

Intro to H2O

Lauren DiPerna // Data Scientist @ H2O

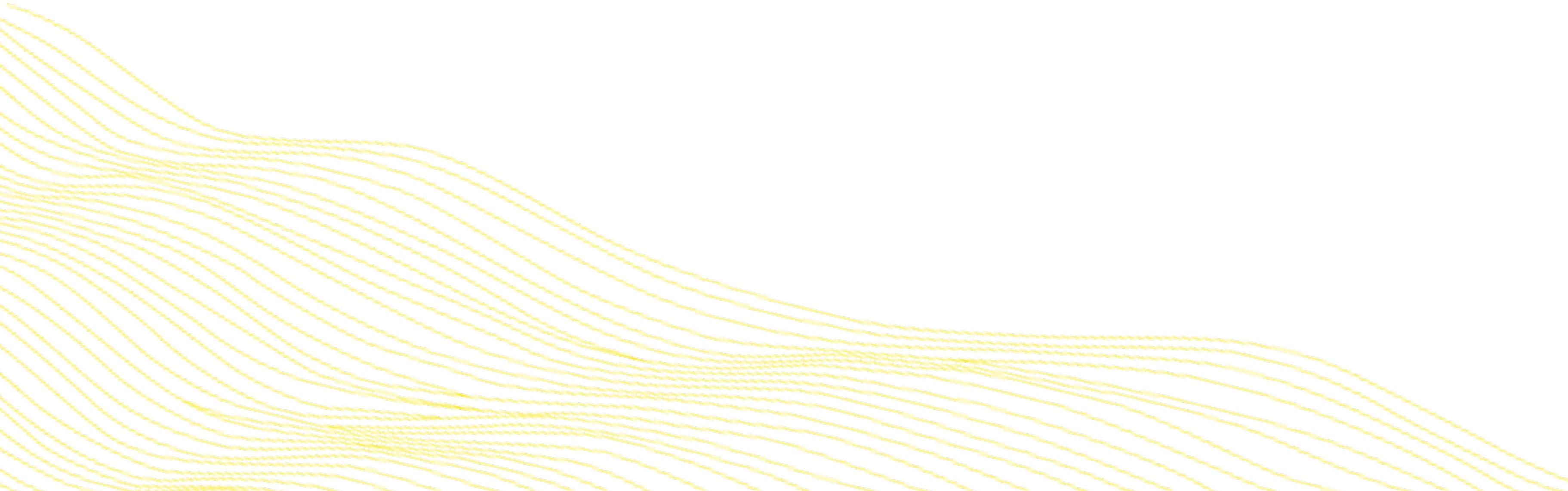
Charlotte, NC // June 11, 2018

The Agenda

- What is H2O
- How does H2O Core work?
- The Python API
- The Use Case Dataset
- How to Learn From Data
- How to use AutoML?

What is H2O.ai?

The AI Company that builds ML Products



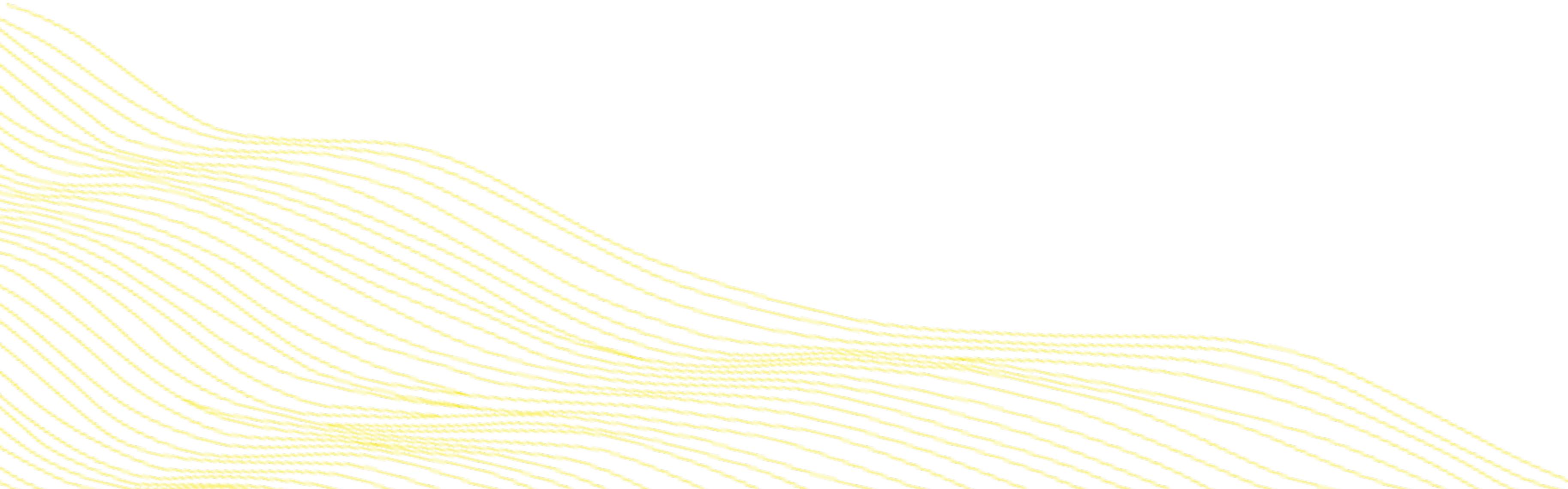
H2O.ai: Company Overview

Founded	2011 Venture-backed, Debuted in 2012
Products	H2O-3 E. Steam Sparkling Water DAI H2O4GPU
Mission	Operationalize Data Science & Platform to Build Beautiful Data Products
Team	75+ employees: Distributed Systems Engineers , ML Experts, World-class Visualization Designers
Headquarters	Mountain View, CA



What is H2O-3?

Powerful Platform for End-to-End Machine Learning



What is H2O-3?

Java-Based Software for In-Memory Data Modeling

Open Source



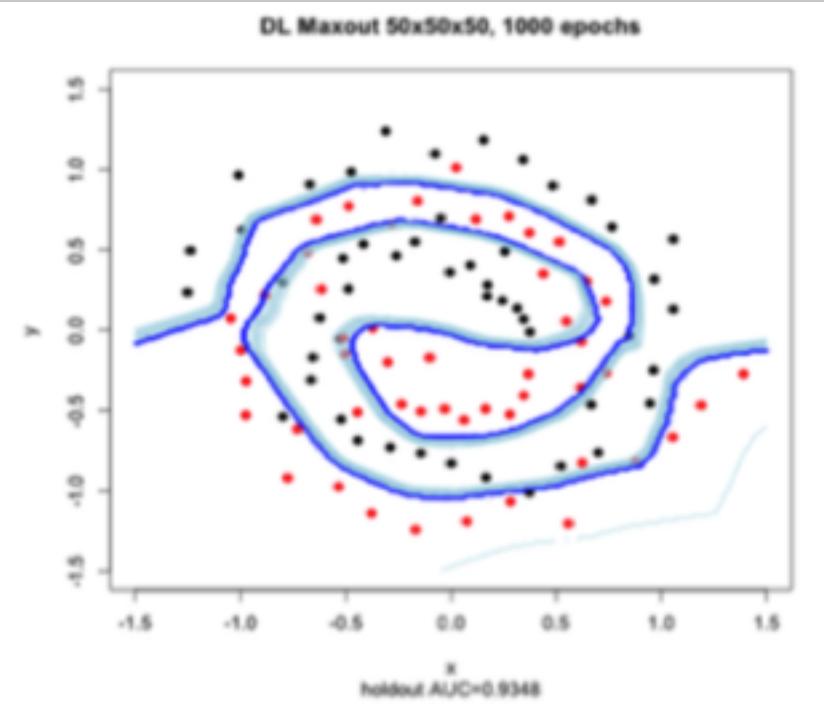
Big Data Ecosystem



Flexible Interface

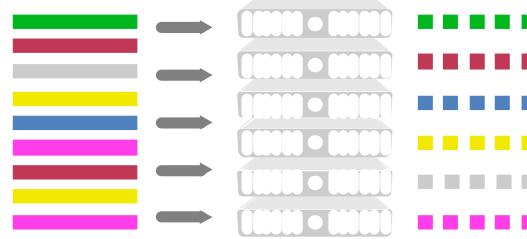
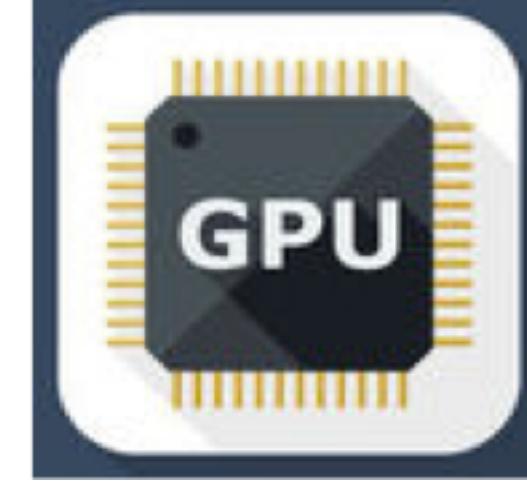


Smart and Fast Algorithms



What is H2O-3?

Java-Based Software for **In-Memory** Data Modeling

Scalability and Performance	Rapid Model Deployment	GPU Enablement*	Cloud Integration
 <ul style="list-style-type: none">• Distributed In-Memory Computing Platform• Distributed Algorithms• Fine-Grain MapReduce	<ul style="list-style-type: none">• Highly portable models deployed in Java (POJO)• Automated and streamlined scoring service deployment with Rest API*		 

Why H2O-3?

x_1	x_2	x_3	...	x_p	y
			...		
			...		
			...		
			...		
			...		
			...		

x_1	x_2	x_3	...	x_p	y
			...		
			...		
			...		
			...		
			...		

x_1	x_2	x_3	...	x_p	y
			...		
			...		
			...		
			...		

x_1	x_2	x_3	...	x_p	y
			...		
			...		
			...		

**No Data Size
Limit**

x_1	x_2	x_3	...	x_p	y
			...		
			...		
			...		

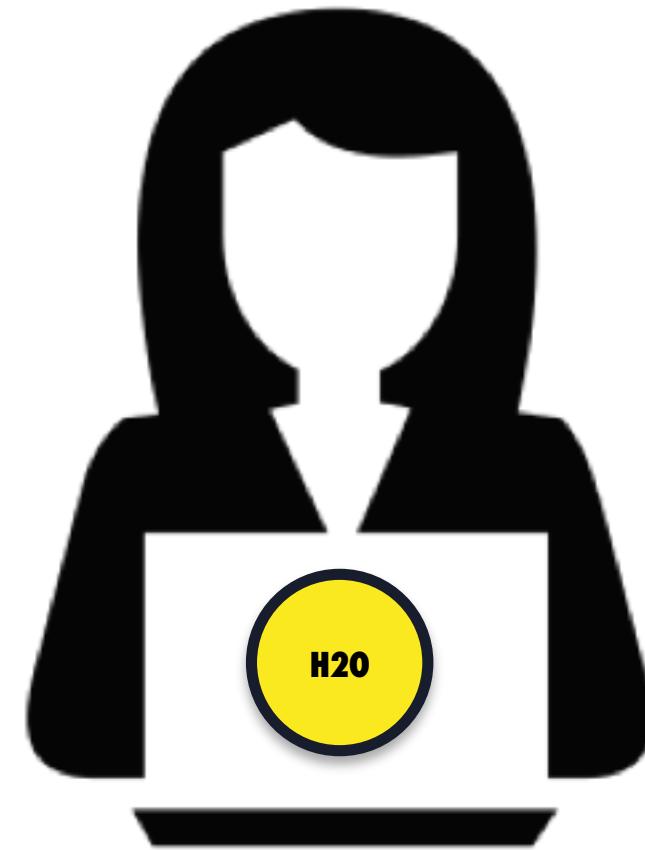
x_1	x_2	x_3	...	x_p	y
			...		
			...		
			...		

Why H₂O-3?



♥ R + ♥ Py. + ♥ UI

Why H₂O-3?



EASY

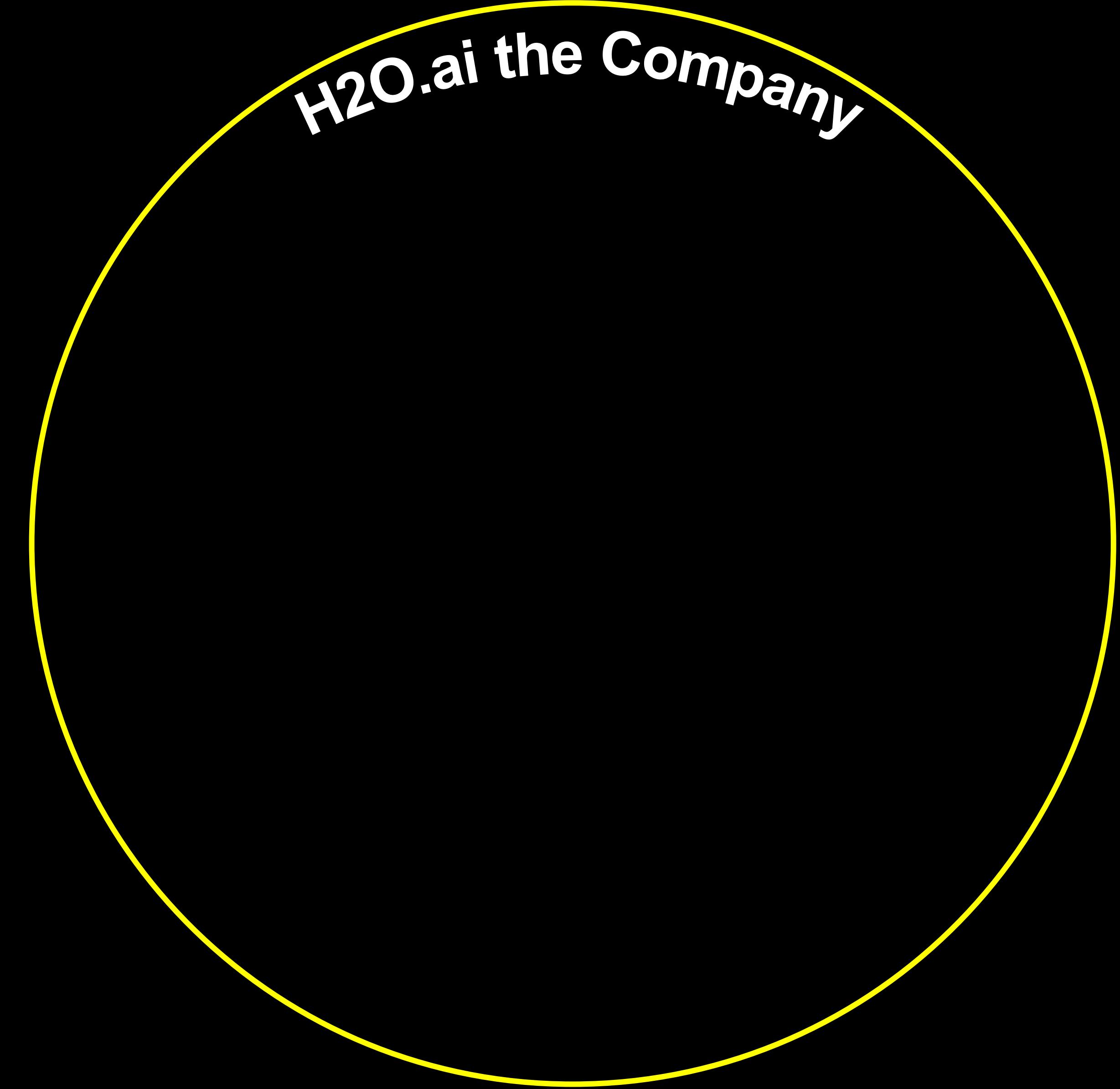


**Live
Results!**

How to think about H2O-3?



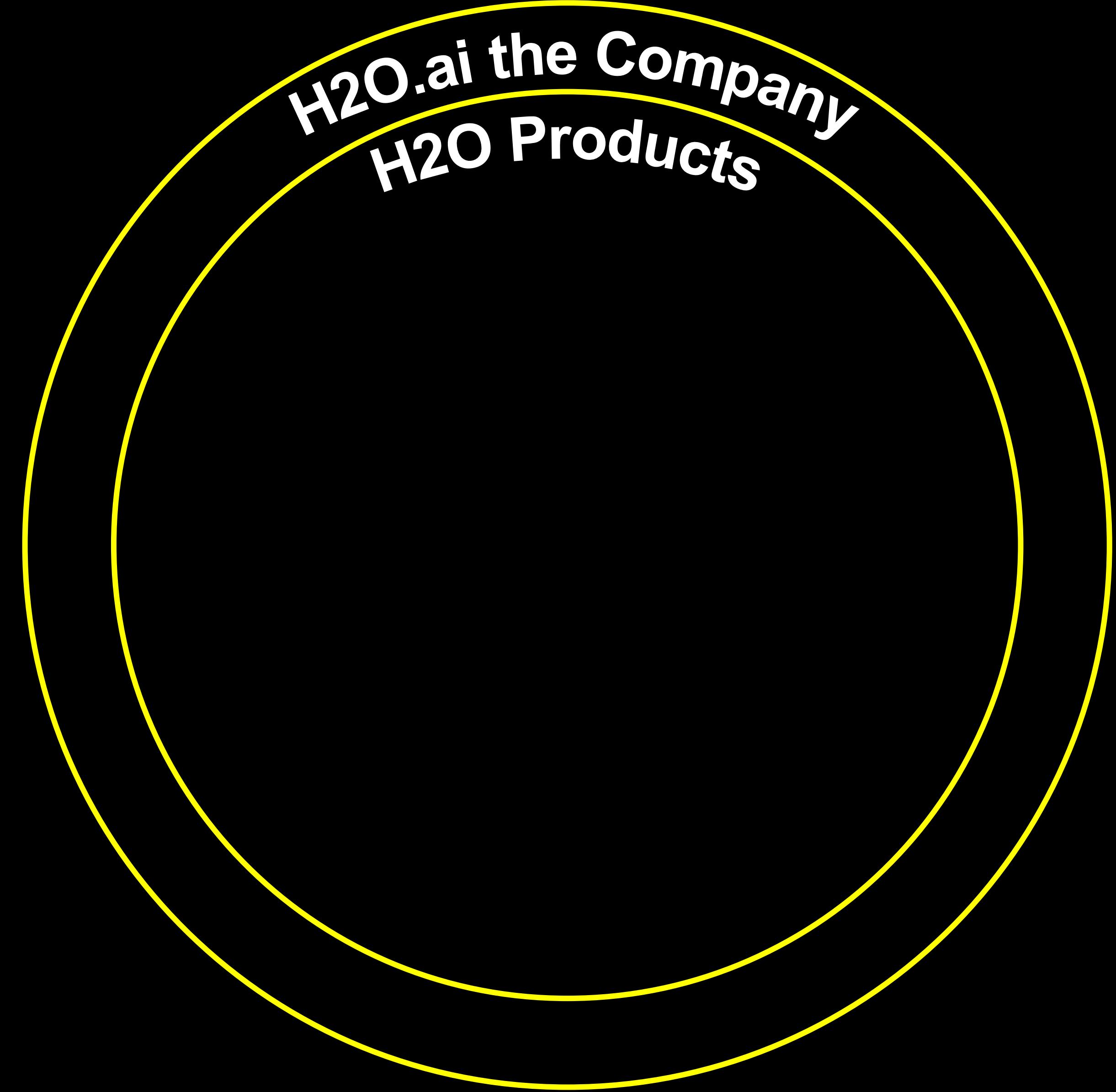
H2O.ai the Company



H2O.ai the Company

H2O.ai the Company

All H2O Products



H2O.ai the Company
H2O Products

H2O.ai the Company
H2O Products

- H2O-3
- Enterprise Steam
- Sparkling Water
- Driverless AI
- H2O4GPU

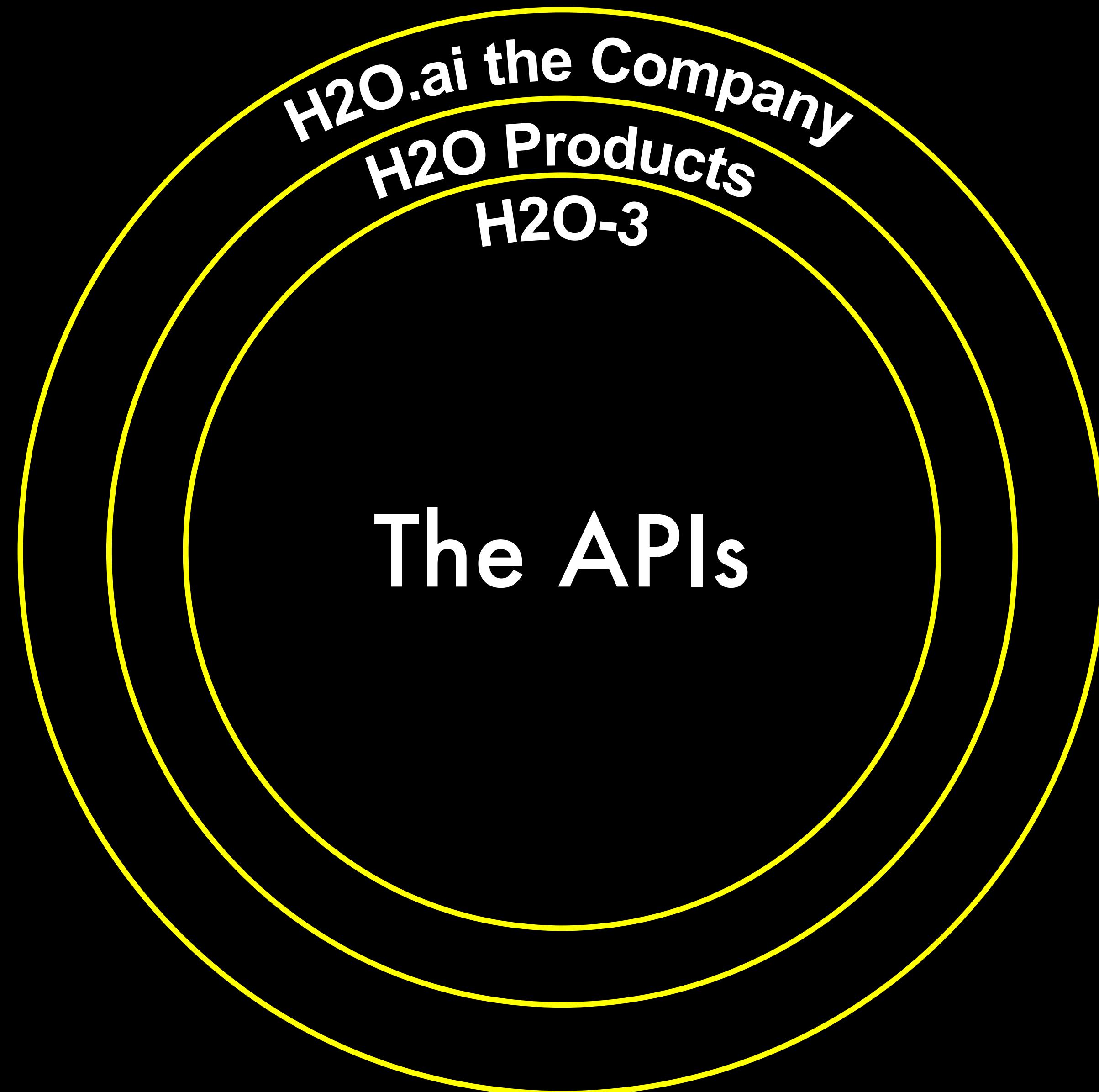
H2O.ai the Company
H2O Products

- **H2O-3**
- Enterprise Steam
- **Sparkling Water**
- Driverless AI
- **H2O4GPU**

