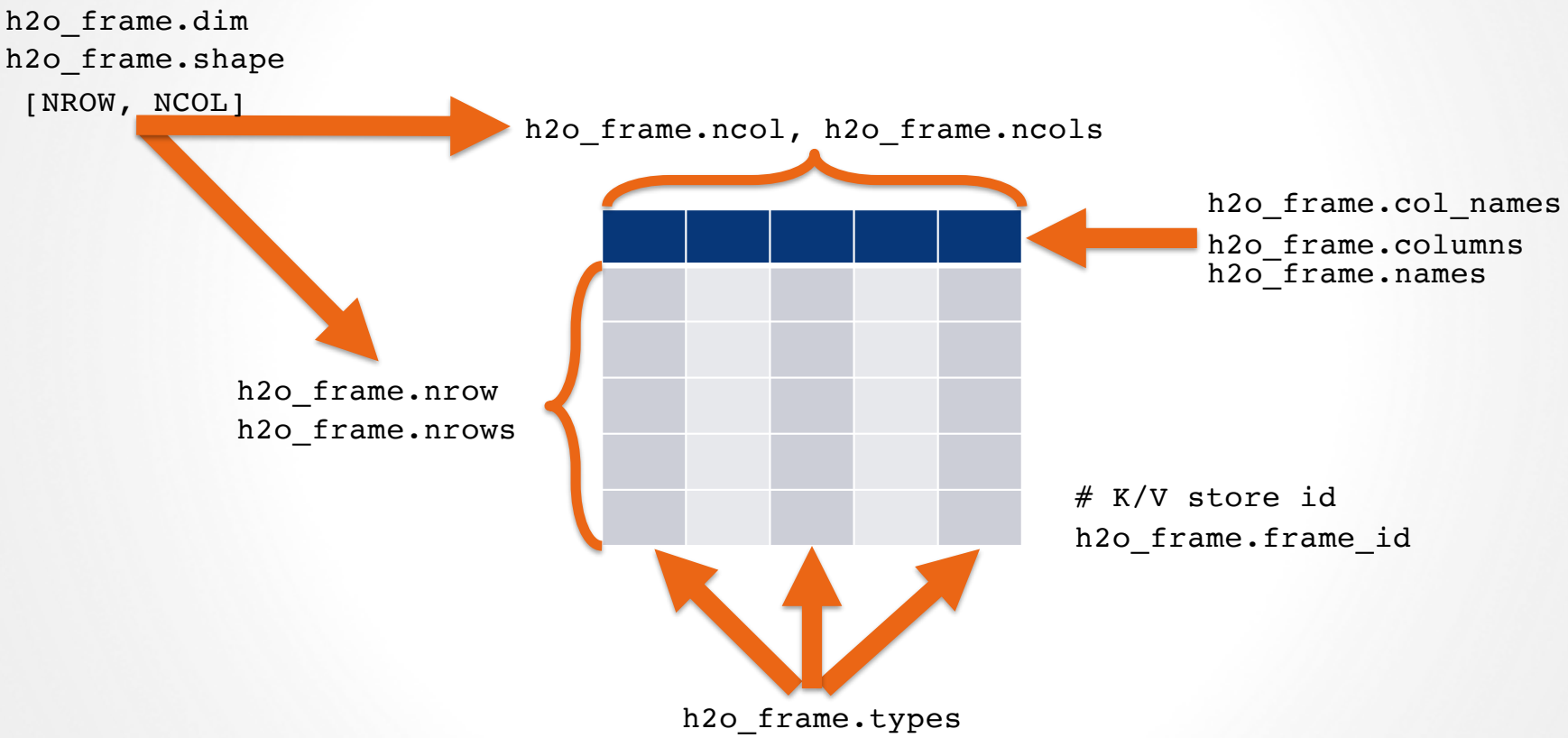
- Turns python pandas frame into an H2OFrame
- Saves dataframe to CSV and then performs `h2o.upload()` to H2O cluster

h2o.as_list OR h2o_frame.as_data_frame

- Turns H2OFrame to lists of list or pandas frame
- Downloads the H2OFrame locally and will parse with pandas if use_pandas is set to True

Note: Be mindful of `h2o.as_list`, it is difficult to fit a large H2OFrame into a Python session

H2OFrame Attributes



H2O Column Types

Enum

`data[x].asfactor()`

Used to convert given variable to an enumerated type variable

(booleans are coded as enum)

Numeric

`data[x].asnumeric()`

Used to convert given variable to an numeric variable

Dates

`data[x].year()`

Convert the entries of an H2OFrame object from milliseconds to years, indexed starting from 1900.

`data[x].month()`

Converts the entries of an H2OFrame object from milliseconds to months (on a 1 to 12 scale).

String

`data[x].ascharacter()`

Used to convert given variable to an string type variable

H2OFrame Example

```
1 import h2o
2 h2o.init()
3 data_path = "https://s3.amazonaws.com/h2o-public-test-data/smalldata/census_income/adult_data.csv"
4 census_data = h2o.import_file(data_path, destination_frame = "census_init")
```

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

```
1 print(census_data.types)
```

```
{'age': 'int',
 'workclass': 'enum',
 'fnlwgt': 'int',
 'education': 'enum',
 'education-num': 'int',
 'marital-status': 'enum',
 'occupation': 'enum',
 'relationship': 'enum',
 'race': 'enum',
 'sex': 'enum',
 'capital-gain': 'int',
 'capital-loss': 'int',
 'hours-per-week': 'int',
 'native-country': 'enum',
 'income': 'enum'}
```

H2OFrame Example

```
1 census_data[0:5].summary() # census_data[0:5].describe()
```

	age	workclass	fnlwgt	education	education-num
type	int	enum	int	enum	int
mins	17.0		12285.0		1.0
mean	38.581646755321145		189778.36651208595		10.080679340315212
maxs	90.0		1484705.0		16.0
sigma	13.640432553581354		105549.97769702227		2.5727203320673966
zeros	0		0		0
missing	0	1836	0	0	0
0	39.0	State-gov	77516.0	Bachelors	13.0
1	50.0	Self-emp-not-inc	83311.0	Bachelors	13.0
2	38.0	Private	215646.0	HS-grad	9.0
3	53.0	Private	234721.0	11th	7.0

Row & Column Selection

Pandas-like convention for slicing and dicing data

(0-based indexes)

Extracting a single column

- `h2o_frame["x"] # column name x`
- `h2o_frame[2]  # 3rd col`
- `h2o_frame[-2]  # 2nd col from end`
- `h2o_frame[:, -1] # Last column`

Filtering rows

- `h2o_frame[0:5, :]`
- `h2o_frame[h2o_frame["x"] > 1, :]`

Extracting multiple columns

- `h2o_frame[["x","y","z"]]`
- `h2o_frame[[1, 5, 6]]`

Filtering rows for select columns

```
h2o_frame[0:50, [1,2,3]]

med = h2o_frame["a"].median()
h2o_frame[h2o_frame["a"] > med, "z"]
```

Filters & Logical Operations

- Logical Operators
  - `h2o_frame[x].logical_negation()`
  - `h2o_frame[x] & h2o_frame[y]`
  - `h2o_frame[x] | h2o_frame[y]`
- Comparison Operators
  - `h2o_frame[x] {==, !=, <, <=, >=, >} value`
  - `h2o_frame[x] {==, !=, <, <=, >=, >} h2o_frame[y]`
- Logical Data Summaries
  - `h2o_frame[x].all()  # includes NAs`
  - `h2o_frame[x].any()  # includes NAs`
  - `h2o_frame[x].any_na_rm()  # disregards NAs`

Filters & Logical Operations

```
test.ifelse(yes, no)
```

Arguments

test	A logical description of the condition to be met (>, <, =, etc...)
yes	The value to return if the condition is TRUE.
no	The value to return if the condition is FALSE.

Equivalent to `[y if t else n for t,y,n in zip(self,yes,no)]`

Note: Only numeric values can be tested, and only numeric results can be returned.

Categorical Data Operations

- Metadata
  - `h2o_frame[x].categories()`
  - `h2o_frame[x].levels()`
  - `h2o_frame[x].nlevels()`
- Categorical Data Manipulation
  - `h2o_frame.relevel(y)`
  - `h2o_frame.set_level(level)`
  - `h2o_frame.set_levels(levels)`

Data Selection Example

```
1 tech_support_income = census_data[census_data["occupation"] == "Tech-support", "income"]
2 print(type(tech_support_income))
```

```
<class 'h2o.frame.H2OFrame'>
```

```
1 print(tech_support_income.table())
```

income	Count
<=50K	645
>50K	283

Missing Data Handling

- Missing Data Code
  - `None`
- Missing Data Filtering
  - `h2o_frame.filter_na_cols(frac=0.2)`
  - `h2o_frame.na_omit()`
- Missing Data Replacement
  - `h2o_frame.impute(…)`
  - `H2OGeneralizedLowRankEstimator`
- Missing Data Inclusion
  - `h2o_frame.insert_missing_values(fraction=0.1, seed=None)`
- Missing Data Summaries
  - `h2o_frame.nacnt()`

Missing Data Handling

```
h2o_frame.impute(column = 0, method = c(“mean", "median",
“mode"), combine_method c("interpolate","average", "lo", "hi"),
by = NULL, groupByFrame = NULL, values = NULL)
```

Arguments

h2o_frame	The dataset containing the column to impute.
column	The column to impute.
method	“mean" replaces NAs with the column mean; "median" replaces NAs with the column median; "mode" replaces with the most common factor (for factor columns only)
combine_method	If method is "median", then choose how to combine quantiles on even sample sizes. This parameter is ignored in all other cases
by	Group by columns
groupByFrame	Impute the column col with this pre-computed grouped frame.
values	A vector of impute values (one per column). NaN indicates to skip the column

Missing Data Handling

```
1 nacnts = dict(zip(census_data.col_names, census_data.nacnt()))
2 print(dict((k, int(v)) for (k, v) in nacnts.items() if v > 0))
```

```
{'workclass': 1836, 'occupation': 1843, 'native-country': 583}
```

```
1 codes = census_data["native-country"].asnumeric()
2 levels = census_data["native-country"].levels()[0]
3 levels.append("Unknown")
4
5 census_data["native-country-clean"] = h2o.H2OFrame.ifelse(codes != None, codes, len(levels))
6 census_data["native-country-clean"] = census_data["native-country-clean"].asfactor()
7 census_data["native-country-clean"] = census_data["native-country-clean"].set_levels(levels)
8
9 print((census_data["native-country-clean"] == "Unknown").table())
```

native-country-clean	Count
0	31978
1	583

Summary & Aggregation

```
h2o_frame[x].table(dense = TRUE)
```

Arguments

h2o_frame	An H2OFrame object with at least one column
x	Column name
dense	A logical for dense representation, which lists only non-zero counts, 1 combination per row. Set to FALSE to expand counts across all combinations.

Value: Returns a tabulated H2OFrame Object

Summary & Aggregation

```
h2o.group_by(data, by, ..., gb.control =
list(na.methods = NULL, col.names = NULL))
```

Arguments

data	an H2OFrame object.
by	a list of column names
gb.control	a list of how to handle NA values in the dataset as well as how to name output columns
. . .	Any supported aggregated function: `mean, min, max, sum, sd, nrow`

Numeric Data Summaries

- `h2o_frame[x].cor(y=None, na_rm=False, use=None)`
- `h2o_frame[x].kurtosis(na_rm=False)`
- `h2o_frame[x].max()`
- `h2o_frame[x].mean(skipna=False)`
- `h2o_frame[x].median(na_rm=False)`
- `h2o_frame[x].min()`
- `h2o_frame[x].prod(na_rm=False)`
- `h2o_frame[x].quantile(…)`
- `h2o_frame[x].sd(na_rm=False)`
- `h2o_frame[x].skewness(na_rm=False)`
- `h2o_frame[x].sum(skipna=True)`
- `h2o_frame[x].var(y=None, na_rm=False, use=None)`

Group By Aggregation

```
grouped_data = census_data[["occupation", "education-num"]].group_by(["occupation"])
stats = grouped_data.count(na = "ignore").median(na = "ignore").mean(na = "ignore").sd(na = "ignore")
stats.get_frame()
```

occupation	nrow	median_education-num	mean_education-num	sdev_education-num
	0	9	9.25339	2.60279
Adm-clerical	3770	10	10.1135	1.69805
Armed-Forces	9	9	10.1111	2.02759
Craft-repair	4099	9	9.11076	2.03865
Exec-managerial	4066	12	11.4491	2.14321
Farming-fishing	994	9	8.60865	2.75607
Handlers-cleaners	1370	9	8.51022	2.20338
Machine-op-inspct	2002	9	8.48751	2.28528
Other-service	3295	9	8.77967	2.29966
Priv-house-serv	149	9	7.36242	3.11104

Cross-Validated Mean Target Encoding

(Feature Engineering Sneak Peak)

```
def mean_target_encoding(data, x, y, fold_column):
    grouped_data = data[[x, fold_column, y]].group_by([x, fold_column])
    grouped_data.sum(na = "ignore").count(na = "ignore")
    df = grouped_data.get_frame().as_data_frame()
    df_list = []
    nfold = int(data[fold_column].max()) + 1
    for j in range(0, nfold):
        te_x = "te_{}".format(x)
        sum_y = "sum_{}".format(y)
        oof = df.loc[df[fold_column] != j, [x, sum_y, "nrow"]]
        stats = oof.groupby([x]).sum()
        stats[x] = stats.index
        stats[fold_column] = j
        stats[te_x] = stats[sum_y] / stats["nrow"]
        df_list.append(stats[[x, fold_column, te_x]])
    return h2o.H2OFrame(pd.concat(df_list))
```

Numeric Data Transformations

- h2o_frame[x].abs()
- h2o_frame[x].acos()
- h2o_frame[x].acosh()
- h2o_frame[x].asin()
- h2o_frame[x].asinh()
- h2o_frame[x].atan()
- h2o_frame[x].atanh()
- h2o_frame[x].ceil()
- h2o_frame[x].cos()
- h2o_frame[x].cosh()
- h2o_frame[x].cospi()
- h2o_frame[x].cut(breaks, …)
- h2o_frame[x].digamma()
- h2o_frame[x].exp()
- h2o_frame[x].expm1()
- h2o_frame[x].floor()
- h2o_frame[x].gamma()
- h2o_frame[x].lgamma()
- h2o_frame[x].log()
- h2o_frame[x].log10()
- h2o_frame[x].log1p()
- h2o_frame[x].log2()
- h2o_frame[x].round(digits=0)
- h2o_frame[x].scale(center=True, scale=True)
- h2o_frame[x].sign()
- h2o_frame[x].signif(digits=6)
- h2o_frame[x].sin()
- h2o_frame[x].sinh()
- h2o_frame[x].sinpi()
- h2o_frame[x].sqrt()
- h2o_frame[x].tan()
- h2o_frame[x].tanh()
- h2o_frame[x].tanpi()
- h2o_frame[x].trigamma()
- h2o_frame[x].trunc()

Numeric Data Transformations

Transformation for skewed data with positive and negative values

```
pseudoLog10(x) = asinh(x/2) / log(10)
```

```
1  import math
2
3  def pseudo_log10(x):
4      return math.asinh(x / 2) / math.log(10)
5
6  print("pseudo_log10(\261{}) = \261{:0.6f}".format(100000, pseudo_log10(100000)))
7  print("pseudo_log10(\261{})  = \261{:0.6f}".format(10000, pseudo_log10(10000)))
8  print("pseudo_log10(\261{})   = \261{:0.6f}".format(1000, pseudo_log10(1000)))
9  print("pseudo_log10(\261{})    = \261{:0.6f}".format(100, pseudo_log10(100)))
10 print("pseudo_log10(\261{})     = \261{:0.6f}".format(10, pseudo_log10(10)))
11 print("pseudo_log10(\261{})      = \261{:0.6f}".format(1, pseudo_log10(1)))
12 print("pseudo_log10({})       = {}".format(0, pseudo_log10(0)))
```

```
pseudo_log10(±100000) = ±5.000000
pseudo_log10(±10000)  = ±4.000000
pseudo_log10(±1000)   = ±3.000000
pseudo_log10(±100)    = ±2.000043
pseudo_log10(±10)     = ±1.004279
pseudo_log10(±1)      = ±0.208988
pseudo_log10(0)       = 0.0
```

Numeric Data Transformations

```
1 import numpy as np
2 breaks = np.linspace(10, 90, 9).tolist()
3 census_data["age_group"] = census_data["age"].cut(breaks)
4
5 census_data["log1p_capital-gain"] = census_data["capital-gain"].log1p()
6 census_data["log1p_capital-loss"] = census_data["capital-loss"].log1p()
7 print(census_data["age_group"].table())
```

age_group	Count
(10.0,20.0]	2410
(20.0,30.0]	8162
(30.0,40.0]	8546
(40.0,50.0]	6983
(50.0,60.0]	4128
(60.0,70.0]	1792
(70.0,80.0]	441
(80.0,90.0]	99

Date/Time Manipulations

- Date/Time Creation
  - `h2o_frame[x].as_date(format)`
- Date Extraction
  - `h2o_frame[x].day()`
  - `h2o_frame[x].dayOfWeek()`
  - `h2o_frame[x].month()`
  - `h2o_frame[x].week()`
  - `h2o_frame[x].year()`
- Time Extraction
  - `h2o_frame[x].hour()`
  - `h2o_frame[x].minute()`
  - `h2o_frame[x].second()`

String Munging

- String Matching
  - `h2o_frame[x].countmatches()`
  - `h2o_frame[x].grep()`
  - `h2o_frame[x].match()`
- Substitute pattern match
  - `h2o_frame[x].gsub() # Replace all occurrences`
  - `h2o_frame[x].sub()  # Replace first occurrence`
- String Cleaning
  - `h2o_frame[x].substring()`
  - `h2o_frame[x].strsplit()`
  - `h2o_frame[x].trim()`
  - `h2o_frame[x].lstrip()`
  - `h2o_frame[x].rstrip()`

Joins Between Two H2OFrames

```
h2o_frame.merge(other, all_x=False, all_y=False, by_x=None,
                by_y=None, method='auto')
```

Arguments

h2o_frame	left/self data set in the join.
other	right/other data set in the join.
all_x	If True, include all rows from the left/self frame.
all_y	If True, include all rows from the right/other frame.
by_x	list of columns in the left/self frame to use as a merge key.
by_y	list of columns in the right/other frame to use as a merge key.
method	string representing the merge method, one of auto(default), radix or hash.

Joins Between Two H2OFrames

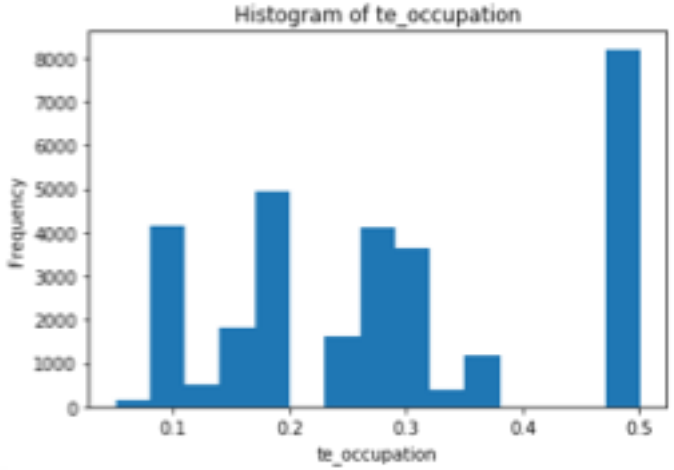
```
1 census_data["cv_fold"] = census_data.kfold_column(n_folds = 5, seed = 123)
2 census_data["high_income"] = census_data["income"] == ">50K"
3 te_occupation = mean_target_encoding(census_data, x = "occupation", y = "high_income",
                                       fold_column = "cv_fold")
```

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

```
1 census_data = census_data.merge(te_occupation, all_x = True)
```

Histogram of Numeric Columns

```
1 import matplotlib.pyplot as plt
2 %matplotlib inline
3
4 import warnings
5 import matplotlib.cbook
6 warnings.filterwarnings("ignore", category = matplotlib.cbook.mplDeprecation)
7
8 census_data["te_occupation"].hist()
```



Case Study: Lending Club Dataset

- Loan data from 2007 up until 2015 including rejected applications and accepted applications.
- Of the 500k accepted applicants about 160k loans have either been completely paid off or defaulted.
- There are about 4 million applicants in the rejected loans dataset.
- **Use Case 1**: Predict the likelihood of a user defaulting based on the information supplied when applying for a loan.
- **Use Case 2**: Determine the interest rate Lending Club would have offered the user based on the information supplied when applying for a loan.
- Full Data: https://www.kaggle.com/wendykan/lending-club-loan-data
- H2O Subset: https://s3.amazonaws.com/h2o-public-test-data/bigdata/laptop/lending-club/loan.csv

Case Study: Lending Club Dataset

	Column Name	Description	Unit	Role
1	loan_amnt	Requested loan amount	US dollars	Predictor
2	term	Longest term length	Months	Predictor
3	**int_term**	**Recommended interest rate**	**Rate**	**Response**
4	emp_length	Employment length	Years	Predictor
5	home_ownership	Housing status	Categorical	Predictor
6	annual_inc	Annual income	US dollars	Predictor
7	purpose	Purpose for the loan	Categorical	Predictor
8	addr_state	State of residence	Categorical	Predictor
9	dti	Debt to income ratio	Percent	Predictor
10	delinq_2yrs	Number of delinquencies in the past 2 years	Count	Predictor
11	revol_util	Revolving credit line utilized	Percent	Predictor
12	total_acc	Number of active accounts	Count	Predictor
13	**bad_loan**	**Bad loan indicator**	**Boolean**	**Response**
14	longest_credit_length	Age of oldest active account	Years	Predictor
15	verification_status	Income verification status	Boolean	Predictor

Case Study: Lending Club Dataset

Predictor Column Name	Transformation
loan_amnt	Manage high and low amounts
term	Recode as 0/1 binary
emp_length	Create missing value indicator
home_ownership	Combine categories
annual_inc	Manage high and low annual incomes
purpose	Cross-Validated Mean Target Encoding
addr_state	Cross-Validated Mean Target Encoding
dti	Manage high and low debt-to-income ratios
delinq_2yrs	Manage high and low delinquency counts
revol_util	Manage high and low utilization
total_acc	Manage high and low number of accounts
longest_credit_length	Manage high and low credit lengths
verification_status	Recode as 0/1 binary

H2O Training

1. Introduction to ML and H2O
2. H2O Auto Machine Learning
3. Productionalizing Models
4. Machine Learning Concepts
5. H2O Feature Engineering
6. H2O Algorithms

Supervised Learning in H2O

- **Regression and Classification Models** Exponential distributions including Poisson, Gamma, and Tweedie are available in addition to Bernoulli, Multinomial, and Gaussian distributions.
- **Fast and CPU Efficient** Parallel and distributed computation across multiple nodes and many cores.
- **Grid Search** Hyperparameter optimization allows the user to run through many parameters before selecting the best models.
- **Early Stopping** The user can specify the metric and the incremental change in the metric as convergence.
- **Stochastic** User can specify the sample rate that the algorithm will sample the column and row by for better generalization.
- **Model Output** The model is exportable as Java code and if you find the model overfitted after a certain number of trees, it is easy to reduce the number of trees in a POJO before putting it in production without rerunning model build.

Supervised Learning:

GENERALIZED LINEAR MODEL

What is the Generalized Linear Model (GLM)?