H2O-3

H2O Products

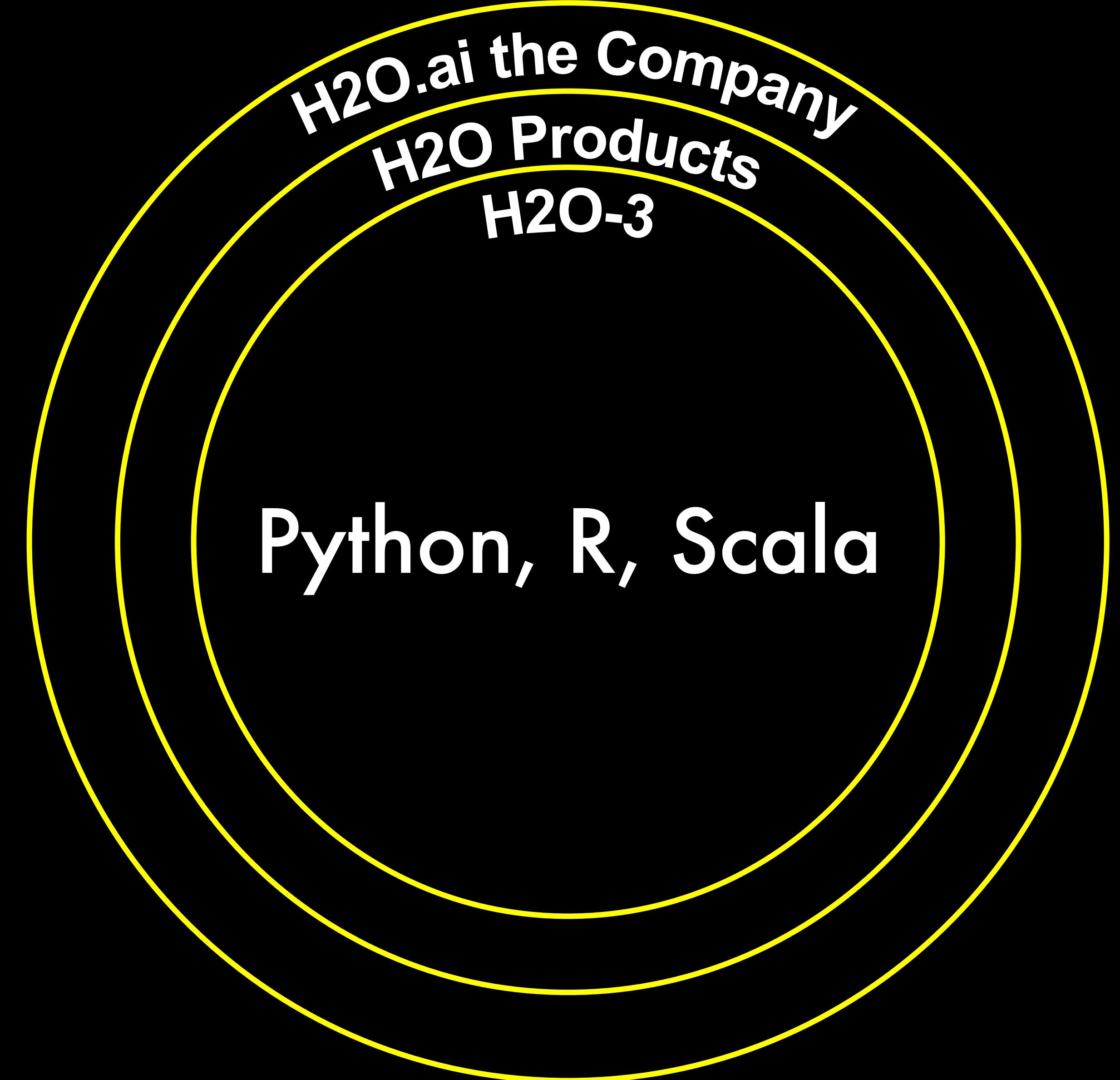
H2O.ai the Company

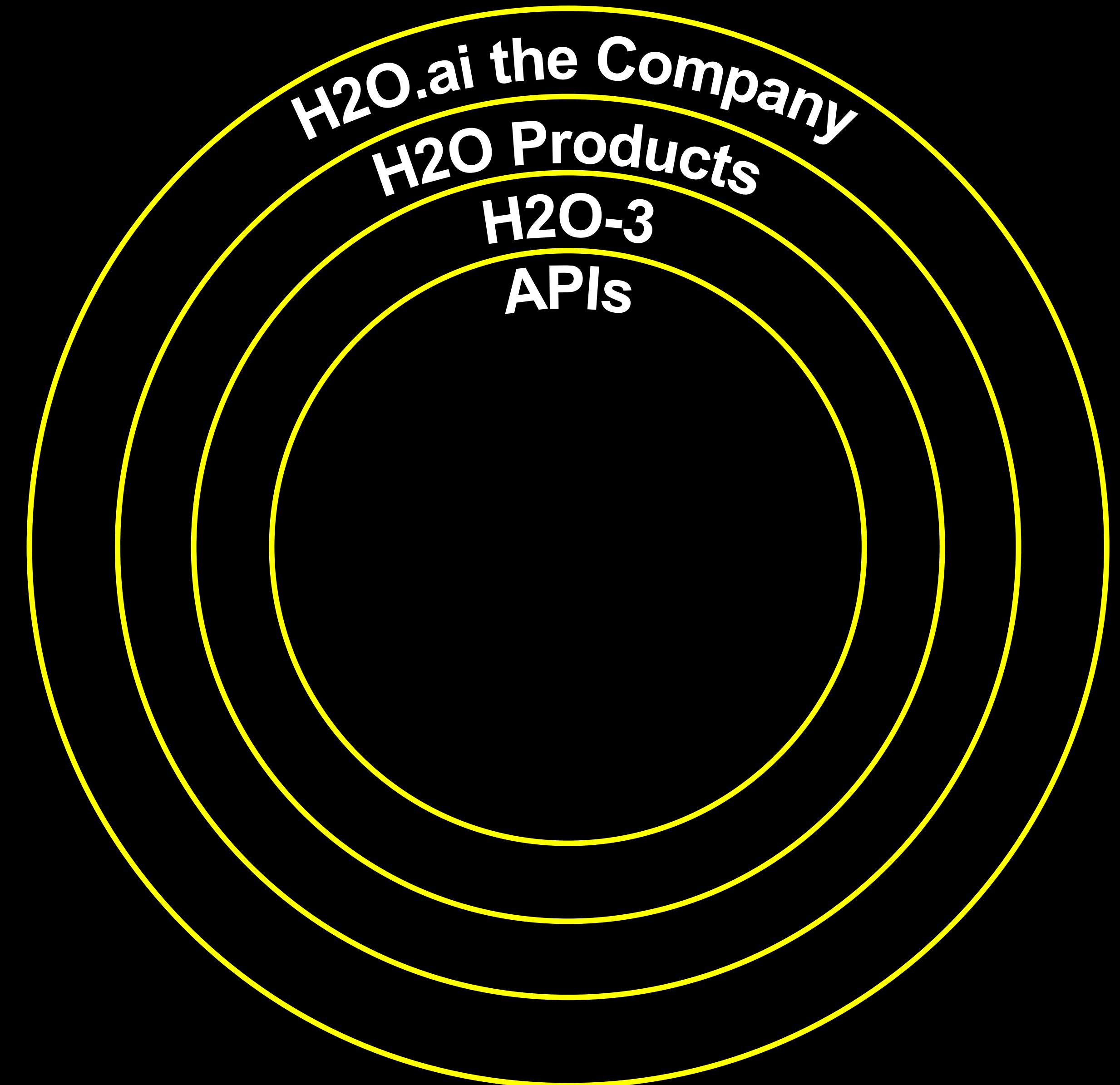


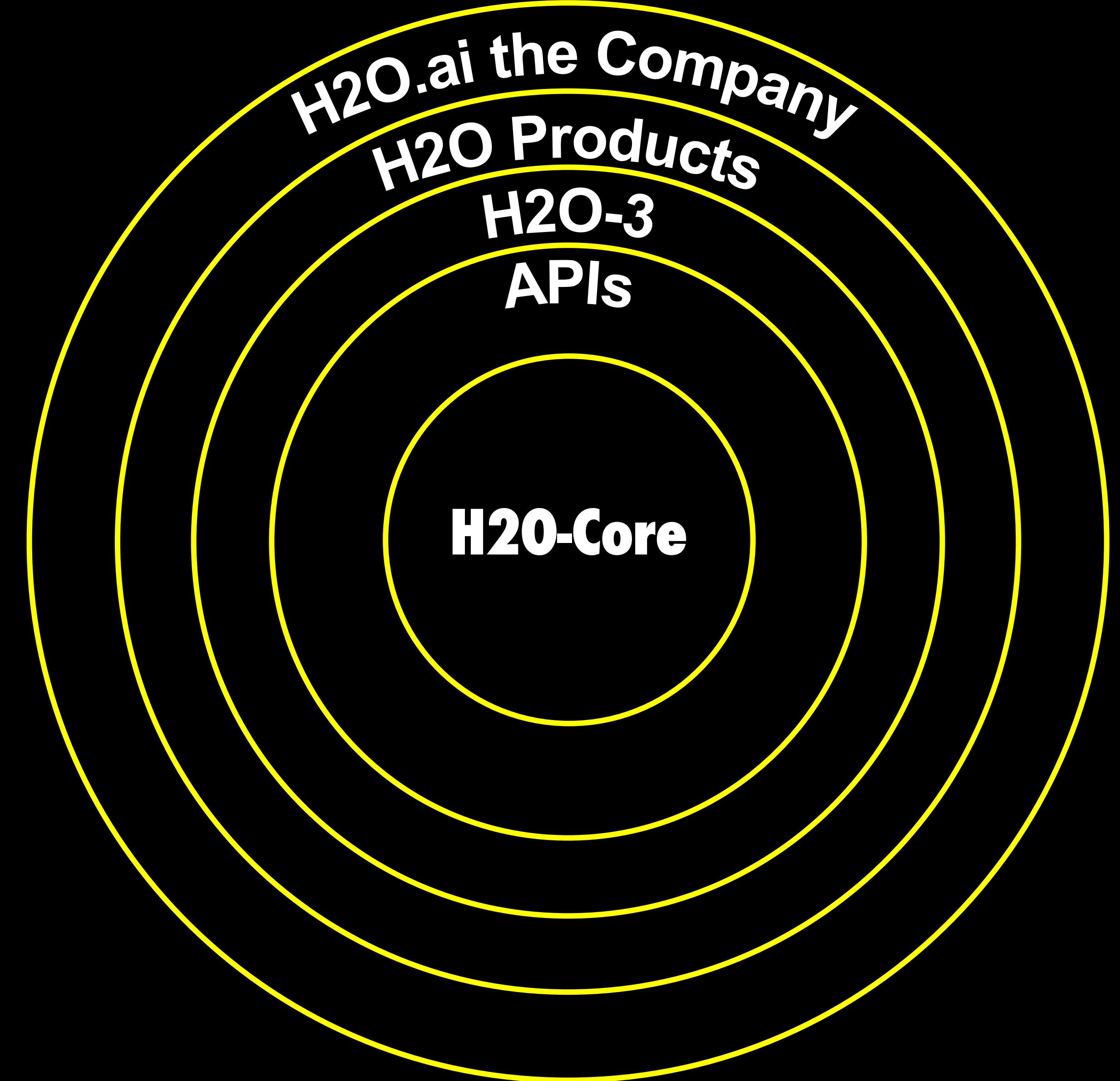
The APIs

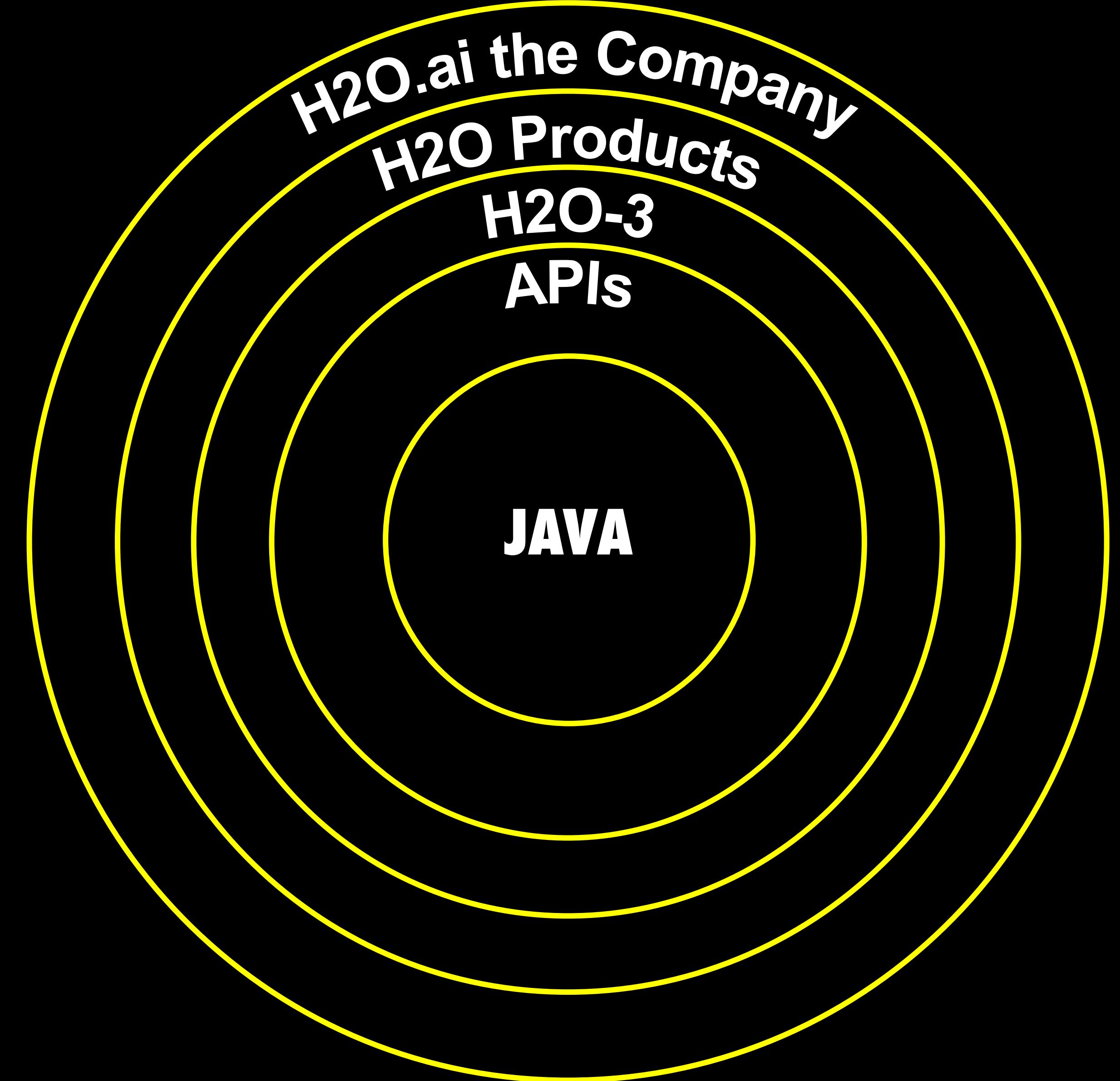
H2O Products
H2O-3

H2O.ai the Company

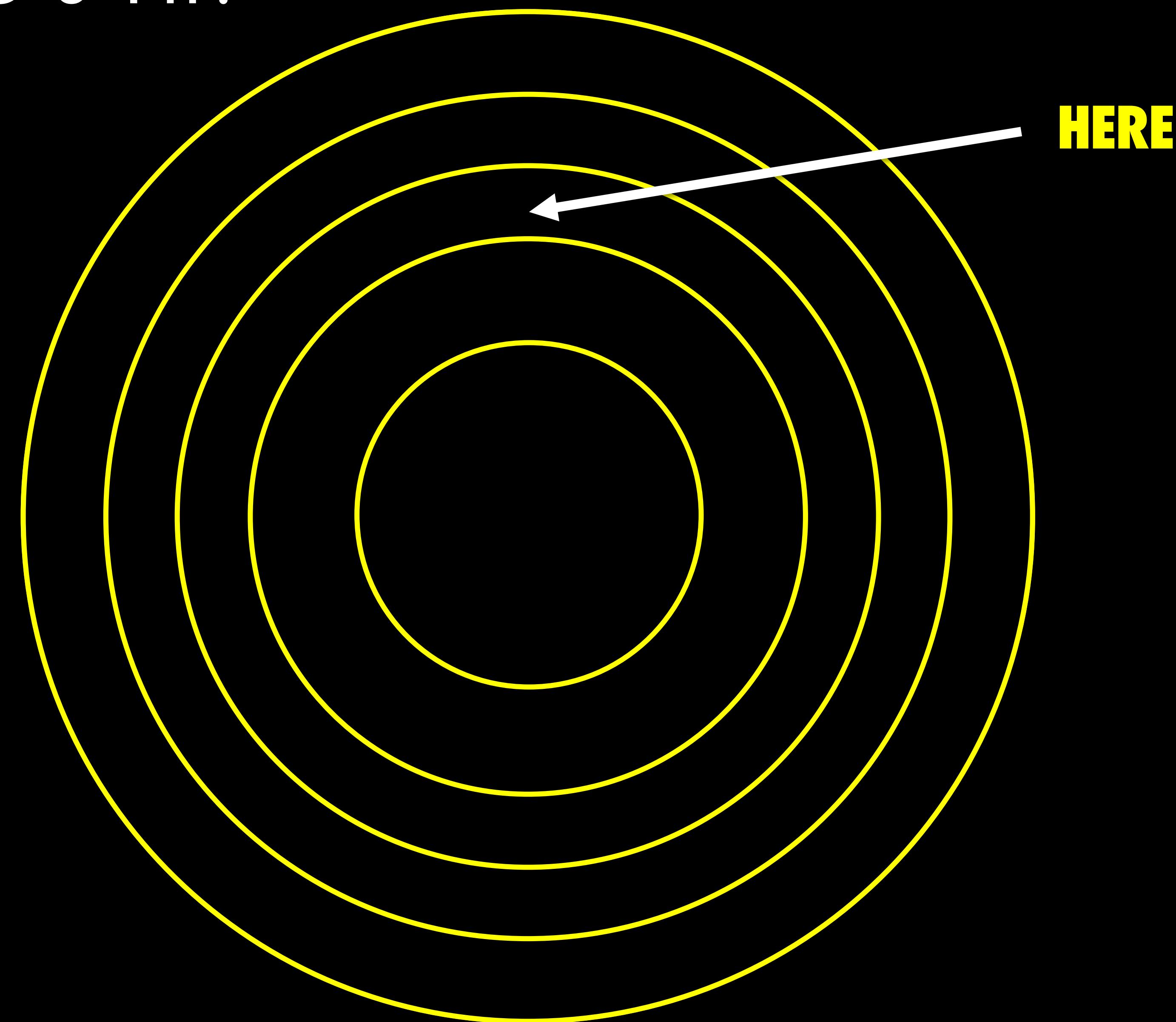








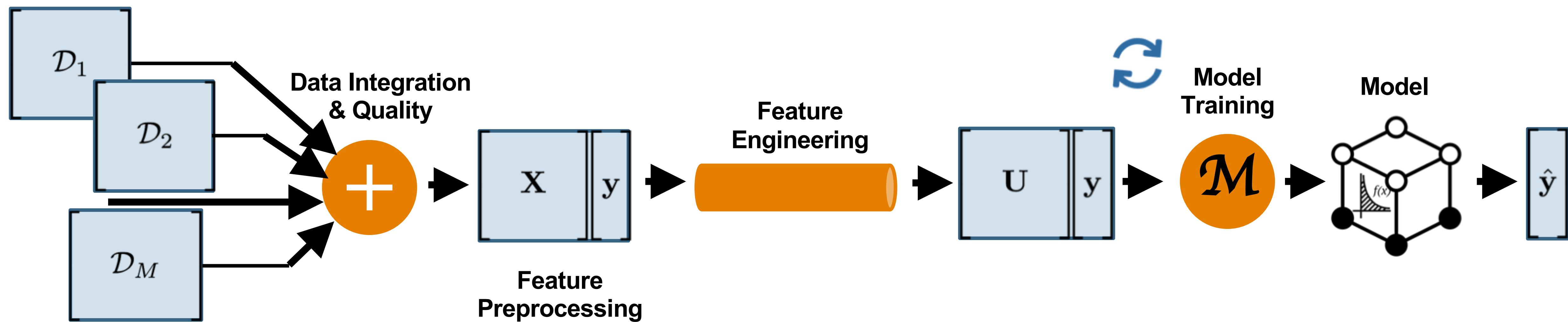
Where does H₂O-3 Fit?



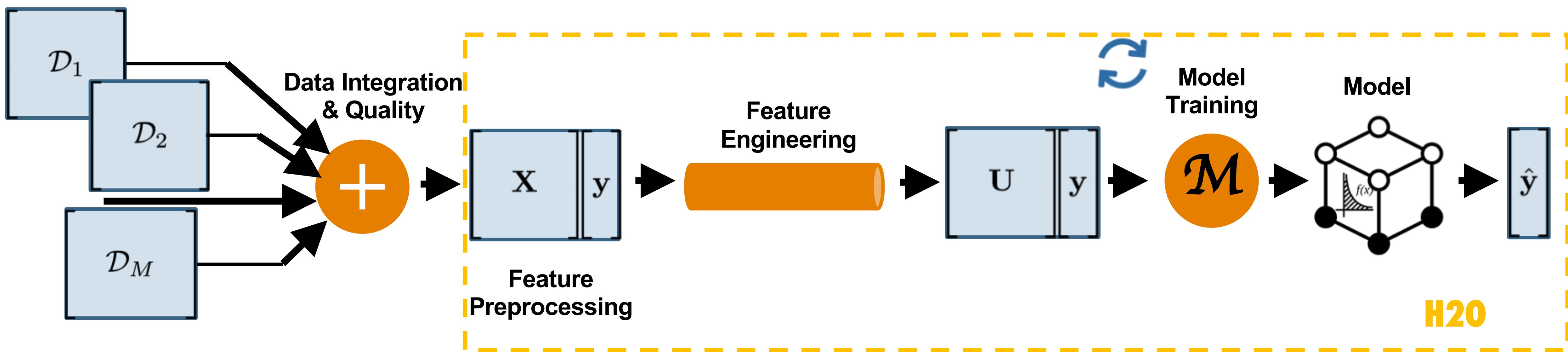


HERE

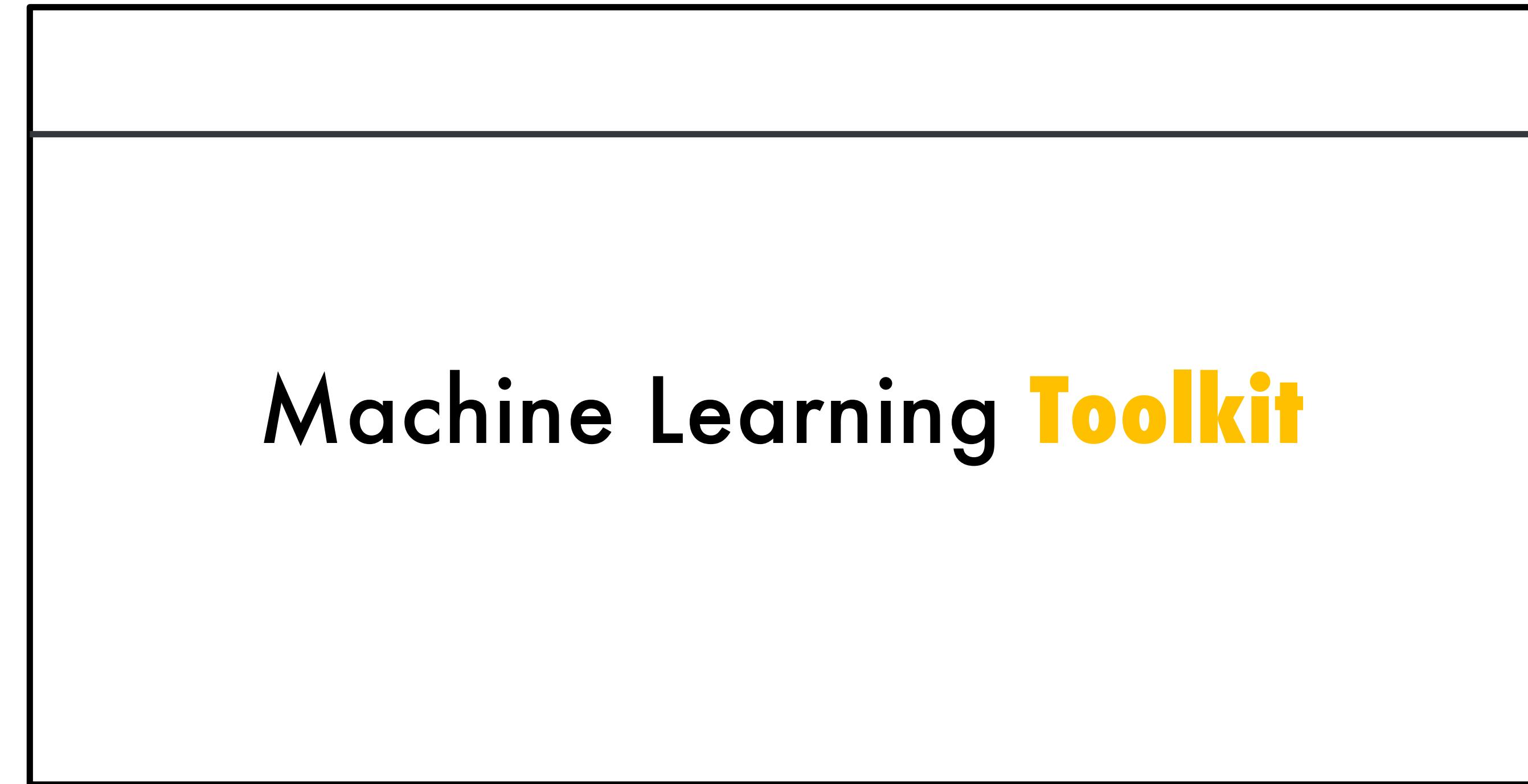
The Machine Learning Pipeline



This is Where H2O-3 Helps



Everything in One Place



Machine Learning **Toolkit**

Everything in One Place

Tackle ML Challenges

Current Algorithm Overview

Statistical Analysis

- Linear Models (GLM)
- Naïve Bayes

Ensembles

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

Deep Neural Networks

- MLP
- Autoencoder
 - Anomaly Detection
 - Deep Features

Clustering

- K-Means (Auto-K)

Dimension Reduction

- Principal Component Analysis
- Generalized Low Rank Models

Word Embedding

- Word2Vec

Time Series

- iSAX

Machine Learning Tuning

- Hyperparameter Search
- Early Stopping

Scientific Advisory Council



Dr. Trevor Hastie

- PhD in Statistics, Stanford University
- John A. Overdeck Professor of Mathematics, Stanford University
- Co-author, *The Elements of Statistical Learning: Prediction, Inference and Data Mining*
- Co-author, *Generalized Additive Models*
- 108,404 citations (via Google Scholar)



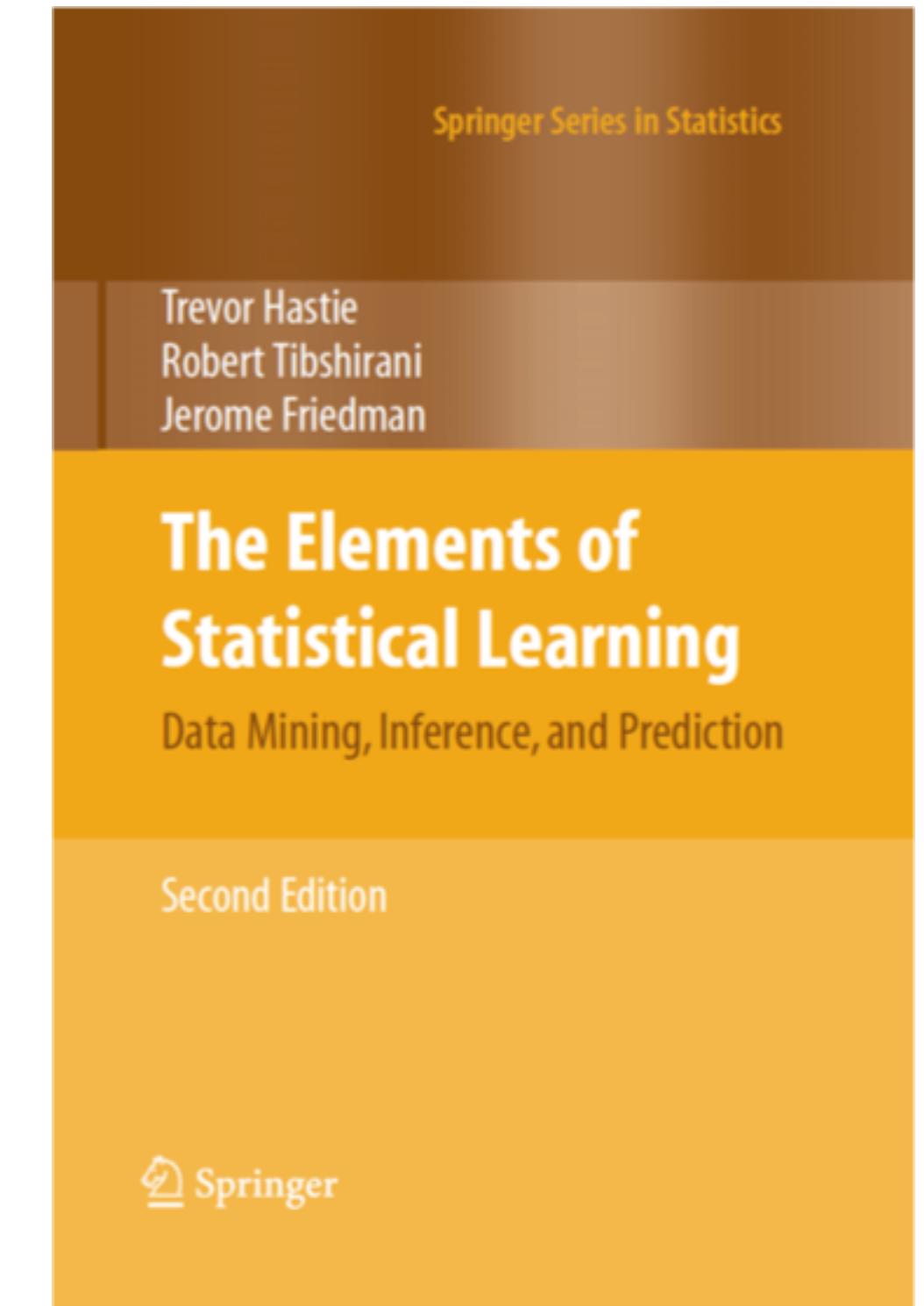
Dr. Robert Tibshirani

- PhD in Statistics, Stanford University
- Professor of Statistics and Health Research and Policy, Stanford University
- COPPS Presidents' Award recipient
- Co-author, *The Elements of Statistical Learning: Prediction, Inference and Data Mining*
- Author, *Regression Shrinkage and Selection via the Lasso*
- Co-author, *An Introduction to the Bootstrap*



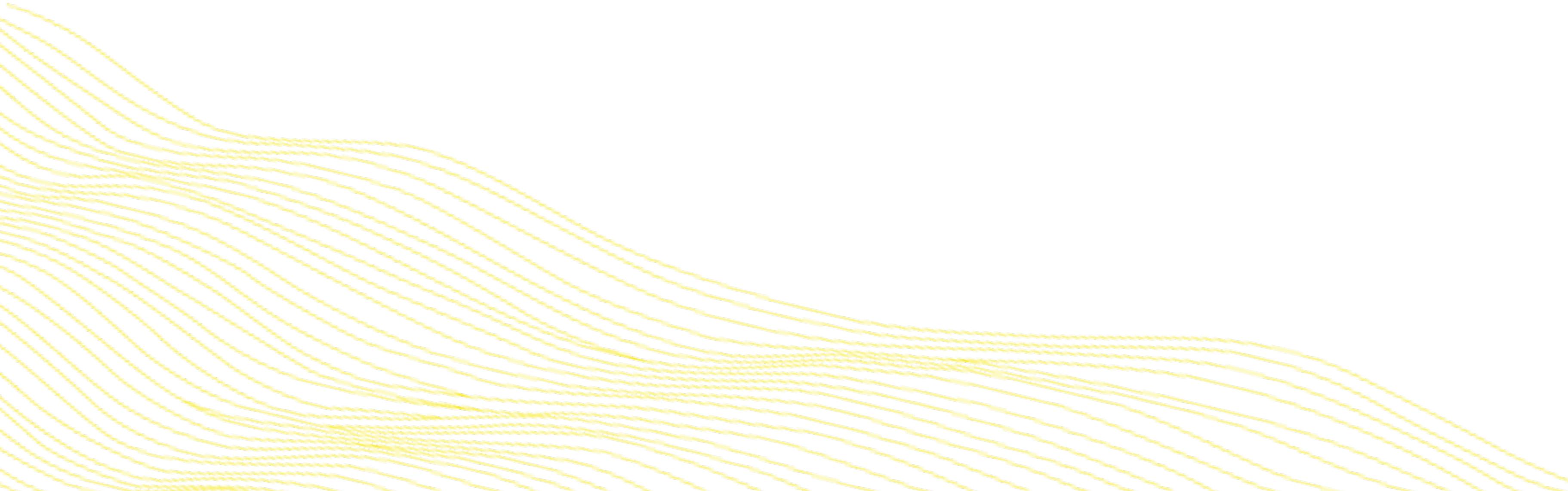
Dr. Steven Boyd

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



H2O Core

What Makes Machine learning at Scale Possible



How H2O Core Works

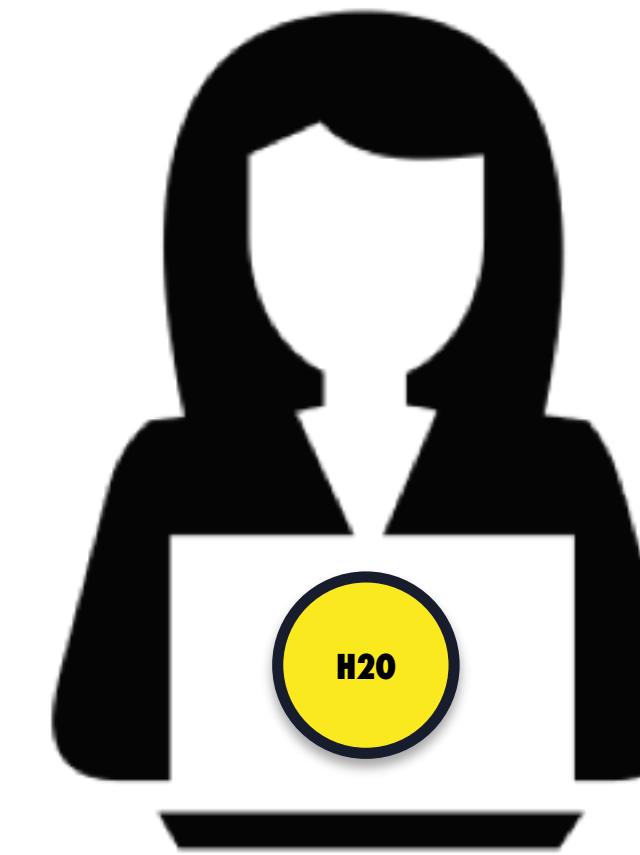


(just a java application)

How H2O Core Works



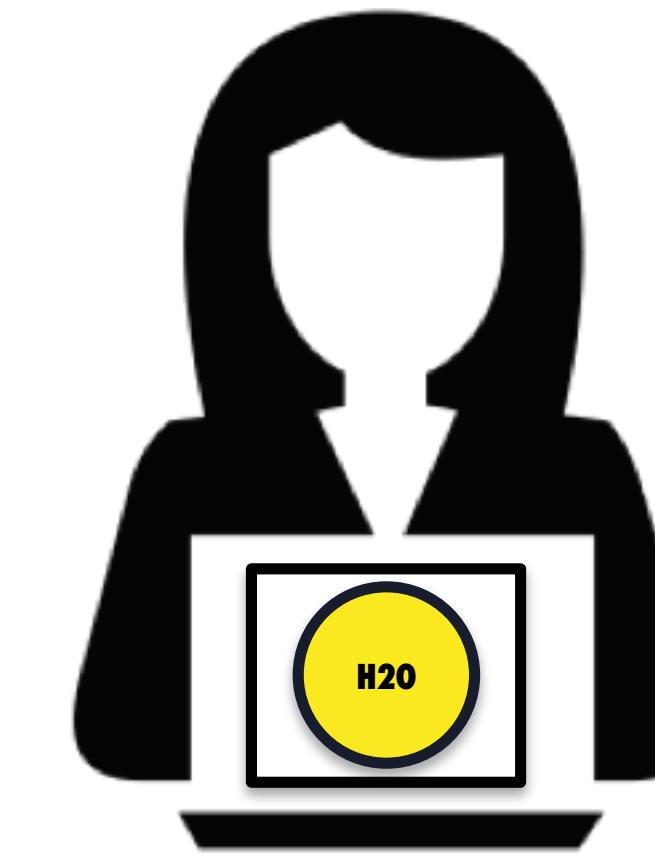
How H2O Core Works



on a laptop



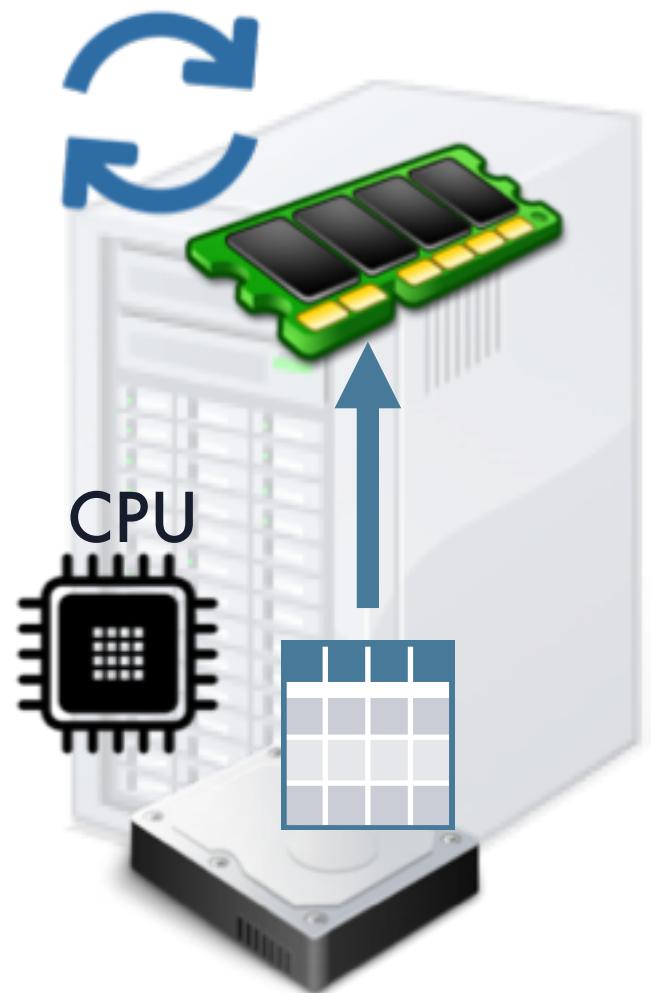
on a virtual machine



in a container

H2O Core

Model Building

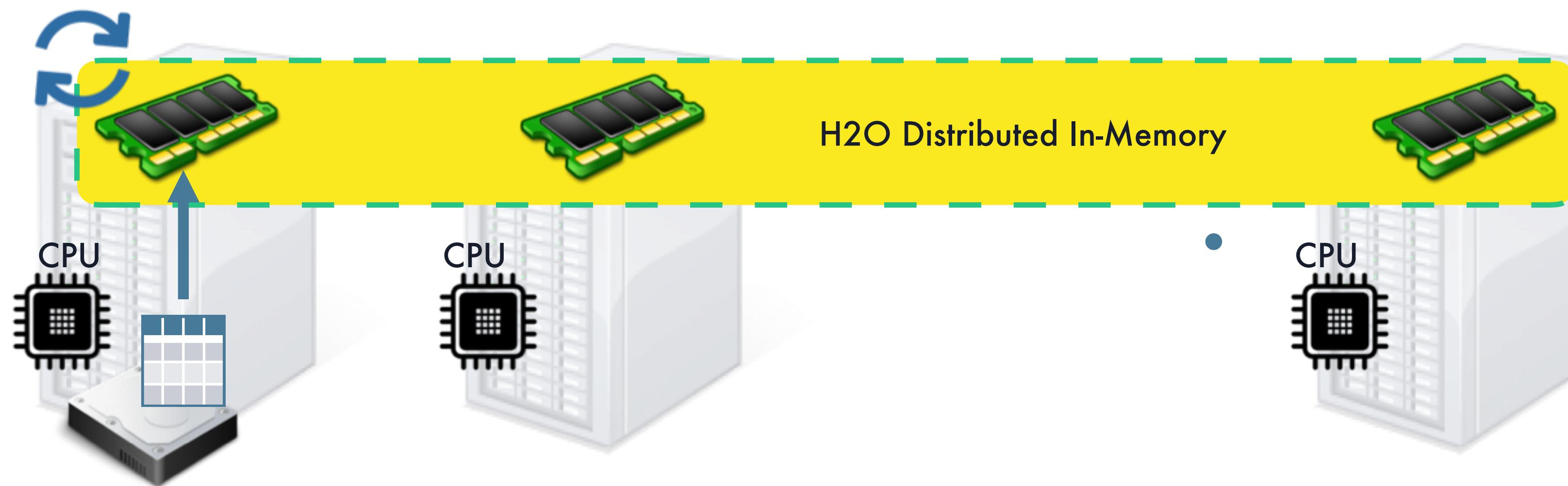


H2O Core

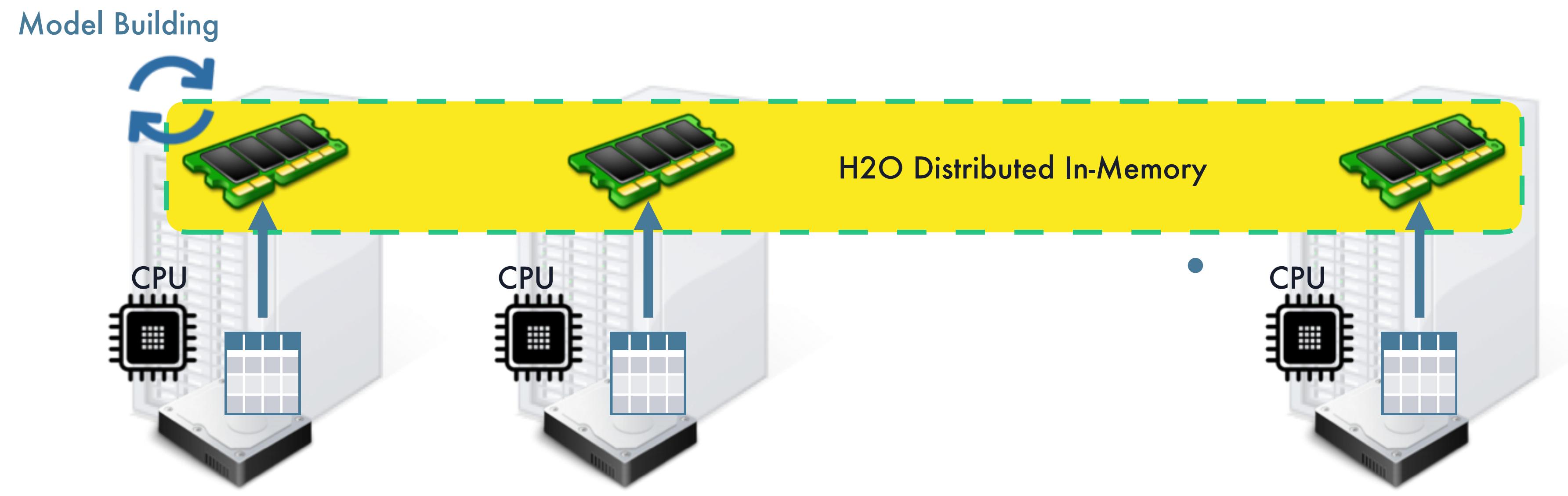


H2O Core

Model Building



H2O Core with Other Data Sources

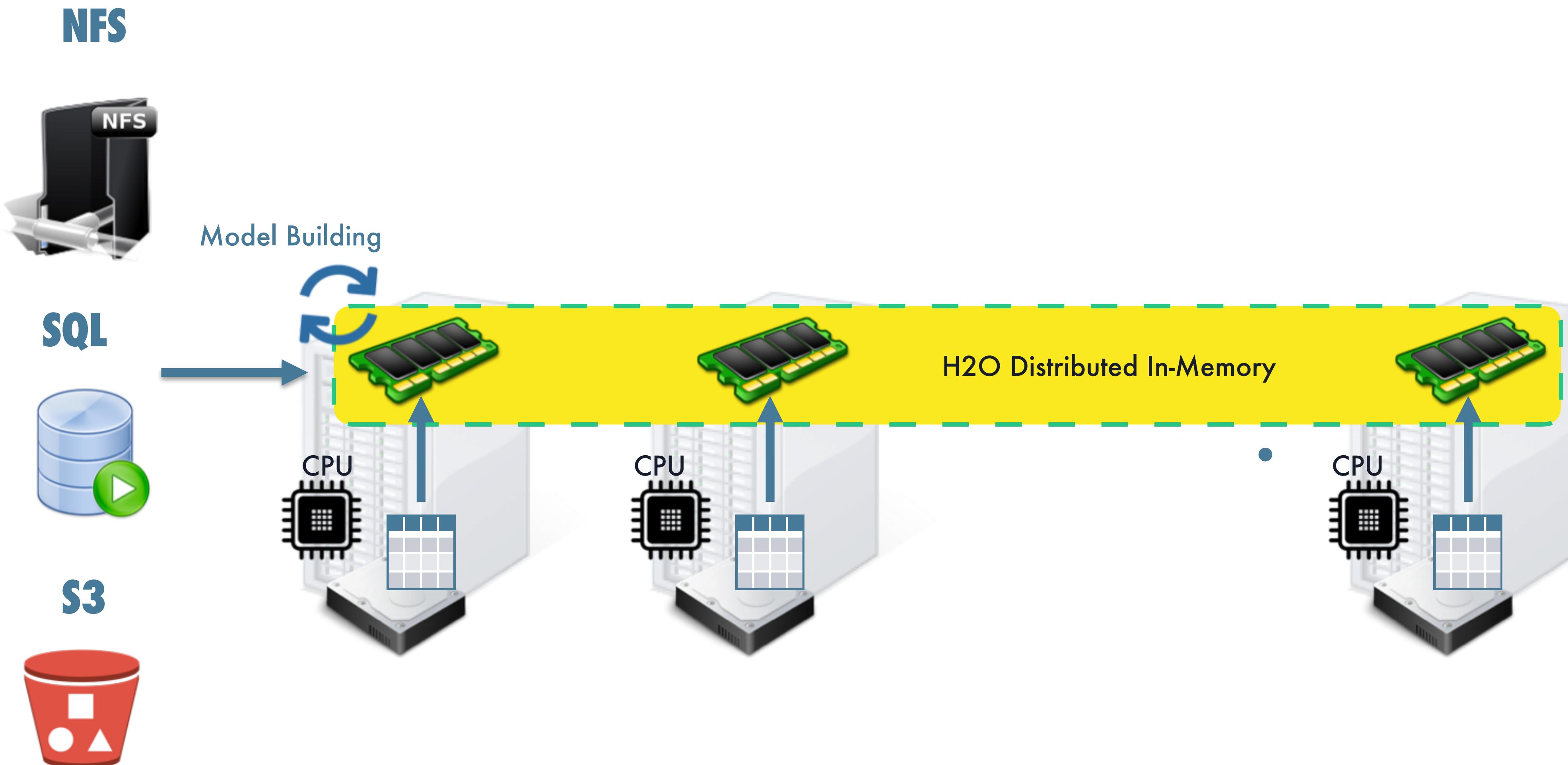


YARN

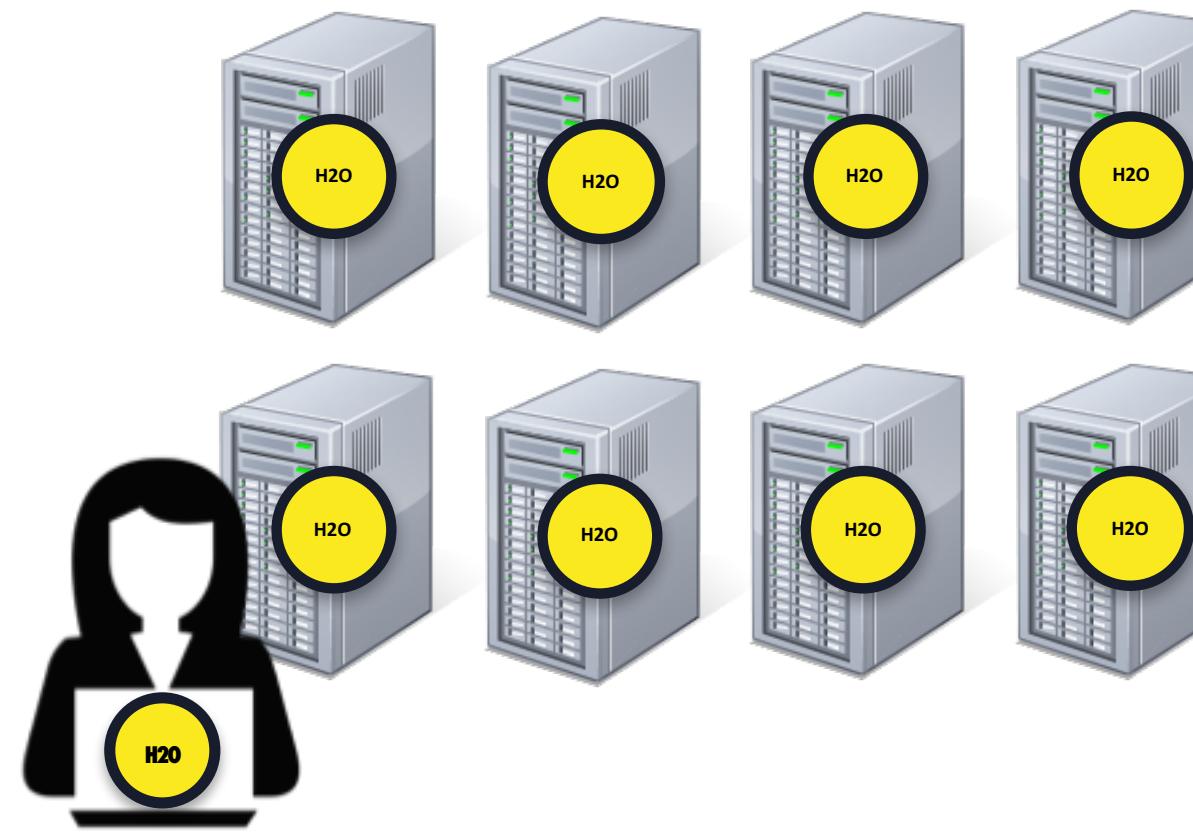
cloudera Hortonworks

MAPR

H2O Core with Other Data Sources



H2O Distributed Environments



distributed across multiple machines



distributed on the cloud

Interfacing with H₂O

(Python, R, and Flow)



Python API

The screenshot shows a Jupyter Notebook interface with a yellow header bar. The menu bar includes File, Edit, View, Insert, Cell, Kernel, Help, and a Python 3 kernel indicator. Below the menu is a toolbar with various icons for file operations like save, new, and copy, along with buttons for Code and CellToolbar.

The main area displays a code cell labeled "In []:" containing Python code for working with the Boston housing dataset using the H2O Generalized Linear Estimator. The code includes importing h2o, initializing the H2O environment, loading the Boston dataset from S3, setting predictor and response variables, splitting the data into training and validation sets, training a model with alpha=0.25, and printing the validation MSE.

```
In [ ]: import h2o
from h2o.estimators.glm import H2OGeneralizedLinearEstimator
h2o.init()

# import the boston dataset:
# this dataset looks at features of the boston suburbs and predicts median housing prices
# the original dataset can be found at https://archive.ics.uci.edu/ml/datasets/Housing
boston = h2o.import_file("https://s3.amazonaws.com/h2o-public-test-data/smalldata/gbm_test/BostonHousing.csv")

# set the predictor names and the response column name
predictors = boston.columns[:-1]
# set the response column to "medv", the median value of owner-occupied homes in $1000's
response = "medv"

# convert the chas column to a factor (chas = Charles River dummy variable (= 1 if tract bounds
# crosses river)
boston['chas'] = boston['chas'].asfactor()

# split into train and validation sets
train, valid = boston.split_frame(ratios = [.8])

# try using the `alpha` parameter:
# initialize the estimator then train the model
boston_glm = H2OGeneralizedLinearEstimator(alpha = .25)
boston_glm.train(x = predictors, y = response, training_frame = train, validation_frame = valid)

# print the mse for the validation data
print(boston_glm.mse(valid=True))
```

Python API

Import the python package

```
import h2o
from h2o.estimators.glm import H2OGeneralizedLinearEstimator
h2o.init()

# import the boston dataset:
# this dataset looks at features of the boston suburbs and predi
# the original dataset can be found at https://archive.ics.uci.e
boston = h2o.import_file("https://s3.amazonaws.com/h2o-public-te
```

Python API

Launch an H2O cluster

```
import h2o
from h2o.estimators.glm import H2OGeneralizedLinearEstimator
h2o.init()  
  
# import the boston dataset:  
# this dataset looks at features of the boston suburbs and predi  
# the original dataset can be found at https://archive.ics.uci.e  
boston = h2o.import_file("https://s3.amazonaws.com/h2o-public-te
```

Python API

Import a Dataset

```
import h2o
from h2o.estimators.glm import H2OGeneralizedLinearEstimator
h2o.init()

# import the boston dataset:
# this dataset looks at features of the boston suburbs and predi
# the original dataset can be found at https://archive.ics.uci.e
boston = h2o.import_file("https://s3.amazonaws.com/h2o-public-te
```

Python API

Build a Model

```
# initialize the estimator then train the model
boston_glm = H2OGeneralizedLinearEstimator(alpha = .25)
boston_glm.train(x = predictors, y = response, training_frame = tra

# print the mse for the validation data
print(boston_glm.mse(valid=True))
```

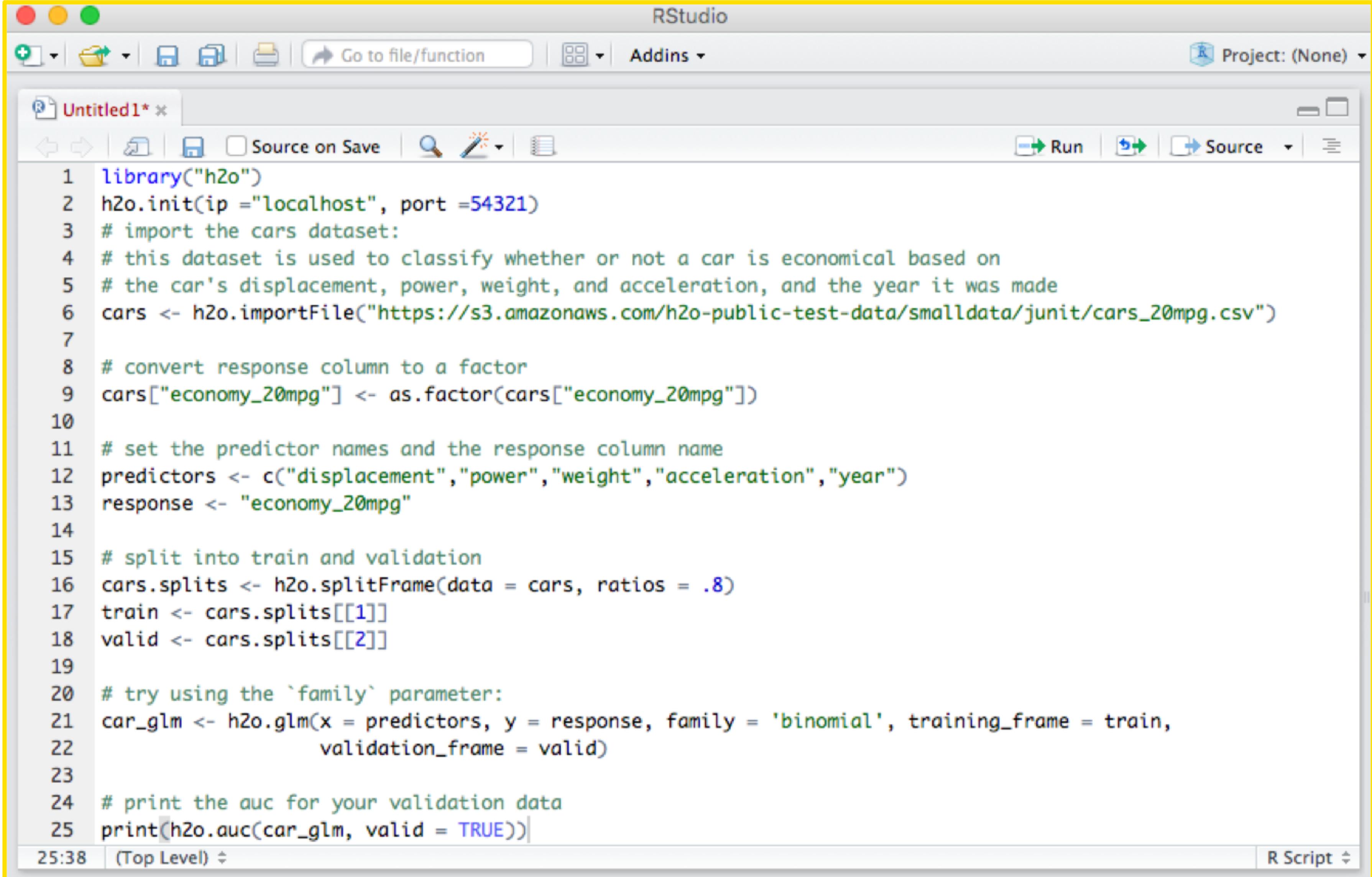
Python API

Get Model Performance

```
# initialize the estimator then train the model
boston_glm = H2OGeneralizedLinearEstimator(alpha = .25)
boston_glm.train(x = predictors, y = response, training_frame = tra

# print the mse for the validation data
print(boston_glm.mse(valid=True))
```

R API



The screenshot shows the RStudio interface with the following details:

- Title Bar:** RStudio
- Toolbar:** Includes standard icons for file operations (New, Open, Save, Print) and a "Go to file/function" search bar.
- Project Bar:** Shows "Project: (None)".
- Code Editor:** An untitled R script titled "Untitled1". The code uses the `h2o` package to load a dataset, split it into training and validation sets, and then fit a logistic regression model using the `h2o.glm` function.

```
library("h2o")
h2o.init(ip = "localhost", port = 54321)
# import the cars dataset:
# this dataset is used to classify whether or not a car is economical based on
# the car's displacement, power, weight, and acceleration, and the year it was made
cars <- h2o.importFile("https://s3.amazonaws.com/h2o-public-test-data/smalldata/junit/cars_20mpg.csv")

# convert response column to a factor
cars["economy_20mpg"] <- as.factor(cars["economy_20mpg"])

# set the predictor names and the response column name
predictors <- c("displacement", "power", "weight", "acceleration", "year")
response <- "economy_20mpg"

# split into train and validation
cars.splits <- h2o.splitFrame(data = cars, ratios = .8)
train <- cars.splits[[1]]
valid <- cars.splits[[2]]

# try using the `family` parameter:
car_glm <- h2o.glm(x = predictors, y = response, family = 'binomial', training_frame = train,
                     validation_frame = valid)

# print the auc for your validation data
print(h2o.auc(car_glm, valid = TRUE))
```

- Status Bar:** Shows "25:38 (Top Level)" and "R Script".
- Bottom Right Corner:** H2O.ai logo.