- Class of models that relate X (inputs) to Y (output)

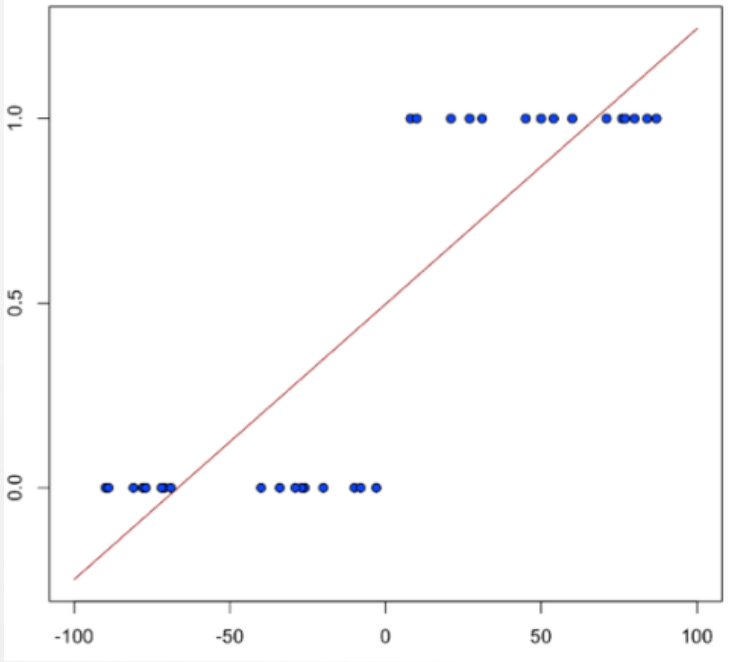
$$\mathrm{E}(\mathbf{Y}) = \boldsymbol{\mu} = g^{-1}(\mathbf{X}\boldsymbol{\beta})$$

$$\mathrm{Var}(\mathbf{Y}) = \mathrm{V}(\boldsymbol{\mu}) = \mathrm{V}(g^{-1}(\mathbf{X}\boldsymbol{\beta})).$$

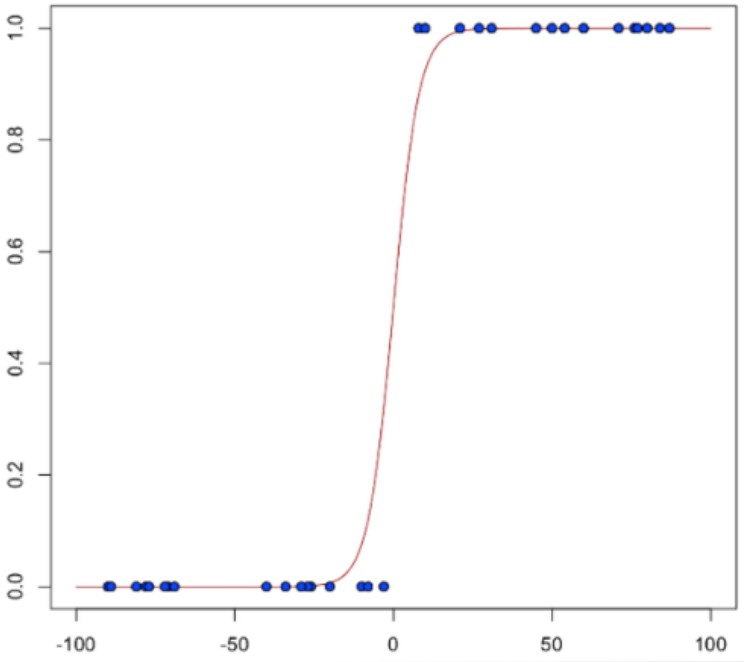
- Allows for a unification of models that have errors of the following form:
  - Normal (Gaussian)
  - Poisson
  - Gamma
  - Tweedie
  - Binomial (Logistic)
  - Multinomial
- MLE is found by iteratively reweighted least squares

Same predictors, different family and link functions

Linear Regression fit
(family=gaussian,link =identity)



Logistic Regression fit
(family=binomial,link = logit)



Pros and Cons of GLM's

Pros

- Fast to fit
- Easy to interpret
- Well understood statistical properties
- Works for continuous, categorical, and count targets
- Transparent scoring functions

Cons

- Dependent of external feature engineering
- Difficult to model non-linearities
- Difficult to model interaction effects
- Ill-defined with multicollinearity
  - Non-unique solutions

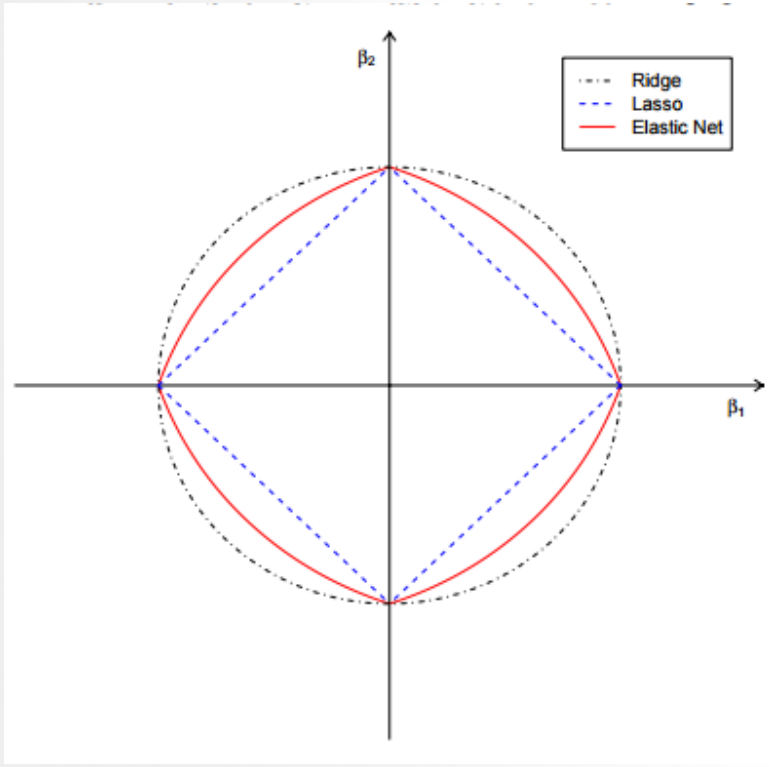
Supervised Learning:

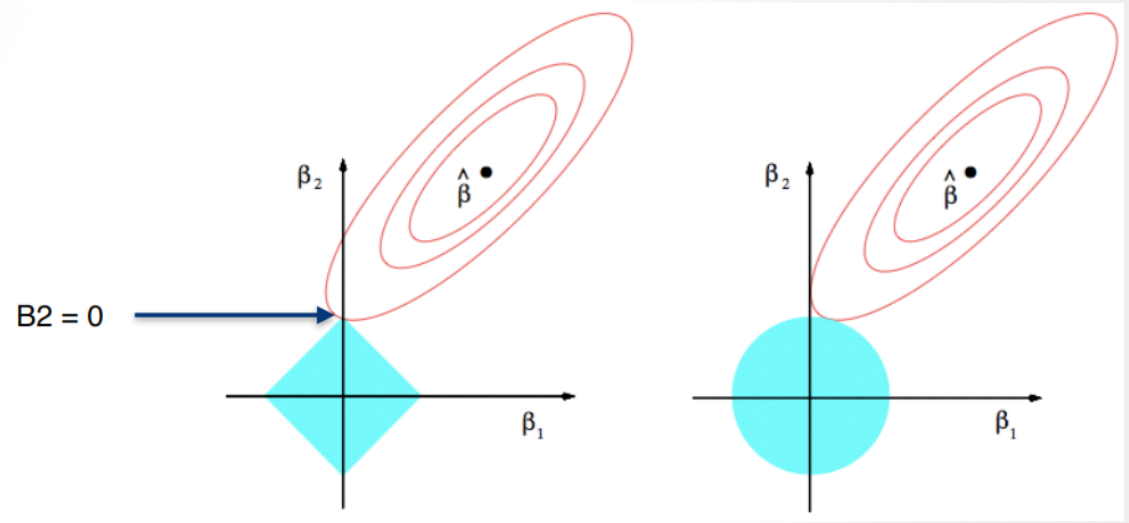
PENALIZED GENERALIZED LINEAR MODEL(LASSO, RIDGE, ELASTIC NET)

Regularized Regression

- Want to build a parsimonious but interpretable model
- Shrink the number of predictors and/or size by imposing penalties on the estimated coefficients (L1, L2)
  - LASSO: $\min_{\beta_0,\beta}\left\{\frac{1}{N}\sum_{i=1}^{N}(y_i-\beta_0-x_i^T\beta)^2\right\}$ subject to $\sum_{j=1}^{p}|\beta_j|\le t.$
    - picks one correlated variable. others discarded. Sparse.
  - Ridge: minimize $\sum_{i=1}^{n}(y_i-\beta^T\mathbf{z}_i)^2$ s.t. $\sum_{j=1}^{p}\beta_j^2\le t$
    - correlated variables coefficients are pushed to the same value
  - Elastic Net: $\arg\min_{\beta}\|\mathbf{y}-\mathbf{X}\beta\|^2+\lambda_2\|\beta\|^2+\lambda_1\|\beta\|_1$
    - sparse solution, correlated variables grouped, enter/ leave the model together

Constraints for L1, L2 and L1&L2





LASSO Regression vs. Ridge Regression

GLM Best Practices

- Regularization Selection
  - Explore a few values for alpha, e.g. 0.01, 0.25, 0.5, 0.75, 0.99
- Wide Data Sets (10K+ columns)
  - Iteratively Reweighted Least Squares (IRLS) fails with lambda = 0
    - IRLS requires p x p Hessian matrix, where p = # of coefficients
    - Could use Limited-memory BFGS (L-BFGS)
  - IRLS + lambda search works and is recommended
    - Use alpha >> 0
    - Can produce 1K+ non-zero coefficients
  - L-BFGS + L2 penalty works
  - L-BFGS + L1 penalty works, but may take a long time

GLM Example

scikit-learn-like interface for modeling

```
y = "income"
x = ["age", "workclass", "fnlwgt", "education", "marital-status", "occupation", "relationship",
     "race", "sex", "capital-gain", "capital-loss","hours-per-week", "native-country"]
```

```
from h2o.estimators.glm import H2OGeneralizedLinearEstimator
glm_0 = H2OGeneralizedLinearEstimator(family = "binomial", lambda_search = True,
                                      nfolds = 5, seed = 123)
glm_0.train(x = x, y = y, training_frame = census_data, model_id = "income_glm_0")
```

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

```
from h2o.grid.grid_search import H2OGridSearch
glm_hyper_parameters = {"alpha": [0.5, 0.75, 1]}
glm_grid = H2OGridSearch(H2OGeneralizedLinearEstimator(family = "binomial", lambda_search = True,
                                                       nfolds = 5, seed = 123),
                         glm_hyper_parameters)
glm_grid.train(x = x, y = y, training_frame = census_data, grid_id = "income_glm_grid")
```

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

Supervised Learning:

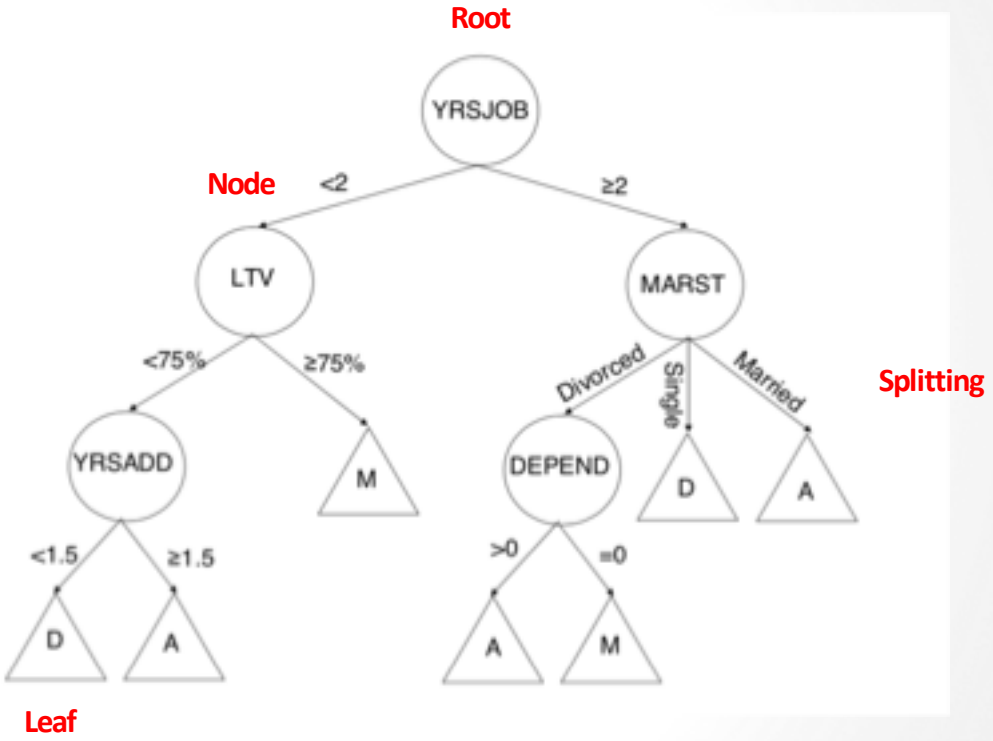
DECISION TREES (RECURSIVE PARTITIONING)

Decision Trees

- supervised learning

$$T : \mathbf{x} \rightarrow y$$

- classification or regression
- traditionally created manually
- tree-growing algorithms to optimize decision

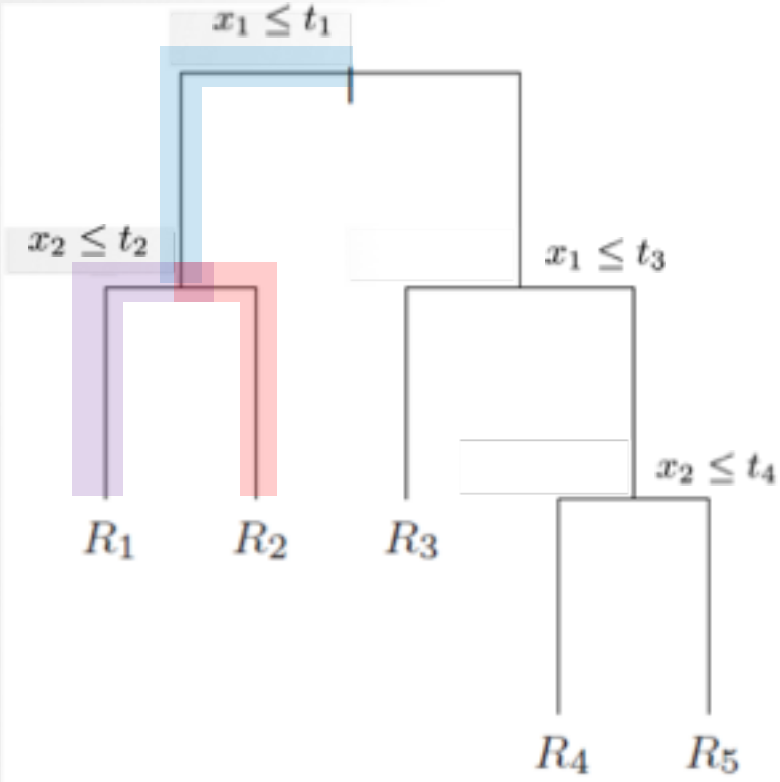


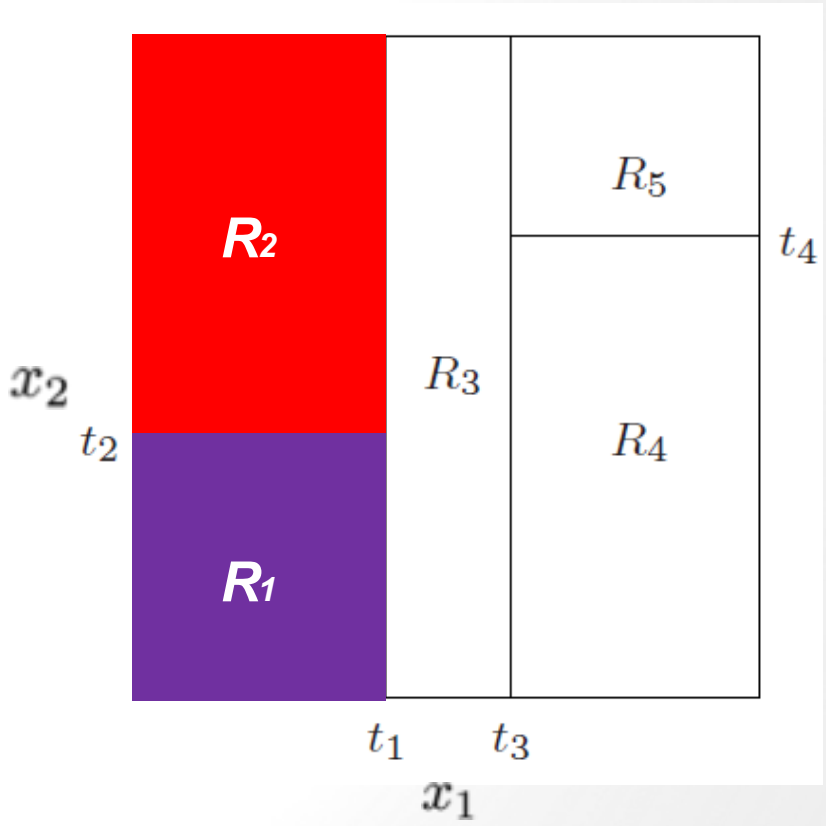
Example: Processing mortgage applications

YRSJOB: Years at current job
YRSADD: Years at current address
DEPEND: # of dependents
MARST: marital status
LTV: loan-to-value ratio

Decision Trees

- Decision tree is a recursive partition of the input space
- Most are *binary*



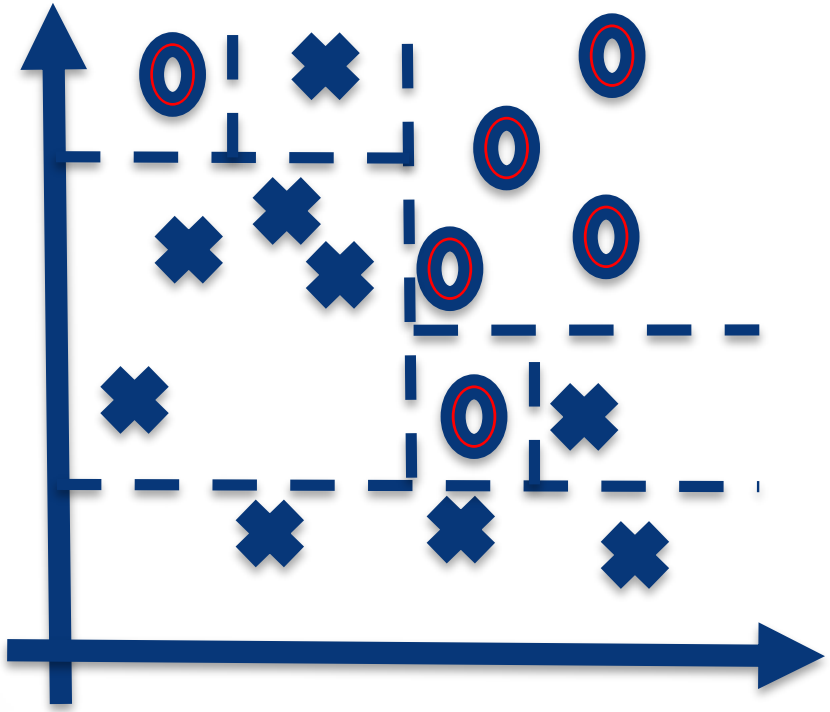


Decision Tree Algorithms

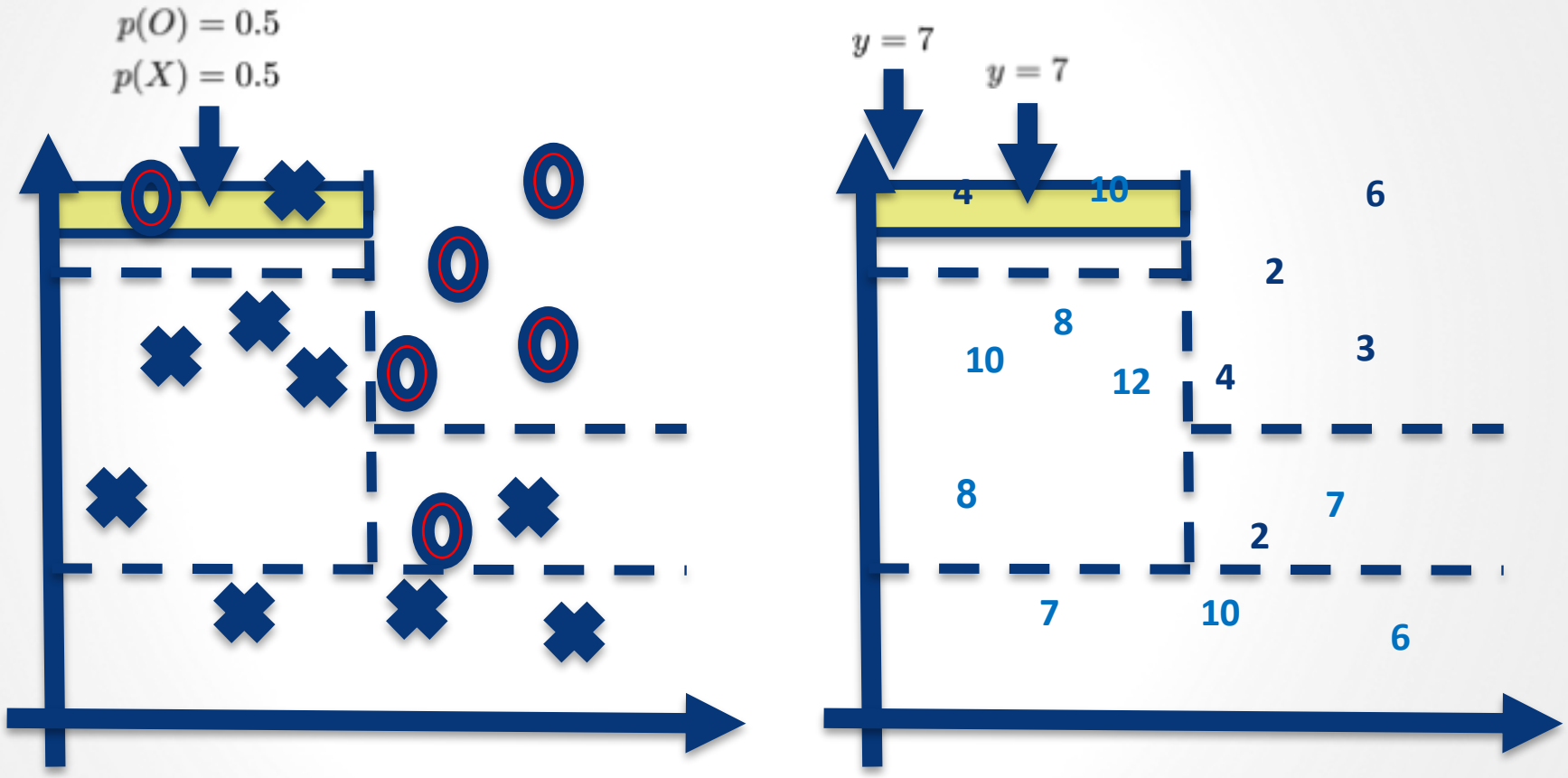
- Breaks sample data into homogenous pieces
  - Sample means
- Handles categorical and continuous data
- Identifies interaction effects and important variables
- Has multiple algorithms
  - ID3
  - C4.5
  - CHAID

```
TreeGrowing (S, A, y, SplitCriterion, StoppingCriterion)
Where:
S - Training Set
A - Input Feature Set
y - Target Feature
SplitCriterion - the method for evaluating a certain split
StoppingCriterion - the criteria to stop the growing process
Create a new tree T with a single root node.
IF StoppingCriterion(S) THEN
    Mark T as a leaf with the most
    common value of y in S as a label.
ELSE
    ∀a_i ∈ A find a that obtain the best SplitCriterion(a_i, S).
    Label t with a
    FOR each outcome v_i of a:
        Set Subtree_i = TreeGrowing (σ_{a=v_i} S, A, y).
        Connect the root node of t_T to Subtree_i with
                an edge that is labelled as v_i
    END FOR
END IF
RETURN TreePruning (S, T, y)
TreePruning (S, T, y)
Where:
S - Training Set
y - Target Feature
T - The tree to be pruned
DO
    Select a node t in T such that pruning it
        maximally improve some evaluation criteria
    IF t ≠ ∅ THEN T = pruned(T, t)
UNTIL t = ∅
RETURN T
```