Flow: Custom Web UI

The screenshot shows the H2O Flow web application interface. At the top, there is a navigation bar with the H2O logo and links for Flow, Cell, Data, Model, Score, Admin, and Help. Below the navigation bar is a toolbar with various icons for file operations like upload, download, and delete, as well as navigation and search functions.

In the main area, there is a code editor window titled "Financial Models" containing the following code:

```
CS buildModel "gbm"
```

The execution time is listed as 21ms. Below the code editor is a section titled "Build a Model".

The "Build a Model" section includes a dropdown menu for "Select an algorithm" set to "Gradient Boosting Machine".

Below the algorithm selection is a "PARAMETERS" section with the following configuration:

Parameter	Value	Description
model_id	gbm-39530a5a-cdd9-494c-912e-11191ac7edd3	Destination id for this model; auto-generated if not specified.
training_frame	(Choose...)	Id of the training data frame.
validation_frame	(Choose...)	Id of the validation data frame.
nfolds	0	Number of folds for K-fold cross-validation (0 to disable or >= 2).
response_column	(Choose...)	Response variable column.
ignored_columns	Search...	

On the right side of the parameters, there are two checkboxes labeled "GRID?" and an "Edit" icon.

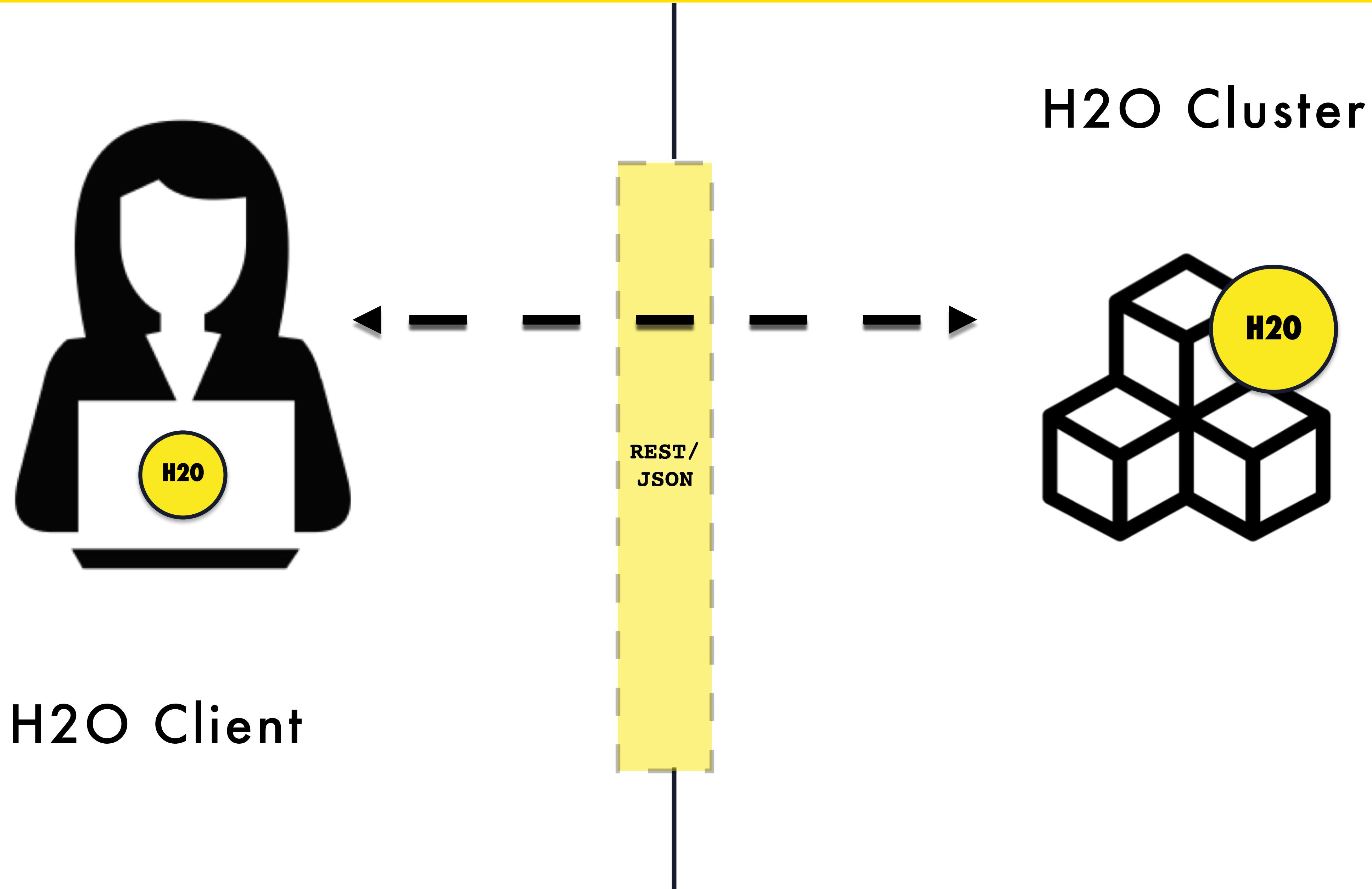
The H2O Python API*

(how the client & server communicate)



*Same applies for R

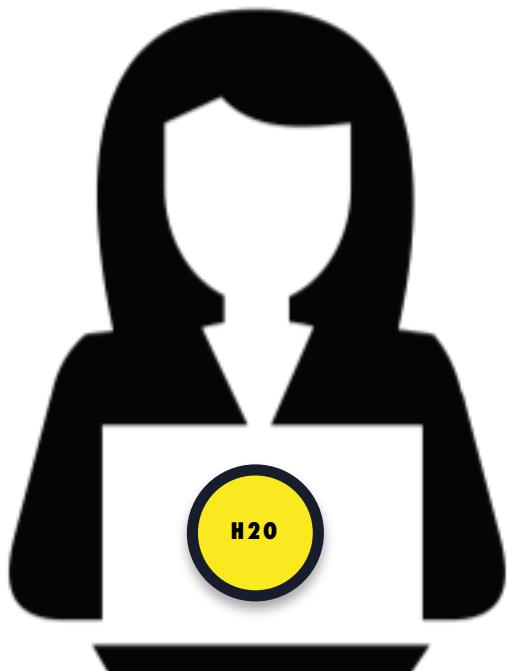
Client & Cluster Communication



Communication Layers: A Command

Importing Big Data with Python Code

Python Commands



the Client

A screenshot of a Jupyter Notebook interface. The menu bar includes File, Edit, View, Insert, Cell, Kernel, and Help. The toolbar includes icons for file operations and a CellToolbar button. The code cell contains the following Python script:

```
In [ ]: import h2o
from h2o.estimators.glm import H2OGeneralizedLinearEstimator
h2o.init()

# import the boston dataset:
# this dataset looks at features of the boston suburbs and predicts median housing prices
# the original dataset can be found at https://archive.ics.uci.edu/ml/datasets/Housing
boston = h2o.import_file("https://s3.amazonaws.com/h2o-public-test-data/smalldata/gbm_test/BostonHousing.csv")

# set the predictor names and the response column name
predictors = boston.columns[:-1]
# set the response column to "medv", the median value of owner-occupied homes in $1000's
response = "medv"

# convert the chas column to a factor (chas = Charles River dummy variable (= 1 if tract bounds
# crosses river)
boston['chas'] = boston['chas'].asfactor()

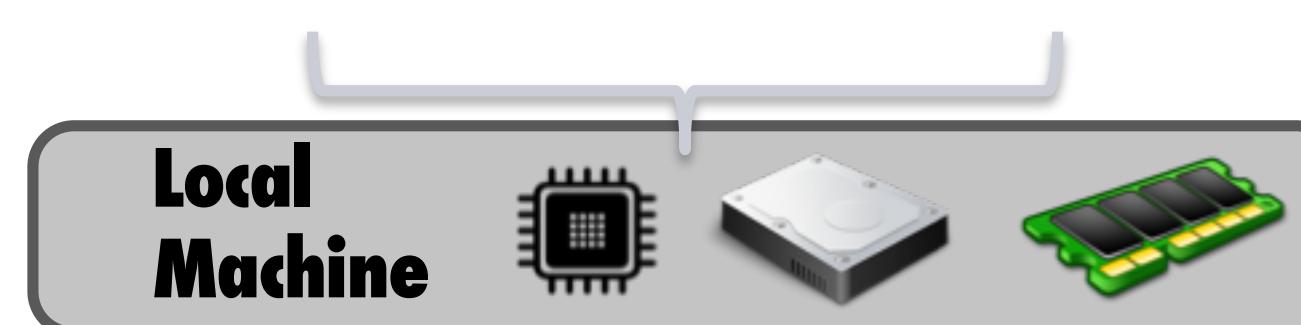
# split into train and validation sets
train, valid = boston.split_frame(ratios = [.8])

# try using the `alpha` parameter:
# initialize the estimator then train the model
boston_glm = H2OGeneralizedLinearEstimator(alpha = .25)
boston_glm.train(x = predictors, y = response, training_frame = train, validation_frame = valid)

# print the mse for the validation data
print(boston_glm.mse(valid=True))
```

Path to Your Dataset

h2o.import_file(...)



Communication



h2o.import_file(...)
requests file import

```
In [ ]: import h2o
from h2o.estimators.glm import H2OGeneralizedLinearEstimator
h2o.init()

# import the boston dataset:
# this dataset looks at features of the boston suburbs and predicts median housing prices
# the original dataset can be found at https://archive.ics.uci.edu/ml/datasets/Housing
boston = h2o.import_file("https://s3.amazonaws.com/h2o-public-test-data/smalldata/gbm_test/BostonHousing.csv")

# set the predictor names and the response column name
predictors = boston.columns[:-1]
# set the response column to "medv", the median value of owner-occupied homes in $1000's
response = "medv"

# convert the chas column to a factor (chas = Charles River dummy variable (= 1 if tract bounds
# crosses river)
boston['chas'] = boston['chas'].asfactor()

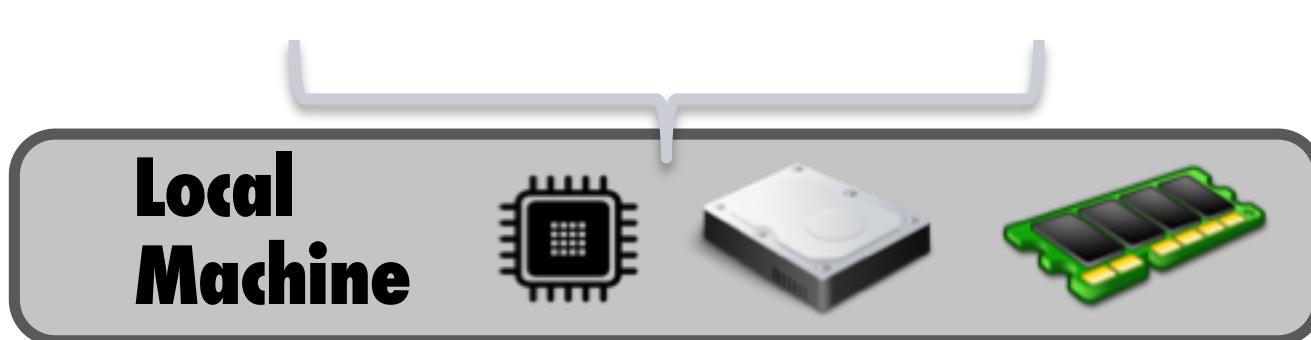
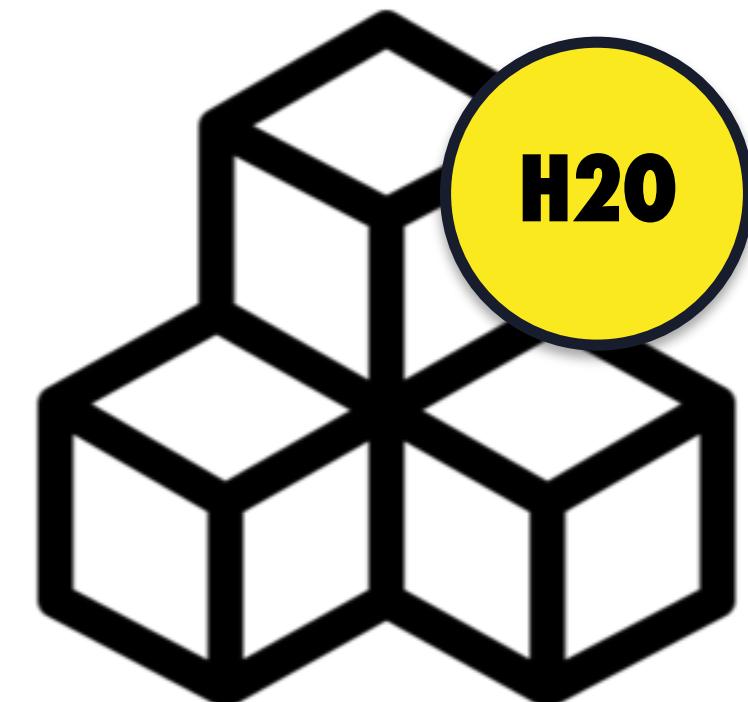
# split into train and validation sets
train, valid = boston.split_frame(ratios = [.8])

# try using the `alpha` parameter:
# initialize the estimator then train the model
boston_glm = H2OGeneralizedLinearEstimator(alpha = .25)
boston_glm.train(x = predictors, y = response, training_frame = train, validation_frame = valid)

# print the mse for the validation data
print(boston_glm.mse(valid=True))
```

REST
/
JSON

H2O Cluster



Cluster Does Heavy Lifting



A screenshot of a Jupyter Notebook interface. The menu bar includes File, Edit, View, Insert, Cell, Kernel, and Help. The toolbar below has icons for file operations and cell execution. The code cell contains the following Python script:

```
In [ ]: import h2o
from h2o.estimators.glm import H2OGeneralizedLinearEstimator
h2o.init()

# import the boston dataset:
# this dataset looks at features of the boston suburbs and predicts median housing prices
# the original dataset can be found at https://archive.ics.uci.edu/ml/datasets/Housing
boston = h2o.import_file("https://s3.amazonaws.com/h2o-public-test-data/smalldata/gbm_test/BostonHousing.csv")

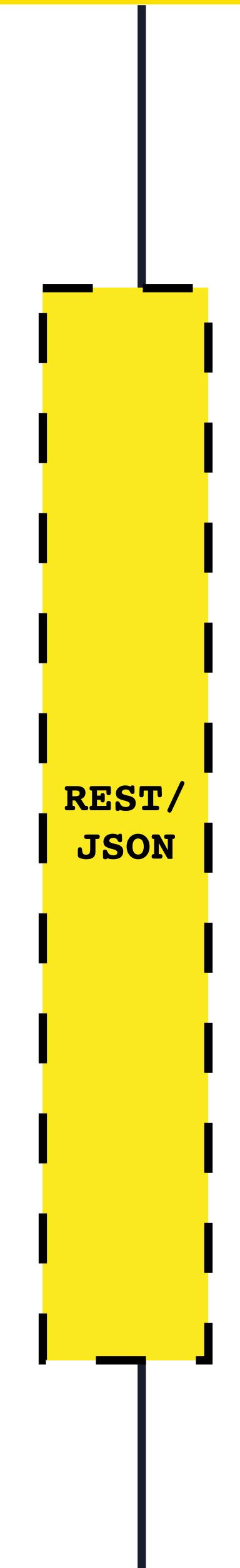
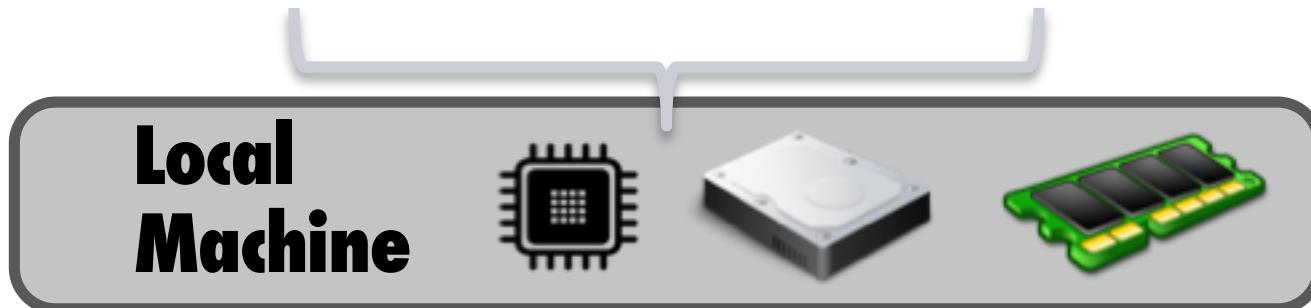
# set the predictor names and the response column name
predictors = boston.columns[:-1]
# set the response column to "medv", the median value of owner-occupied homes in $1000's
response = "medv"

# convert the chas column to a factor (chas = Charles River dummy variable (= 1 if tract bounds
# crosses river)
boston['chas'] = boston['chas'].asfactor()

# split into train and validation sets
train, valid = boston.split_frame(ratios = [.8])

# try using the `alpha` parameter:
# initialize the estimator then train the model
boston_glm = H2OGeneralizedLinearEstimator(alpha = .25)
boston_glm.train(x = predictors, y = response, training_frame = train, validation_frame = valid)

# print the mse for the validation data
print(boston_glm.mse(valid=True))
```



H2O Cluster



Cluster Does Heavy Lifting



```
In [ ]: import h2o
from h2o.estimators.glm import H2OGeneralizedLinearEstimator
h2o.init()

# import the boston dataset:
# this dataset looks at features of the boston suburbs and predicts median housing prices
# the original dataset can be found at https://archive.ics.uci.edu/ml/datasets/Housing
boston = h2o.import_file("https://s3.amazonaws.com/h2o-public-test-data/smalldata/gbm_test/BostonHousing.csv")

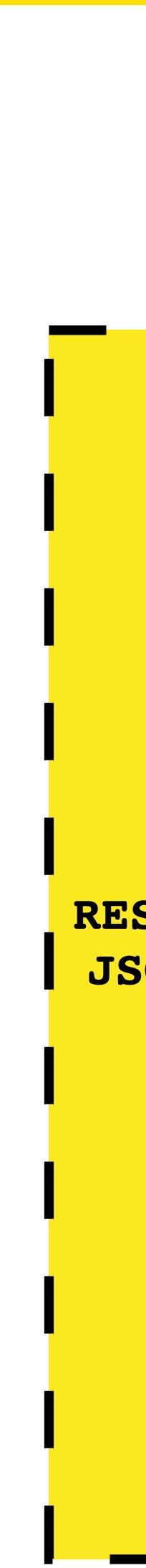
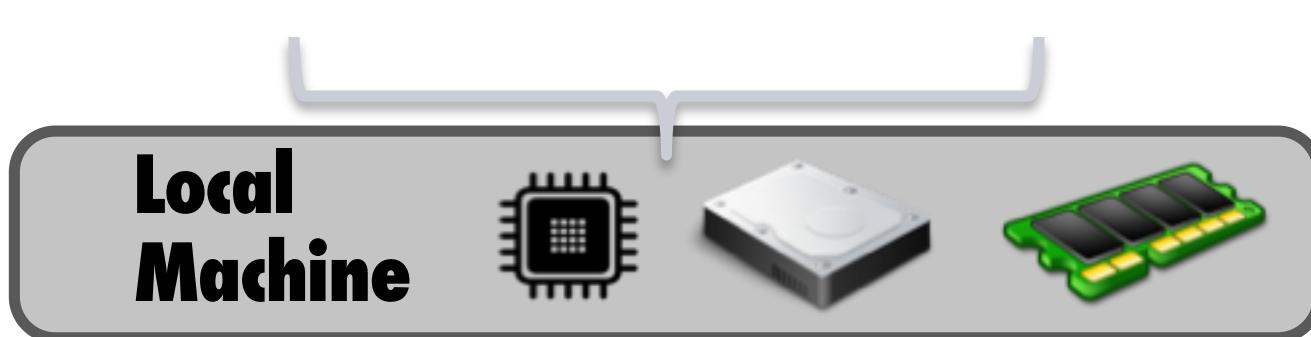
# set the predictor names and the response column name
predictors = boston.columns[:-1]
# set the response column to "medv", the median value of owner-occupied homes in $1000's
response = "medv"

# convert the chas column to a factor (chas = Charles River dummy variable (= 1 if tract bounds
# crosses river)
boston['chas'] = boston['chas'].asfactor()

# split into train and validation sets
train, valid = boston.split_frame(ratios = [.8])

# try using the `alpha` parameter:
# initialize the estimator then train the model
boston_glm = H2OGeneralizedLinearEstimator(alpha = .25)
boston_glm.train(x = predictors, y = response, training_frame = train, validation_frame = valid)

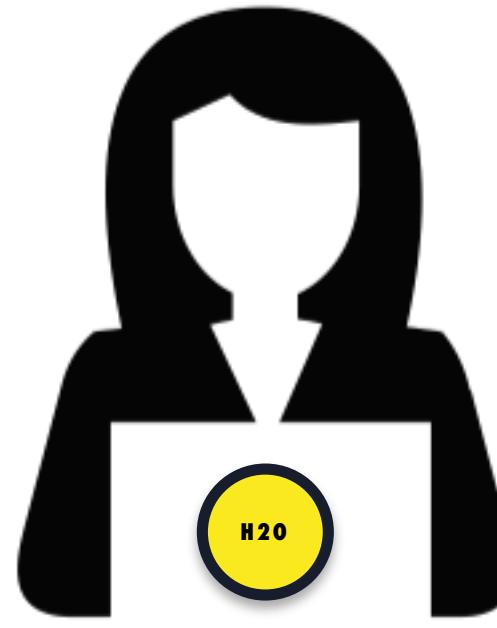
# print the mse for the validation data
print(boston_glm.mse(valid=True))
```



H2O Cluster



Cluster Does Heavy Lifting



```
In [ ]: import h2o
from h2o.estimators.glm import H2OGeneralizedLinearEstimator
h2o.init()

# import the boston dataset:
# this dataset looks at features of the boston suburbs and predicts median housing prices
# the original dataset can be found at https://archive.ics.uci.edu/ml/datasets/Housing
boston = h2o.import_file("https://s3.amazonaws.com/h2o-public-test-data/smalldata/gbm_test/BostonHousing.csv")

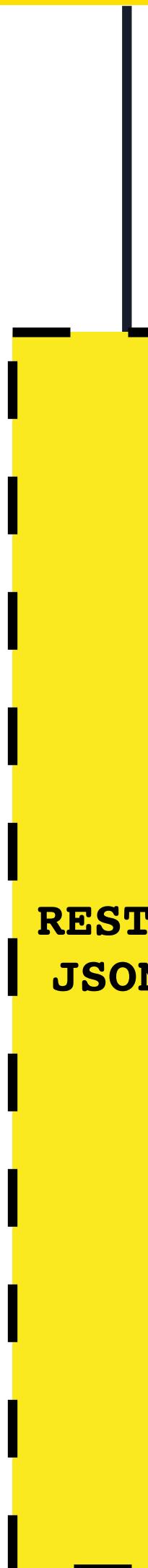
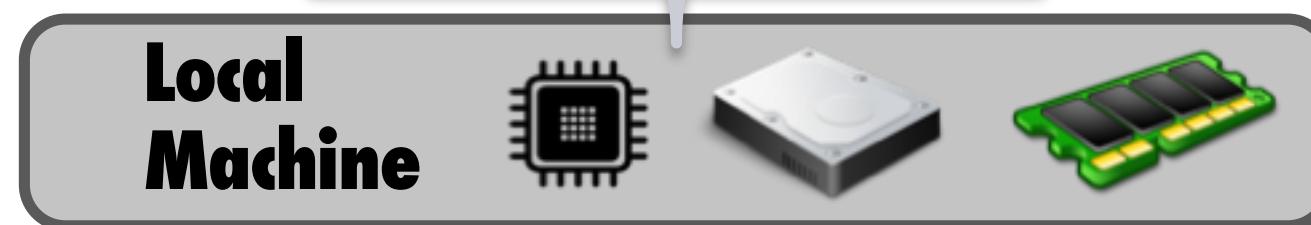
# set the predictor names and the response column name
predictors = boston.columns[:-1]
# set the response column to "medv", the median value of owner-occupied homes in $1000's
response = "medv"

# convert the chas column to a factor (chas = Charles River dummy variable (= 1 if tract bounds
# crosses river)
boston['chas'] = boston['chas'].asfactor()

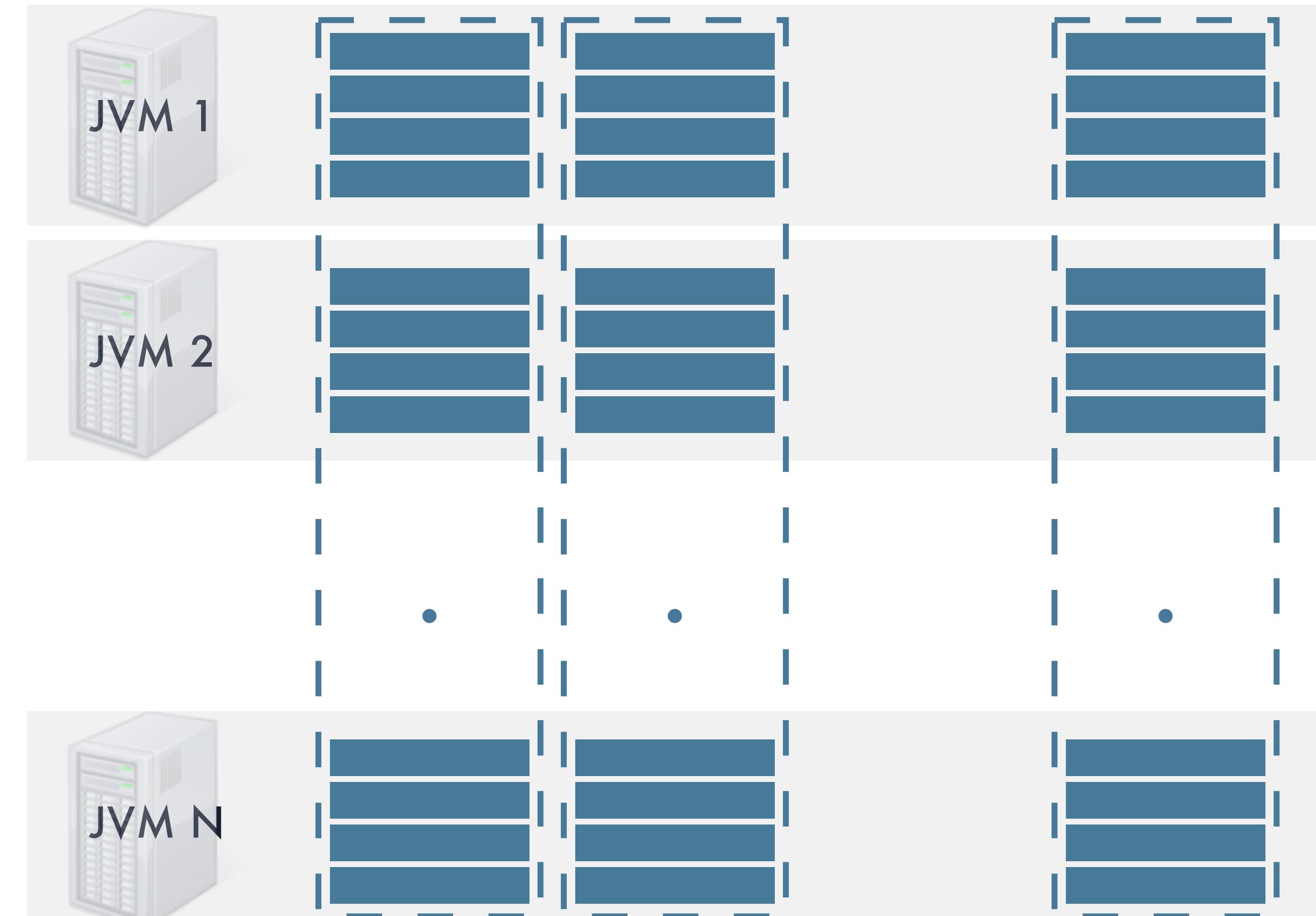
# split into train and validation sets
train, valid = boston.split_frame(ratios = [.8])

# try using the `alpha` parameter:
# initialize the estimator then train the model
boston_glm = H2OGeneralizedLinearEstimator(alpha = .25)
boston_glm.train(x = predictors, y = response, training_frame = train, validation_frame = valid)

# print the mse for the validation data
print(boston_glm.mse(valid=True))
```



H2O Cluster



Cluster Does Heavy Lifting



```
In [ ]: import h2o
from h2o.estimators.glm import H2OGeneralizedLinearEstimator
h2o.init()

# import the boston dataset:
# this dataset looks at features of the boston suburbs and predicts median housing prices
# the original dataset can be found at https://archive.ics.uci.edu/ml/datasets/Housing
boston = h2o.import_file("https://s3.amazonaws.com/h2o-public-test-data/smalldata/gbm_test/BostonHousing.csv")

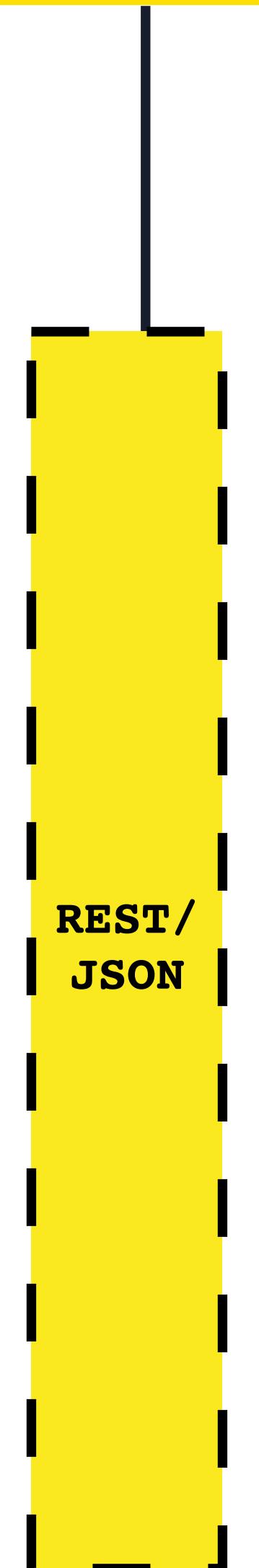
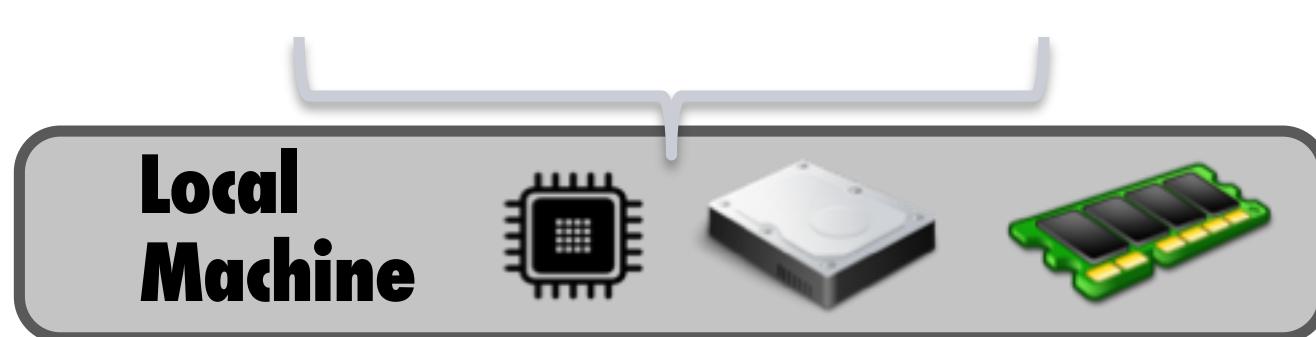
# set the predictor names and the response column name
predictors = boston.columns[:-1]
# set the response column to "medv", the median value of owner-occupied homes in $1000's
response = "medv"

# convert the chas column to a factor (chas = Charles River dummy variable (= 1 if tract bounds
# crosses river)
boston['chas'] = boston['chas'].asfactor()

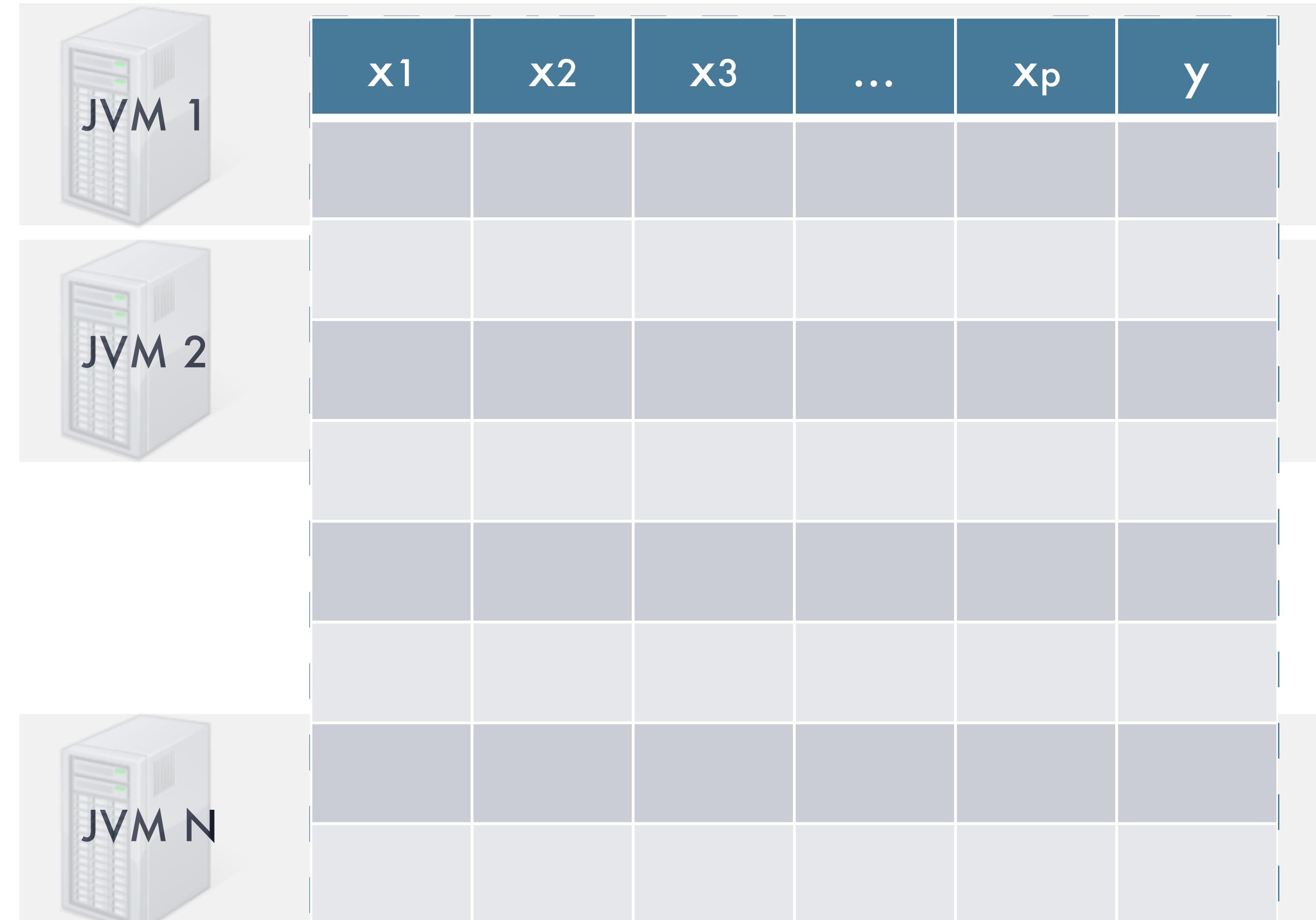
# split into train and validation sets
train, valid = boston.split_frame(ratios = [.8])

# try using the `alpha` parameter:
# initialize the estimator then train the model
boston_glm = H2OGeneralizedLinearEstimator(alpha = .25)
boston_glm.train(x = predictors, y = response, training_frame = train, validation_frame = valid)

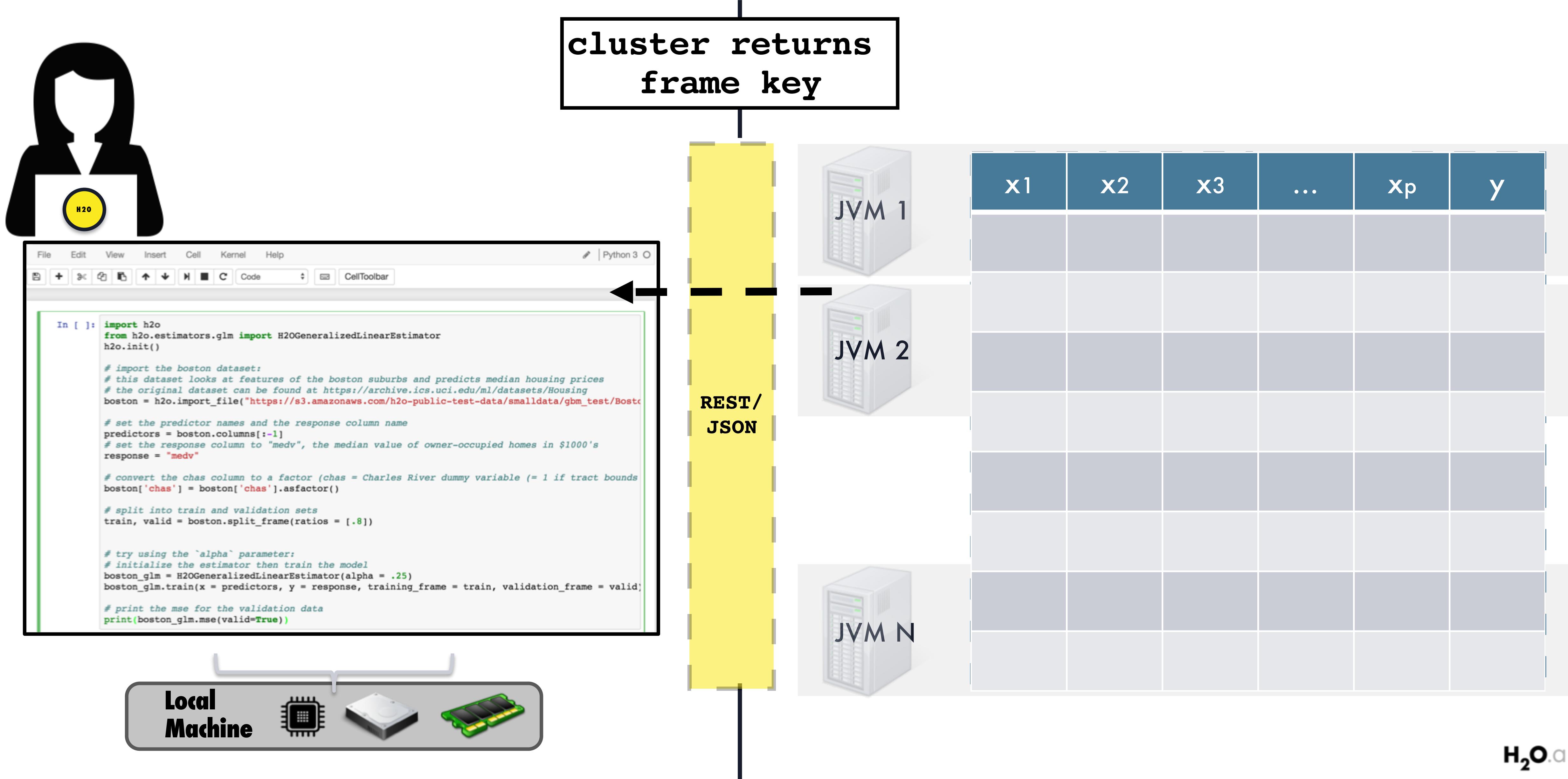
# print the mse for the validation data
print(boston_glm.mse(valid=True))
```



H2O Cluster



H2O Client



The Use Case

(about our data)



Backorders Dataset



The Backorders Data

	sku	national_inv	lead_time	in_transit_qty	forecast_3_month	forecast_6_month
1	1113121	0	8	1	6	6
2	1113268	0	8	0	2	3
3	1113874	20	2	0	45	99
4	1114222	0	8	0	9	14
5	1114823	0	12	0	31	31
6	1115453	55	8	0	216	360

The Use Case: End Goal

Predict if a product will go on backorder

The Use Case: End Goal

Predict if a product will go on backorder
(based on our historic backorders data)

The Use Case: End Goal

Build a model to predict
our **target**:
Yes | No

The Use Case: End Goal

Build a model to predict
our target (aka \hat{Y}):
Yes | No

The Use Case: End Goal

Build a model to predict
our target (**aka response column**):
Yes | No

The Use Case: End Goal

Build a model to predict
our target:
Yes | No

went_on_backorder
Yes

Given Data: How to Learn?

How does an algorithm **learn from data** ?

Learning from Data

Modeling Table

Learning from Data

sku	national_inv	lead_time	...	went_on_backorder
1113121	0	8	...	YES
1113268	0	8	...	NO
1113874	20	2	...	YES
1114222	0	8	...	YES
1114823	0	12	...	NO
1115453	55	8	...	NO
1115620	-34	8	...	NO

Learning from Data

observations

sku	national_inv	lead_time	...	went_on_backorder
1113121	0	8	...	YES
1113268	0	8	...	NO
1113874	20	2	...	YES
1114222	0	8	...	YES
1114823	0	12	...	NO
1115453	55	8	...	NO
1115620	-34	8	...	NO

Learning from Data

training ex.

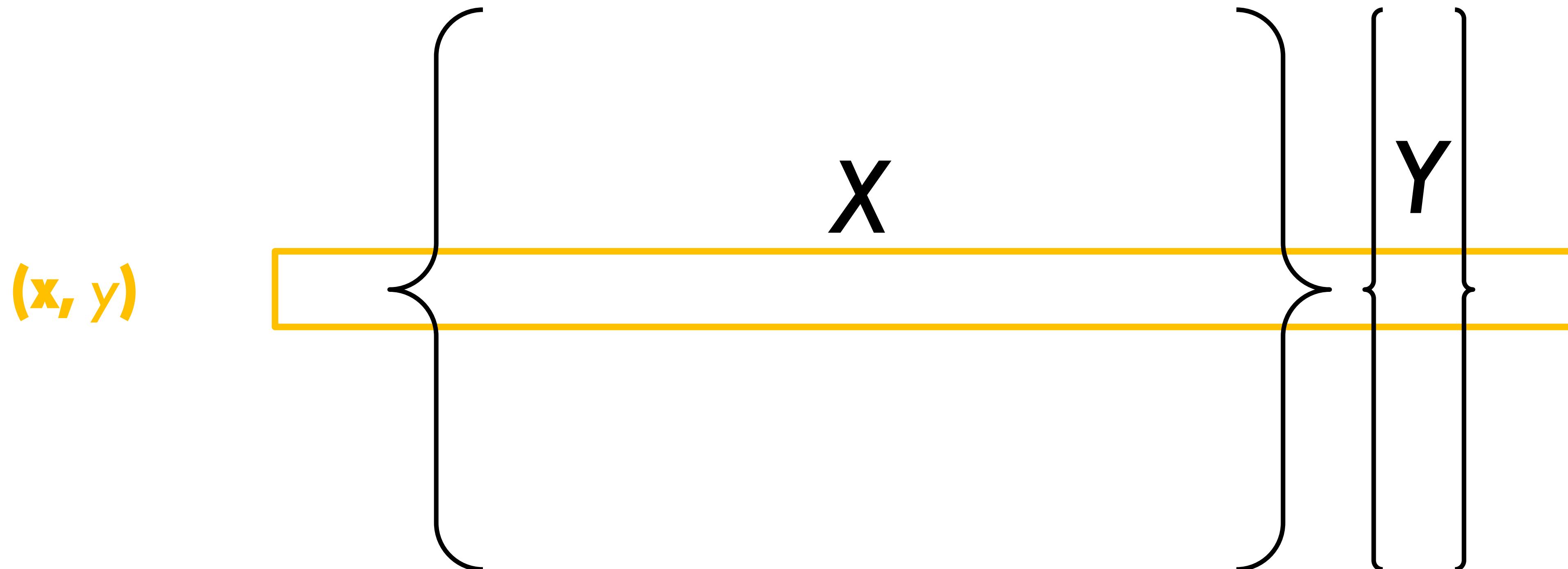
sku	national_inv	lead_time	...	went_on_backorder
1113121	0	8	...	YES
1113268	0	8	...	NO
1113874	20	2	...	YES
1114222	0	8	...	YES
1114823	0	12	...	NO
1115453	55	8	...	NO
1115620	-34	8	...	NO

Learning from Data

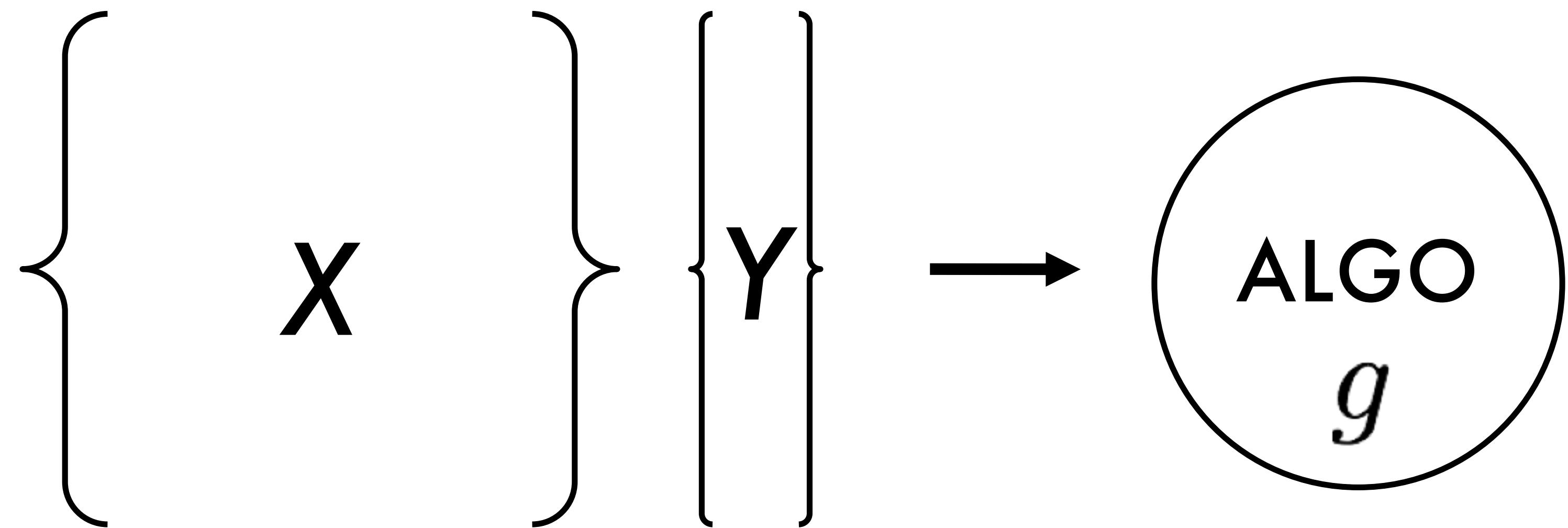
(x, y)

sku	national_inv	lead_time	...	went_on_backorder
1113121	0	8	...	YES
1113268	0	8	...	NO
1113874	20	2	...	YES
1114222	0	8	...	YES
1114823	0	12	...	NO
1115453	55	8	...	NO
1115620	-34	8	...	NO

Learning from Data



Learning from Data

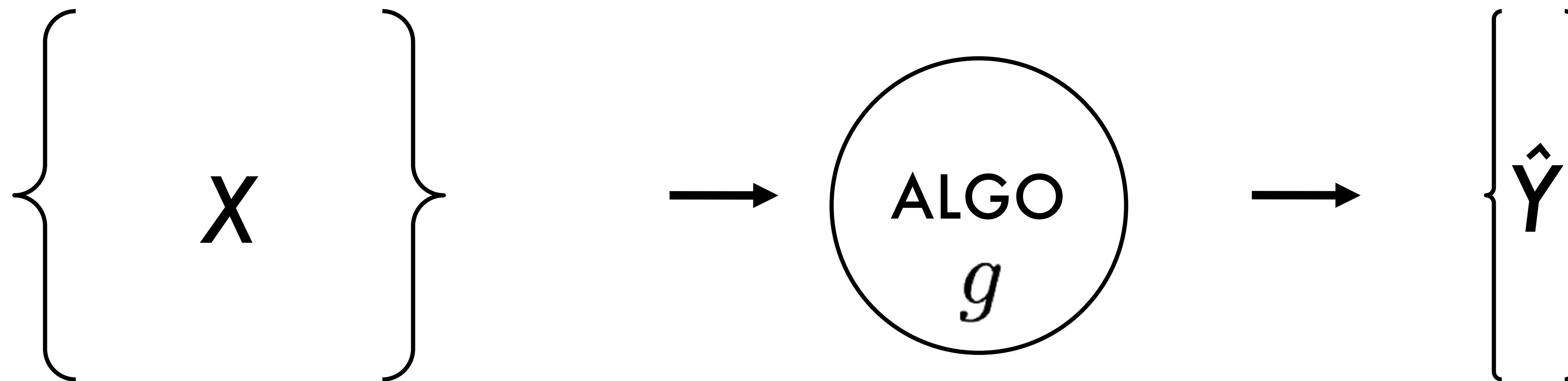


$$g : x \rightarrow y$$

Learn the Pattern

Learning from Data

New Input Data



Supervised Learning

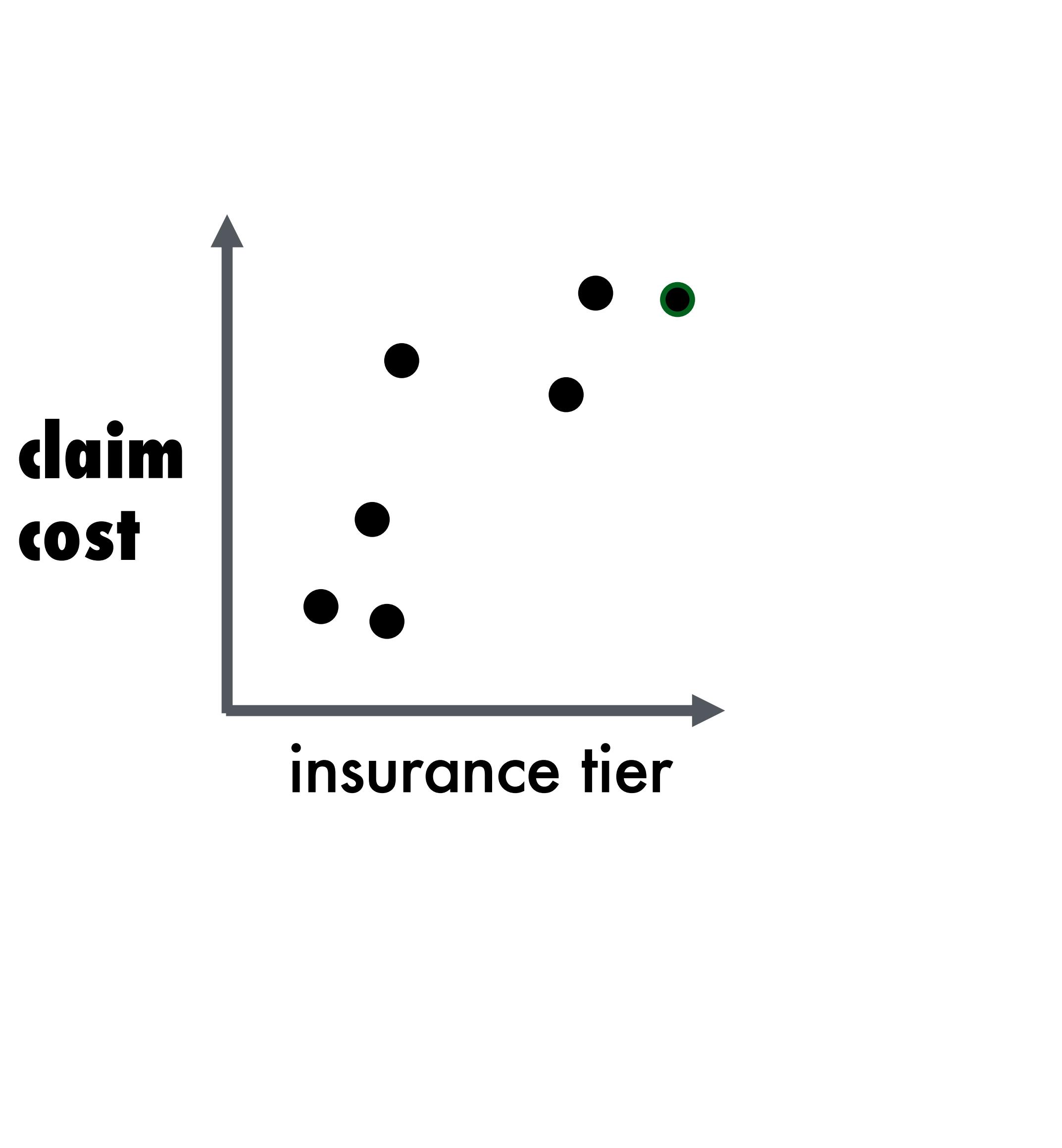
REGRESSION

How much will a claim cost?