Growing a Tree: Splitting



Decision Tree Inference



Growing a Tree: Splitting Criteria

- If starting with criterion to minimize, then learning an optimal decision tree is an NP-complete problem
- heuristic algorithms employ greedy procedure
- splitting criteria:

Classification: k classes

Entropy:

$$\sum_{i=1}^{k} \hat{p}_i \log(\hat{p}_i)$$

Gini:

$$\sum_{i=1}^{k} \hat{p}_i (1 - \hat{p}_i)$$

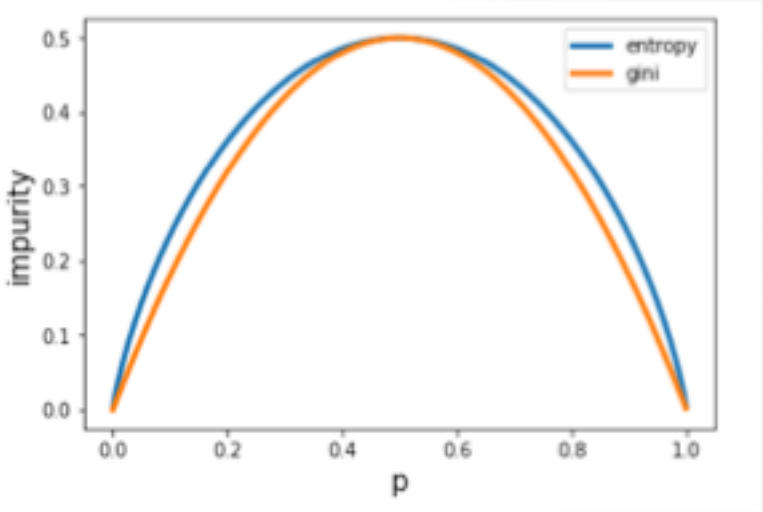
Squared error:

$$\sum_{i=1}^{n} (y_i - \hat{y})^2$$

Regression

Squared error

$$\sum_{i=1}^{n} (y_i - \hat{y})^2$$



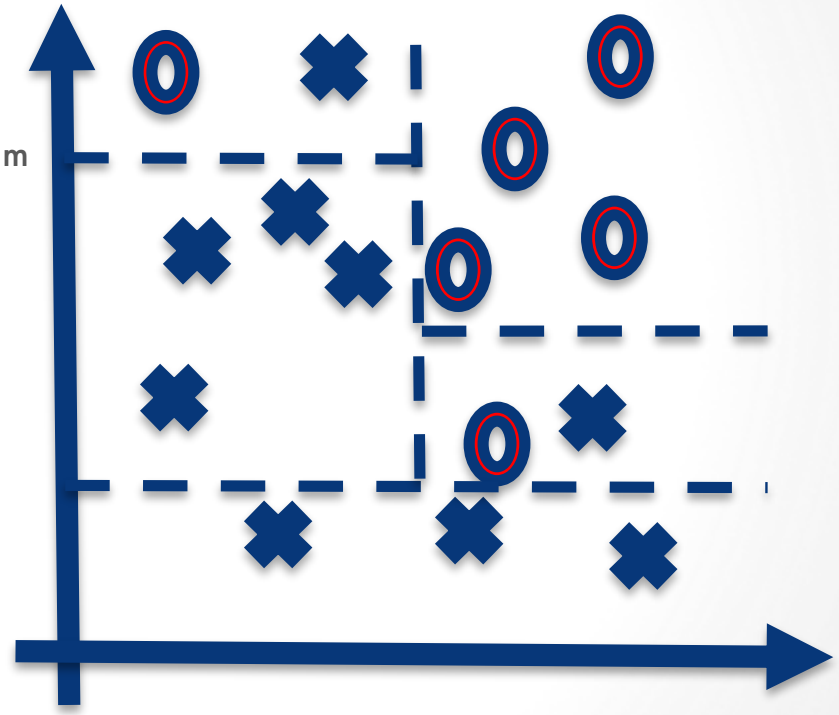
Growing a Tree: Stopping Criteria

1) All instances in leaf have the same y value

2) A maximum tree depth has been reached

3) Number of instances in leaf is below some minimum

4) Splitting criteria is below some threshold



Missing and Unseen Values

- Missing Data in Training Predictors: Treated as a “level”
- Missing Value in Training Target: Ignored
- Unseen Categorical Levels During Scoring: Treated as an “NA” value and classified with outliers

Pros/Cons of Decision Trees

simple to understand/interpret

little data prep
- natural handling of "mixed" data types
- handling of missing values
- handing of multi-class outputs

robustness to outliers in input space

insensitive to monotonic transformations of inputs

computational scalability (large N) $\mathcal{O}(n \log n)$

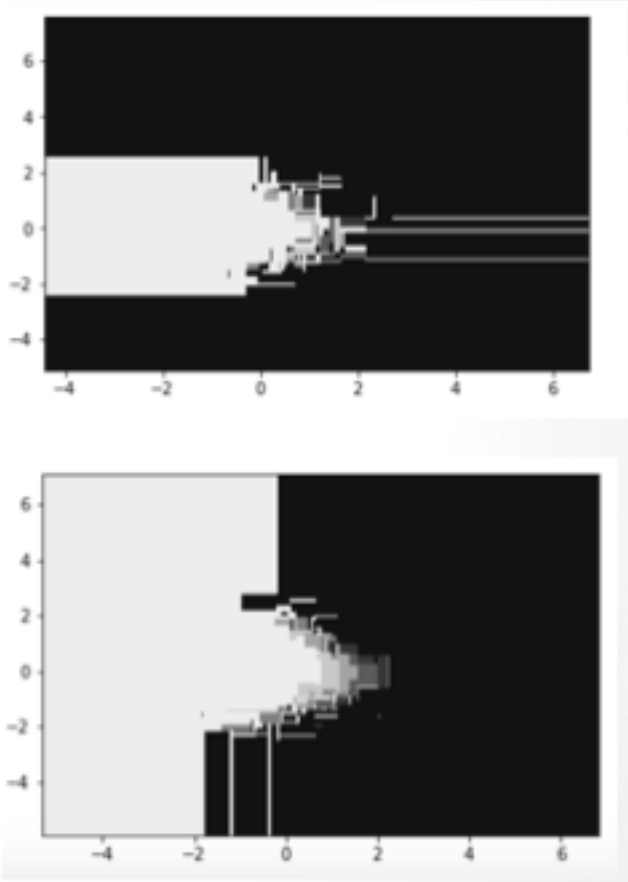
automatically ignores irrelevant inputs

✗ weak predictors

✗ can be unstable to small variations in the data

✗ poor ability to extract linear relationships

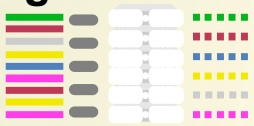
✗ can create biased trees if classes unbalanced



Exercise 1: Creating Decision Tree in sklearn

Scalable Implementation in H2O

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

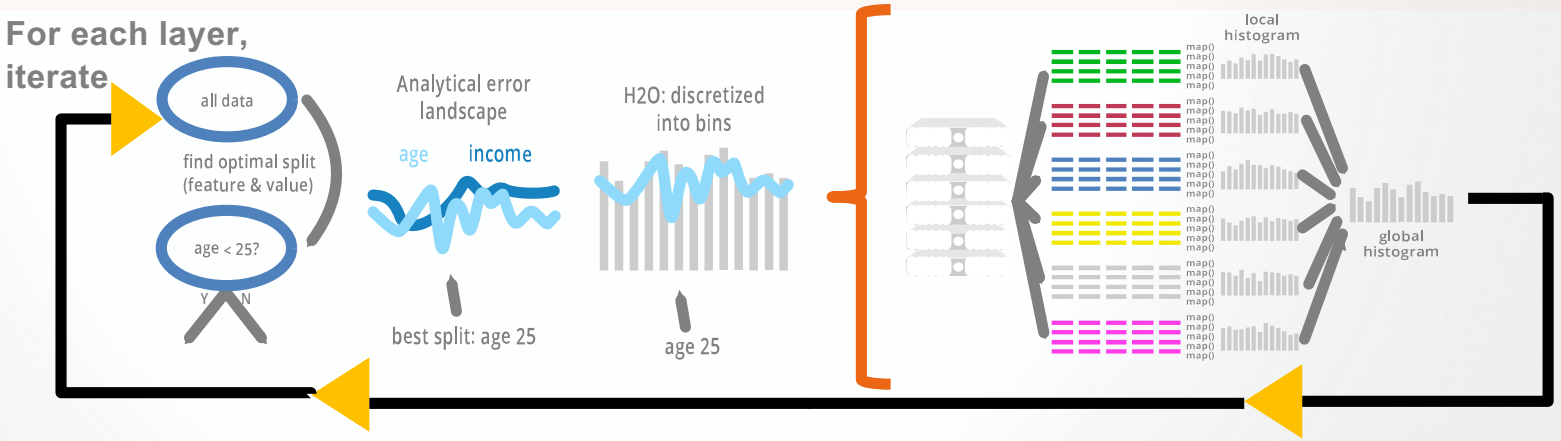
Basis for fine-grain Map/Reduce for histogram calculation

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

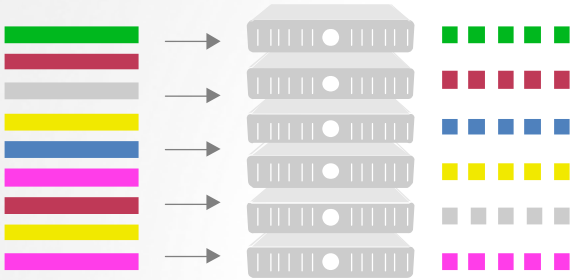
Start with root node and build layers of tree nodes **[ILLUSTRATION BELOW]**

For each layer, repeat the following:

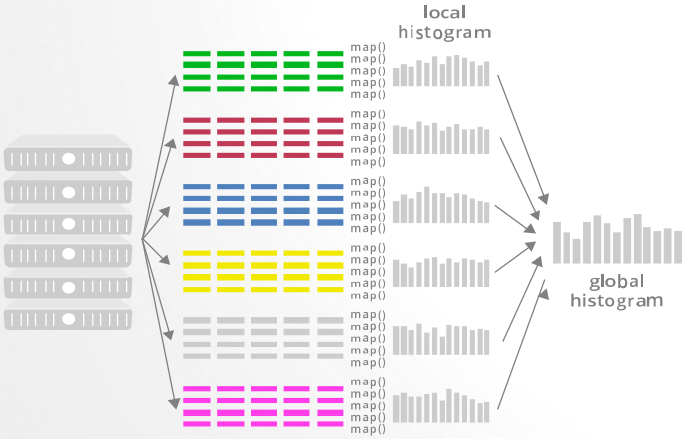
- For a set of features, split the data at every possible split point
- Find the split that leads to best model improvement
- Use discretization to limit the number of potential splits
  - To find the split, local histograms are calculated on each node and then aggregated into a global histogram
  - From the global histogram, the best split column is chosen



Scalable Implementation in H2O

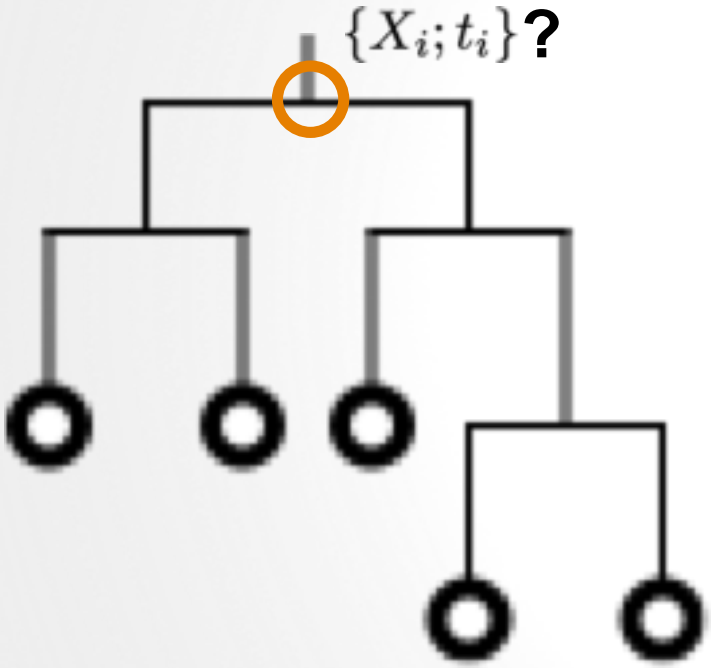


Parallel Parse into **Distributed Rows**

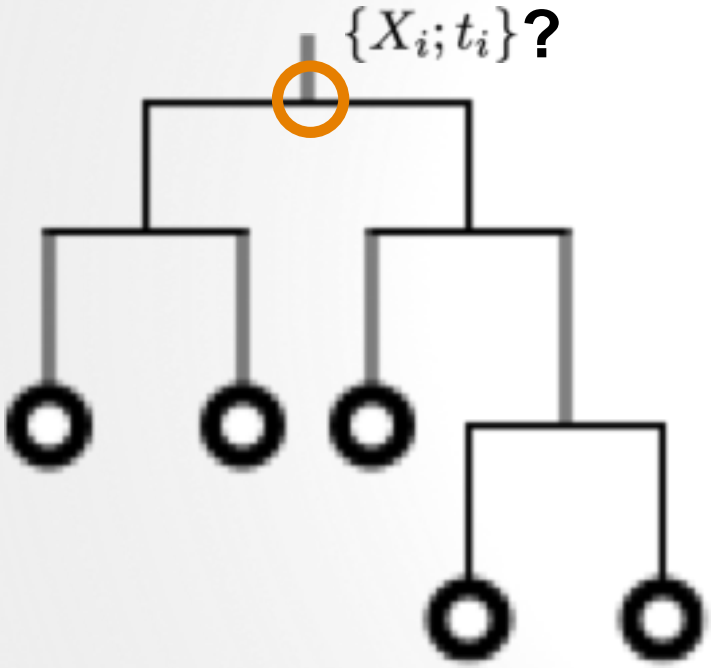


Fine Grain Map Reduce Illustration: Scalable Distributed Histogram Calculation for GBM