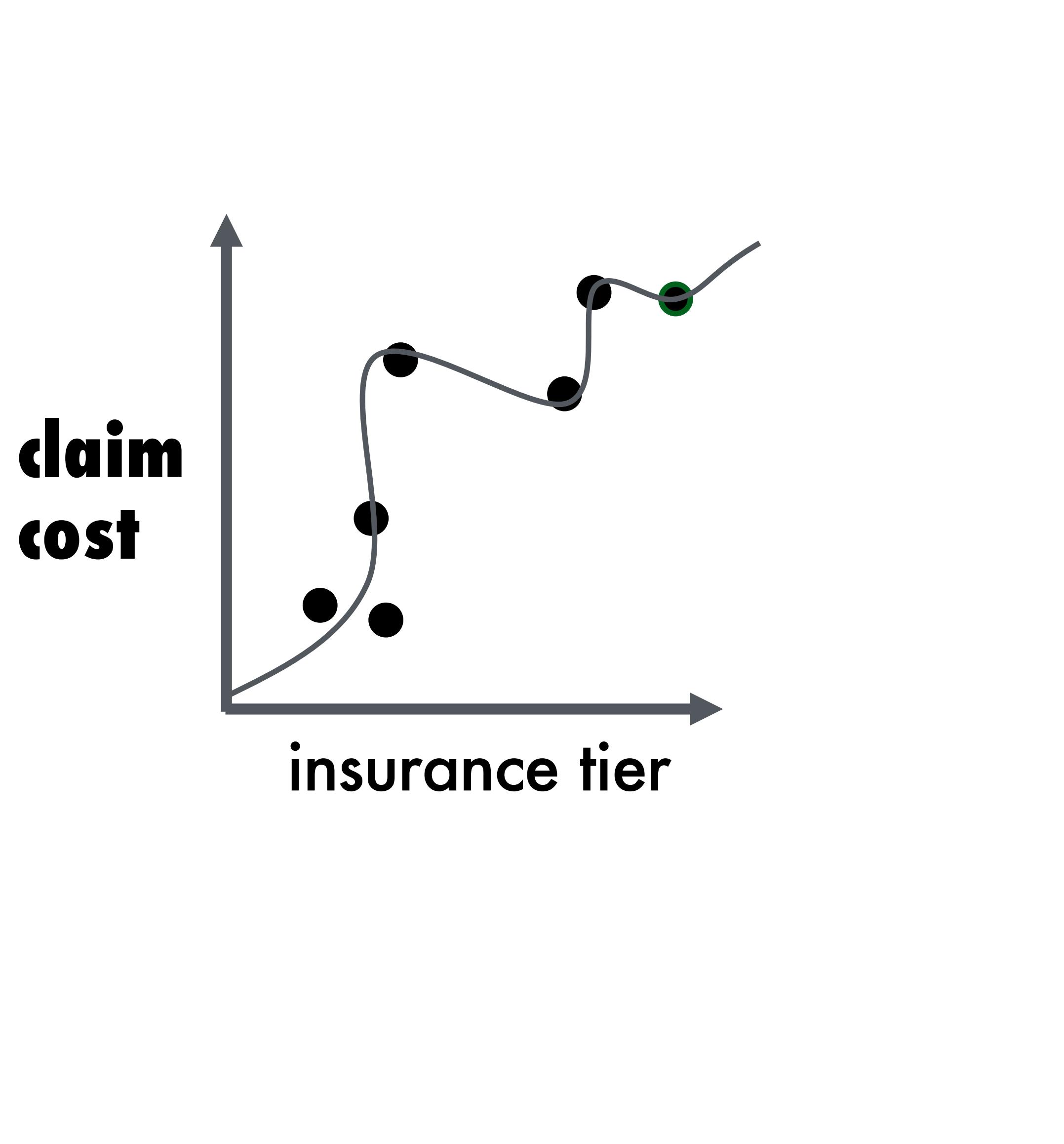
CLASSIFICATION

Will a borrower default on loan?

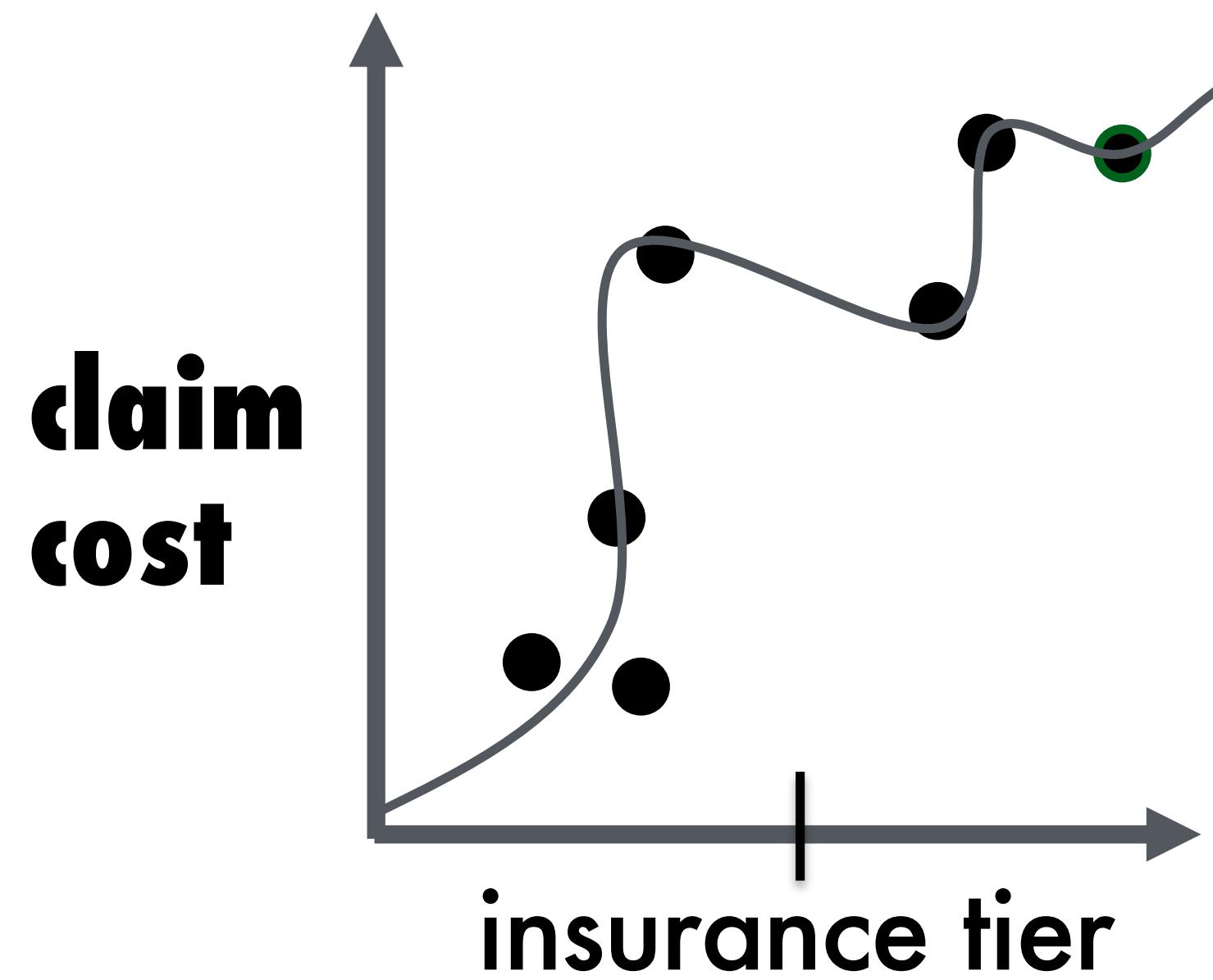
Supervised Learning



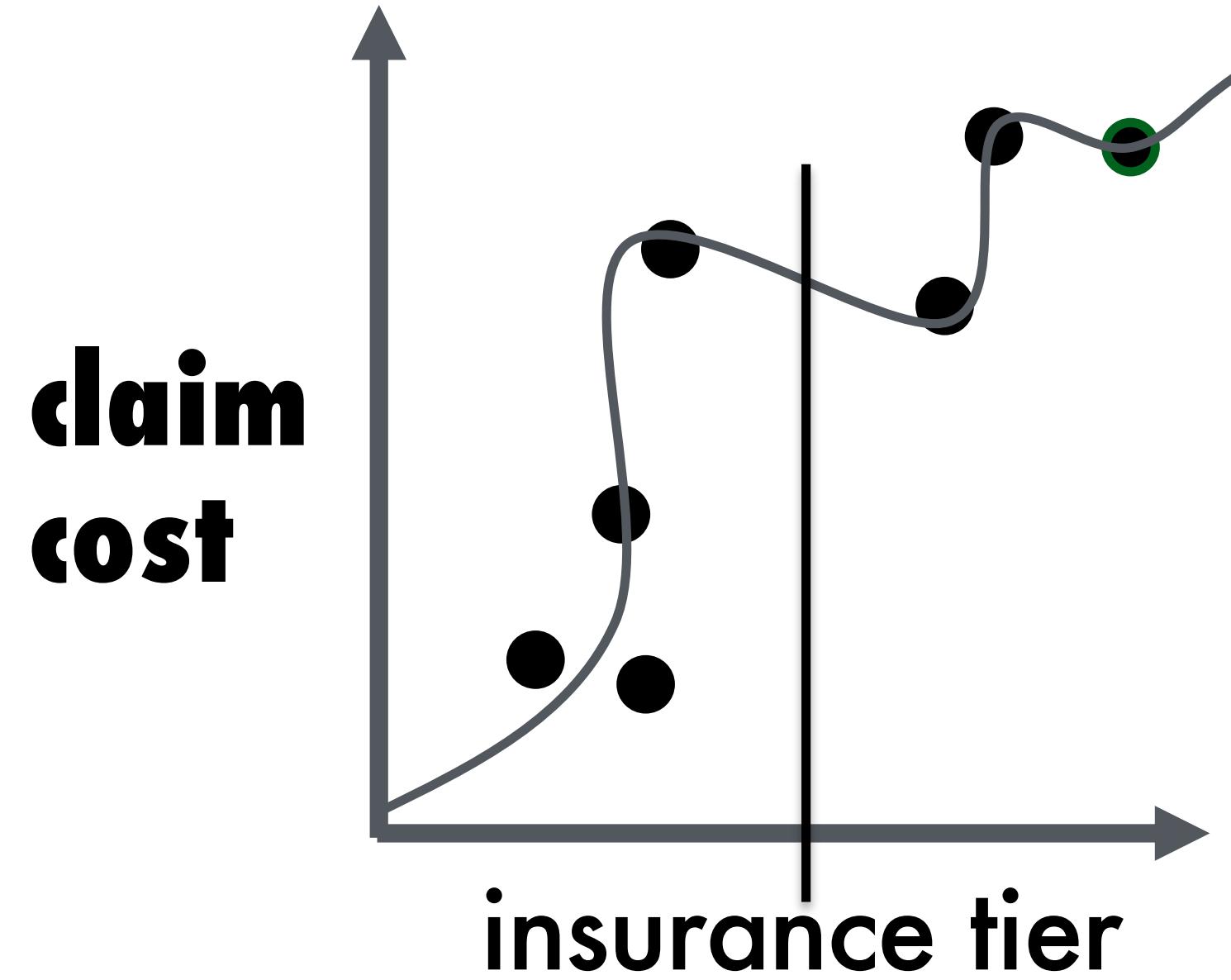
Supervised Learning



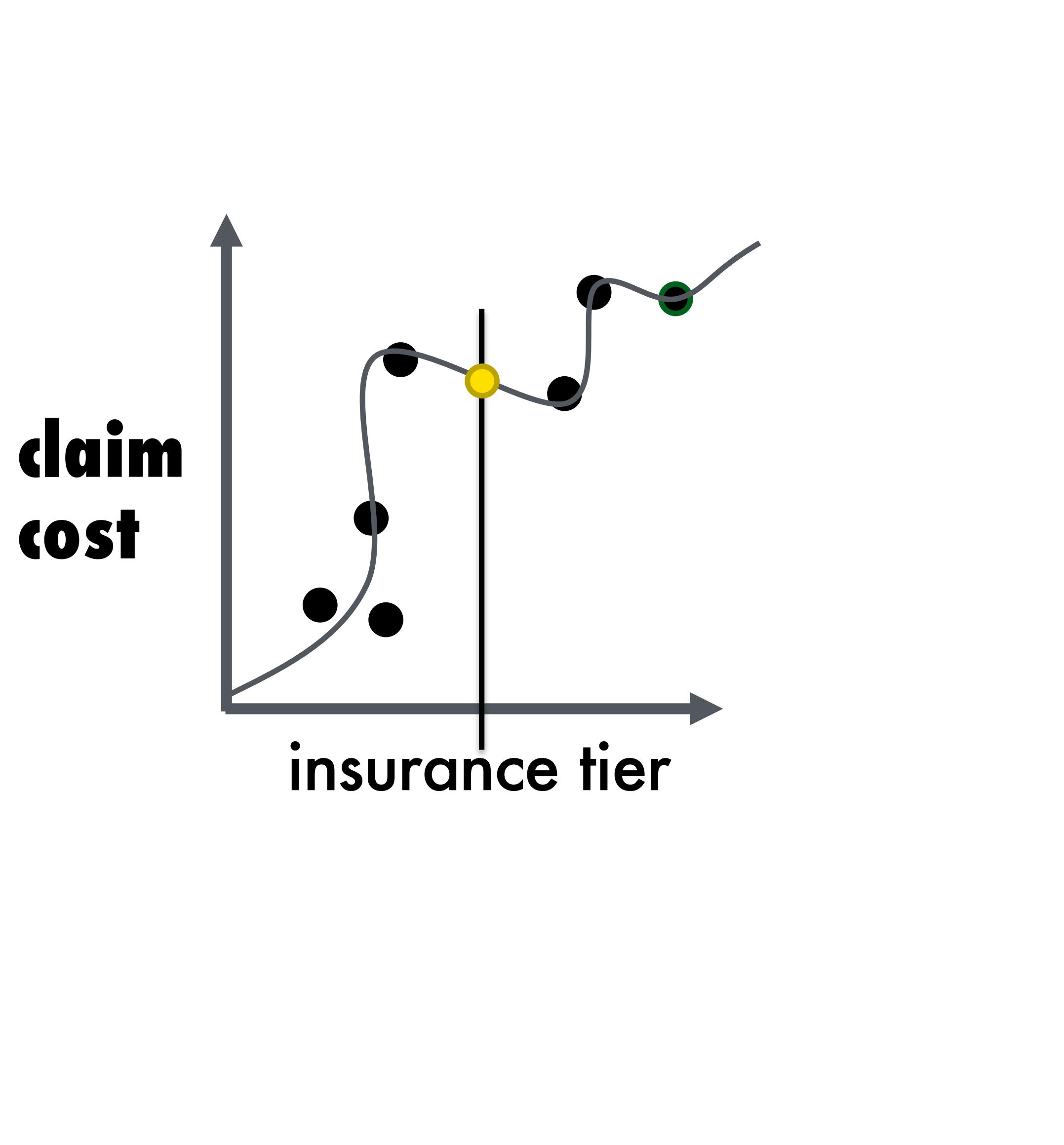
Supervised Learning



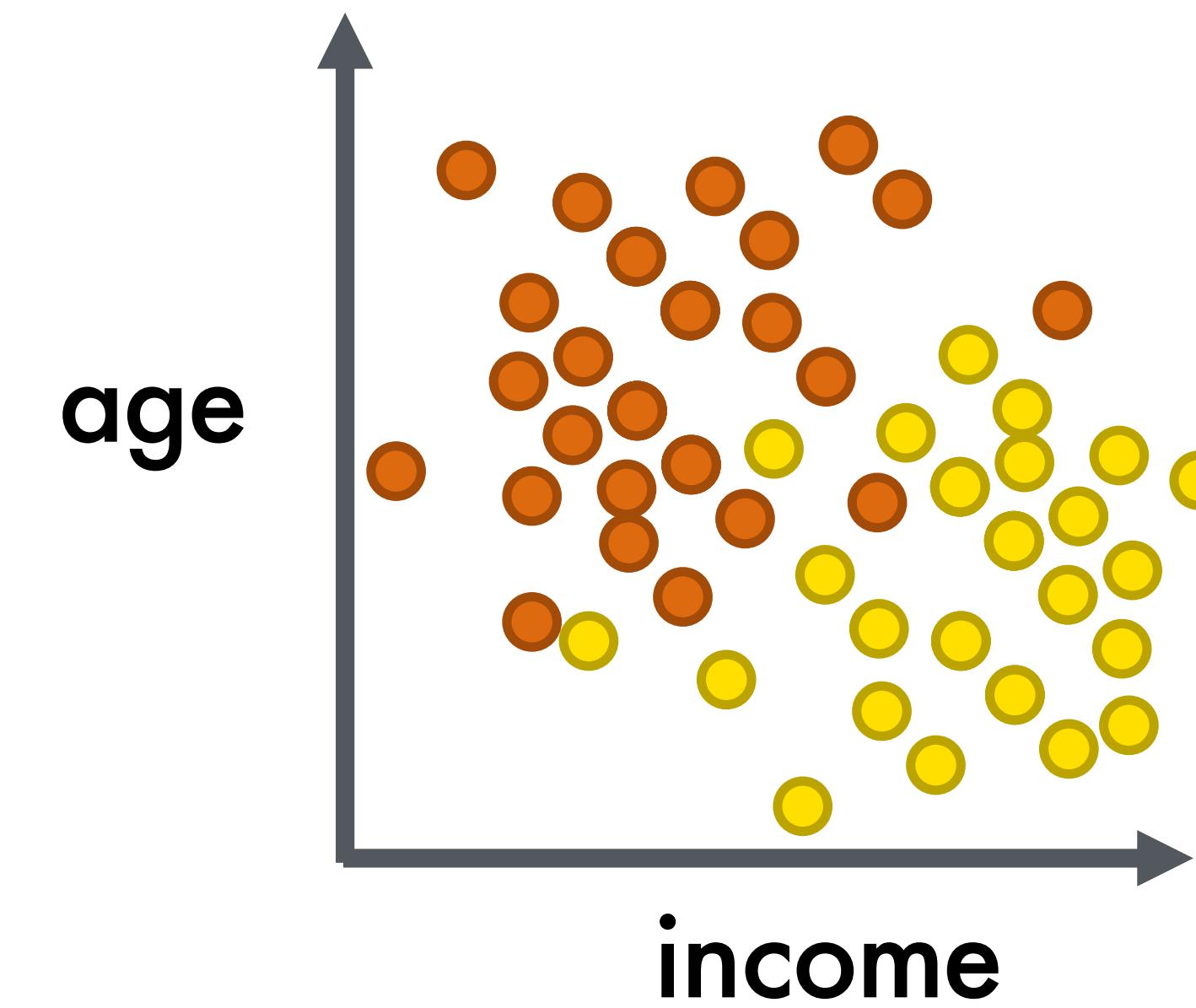
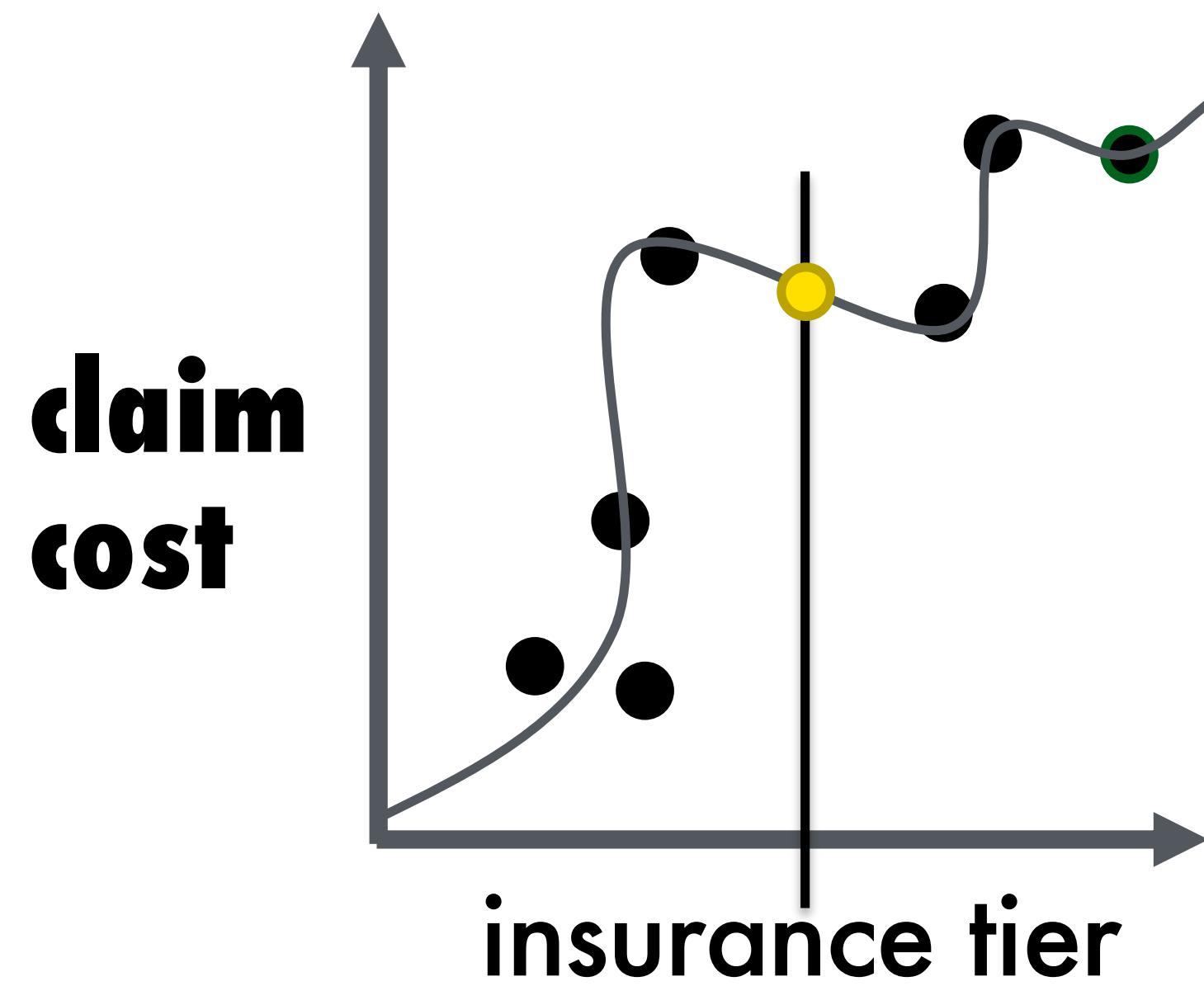
Supervised Learning



Supervised Learning

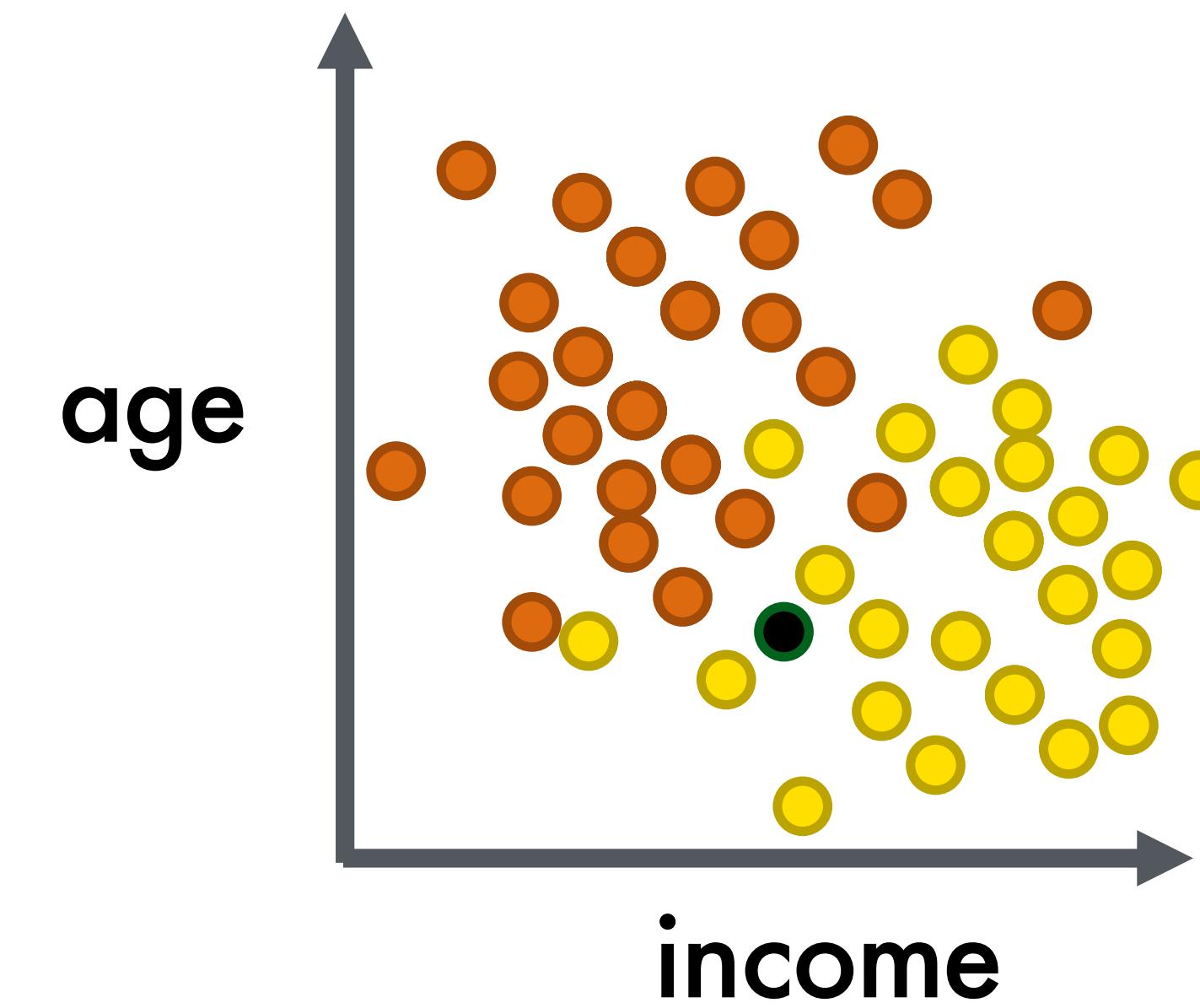
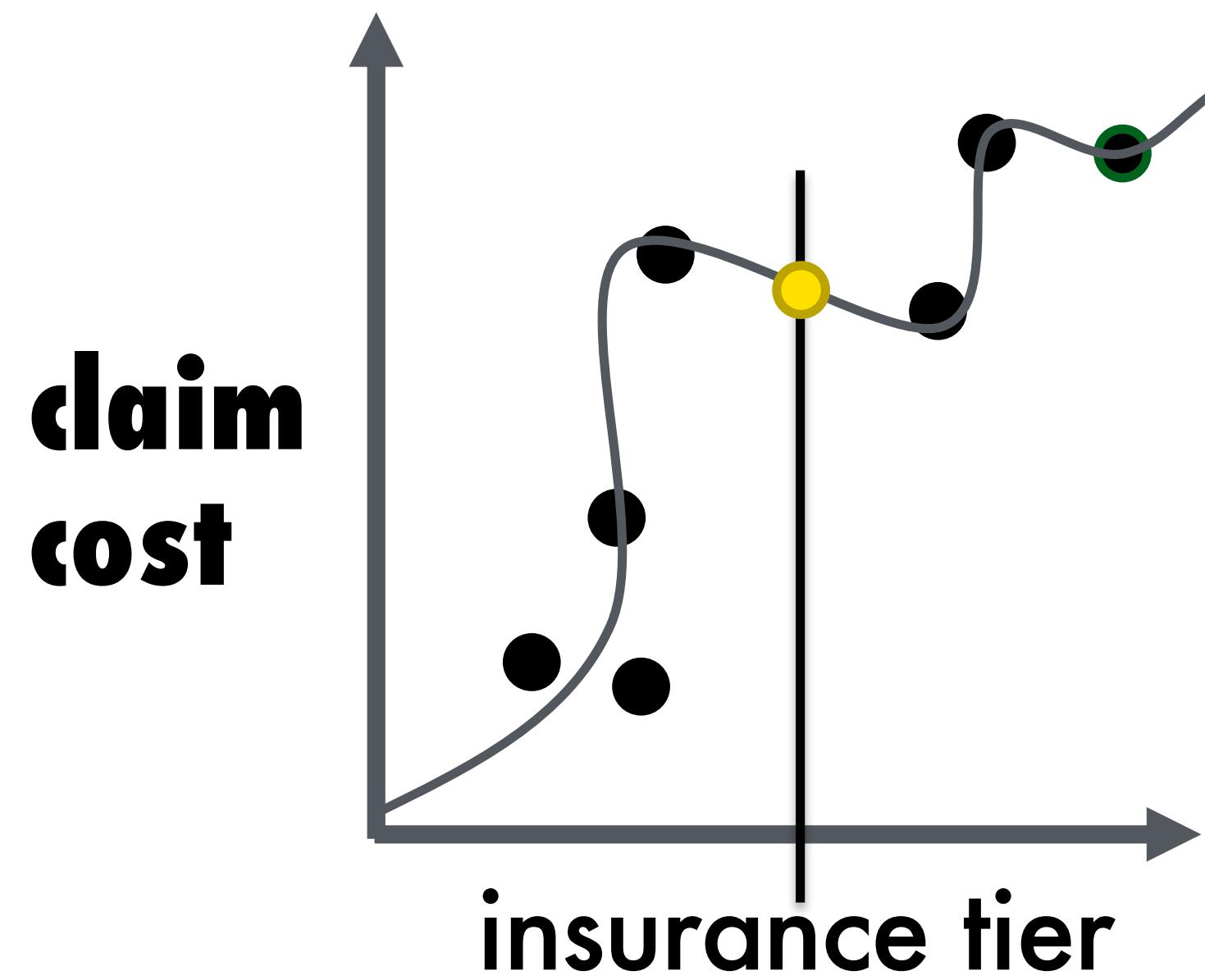


Supervised Learning

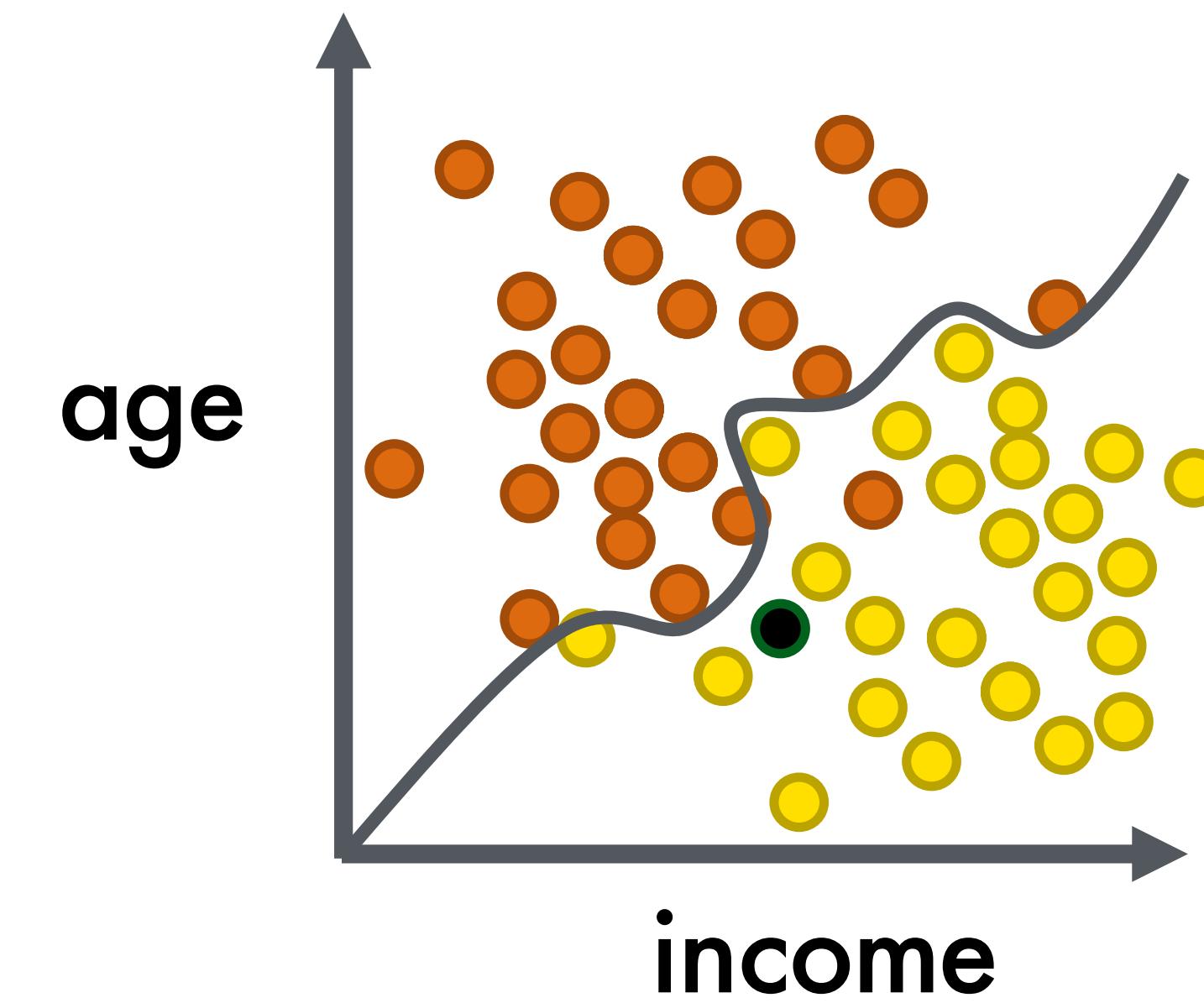
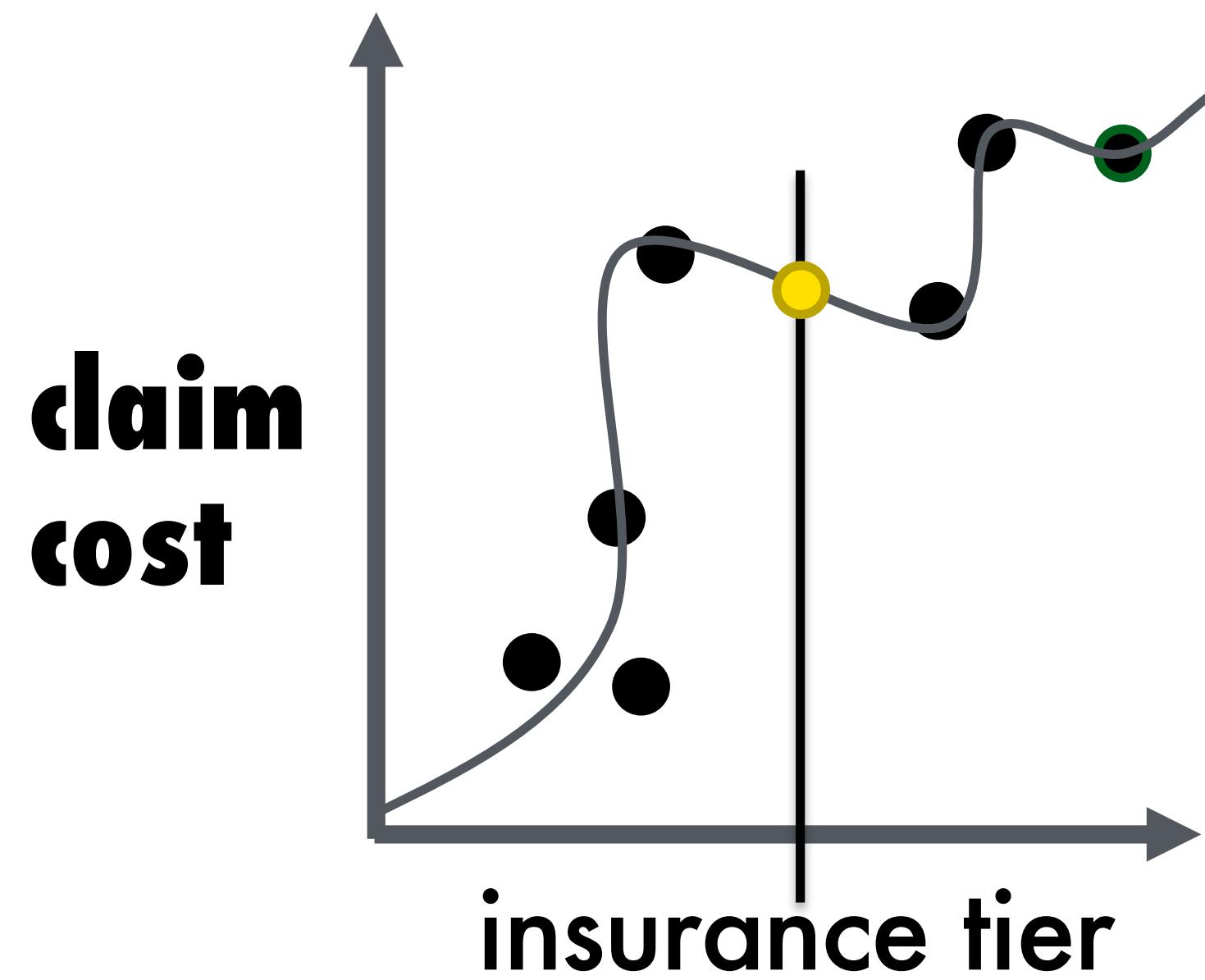


no
yes

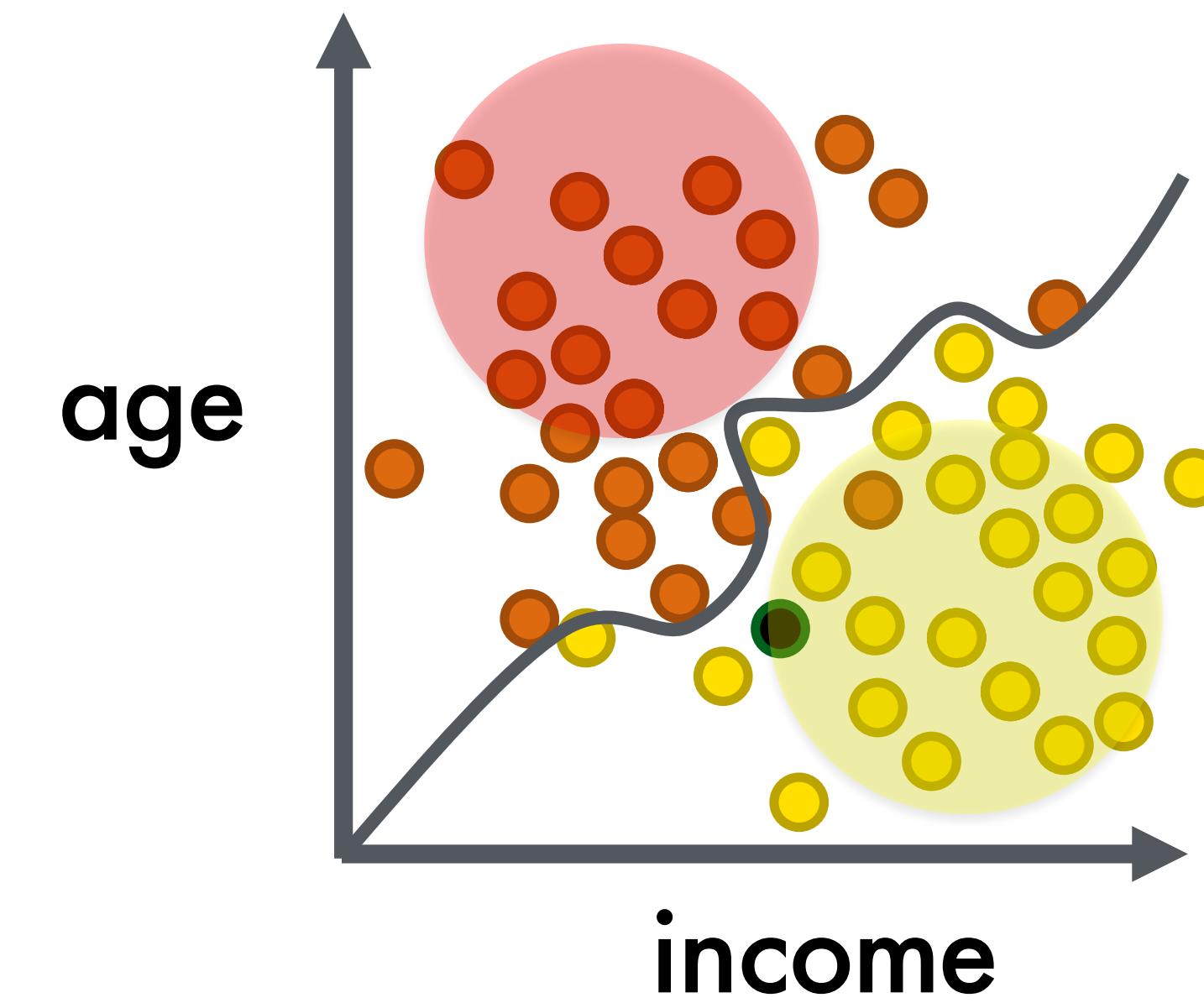
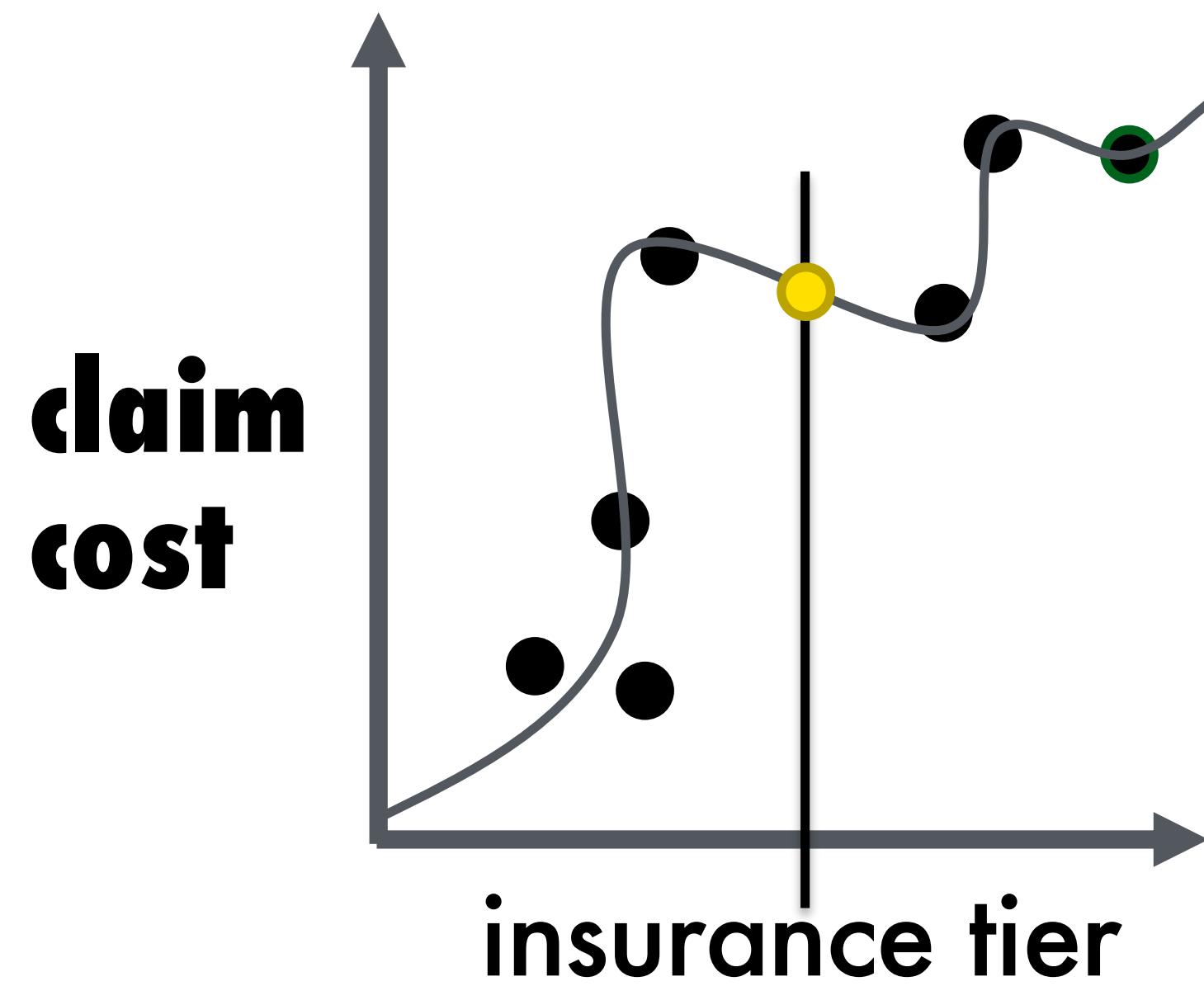
Supervised Learning



Supervised Learning



Supervised Learning



no
yes

Current Algorithm Overview

Supervised Algorithms

Statistical Analysis

- Linear Models (GLM)
- Naïve Bayes

Ensembles

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

Deep Neural Networks

- MLP
- Autoencoder
 - Anomaly Detection
 - Deep Features

Clustering

- K-Means (Auto-K)

Dimension Reduction

- Principal Component Analysis
- Generalized Low Rank Models

Word Embedding

- Word2Vec

Time Series

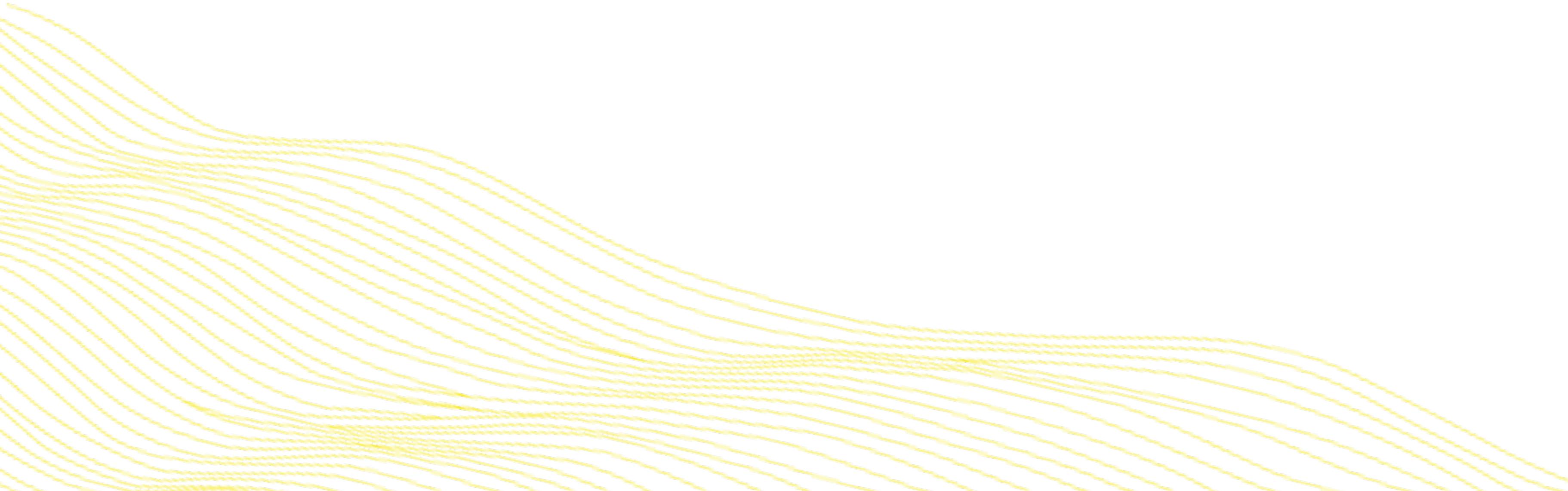
- iSAX

Machine Learning Tuning

- Hyperparameter Search
- Early Stopping

Intro to H2O's AutoML

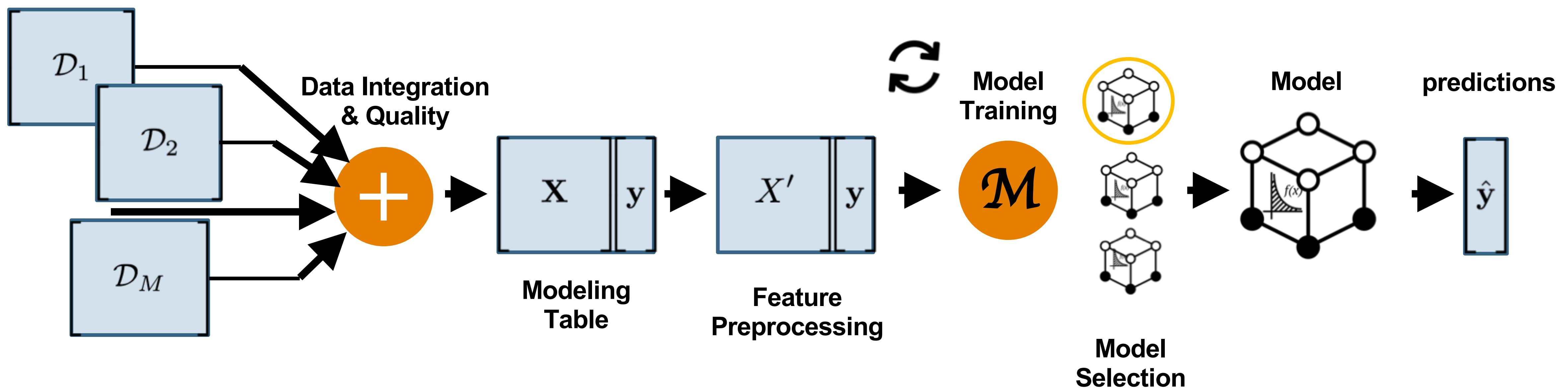
Scalable Automatic Machine Learning



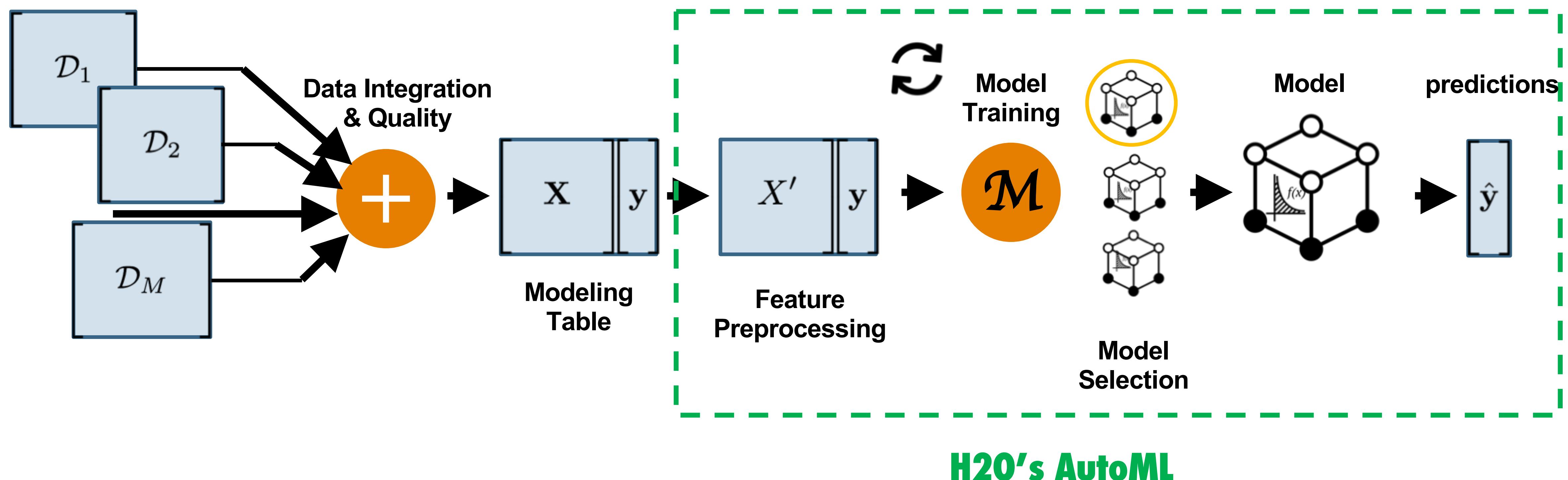
AutoML Agenda

- The Machine Learning Pipeline
- What is Automatic Machine Learning?
- How to Use H2O's AutoML