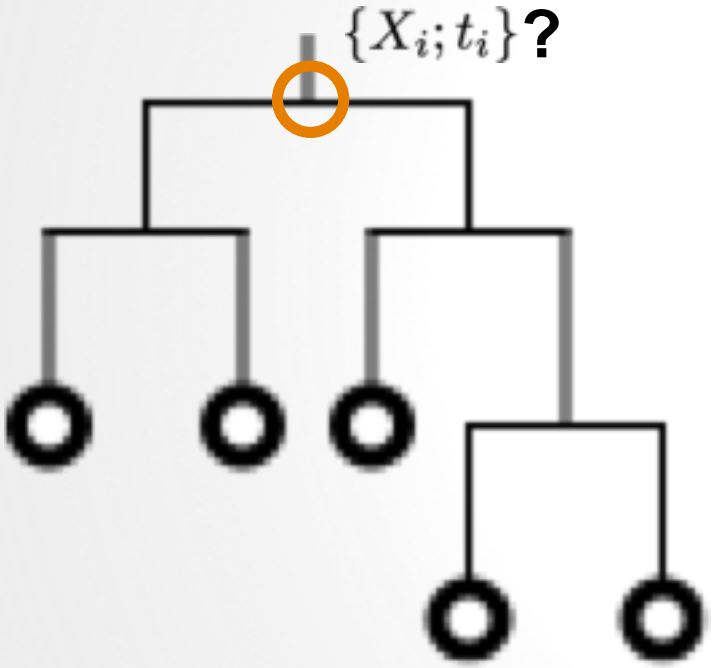
Tree Growth with Data Parallelism

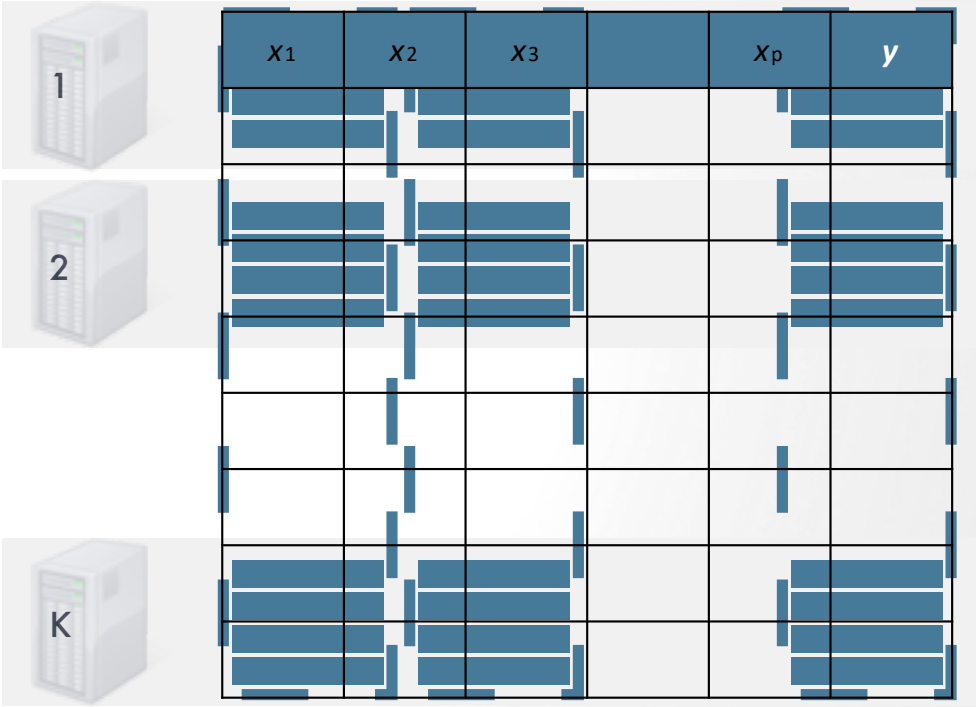


Tree Growth with Data Parallelism

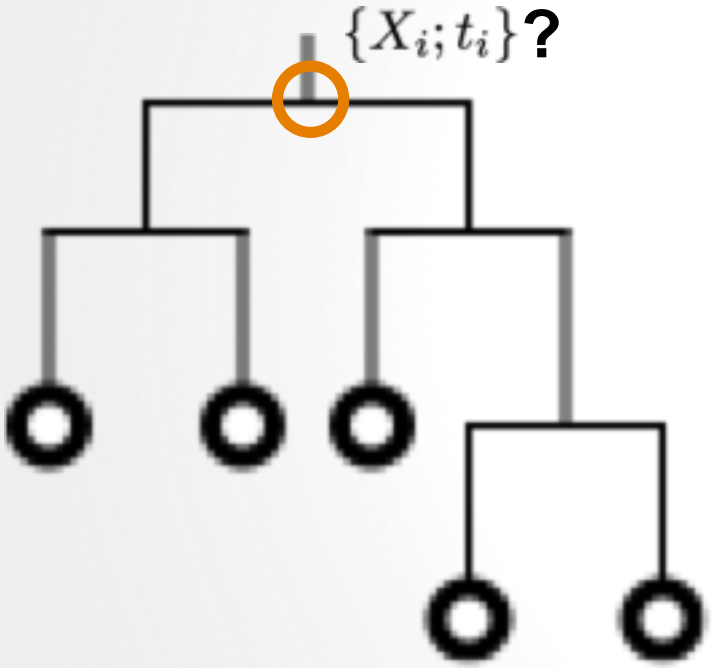


Tree Growth with Data Parallelism

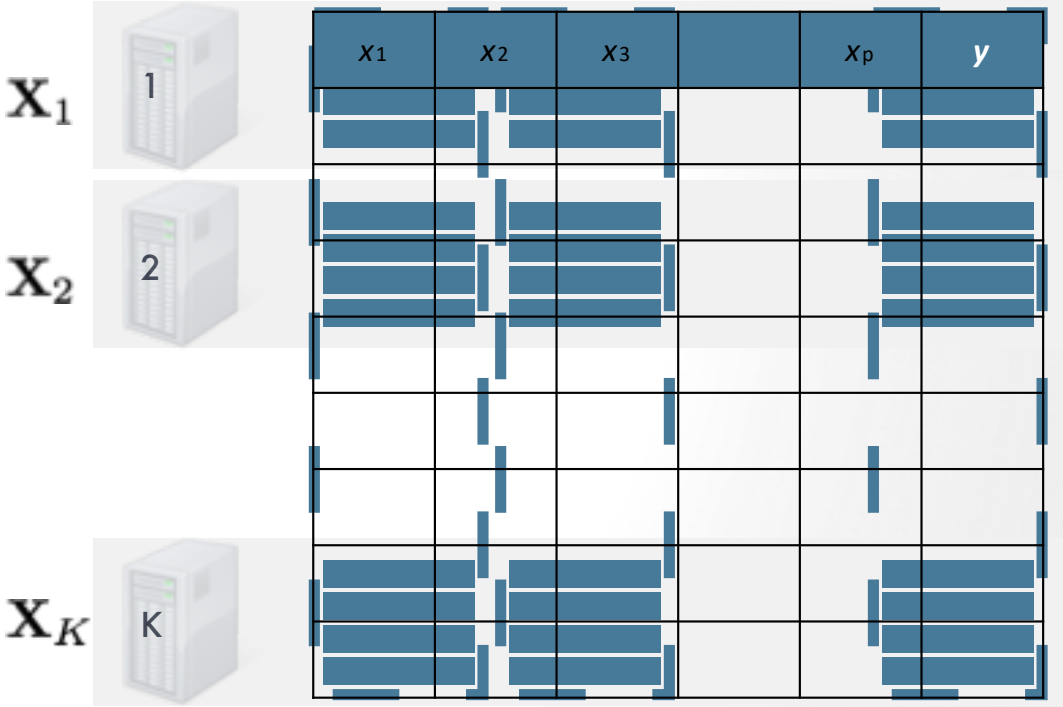




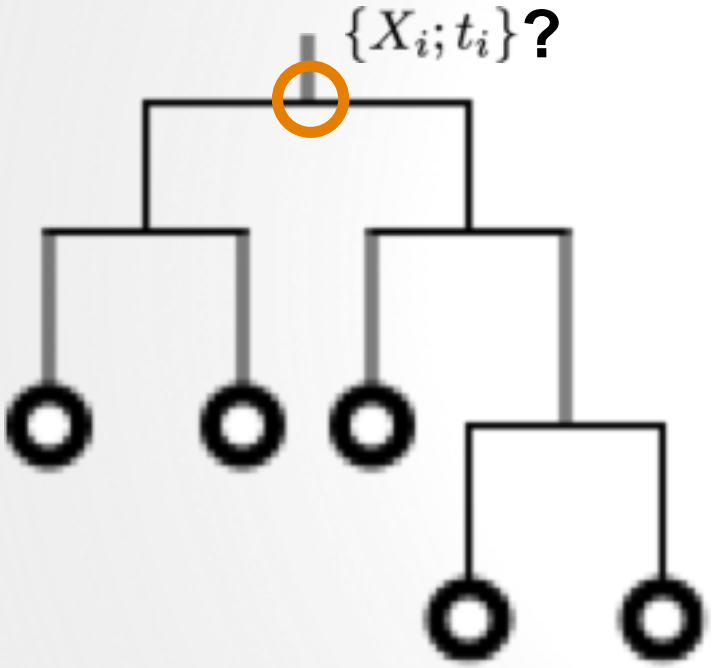
Tree Growth with Data Parallelism

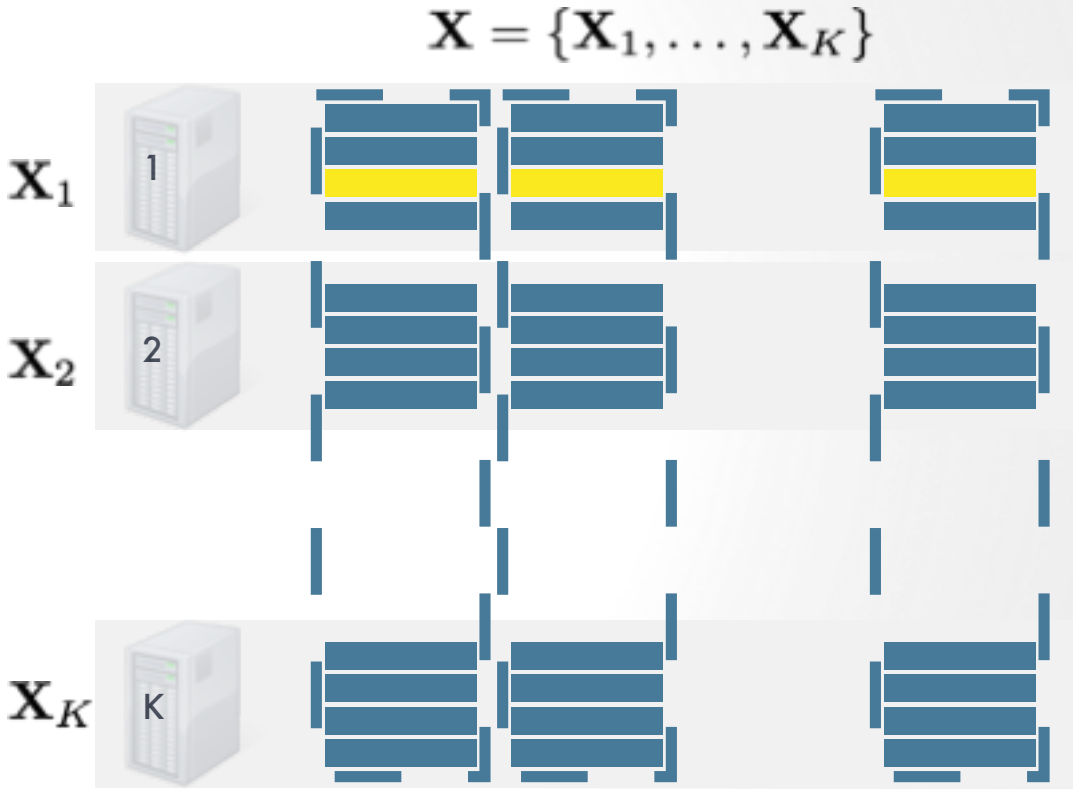


$$\mathbf{X} = \{\mathbf{X}_1, \ldots, \mathbf{X}_K\}$$

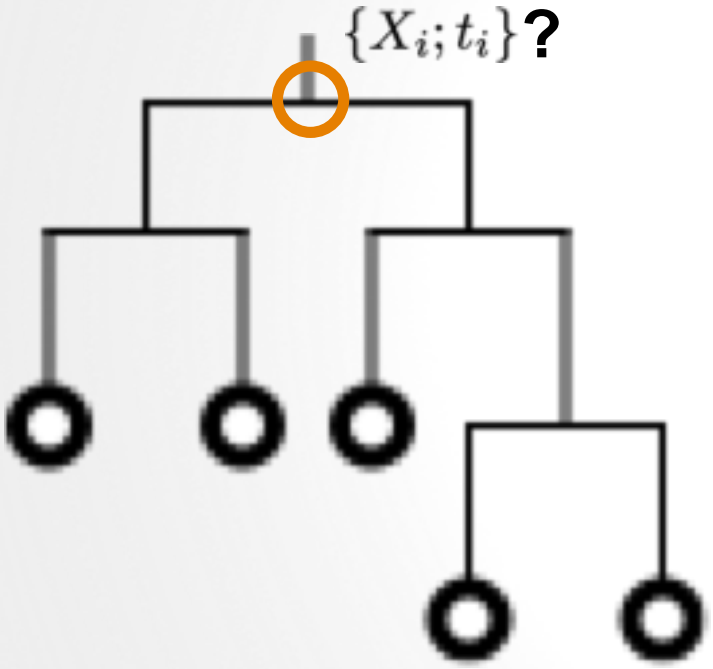


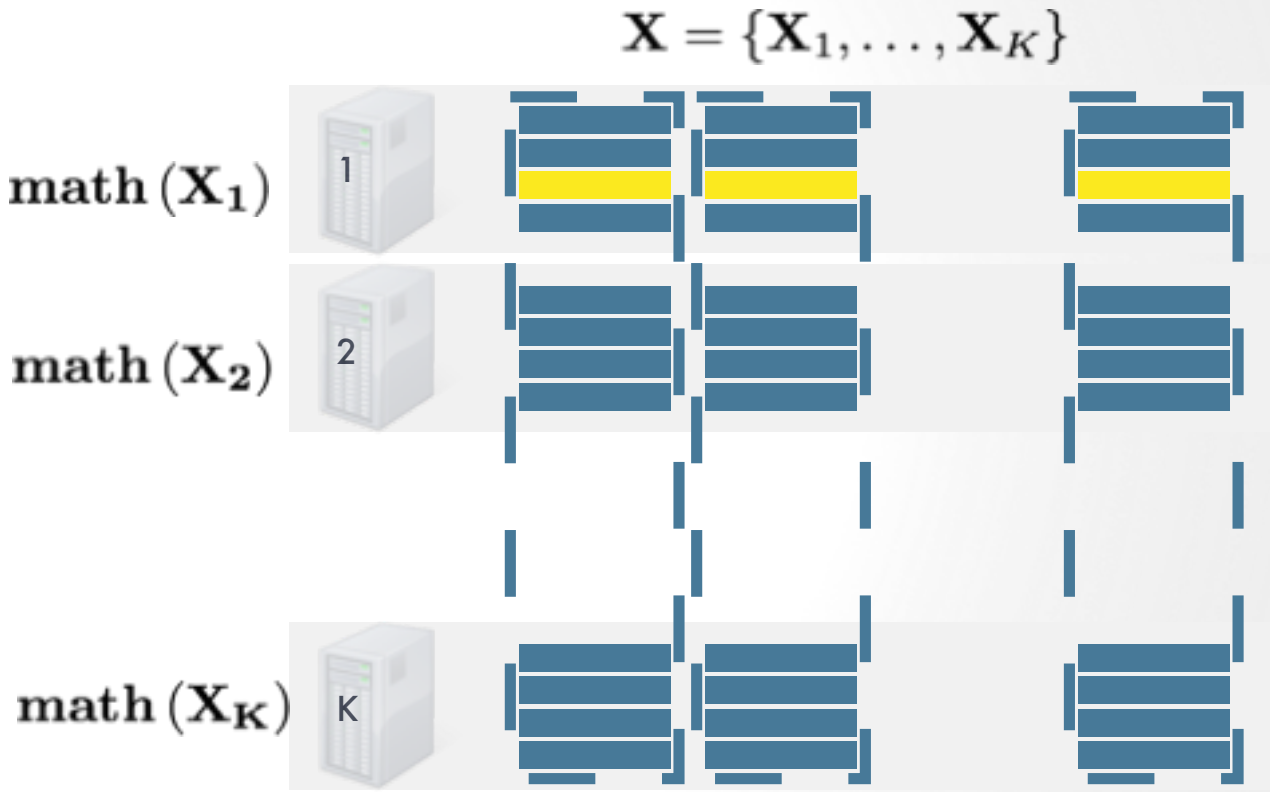
Tree Growth with Data Parallelism





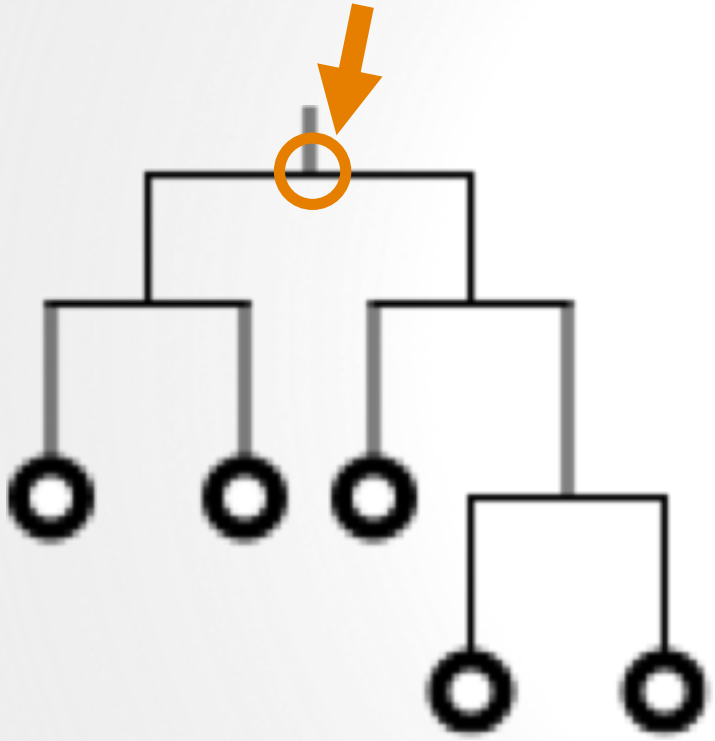
Tree Growth with Data Parallelism





Tree Growth with Data Parallelism

$$\{X_i; t_i\} = f(\mathbf{math}\,(\mathbf{X_1}), \ldots, \mathbf{math}\,(\mathbf{X_K}))$$



$$\mathbf{X} = \{\mathbf{X}_1, \ldots, \mathbf{X}_K\}$$

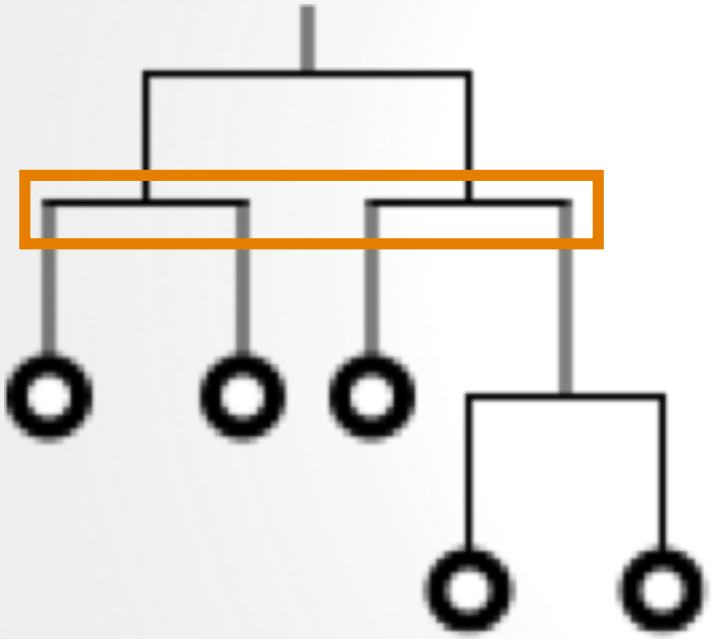
$\mathbf{math}\,(\mathbf{X_1})$

$\mathbf{math}\,(\mathbf{X_2})$

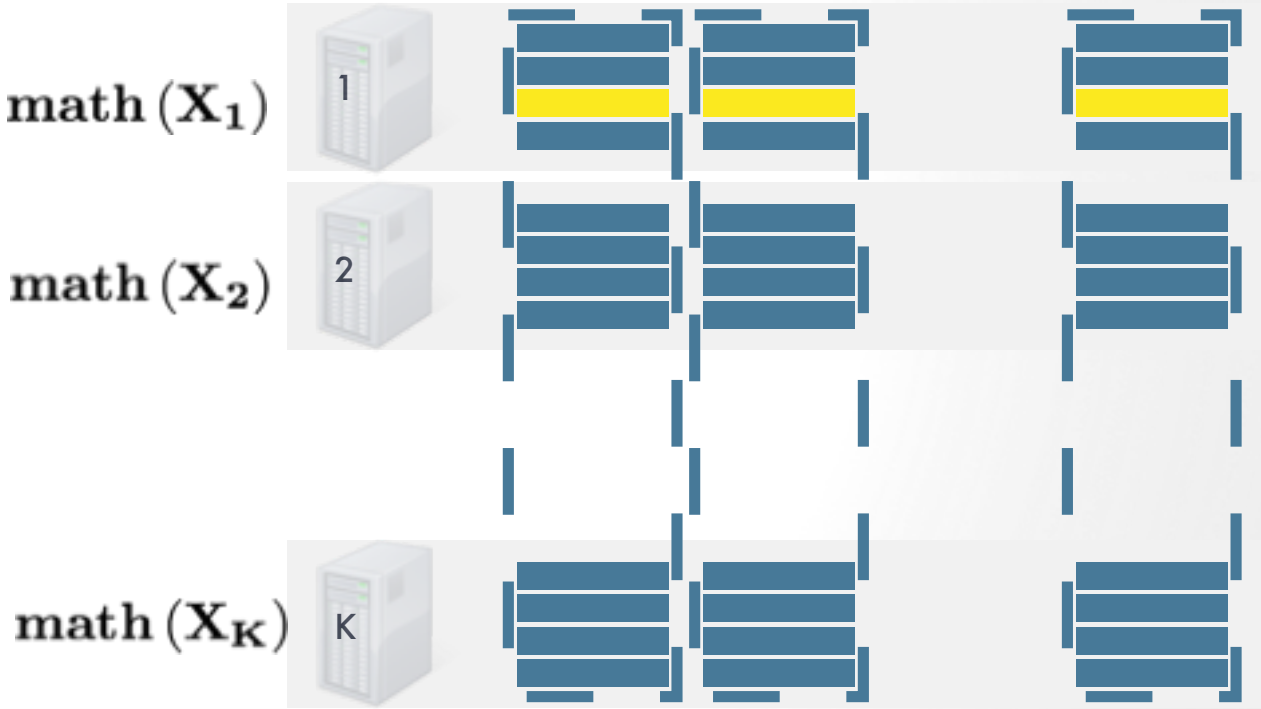
$\mathbf{math}\,(\mathbf{X_K})$

Tree Growth with Data Parallelism

$$\{X_i; t_i\} = f(\mathbf{math}\,(\mathbf{X_1}), \ldots, \mathbf{math}\,(\mathbf{X_K}))$$

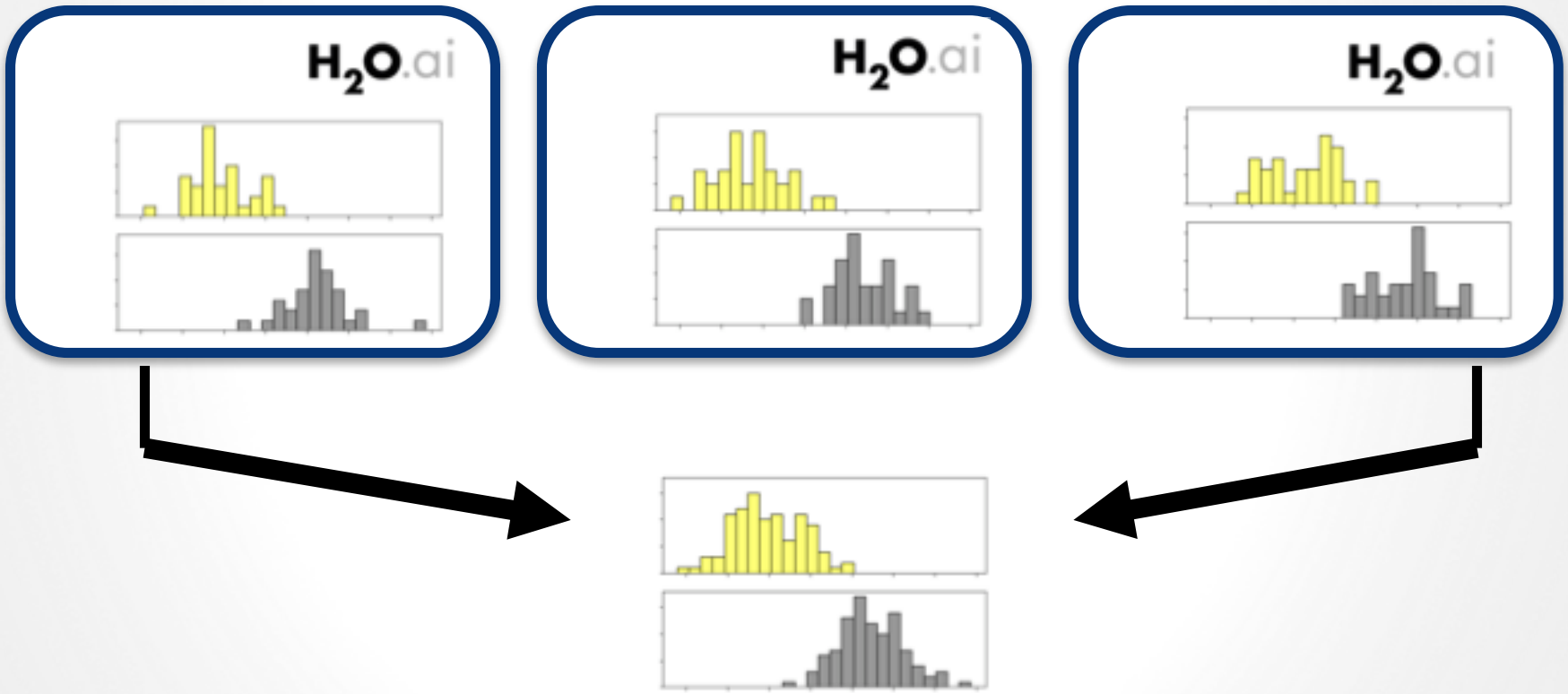


$$\mathbf{X} = \{\mathbf{X}_1, \ldots, \mathbf{X}_K\}$$

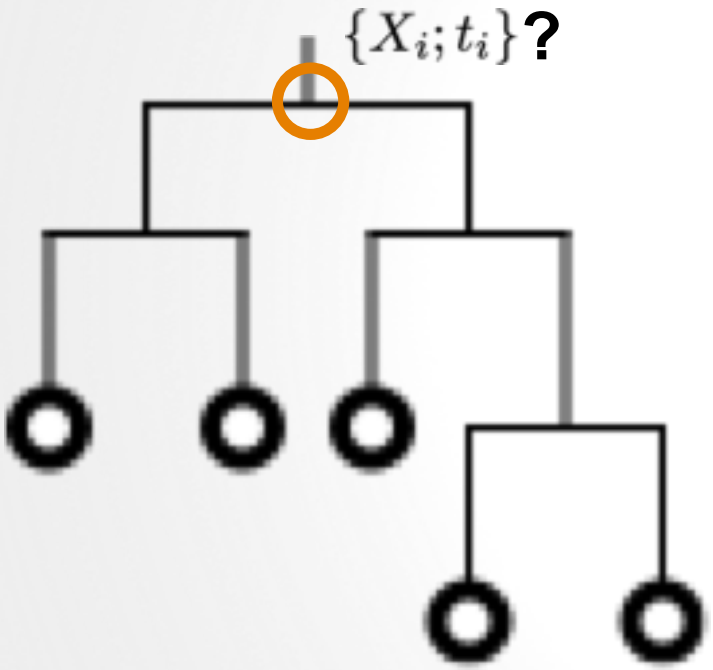


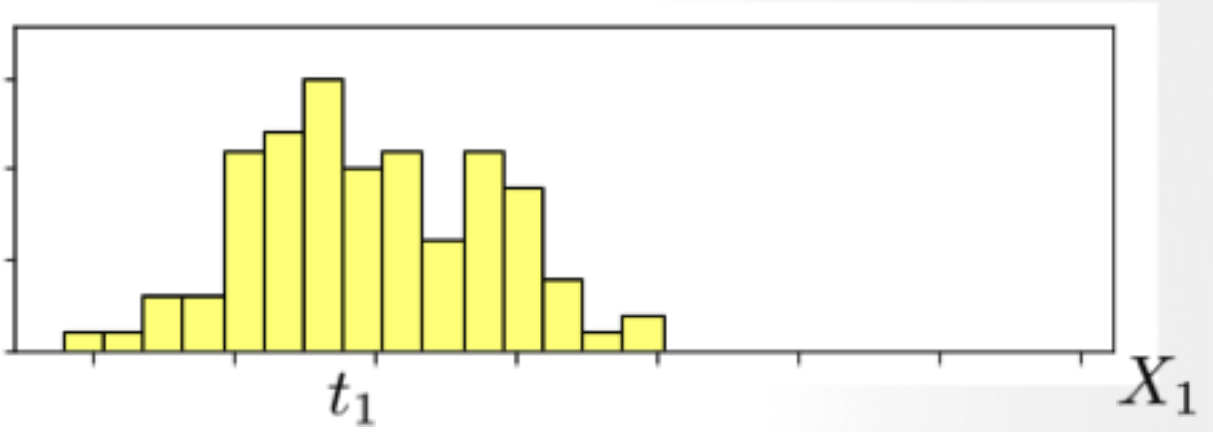
Full Data Parallelism for Each Level of Tree Growth!

Splitting with Distributed Data



Numerical Binning (Histogramming)





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

Categorical Binning

- Lexigraphical ordering (i.e. alphabetical ordering)

- Example: {Red, Blue, Yellow, Orange, Purple, Green}
  - Lexigraphical 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}

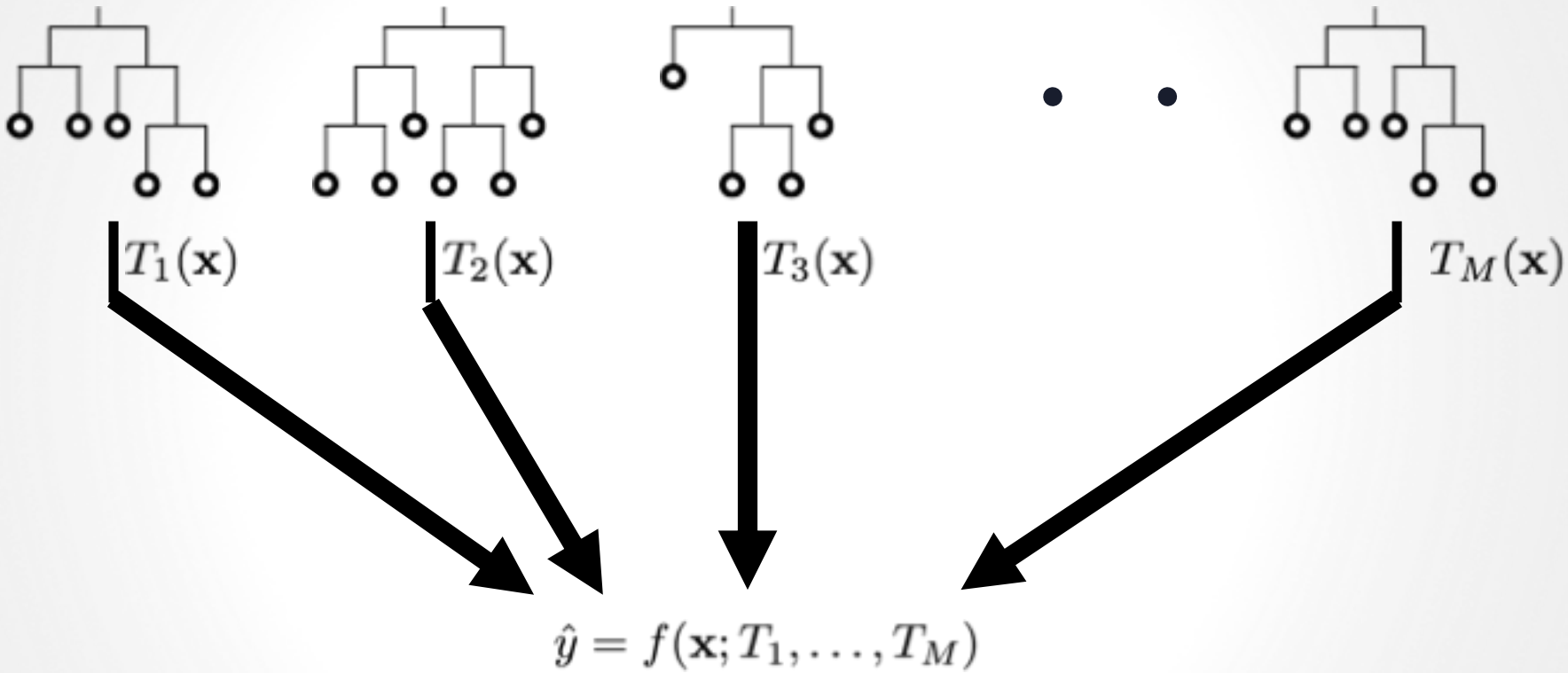
Binning in H2O Decision Trees

- Binning Numeric Features
  - Traditionally split points are chosen by sorting the each feature and inspecting an induced split.
  - For big data even when running parallel and distributed this can be computationally expensive so we approximate sorting with binning.
  - **More Bins, More Accurate** The number of bins can be specified by the user and it is the minimum number of bins required in a histogram built for each feature.
- Binning Categorical Features
  - **High Cardinality Features** slow model builds by inducing splits by each level.
  - Bin the levels in a categorical column according to “nbins_cat” parameter.
  - **More Bins, More Likely To Overfit** Increasing the number of bins can lead to splits on a single category, which can lead to overfitting.

Supervised Learning:

TREE ENSEMBLES

Ensemble of Trees

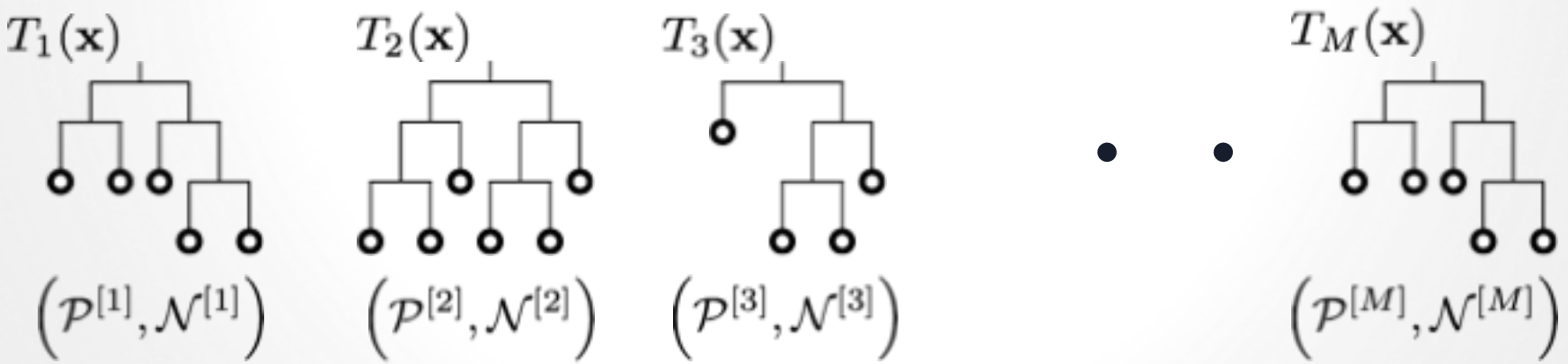


Regularization: Observation and Feature Sampling

- For each tree, select from *P* features (columns) and *N* rows

$$\mathcal{P} \in \{1, 2, \ldots, P\} \qquad \mathcal{N} \in \{1, 2, \ldots, N\}$$

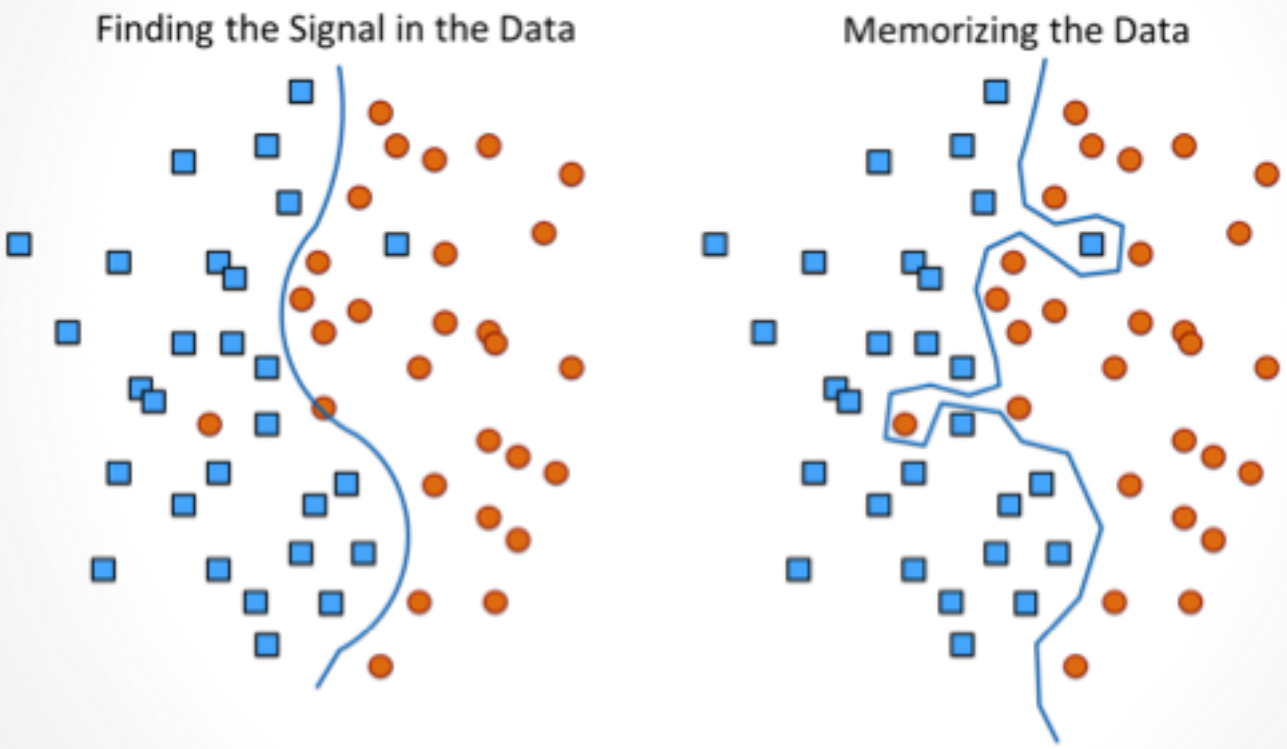
$$\mathcal{P}^{[i]} \subseteq \mathcal{P},\ |\mathcal{P}^{[i]}| = P^{[i]} = \gamma_{\text{COL}} P \qquad \mathcal{N}^{[i]} \subseteq \mathcal{N},\ |\mathcal{N}^{[i]}| = N^{[i]} = \gamma_{\text{ROW}} N$$



Scoring and Stopping

- How often should we check our validation error? (Computation time versus generalization)
  - **`score_each_iteration`**: score model after each tree
  - **`score_tree_interval`**: score model after n trees

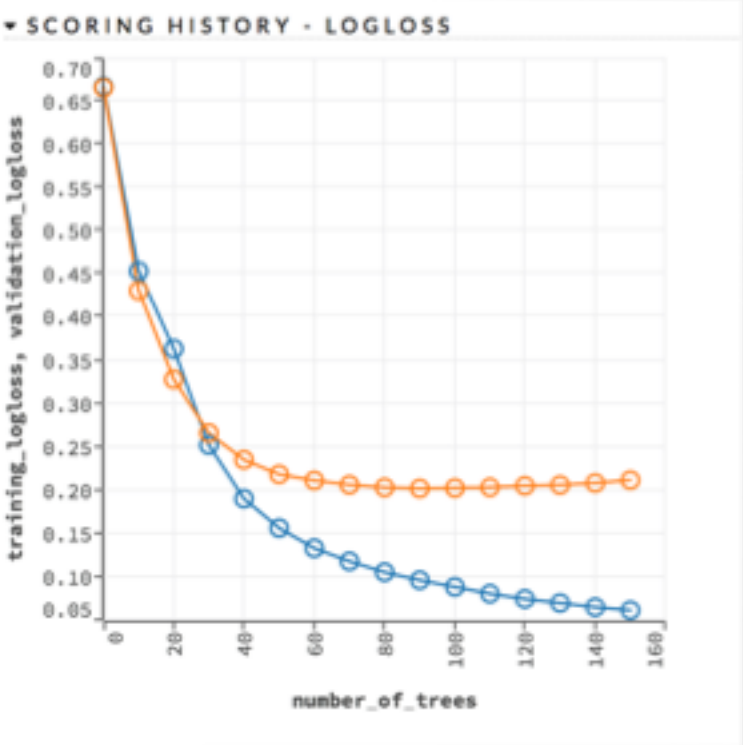
- Setting criteria for stopping
  - **`stopping_rounds`**: early stop if stopping metric's moving average does not improve for this many rounds
  - **`stopping_metric`**: metric for early stopping
  - **`stopping_tolerance`**: Relative tolerance for metric-based stopping criterion (stop if relative improvement is not at least this much)
  - **`max_runtime_secs:`** maximum runtime to allow for model building


Finding the Signal in the Data
Memorizing the Data

Early Stopping

Overfitting

Early Stopping



Cross-Validation

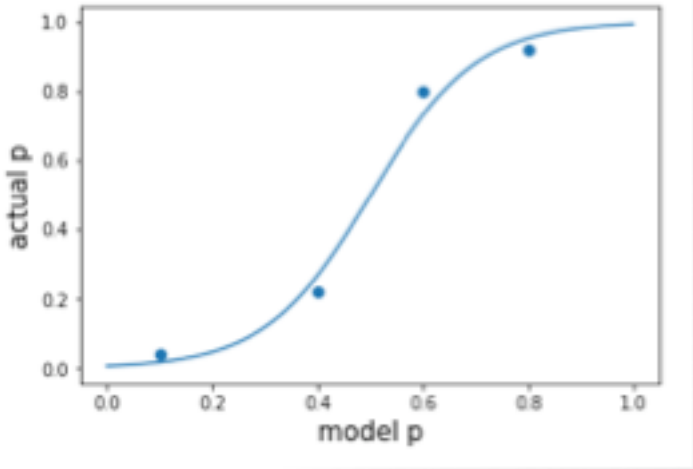
- **`nfolds`**: number of folds for N-fold cross-validation (default: 0, disabled)
- **`keep_cross_validation_predictions`**: keep the predictions of the cross-validation models.
- **`keep_cross_validation_fold_assignment`**: keep the cross-validation fold assignment.
- **`fold_assignment`**: cross-validation fold assignment scheme, if fold column is not specified. The ”Stratified” option will stratify the folds based on the response variable, for classification problems. Must be one of: ”AUTO”, ”Random”, ”Modulo”, ”Stratified”.
- **`fold_column`**: column with cross-validation fold index assignment per observation.

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:

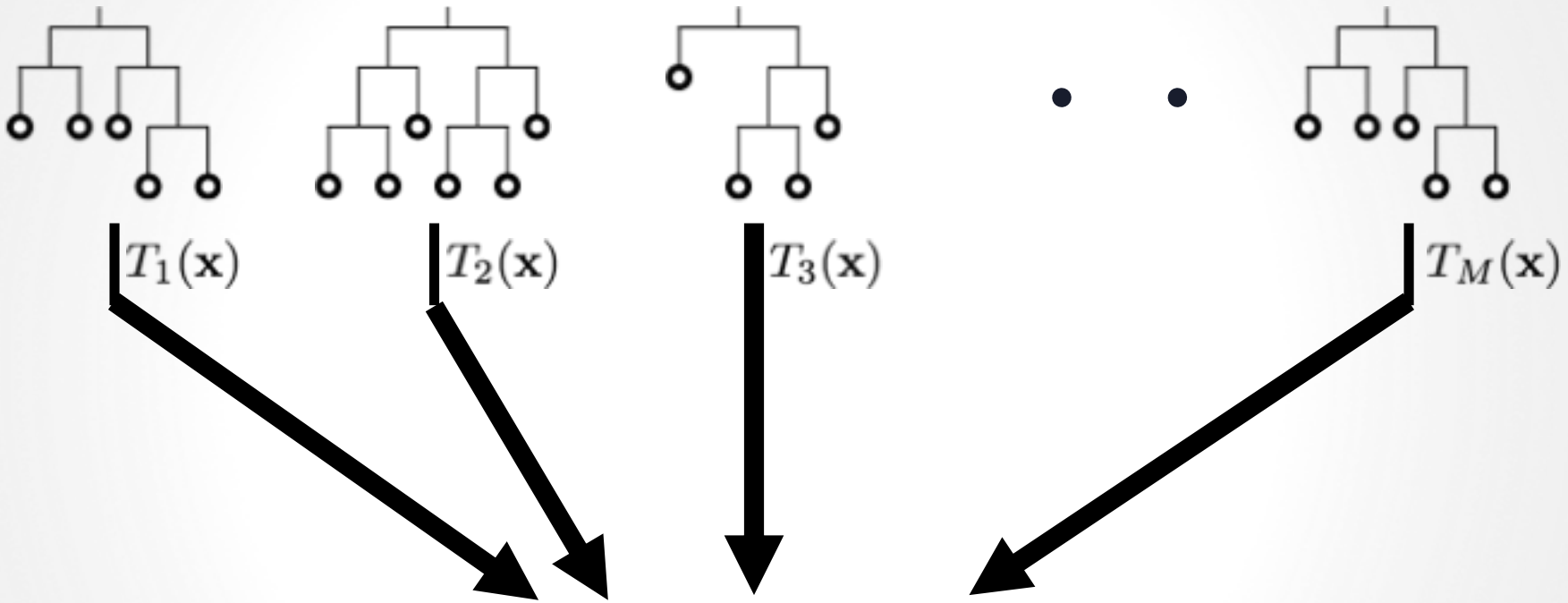
$$P(\mathbf{x}) = \frac{1}{1 + \exp(Af(\mathbf{x}) + B)}$$

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

Supervised Learning:

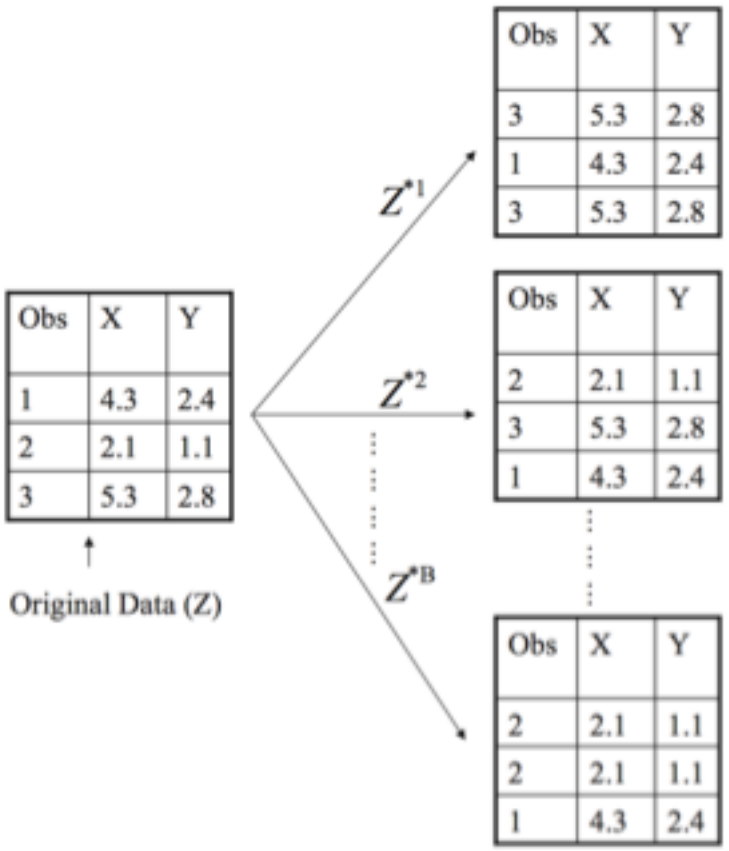
RANDOM FORESTS

Random Forest



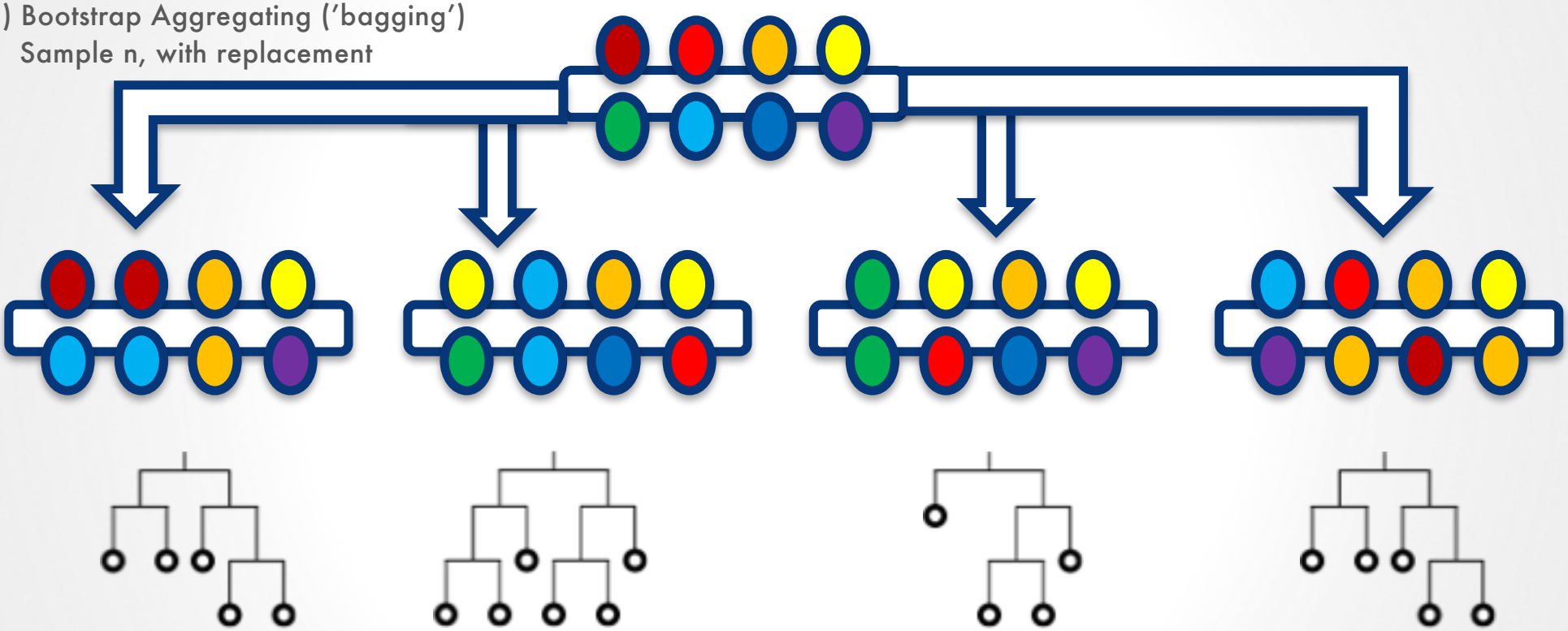
$$f_M(\mathbf{x}) = \sum_{i=1}^{M} T_i(\mathbf{x})$$

The Bootstrap



RF Tree Variance: Bagging

1) Bootstrap Aggregating ('bagging')
   Sample n, with replacement



$$f_M(\mathbf{x}) = \sum_{i=1}^{M} T_i(\mathbf{x})$$

RF Tree Variance: Splits

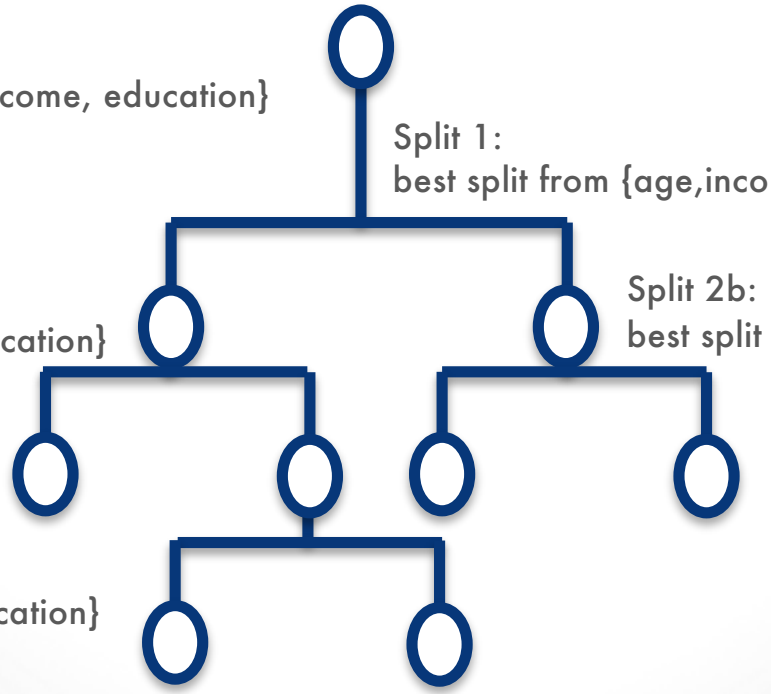
2) Do not consider all of the features for each split

- default is often $\sqrt{n}$
- also speeds up computational time

Additional methods for variance:

- do not consider all features for each *tree*
- H2O: random histograms

Example:
features = {age, sex, income, education}

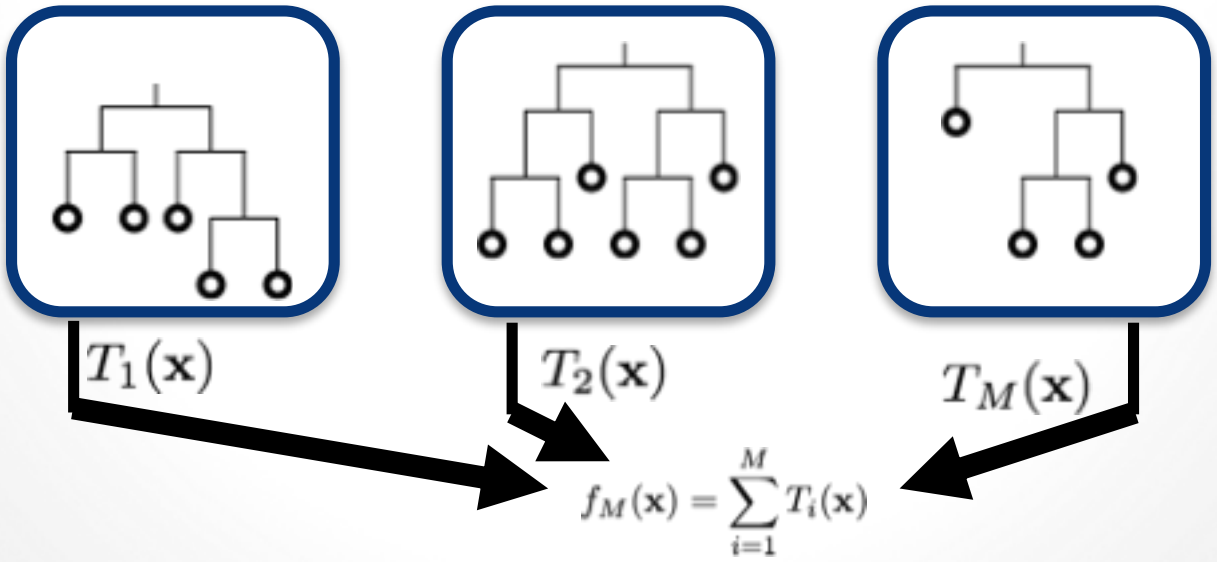


Distributed RFs

Method 1: Parallelize by tree

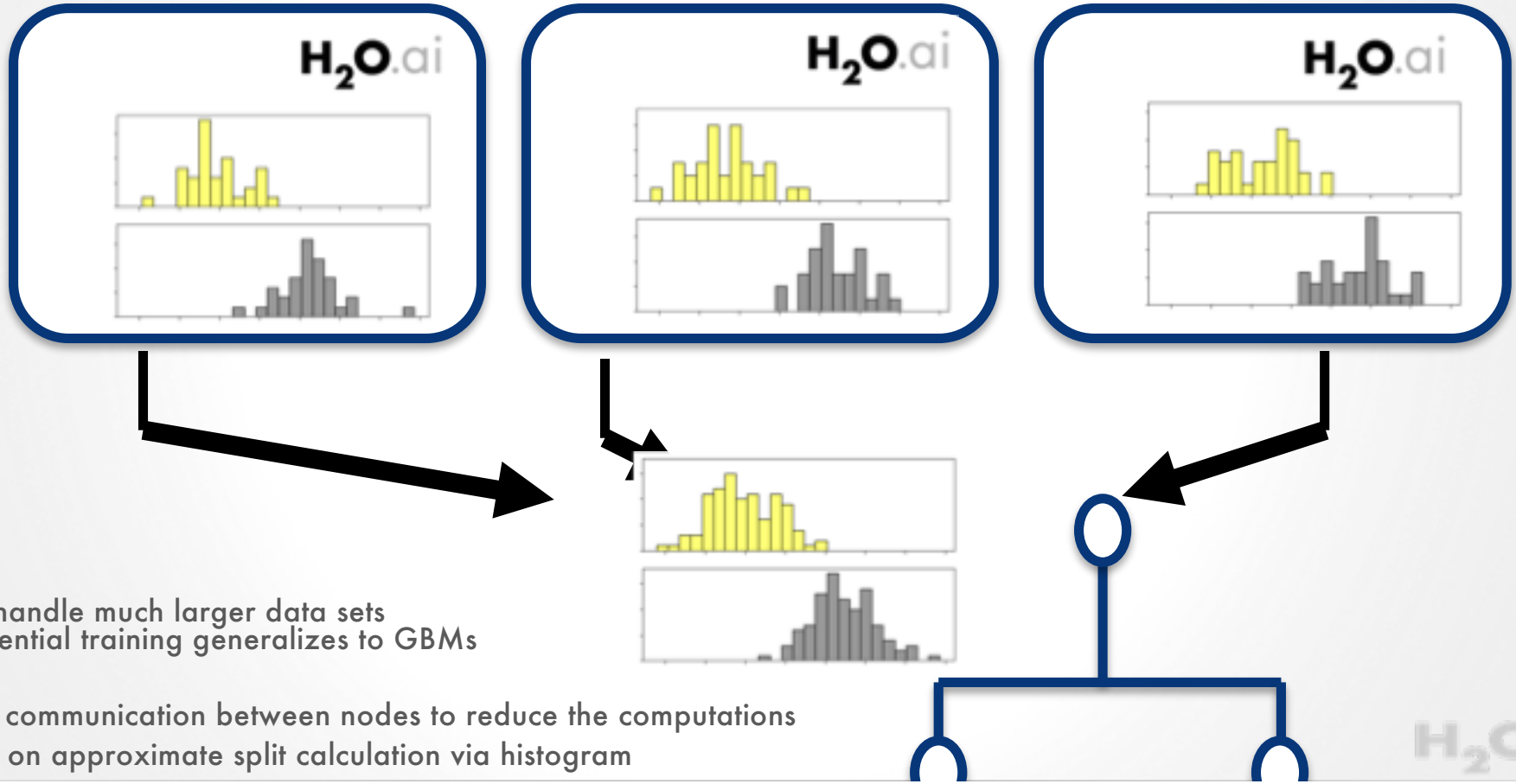
low communication between nodes (only to reduce as part of map/reduce)

✗ every node must have all data in memory
✗ does not generalize to GBMs



Distributed RFs in H2O

Method 2: Parallelize by data



can handle much larger data sets
sequential training generalizes to GBMs

✗ more communication between nodes to reduce the computations
✗ relies on approximate split calculation via histogram

RF Parameters: Setting Up a Model

Python API

```
h2o.estimators.random_forest.H2ORandomForestEstimator
- h2o.estimators.estimator_base.H2OEstimator
-- h2o.model.model_base.ModelBase
```

`model_id:`	specify a custom name for the model to use as a reference. (default: randomly generated)
`checkpoint:`	load a previously generated model
`seed:`	specify a pseudorandom seed, for reproducibility
`verbose:`	print verbose scoring history to console

RF Parameters: Defining Data

- `training_frame:` H2OFrame used to train model
- `validation_frame:` H2OFrame used to validate model (optional)
- `y:` column name or index of target variable
- `x:` column names (of indices) of predictor variables (default: all but y and ignored_columns)

- `ignored_columns:` columns to ignore (e.g. ID column)
- `ignore_const_cols:` ignore constant columns (default: True)
- `weights_column:` column by which to weight the data points

- `categorical_encoding:` encoding scheme for categorical variables: {"enum", "one_hot_explicit", "binary", "eigen", "label_encoder", "sort_by_response"} (default: "enum")

RF Parameters: Loss Function

- `distribution:` specifies the loss function
  - `'bernoulli'`
  - `'multinomial'`
  - `'gaussian'`
  - `'poisson'`
  - `'gamma'`
  - `'laplace'`
  - `'quantile'`
  - `'huber'`
  - `'tweedie'`

- `huber_alpha, quantile_alpha`

Choosing a Distribution Function For Numeric Response

Distribution	Loss	Details
Gaussian	Squared Error Loss	Sensitive to outliers
Laplace	Absolute Error Loss	Very robust to outliers
Huber	Hybrid of Squared and Absolute Error Loss	Robust to outliers
Poisson	Poisson Loss (Deviance)	Used for estimating counts
Gamma	Gamma Loss	Used for estimating total values
Tweedie	Tweedie Loss	Used for estimating densities
Quantile	Quantile Regression Loss	Used for estimating a specified percentile

RF Parameters: Ensemble

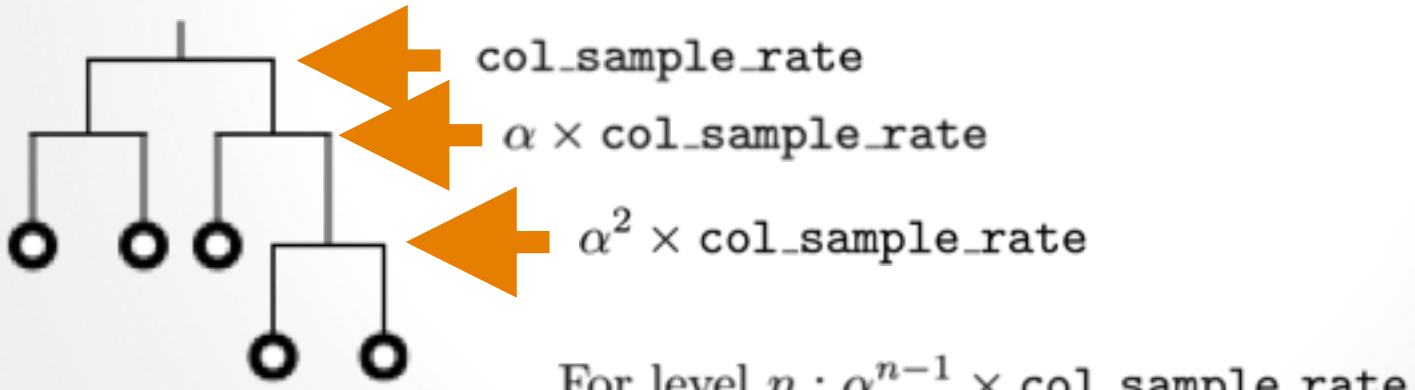
- `ntrees:` number of trees

- `sample_rate:` row sampling rate per tree (default: 0.6320000291)
- `col_sample_rate_per_tree:` column sampling rate per tree (default: 1)

RF Parameters: Individual Trees

- **Column sampling for split:**
- **mtries:** number of columns to sample on each split
  (default: sqrt(n) for classification, n/3 for regression)
- **col_sample_rate_change_per_level:** factor by which to increase or decrease mtries per level of tree

$$\alpha = \texttt{col\_sample\_rate\_change\_per\_level}\ (0 \leq \alpha \leq 2)$$



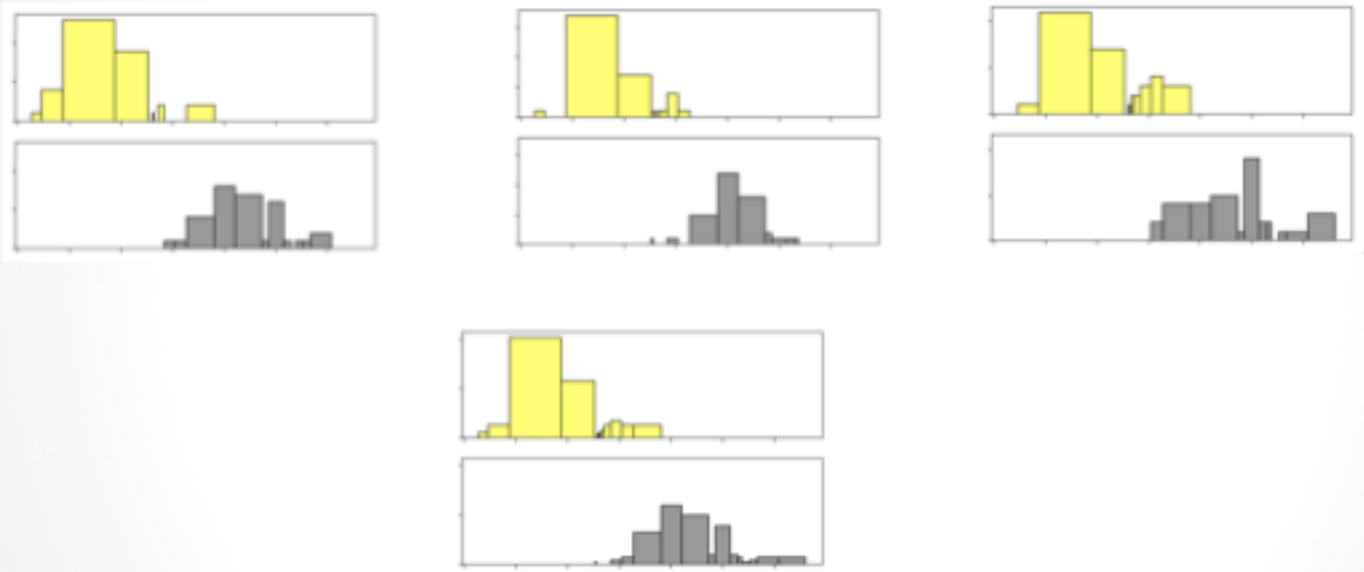
RF Parameters: Individual Trees

- **Column sampling for split:**
- **mtries:** number of columns to sample on each split (default: sqrt(n) for classification, n/3 for regression)
- **col_sample_rate_change_per_level:** factor by which to increase or decrease mtries per level of tree

- **When to stop splitting?**
- **max_depth:** maximum depth of each tree
- **min_rows:** minimum rows in a leaf (i.e. stop splitting when data size is this small)
- **min_split_improvement:** **minimum relative improvement** in split criterion for a split to occur

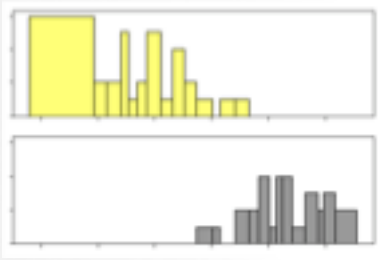
- **Histogramming**
- **nbins**: number of bins for numeric variables (default: 20)
- **nbins_top_level**: can be used instead of nbins; nbins will then decrease by 2 each level
- **nbins_cats**: number of bins for categorical variables (default: 1024)
- **histogram_type**: method for binning {"Uniform Adaptive", "Random", "QuantilesGlobal", "RoundRobin"}

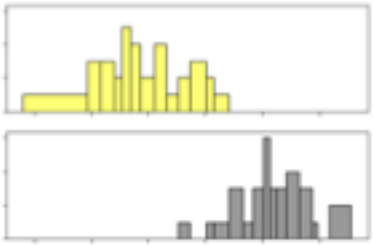
Random Forests in H2O

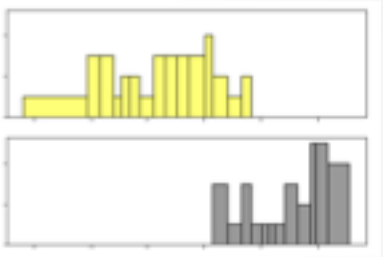


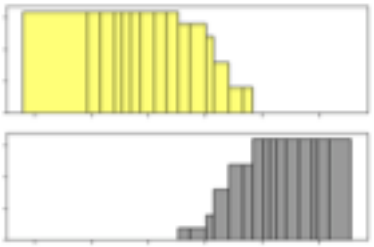
```
histogram_type = Random
```

Random Forests in H2O







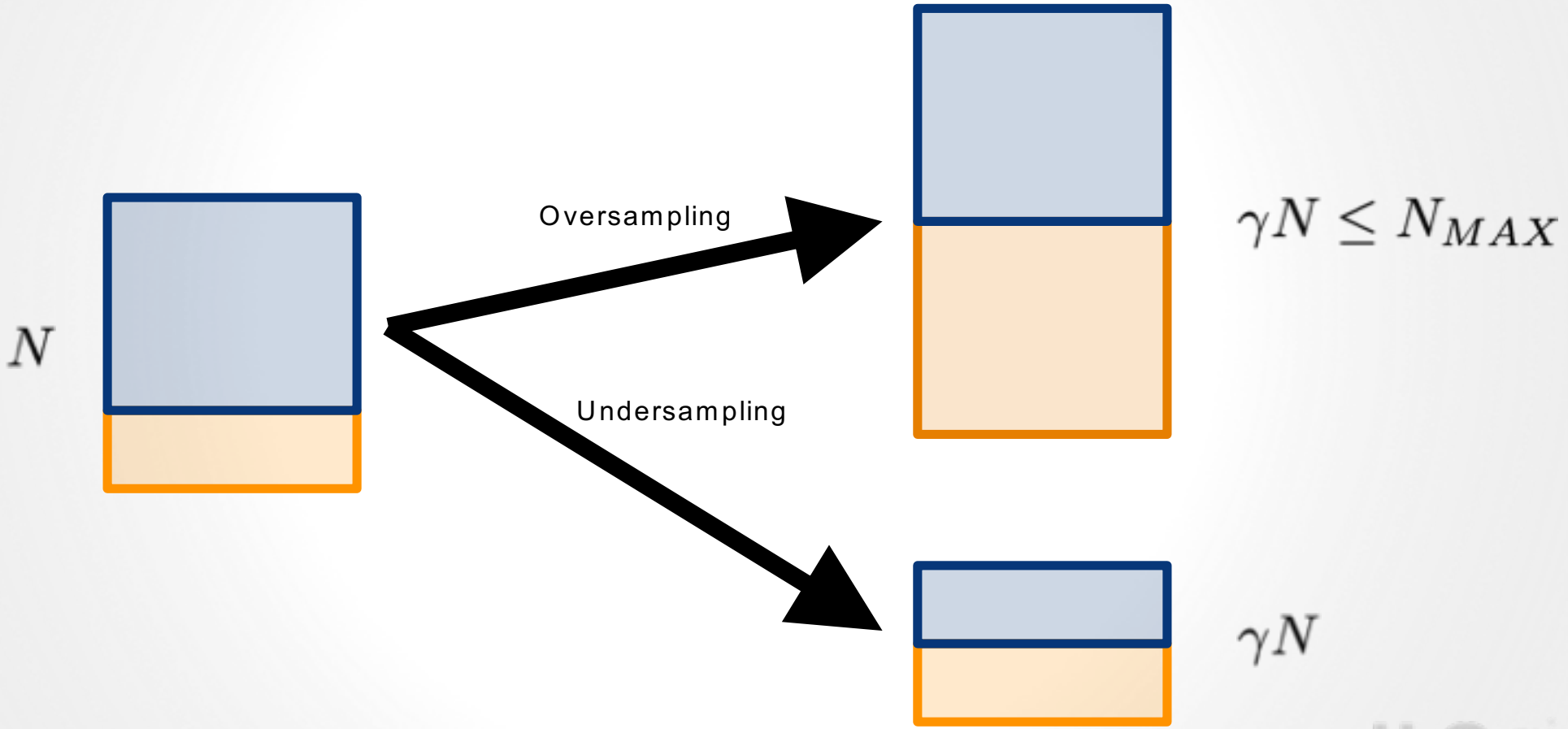


```
histogram_type = Quantiles
```

RF Parameters: Class Imbalances

- **`balance_classes`**: balance training data class counts via over/under-sampling (default: False)
- **`class_sampling_factors`**: desired over/under-sampling ratios per class (in lexicographic order). If not specified, sampling fac- tors will be automatically
- **`max_after_balance_size`**: maximum relative size of the training data after balancing class counts (can be less than 1.0). Requires balance classes. Defaults to 5.0.
- **`sample_rate_per_class:`** variable row sampling rate per class

RF Parameters: Class Imbalances



RF Parameters: Scoring & Stopping

- **score_each_iteration**: score model after each tree (default: false)
- **score_tree_interval**: score model after n trees

- **stopping_rounds**: early stop if stopping metric's moving average does not improve for this many rounds
- **stopping_metric**: metric for early stopping (AUTO: logloss for classification, deviance for regression) Must be one of: "AUTO", "deviance", "logloss", "MSE", "RMSE", "MAE", "RMSLE", "AUC", "lift_top_group", "misclassification", "mean_per_class_error". Defaults to AUTO.
- **stopping_tolerance**: Relative tolerance for metric-based stopping criterion (stop if relative improvement is not at least this much) Defaults to 0.001.
- **max_runtime_secs:** maximum runtime to allow for model building (default: 0, disabled)

RF Parameters: Cross Validation

- **`nfolds`**: number of folds for N-fold cross-validation (default: 0, disabled)
- **`keep_cross_validation_predictions`**: keep the predictions of the cross-validation models (default: False)
- **`keep_cross_validation_fold_assignment`**: keep the cross-validation fold assignment (default: False)
- **`fold_assignment`**: cross-validation fold assignment scheme, if fold column is not specified. The ”Stratified” option will stratify the folds based on the response variable, for classification problems. Must be one of: ”AUTO”, ”Random”, ”Modulo”, ”Stratified”. (defaults: AUTO)
- **`fold_column`**: column with cross-validation fold index assignment per observation.

RF Misc. Parameters

- **binomial_double_trees**

- **offset_column**
- **build_tree_one_node**
- **calibrate_model**
- **calibration_frame**
- **max_hit_ratio_k**: Max. number (top K) of predictions to use for hit ratio computation (for multiclass only, 0 to disable) Defaults to 0.

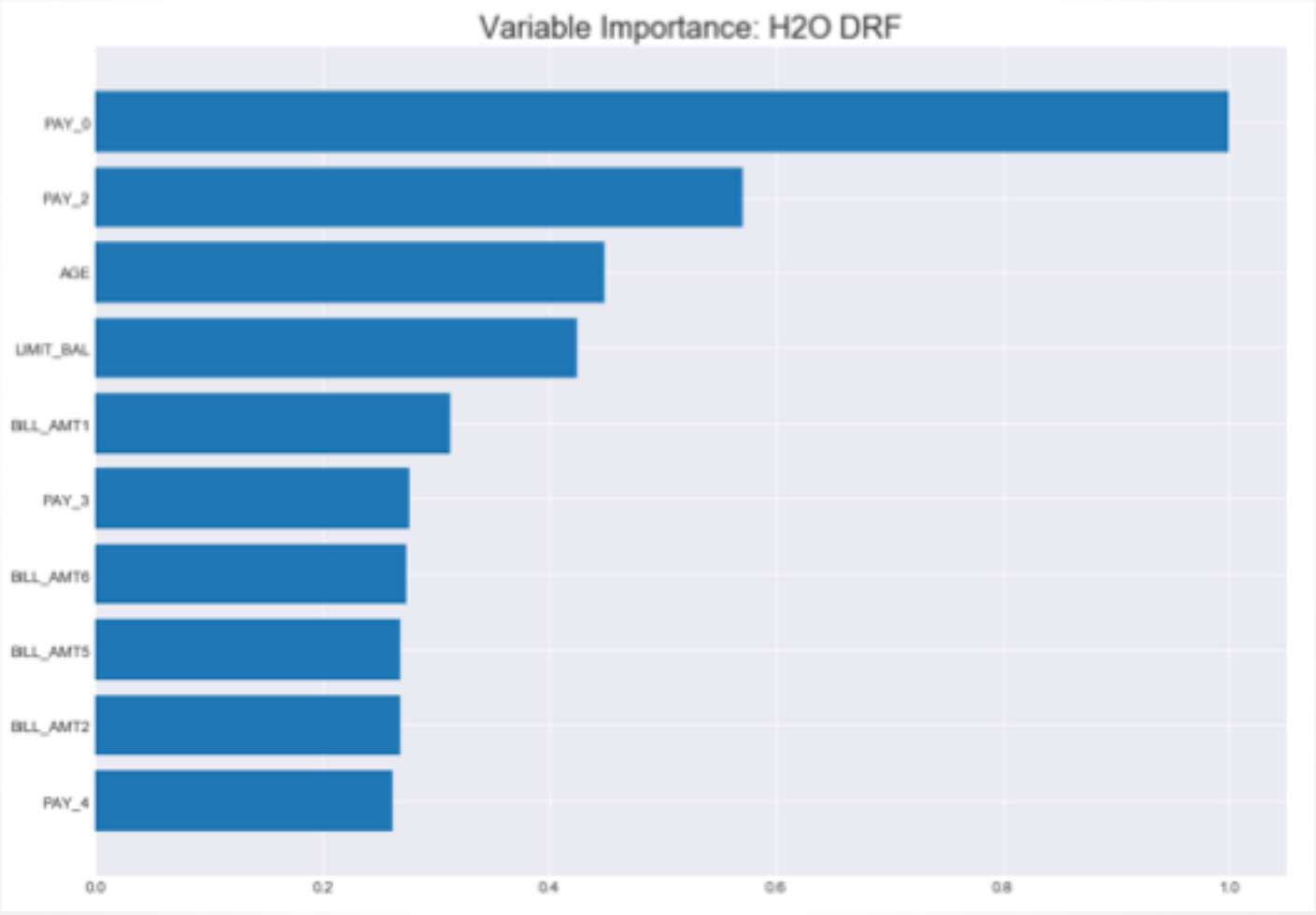
Random Forests in H2O

`.download_pojo()`

```
class rf_Tree_4_class_0 {
  static final double score0(double[] data) {
    double pred =      (Double.isNaN(data[7]) || data[7 /* PAY_3 */] <1.5f ?
          (data[0 /* LIMIT_BAL */] <135683.5f ?
              (Double.isNaN(data[11]) || data[11 /* BILL_AMT1 */] <35312.5f ?
                  (Double.isNaN(data[17]) || data[17 /* PAY_AMT1 */] <3945.5f ?
                    0.7474255f :
                    0.8430034f) :
                  (Double.isNaN(data[4]) || data[4 /* AGE */] <41.5f ?
                    0.79907835f :
                    0.8511166f)) :
              (Double.isNaN(data[5]) || data[5 /* PAY_0 */] <1.5f ?
                  (data[17 /* PAY_AMT1 */] <2876.5f ?
                    0.8268641f :
                    0.9220396f) :
                  (Double.isNaN(data[20]) || data[20 /* PAY_AMT4 */] <11648.5f ?
                    0.44347826f :
                    0.13333334f))) :
          (Double.isNaN(data[9]) || data[9 /* PAY_5 */] <1.0f ?
              (Double.isNaN(data[14]) || data[14 /* BILL_AMT4 */] <165892.5f ?
                  (data[9 /* PAY_5 */] <-0.5f ?
                    0.6821192f :
                    0.5637931f) :
                  (data[15 /* BILL_AMT5 */] <178671.5f ?
                    0.0f :
                    0.42857143f)) :
              (data[3 /* MARRIAGE */] <1.5f ?
                  (Double.isNaN(data[22]) || data[22 /* PAY_AMT6 */] <11535.0f ?
                    0.3069307f :
                    0.8333333f) :
                  (data[12 /* BILL_AMT2 */] <4311.5f ?
                    0.25f :
                    0.4448276f))));
    return pred;
  } // constant pool size = 62B, number of visited nodes = 15, static init size = 0B
}
```

RF Feature Importance

```
.varimp()
.varimp_plot()
```



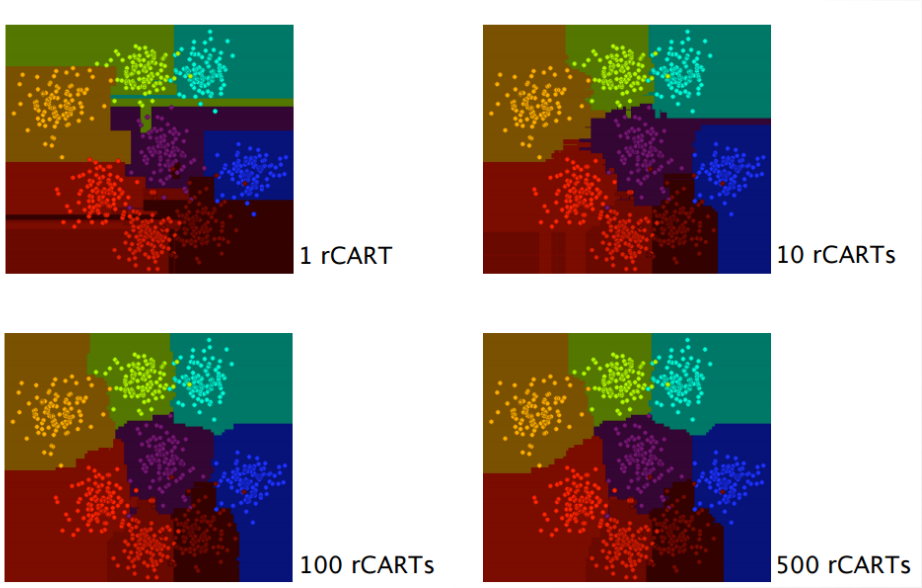
Pros and Cons of Random Forests

Pros

- Few tuning parameters
- Easy to parallelize

Cons

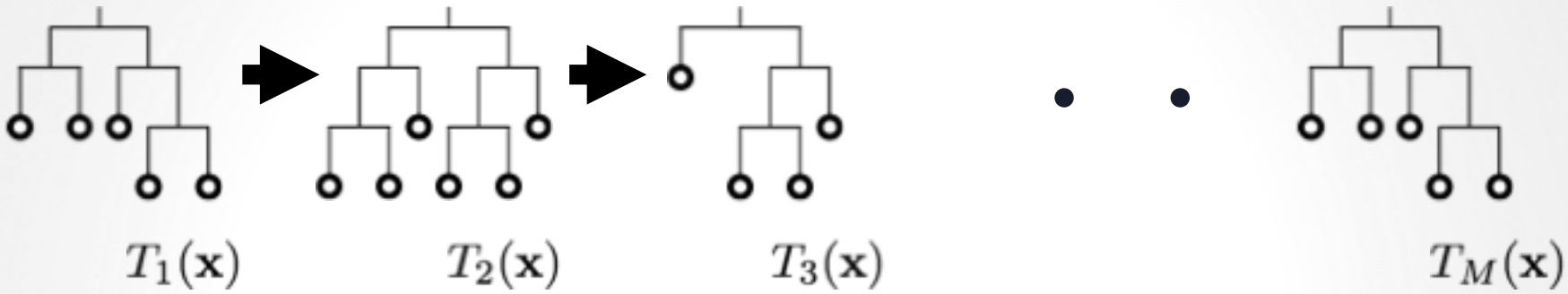
- Slow to score
- Not transparent



Supervised Learning:

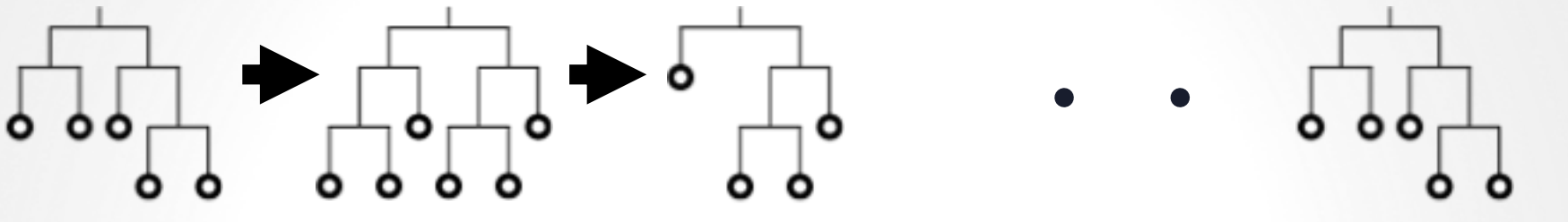
GRADIENT BOOSTING MACHINES

Gradient Boosting Machine (GBM)



$$f_i(\mathbf{x}) = f_{i-1}(\mathbf{x}) + T_i(\mathbf{x}; \hat{\Theta}_i)$$

Gradient Boosting Machine (GBM)



Algorithm 1 Forward Stagewise Additive Training

1: Initialize $f_0(\mathbf{x}) = 0$.
2: **for** $i = 1$ to M **do**
3: $\quad \hat{\Theta}_i = \arg\min_{\Theta_i} \sum_{i=1}^{n} L(y_i, f_{i-1}(x_i) + T(x_i; \Theta_i))$
4: $\quad f_i(\mathbf{x}) = f_{i-1} + T(\mathbf{x}; \hat{\Theta}_i)$
5: **end for**
6: $f(\mathbf{x}) = f_M(\mathbf{x})$

GBM Algorithm

Algorithm 10.3 *Gradient Tree Boosting Algorithm.*

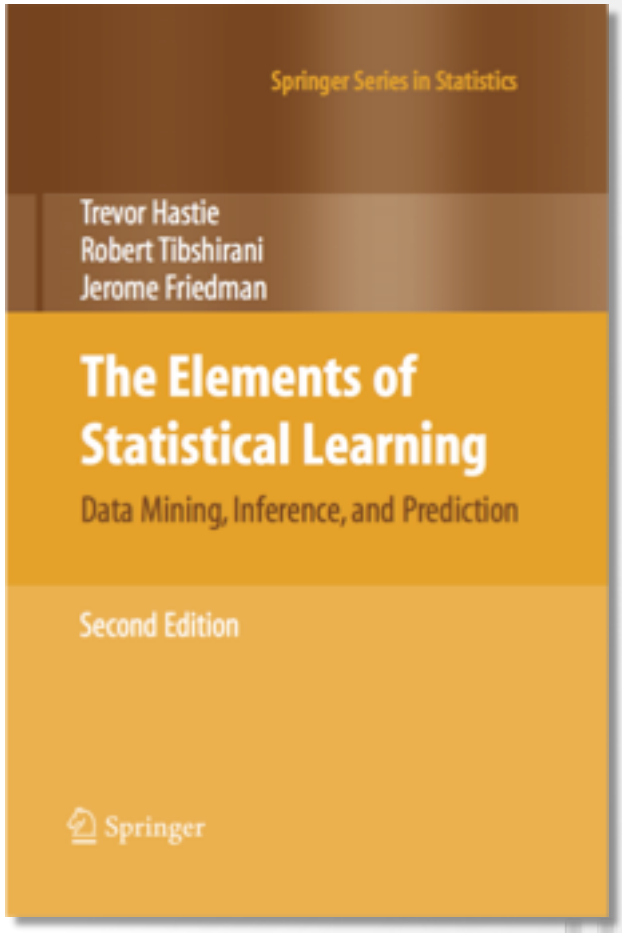
1. Initialize $f_0(x) = \arg\min_\gamma \sum_{i=1}^{N} L(y_i, \gamma)$.
2. For $m = 1$ to M:
   (a) For $i = 1, 2, \ldots, N$ compute

$$r_{im} = -\left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f=f_{m-1}}.$$

   (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 $\hat{f}(x) = f_M(x)$.



Gradient Boosting Machine (GBM) Implementations

- H2O GBM
- XGBoost (in H2O)
  - LightGBM
  - DART

GBM Parameters: Setting Up a Model

Python API

```
h2o.estimators.gbm.H2OGradientBoostingEstimator
- h2o.estimators.estimator_base.H2OEstimator
-- h2o.model.model_base.ModelBase
```

`model_id:`	specify a custom name for the model to use as a reference. (default: randomly generated)
`checkpoint:`	load a previously generated model
`seed:`	specify a pseudorandom seed, for reproducibility
`verbose:`	print verbose scoring history to console

GBM Parameters: Defining Data

(same as RF)

- `training_frame:` H2OFrame used to train model
- `validation_frame:` H2OFrame used to validate model (optional)
- `y:` column name or index of target variable
- `x:` column names (of indices) of predictor variables (default: all but y and ignored_columns)

- `ignored_columns:` columns to ignore (e.g. ID column)
- `ignore_const_cols:` ignore constant columns (default: True)
- `weights_column:` column by which to weight the data points

- `categorical_encoding:` encoding scheme for categorical variables: {"enum", "one_hot_explicit", "binary", "eigen", "label_encoder", "sort_by_response"} (default: "enum")

GBM Parameters: Loss Function

(same as RF)

- **distribution:** specifies the loss function, for use in determining gradients
  - 'bernoulli'
  - 'multinomial'
  - 'gaussian'
  - 'poisson'
  - 'gamma'
  - 'laplace'
  - 'quantile'
  - 'huber'
  - 'tweedie'

- **huber_alpha, quantile_alpha**

GBM Parameters: Ensemble

(same as RF)

- `ntrees:` number of trees

- `sample_rate:` row sampling rate per tree (default:1)
- `col_sample_rate_per_tree:` column sampling rate per tree (default: 1)

GBM Parameters: Individual Trees

- **Column sampling for split:**
- **col_sample_rate:** number of columns to sample on each split (default: sqrt(n) for classification, n/3 for regression)
- **col_sample_rate_change_per_level:** factor by which to increase or decrease mtries per level of tree

- **When to stop splitting?**
- **max_depth:** maximum depth of each tree
- **min_rows:** minimum rows in a leaf (i.e. stop splitting when data size is this small)
- **min_split_improvement:** **minimum relative improvement** in split criterion for a split to occur

- **Histogramming**
- **nbins**: number of bins for numeric variables (default: 20)
- **nbins_top_level**: can be used instead of nbins; nbins will then decrease by 2 each level
- **nbins_cats**: **histogram_type**: number of bins for categorical variables (default: 1024)

- **max_abs_leafnode_pred**

GBM Parameters: Boosting

- `learn_rate:` boosting factor, eta (default=0.1)
- `learn_rate_annealing:` factor by which to reduce the learn_rate every tree (default=1)

GBM Parameters: Class Imbalances

(same as RF)

- **`balance_classes`**: balance training data class counts via over/under-sampling (default: False)
- **`class_sampling_factors`**: desired over/under-sampling ratios per class (in lexicographic order). If not specified, sampling fac- tors will be automatically
- **`max_after_balance_size`**: maximum relative size of the training data after balancing class counts (can be less than 1.0). Requires balance classes. Defaults to 5.0.
- **`sample_rate_per_class:`** variable row sampling rate per class

GBM Parameters: Scoring & Stopping

(same as RF)

- **score_each_iteration**: score model after each tree (default: false)
- **score_tree_interval**: score model after n trees

- **stopping_rounds**: early stop if stopping metric's moving average does not improve for this many rounds
- **stopping_metric**: metric for early stopping (AUTO: logloss for classification, deviance for regression) Must be one of: "AUTO", "deviance", "logloss", "MSE", "RMSE", "MAE", "RMSLE", "AUC", "lift_top_group", "misclassification", "mean_per_class_error". Defaults to AUTO.
- **stopping_tolerance**: Relative tolerance for metric-based stopping criterion (stop if relative improvement is not at least this much) Defaults to 0.001.
- **max_runtime_secs**: maximum runtime to allow for model building (default: 0, disabled)

GBM Parameters: Cross Validation

(same as RF)

- **`nfolds`**: number of folds for N-fold cross-validation (default: 0, disabled)
- **`keep_cross_validation_predictions`**: keep the predictions of the cross-validation models (default: False)
- **`keep_cross_validation_fold_assignment`**: keep the cross-validation fold assignment (default: False)
- **`fold_assignment`**: cross-validation fold assignment scheme, if fold column is not specified. The ”Stratified” option will stratify the folds based on the response variable, for classification problems. Must be one of: ”AUTO”, ”Random”, ”Modulo”, ”Stratified”. Defaults to AUTO.
- **`fold_column`**: column with cross-validation fold index assignment per observation.

GBM Parameters: Misc.

- **offset_column**
- **build_tree_one_node**
- **calibrate_model**
- **calibrate_frame**
- **pred_noise_bandwith**
- **max_hit_ratio_k**: Max. number (top K) of predictions to use for hit ratio computation (for multiclass only, 0 to disable) Defaults to 0.

H2O GBM Tuning Tutorial for Python

https://github.com/h2oai/h2o-3/blob/master/h2o-docs/src/product/tutorials/gbm/gbmTuning.ipynb

H2O GBM Tuning Tutorial for Python

Arno Candel, PhD, Chief Architect, H2O.ai

Ported to Python by Navdeep Gill, M.S., Hacker/Data Scientist, H2O.ai

In this tutorial, we show how to build a well-tuned H2O GBM model for a supervised classification task. We specifically don't focus on feature engineering and use a small dataset to allow you to reproduce these results in a few minutes on a laptop. This script can be directly transferred to datasets that are hundreds of GBs large and H2O clusters with dozens of compute nodes.

You can download the source from H2O's github repository.

Ports to R Markdown and Flow UI (now part of Example Flows) are available as well.

XGBoost

- Popular open-source GBM library
- Used by many Kagglers and a successful algorithm implementation to win competitions

XGBoost Framework

- DMatrix: Data set abstraction and data structure
- Booster: GBM algorithm abstraction

XGBoost in H2O

- No algorithmic difference between H2O XGBoost and “regular” XGBoost
  - H2O just calls the regular XGBoost backend
  - H2O uses JNI to communicate to native C++ XGBoost libraries

- Two modules
  - **`h2o-ext-xgboost`**: contains actual XGBoost model and model builder code
  - **`h2o-genmodel-ext-xgboost`**: extends **`h2o-genmodel`** module and registers XGBoost-specific MOJO

Current XGBoost Support

1. XGBoost is not supported on Windows.
2. XGBoost is initialized for single-node H2O clusters; however multi-node XGBoost support is available as a Beta feature.
3. The list of supported platforms includes:

Platform	Minimal XGBoost	OMP	GPU	Compilation OS
Linux	yes	yes	yes	Ubuntu 14.04, g++ 4.7
OS X	yes	no	no	OS X 10.11
Windows	no	no	no	NA

Note: Minimal XGBoost configuration includes support for a single CPU.

4. Because we are using native XGBoost libraries that depend on OS/platform libraries, it is possible that on older operating systems, XGBoost will not be able to find all necessary binary dependencies, and will not be initialized and available.
5. XGBoost GPU libraries are compiled against CUDA 8, which is a necessary runtime requirement in order to utilize XGBoost GPU support.

LightGBM

- released by Microsoft in 2016; attempt to improve speed and memory usage ("Light")

  1. Data parallelization and histogramming (like H2O)

  2. Gradient-based one-sided sampling (GOSS):
     don't test all the data; prefer data points with large gradients

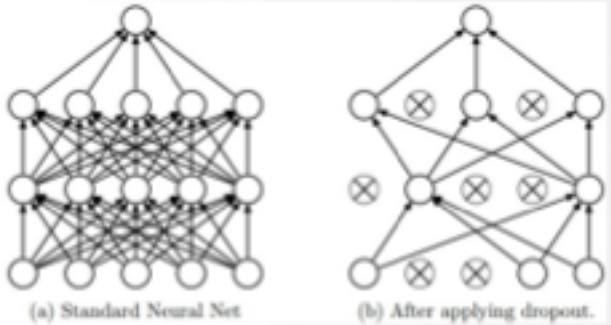
  3. Exclusive Feature Bundling:
     method of feature engineering to reduce sparsity

  4. Leaf-wise growth:

- otherwise includes most of the XGBoost options (e.g. L1/L2 regularization, DART)

Dropouts: DART

- Idea of dropouts comes from deep neural networks (CNN)



(a) Standard Neural Net (b) After applying dropout.

- DART: Dropouts meet Multiple Additive Regression Trees

1) Use only a subset of previous trees for computing gradient for next tree

2) Normalization factors to account for the fact trees were removed

Algorithm 1 The DART algorithm

Let N be the total number of trees to be added to the ensemble
$S_1 \leftarrow \{x, -L'_x(0)\}$
T_1 be a tree trained on the dataset S_1
$M \leftarrow \{T_1\}$
for $t = 2, \ldots, N$ **do**
  $D \leftarrow$ the subset of M such that $T \in M$ is in D with probability p_{drop}
  if $D = \emptyset$ **then** $D \leftarrow$ a random element from M
  end if
  $\hat{M} \leftarrow M \setminus D$
  $S_t \leftarrow \left\{x, -L'_x\left(\hat{M}(x)\right)\right\}$
  T_t be a tree trained on the dataset S_t
  $M \leftarrow M \cup \left\{\frac{T_t}{|D|+1}\right\}$
  for $T \in D$ **do**
    Multiply T in M by a factor of $\frac{|D|}{|D|+1}$
  end for
end for
Output M

Rashmi, K. and Gilad, R. (2015). DART: Dropouts meet Multiple Additive Regression Trees

GPU XGBoost

Accelerating the XGBoost algorithm using GPU computing

Artificial Intelligence | Data Mining and Machine Learning

Rory Mitchell[1], Eibe Frank[1]

April 4, 2017

GPU Accelerated XGBoost

dmlc.ml/2016/12/14/GPU-accelerated-xgboost.html

DMLC Archive

GPU Accelerated XGBoost

Dec 14, 2016 • Rory Mitchell

Update 2016/12/23: Some of our benchmarks were incorrect due to a wrong compiler flag. These have all been updated below.

Decision tree learning and gradient boosting have until recently been the domain of multicore CPUs. Here we showcase a new plugin providing GPU acceleration for the XGBoost library. The plugin provides significant speedups over multicore CPUs for large datasets.

The plugin can be found at: https://github.com/dmlc/xgboost/tree/master/plugin/updater_gpu

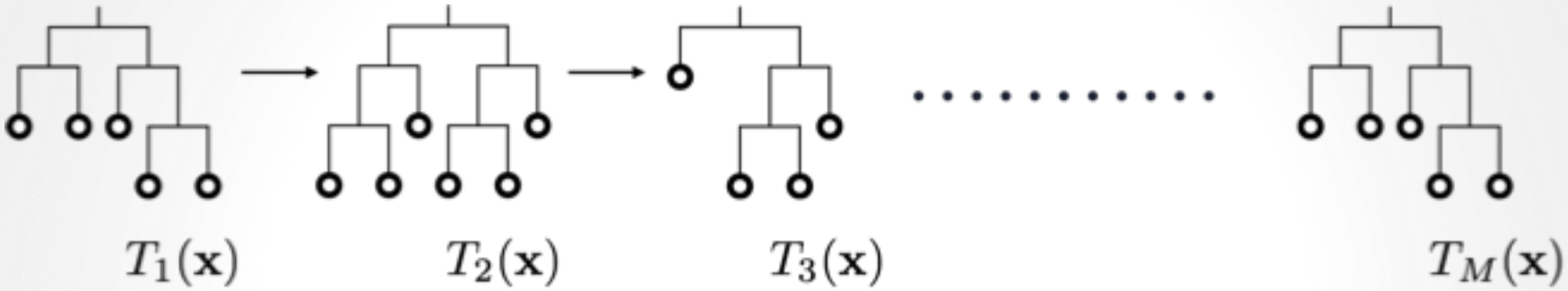
Before talking about the GPU plugin we briefly explain the XGBoost algorithm.

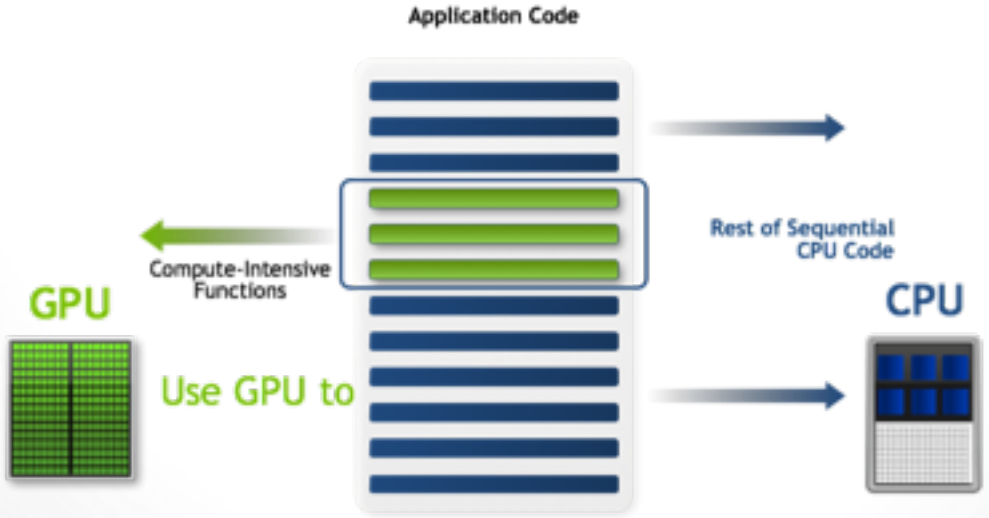
XGBoost for classification and regression

XGBoost is a powerful tool for solving classification and regression problems in a supervised learning setting. It is an implementation of a generalised gradient boosting algorithm designed to offer high-performance, multicore scalability and distributed machine scalability.

The gradient boosting algorithm is an ensemble learning technique that builds many predictive models. Together these smaller models produce much stronger predictions than any single model alone. In particular for gradient boosting, we create these smaller models sequentially, where each new model directly addresses the weaknesses in the previous models.

GBM on GPU





Pros and Cons of Gradient Boosting Machines

Pros

- Robust to correlated features
- Robust to missing values
- Handles non-linearities in data
- Handles interaction effects

Cons

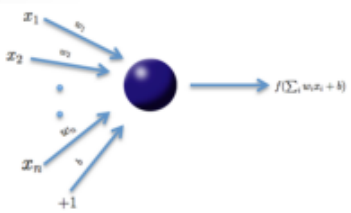
- Requires care in tuning params
  - Tends to overfit the training set
- Sensitive to noise and outliers

Supervised Learning:

DEEP LEARNING

What is Deep Learning?

What it is:

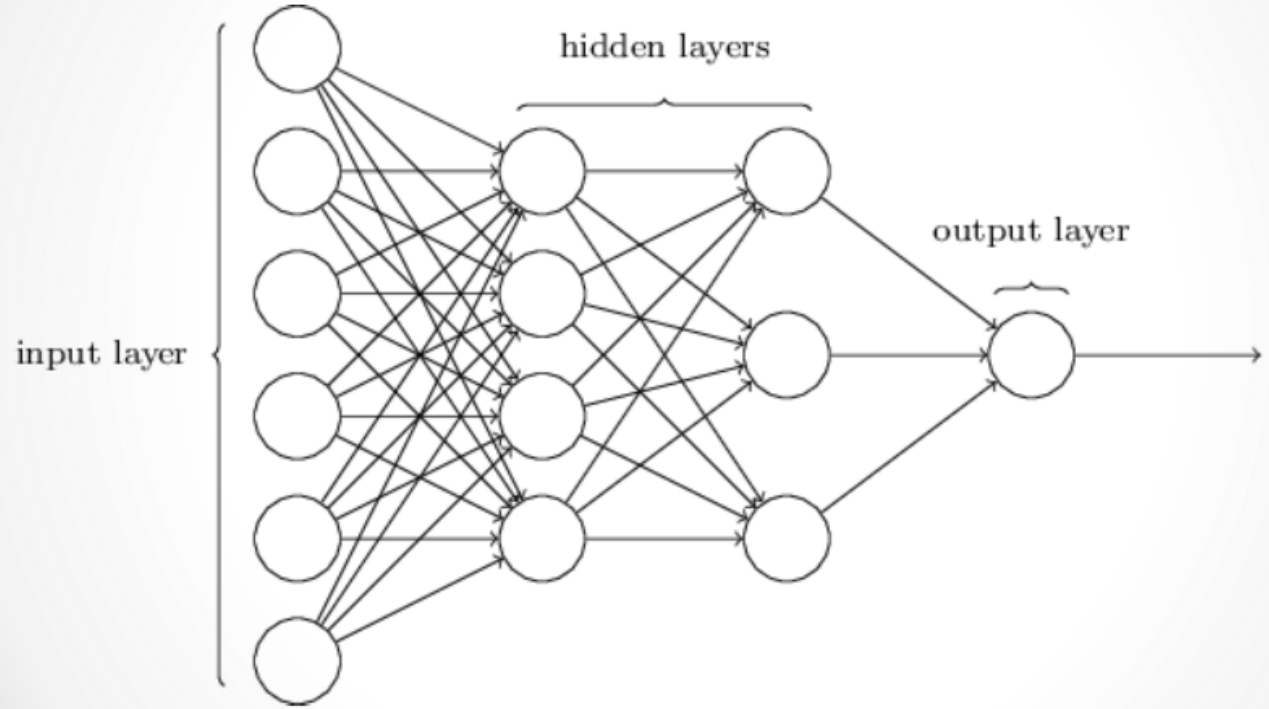


- "A branch of machine learning based on a set of algorithms that attempt to model high-level abstractions in data by using model architectures, composed of multiple non-linear transformations." (Wikipedia, 2015)
- Deep neural networks have more than one hidden layer in their architecture. That's what's "deep."
- Very useful for complex input data such as images, video, audio.

What it's not:

Deep learning architectures, specifically artificial neural networks (ANNs) have been around since 1980, so they are not new. However, there were breakthroughs in training techniques that lead to their recent resurgence (mid 2000's). Combined with modern computing power, they are quite effective.

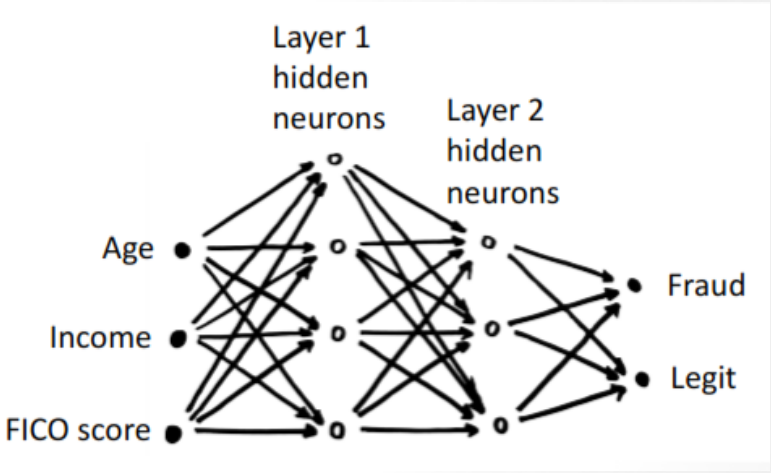
An example “Deep” Neural Network



Example of a deep neural net architecture.

Value of Deep Neural Networks

- Deep Learning learns a **hierarchy of non-linear transformations**
- Neurons transform their input in a non-linear way
- **Black-box, brute-force method, really good at pattern recognition**
- Deep Learning got a boost in the last decade due to **faster hardware and algorithmic advances**



Deep Learning model
=
set of connecting weights
+
type of non-linearity

Pros and Cons of Deep Neural Networks

Pros

- non linear
- robust to correlated features
- learned features can be extracted
- can stop training at any time
- can be fine-tuned with more data
- great ensemble member
- efficient for multi-class problems
- world-class at pattern recognition

Cons

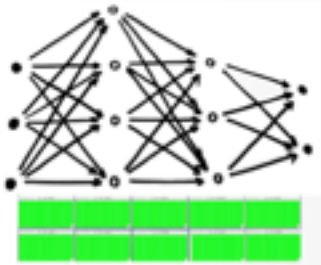
- slow to train
- slow to score
- not interpretable
- results not fully reproducible
- theory not well understood
- overfits, needs regularization
- many hyper-parameters
- expands categorical variables
- must impute missing values

What is H2O Deep Learning?

H2O Deep Learning:

Multi-layer fully-connected feed-forward Neural Network

+ **distributed** processing on multi-node clusters
+ **multi-threaded** speedup on multi-core CPUs
+ **fully featured** for **fast** & **accurate** results

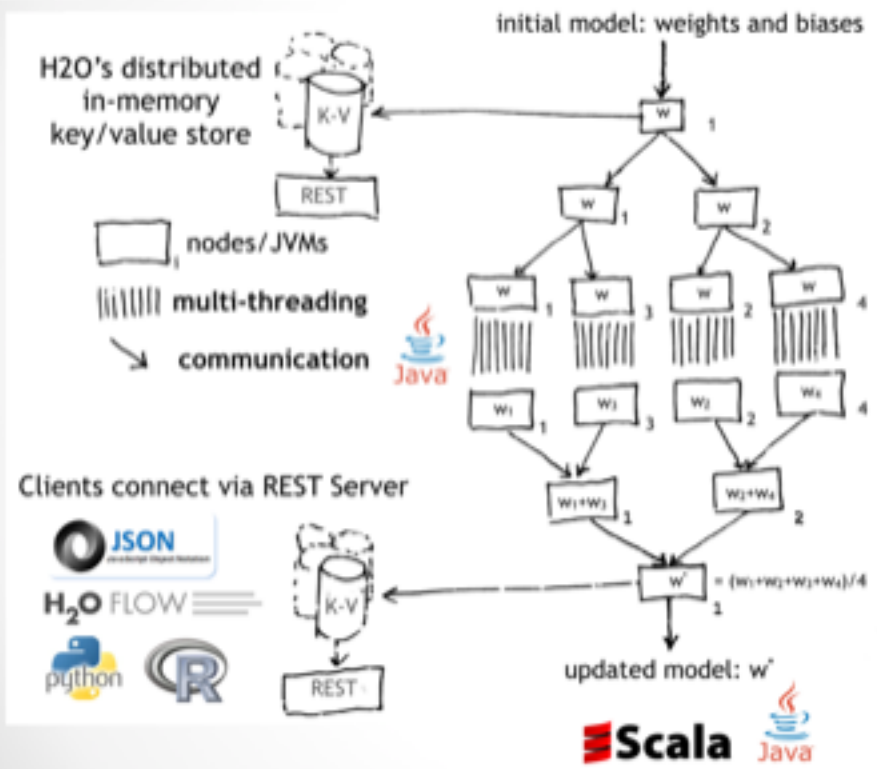


all 320 cores on 10 nodes maxed out

(automatic standardization, automatic handling of categorical and missing values, train/test data adaptation, model initialization, activation functions, multiple loss functions, autoencoder, load balancing, auto-tuning, adaptive learning rate, rate decay, momentum, L1/L2 penalty, dropout, hyper-parameter search, N-fold cross-validation, checkpointing, early stopping, variable importances, feature extraction, realtime model inspection, optimizations for sparse data and networks, etc.)

= **Easy-to-use scalable Deep Learning for large real-world datasets**
(insurance, healthcare, finance, fraud, churn, risk, IoT, etc.)

H2O Deep Learning Architecture



Map/Reduce Iterations

Map: training (fprop+bprop)
parallelized across all nodes and all CPU cores

Reduce: model aggregation
for global convergence

Shown here: Model averaging
New: Elastic averaging

Communication frequency is auto-tuned and user-controllable (affects convergence)

Supervised Learning:

ENSEMBLES

Ensembles

What it is:

- "Ensemble methods use multiple learning algorithms to obtain better predictive performance that could be obtained from any of the constituent learning algorithms." (Wikipedia, 2015)
- Random Forests and Gradient Boosting Machines (GBM) are both ensembles of decision trees.
- Stacking, or Super Learning, is technique for combining various learners into a single, powerful learner using a second-level metalearning algorithm.

What it's not:

Ensembles typically achieve superior model performance over singular methods. However, this comes at a price — computation time.

Case Study: Lending Club Dataset

- Loan data from 2007 up until 2015 including rejected applications and accepted applications.
- Of the 500k accepted applicants about 160k loans have either been completely paid off or defaulted.
- There are about 4 million applicants in the rejected loans dataset.
- **Use Case 1**: Predict the likelihood of a user defaulting based on the information supplied when applying for a loan.
- **Use Case 2**: Determine the interest rate Lending Club would have offered the user based on the information supplied when applying for a loan.
- Full Data: https://www.kaggle.com/wendykan/lending-club-loan-data
- H2O Subset: https://s3.amazonaws.com/h2o-public-test-data/bigdata/laptop/lending-club/loan.csv

Case Study: Lending Club Dataset

	Column Name	Description	Unit	Role
1	loan_amnt	Requested loan amount	US dollars	Predictor
2	term	Longest term length	Months	Predictor
3	**int_term**	**Recommended interest rate**	**Rate**	**Response**
4	emp_length	Employment length	Years	Predictor
5	home_ownership	Housing status	Categorical	Predictor
6	annual_inc	Annual income	US dollars	Predictor
7	purpose	Purpose for the loan	Categorical	Predictor
8	addr_state	State of residence	Categorical	Predictor
9	dti	Debt to income ratio	Percent	Predictor
10	delinq_2yrs	Number of delinquencies in the past 2 years	Count	Predictor
11	revol_util	Revolving credit line utilized	Percent	Predictor
12	total_acc	Number of active accounts	Count	Predictor
13	**bad_loan**	**Bad loan indicator**	**Boolean**	**Response**
14	longest_credit_length	Age of oldest active account	Years	Predictor
15	verification_status	Income verification status	Boolean	Predictor

UNSUPERVISED LEARNING MODELS

Unsupervised Learning Models

- No target!
- Ways to characterize or simplify data

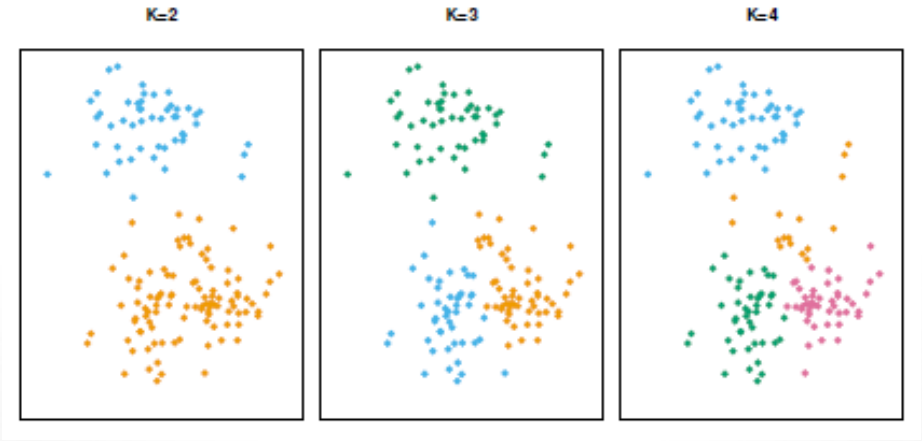
- Uses
  - Clustering (Put rows together)
  - Variable reduction (Put columns together)
  - Anomaly detection
  - Missing Value Imputation

Unsupervised Learning:

K-MEANS CLUSTERING

K-Means Clustering

- K-Means clustering groups observations based on numeric features
  - Assumes clusters are roughly the same sized hyperspheres
  - Minimize Euclidean distance between observations and cluster centers
- Number of methods for choosing the number of clusters, k
  - Choose several and evaluate performance
  - Use business rules



Pros and Cons of K-Means Clustering

Pros

- Fast, Scalable Algorithm

Cons

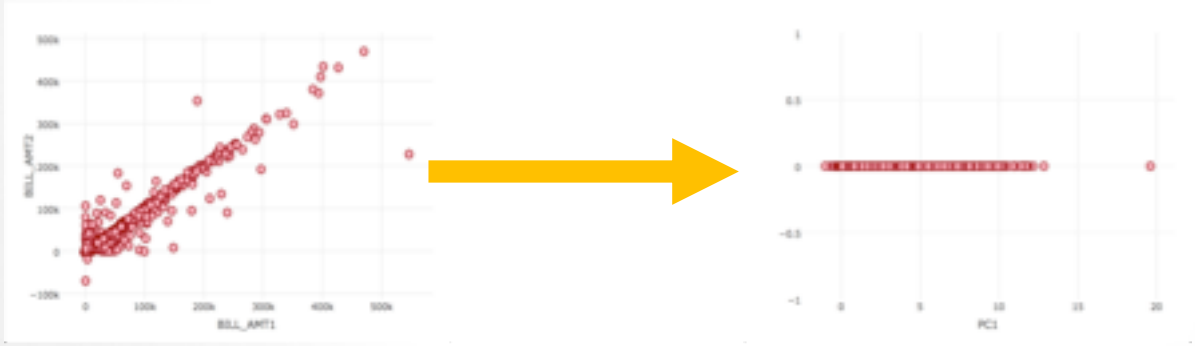
- Choice of k can be tricky
- Euclidean distance not robust
  - Hyperspheres not common
  - Sensitive to correlated measures
  - Sensitive to scaling
  - Sensitive to skewed measures
  - Sensitive to outliers
- Categorical data requires preprocessing
  - Multiple Correspondence Analysis
  - Multi-Dimensional Scaling

Unsupervised Learning:

PRINCIPAL COMPONENTS ANALYSIS (PCA)

Principal Components Analysis

- Orthogonal rotation of covariance or correlation matrix that orders derived measures from highest to lowest variation
- Useful for dimensionality reduction / removing collinearities



Pros and Cons of Principal Components Analysis

Pros

- Can be computed using covariance/correlation matrix

Cons

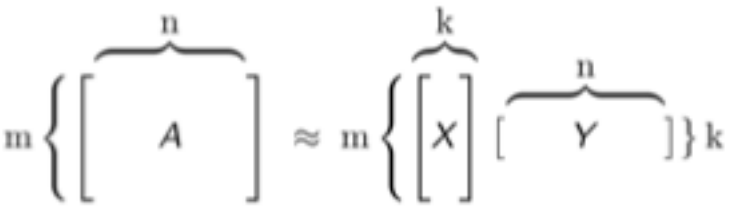
- Sensitive to skewed measures
- Sensitive to outliers
- Categorical data requires preprocessing
  - Multiple Correspondence Analysis
  - Multi-Dimensional Scaling

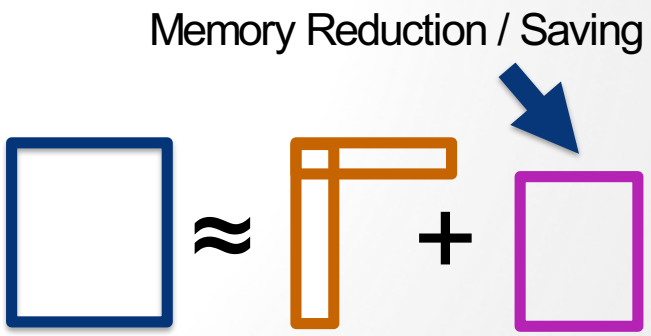
Unsupervised Learning:

GENERALIZED LOW-RANK MODEL (GLRM)

GLRM Overview

- GLRM is an extension of well-known matrix factorization methods such as Principal Component Analysis (PCA).
- Unlike PCA which is limited to numerical data, GLRM can also handle categorical, ordinal and Boolean data.
- **Given**: Data table *A* with *m* rows and *n* columns
- **Find**: Compressed representation as numeric tables *X* and *Y* where *k* is a small user-specified number





- *Y* = archetypal features created from columns of *A*
- *X* = row of A in reduced feature space
- GLRM can approximately reconstruct *A* from product *XY*

GLRM Loss Functions

Each column can use different loss function

Loss	Principal Components Analysis
Quadratic	PCA
Absolute	Robust PCA
Huber	Huber PCA (Hybrid of Quadratic and Robust)
Poisson	Poisson PCA
Hinge	Boolean PCA
Logistic	Logistic PCA
Periodic	Periodic PCA
Categorical	Categorical PCA
Ordinal	Ordinal PCA

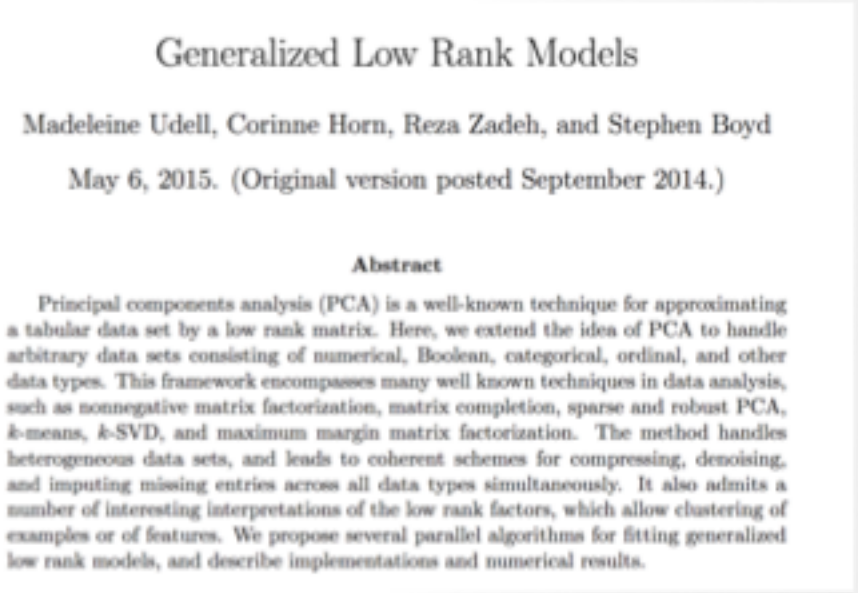
GLRM Key Features

- Memory
  - Compressing large data set with minimal loss in accuracy
- Speed
  - Reduced dimensionality = short model training time
- Feature Engineering
  - Condensed features can be analysed visually
- Missing Data Imputation
  - Reconstructing data set will automatically impute missing values

GLRM Technical References

- Paper
  - arxiv.org/abs/1410.0342

- Other Resources
  - H2O World Video
  - Tutorials



Generalized Low Rank Models

Madeleine Udell, Corinne Horn, Reza Zadeh, and Stephen Boyd

May 6, 2015. (Original version posted September 2014.)

Abstract

Principal components analysis (PCA) is a well-known technique for approximating a tabular data set by a low rank matrix. Here, we extend the idea of PCA to handle arbitrary data sets consisting of numerical, Boolean, categorical, ordinal, and other data types. This framework encompasses many well known techniques in data analysis, such as nonnegative matrix factorization, matrix completion, sparse and robust PCA, k-means, k-SVD, and maximum margin matrix factorization. The method handles heterogeneous data sets, and leads to coherent schemes for compressing, denoising, and imputing missing entries across all data types simultaneously. It also admits a number of interesting interpretations of the low rank factors, which allow clustering of examples or of features. We propose several parallel algorithms for fitting generalized low rank models, and describe implementations and numerical results.

Online Resources

- H2O Tutorials and Training Material
  - https://github.com/h2oai/h2o-tutorials
  - https://github.com/h2oai/h2o-tutorials/tree/master/training/lending_club_exercise
  - https://github.com/h2oai/app-consumer-loan
- Datasets
  - https://s3.amazonaws.com/h2o-public-test-data/bigdata/laptop
  - https://s3.amazonaws.com/h2o-public-test-data/bigdata/laptop/lending-club/loan.csv