Updated Machine Learning Pipeline



Updated Machine Learning Pipeline

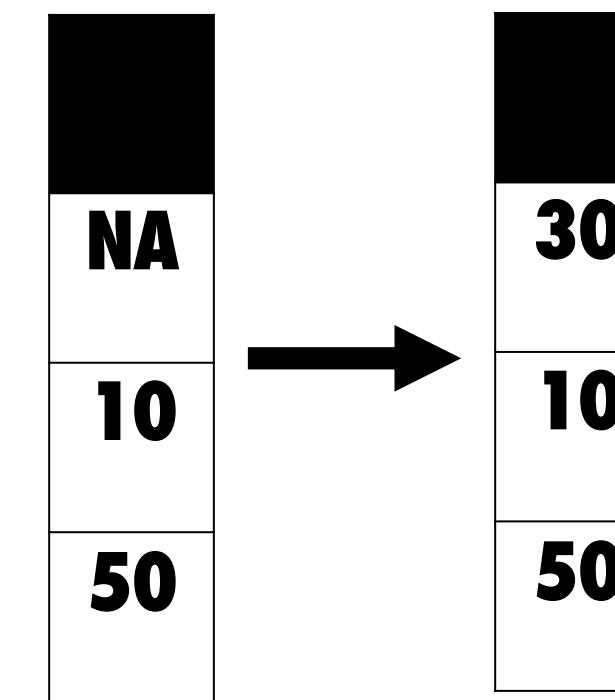


Automatic ML Overview

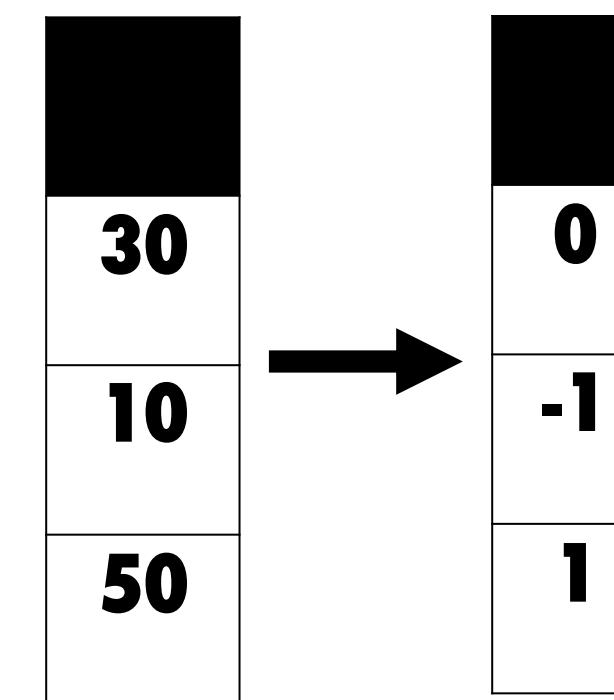
- **Feature Preprocessing**
- Hyperparameter Search
- Ensembles

Feature Preprocessing

Missing Values

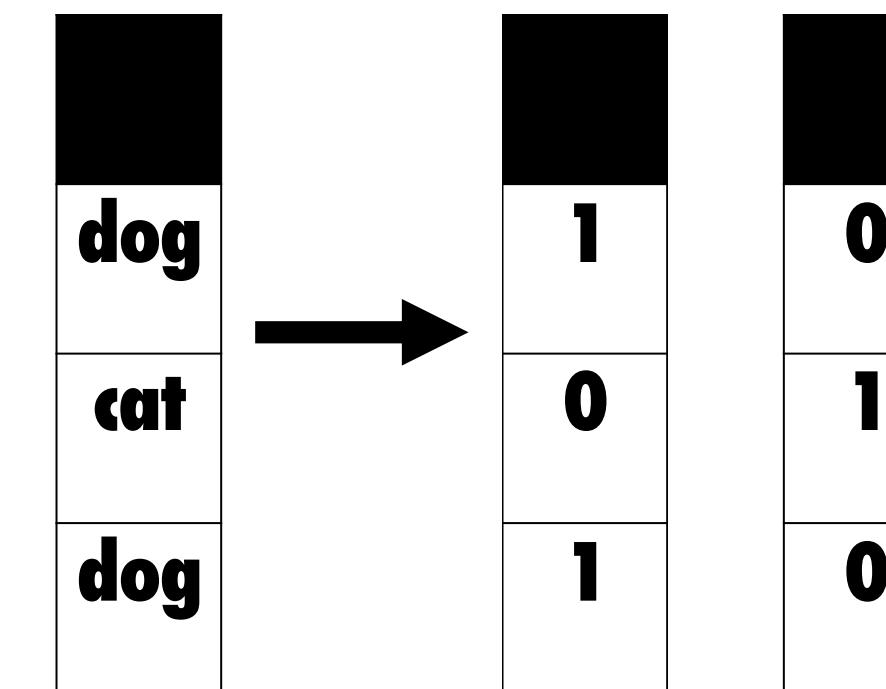


Standardization

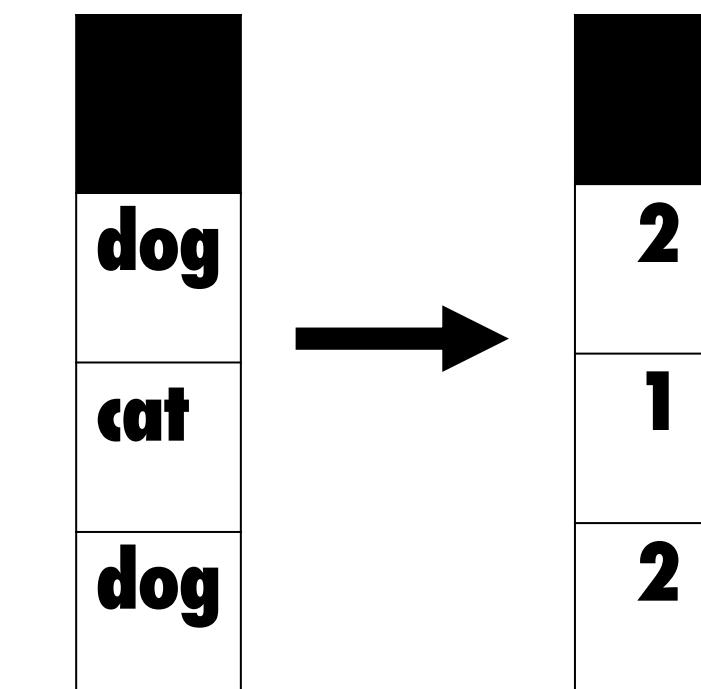


Feature Preprocessing

Handling Categorical Columns



One-hot Encoding



Label Encoder

Feature Preprocessing ≠ Feature Engineering



Domain Expertise

Timestamp
jan/02/2017
march/03/2019
dec/11/2018



Month	Day	Year
01	02	2017
03	03	2019
12	11	2018

Date Expansion

Automatic ML Overview

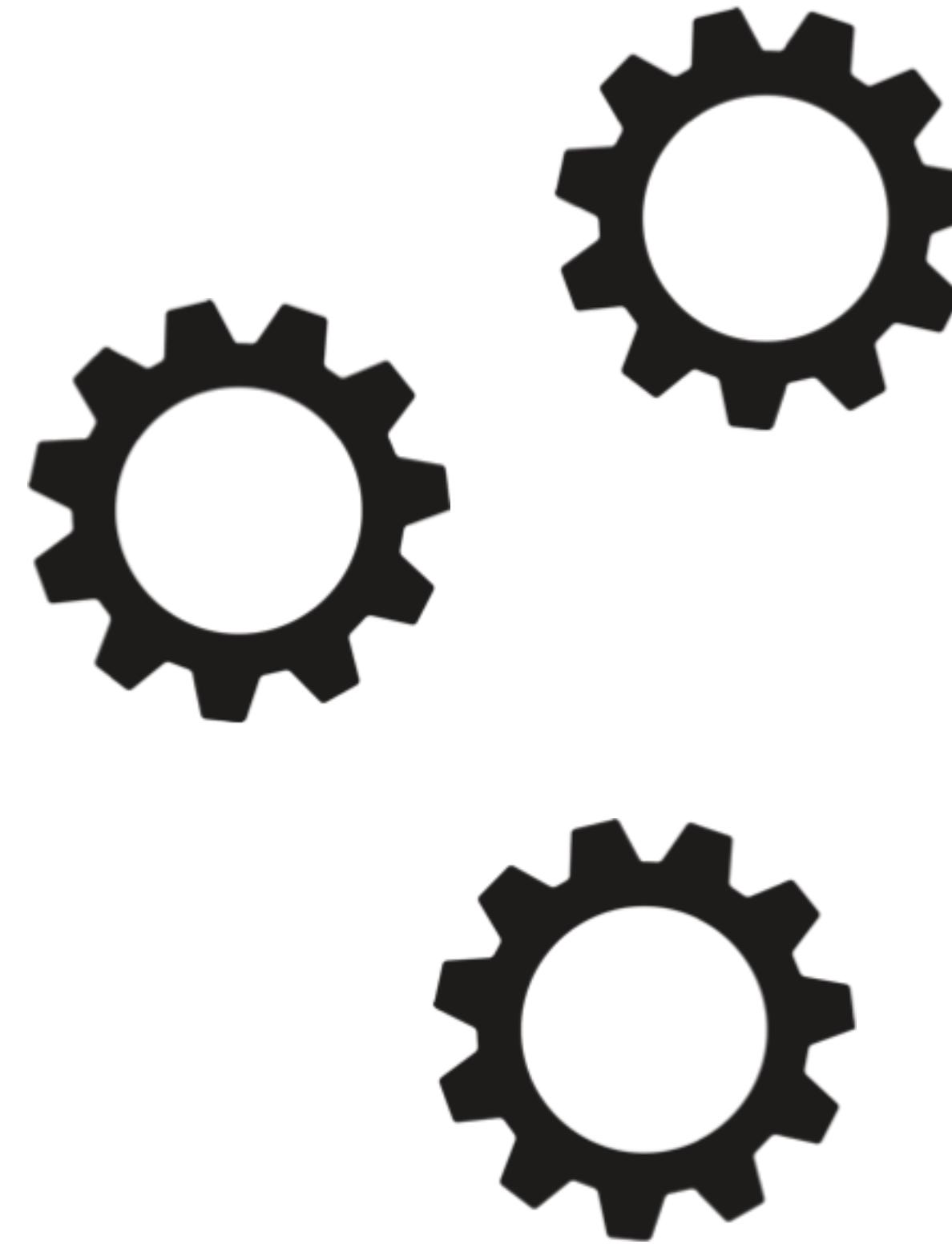
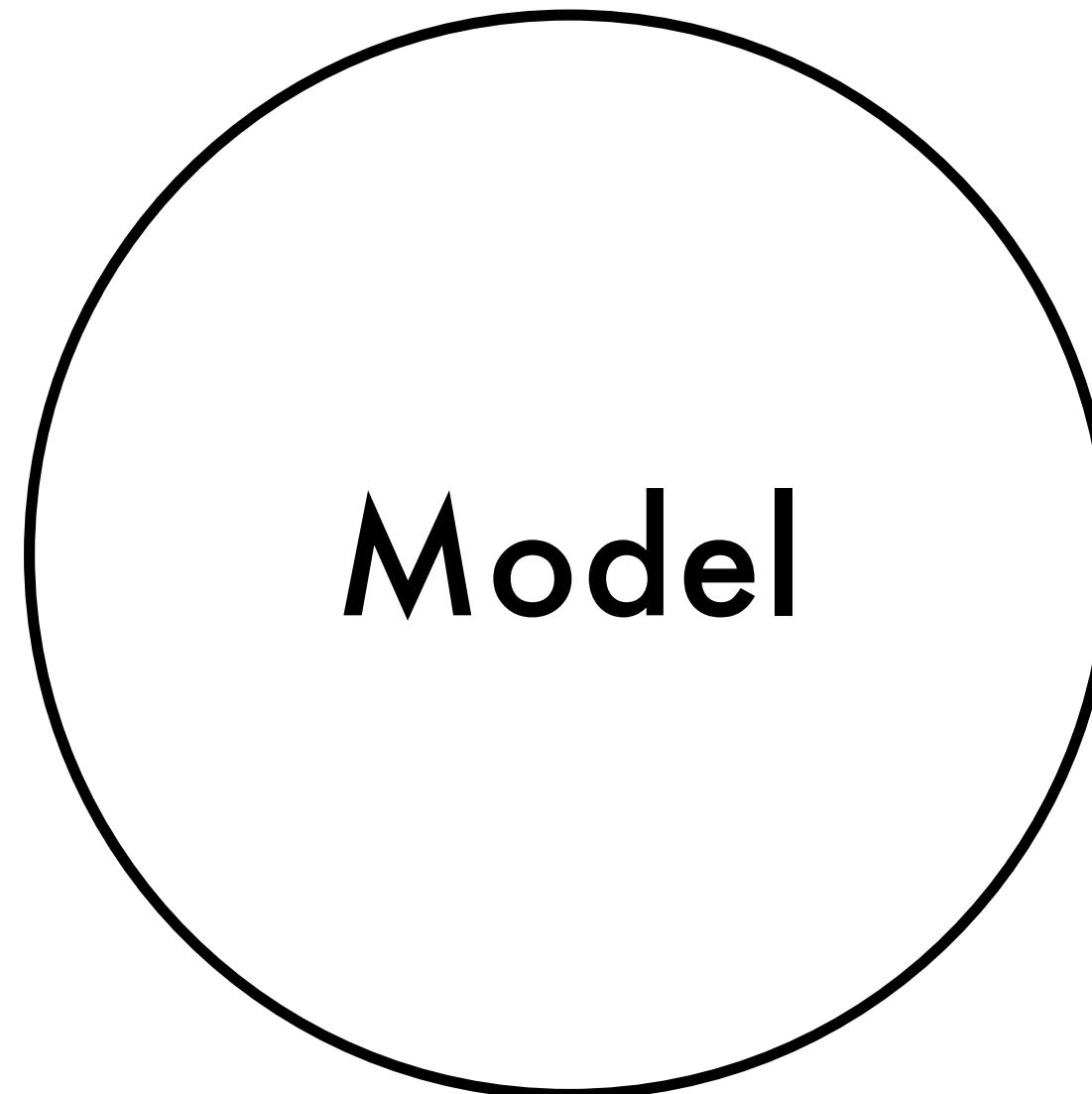
• Feature Preprocessing

- **Hyperparameter Search**
- **Ensembles**

What are Hyperparameters?

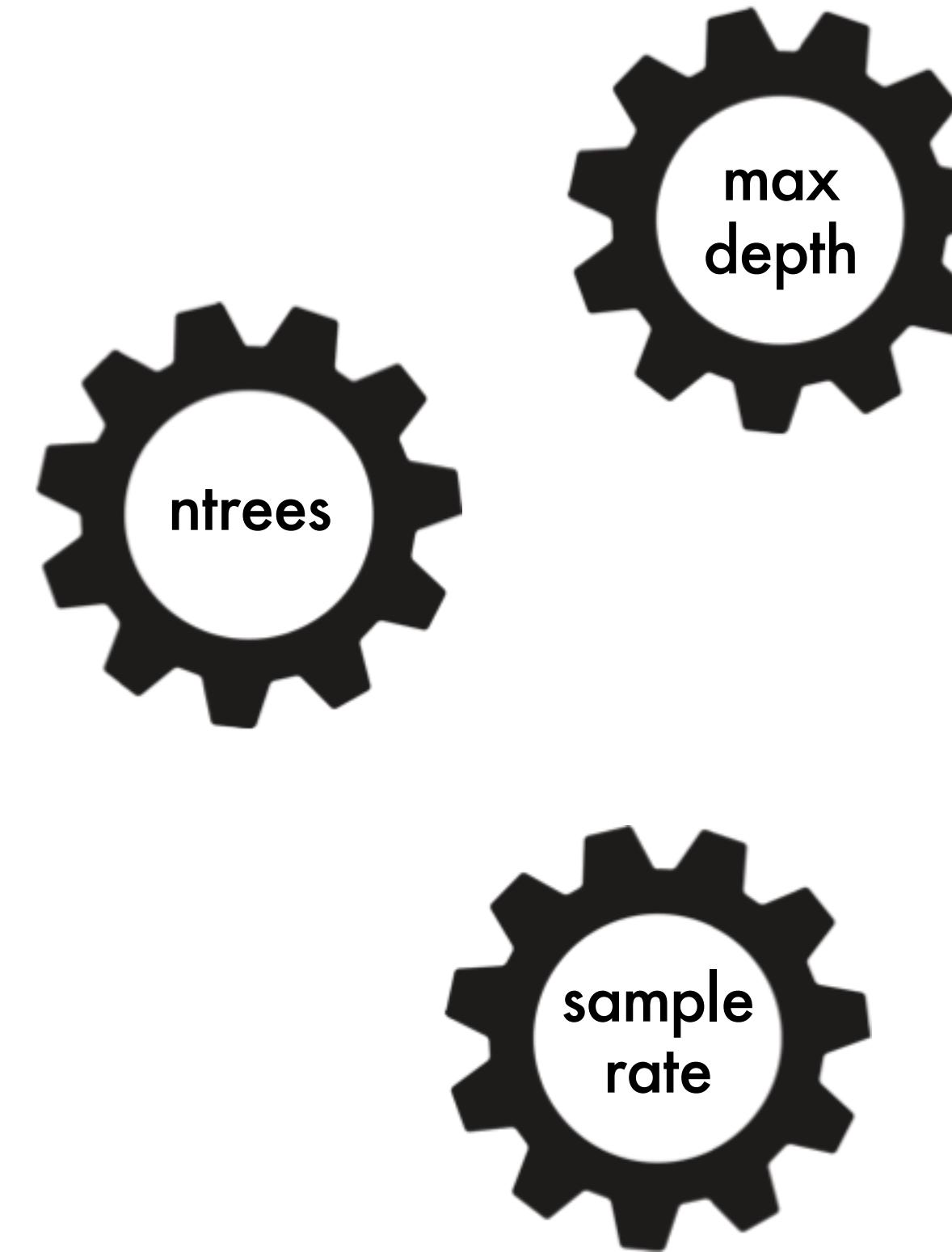
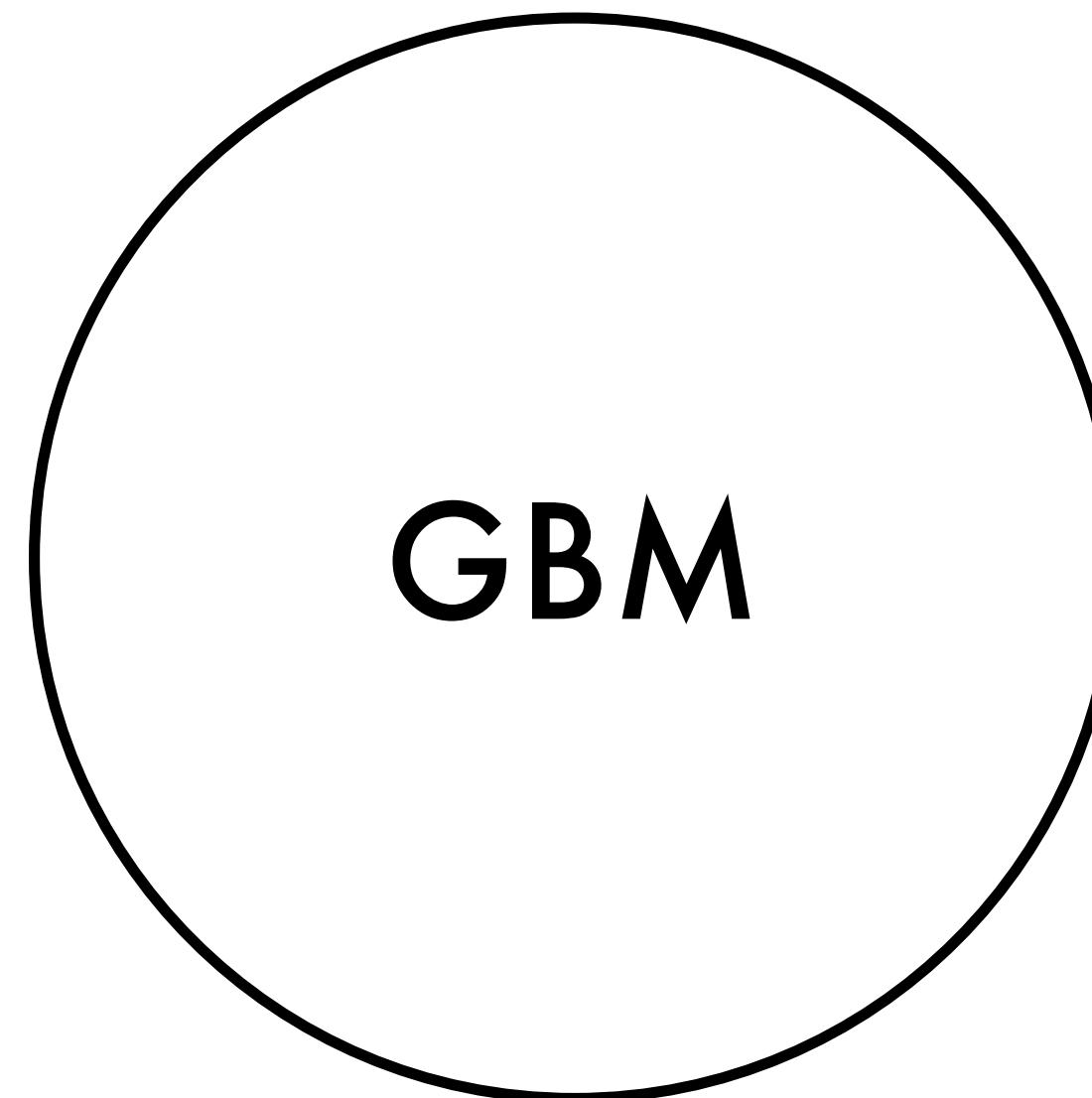
(a quick review)

What are Model Hyperparameters?



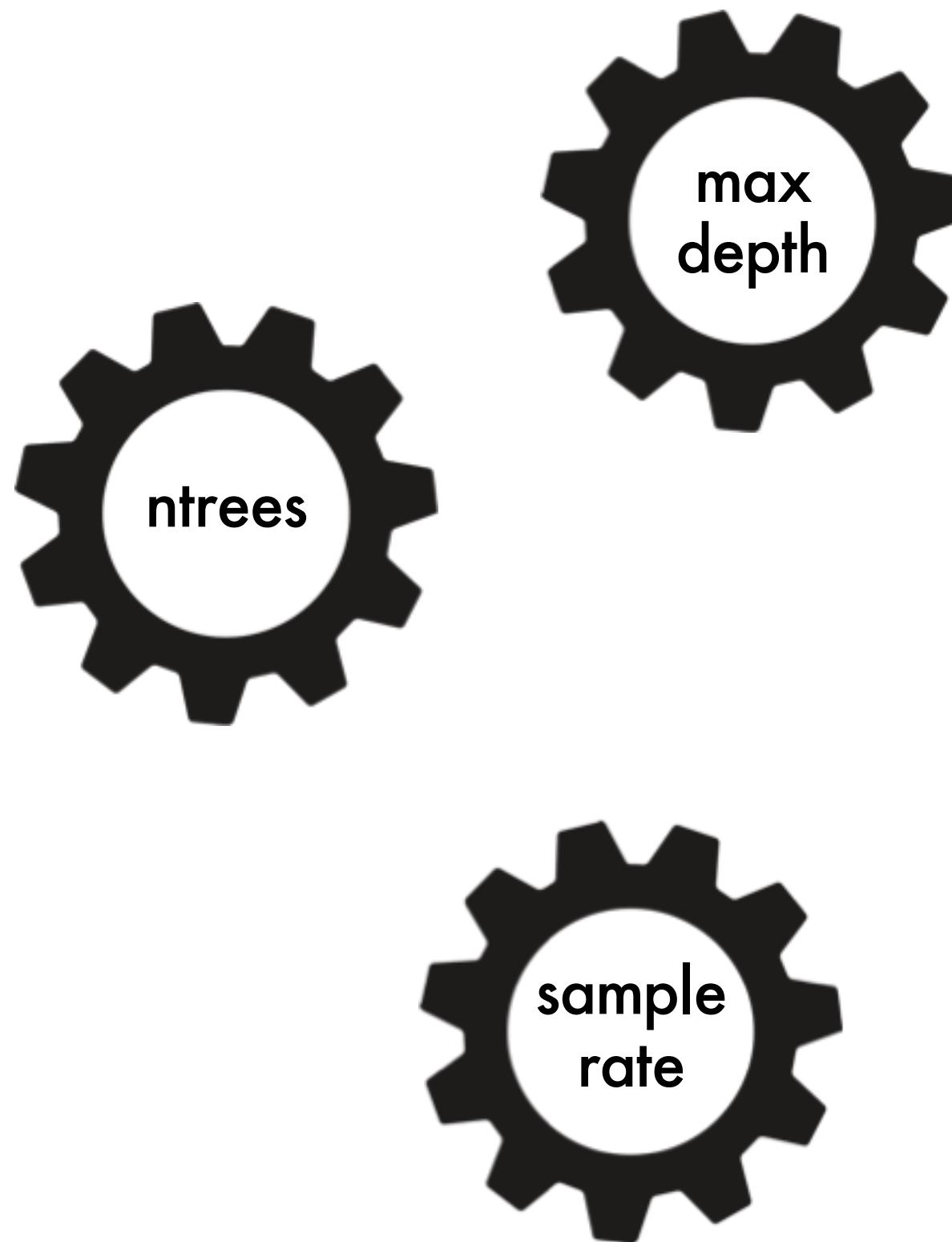
Every model comes with knobs

What are Model Hyperparameters?



Every model comes with knobs

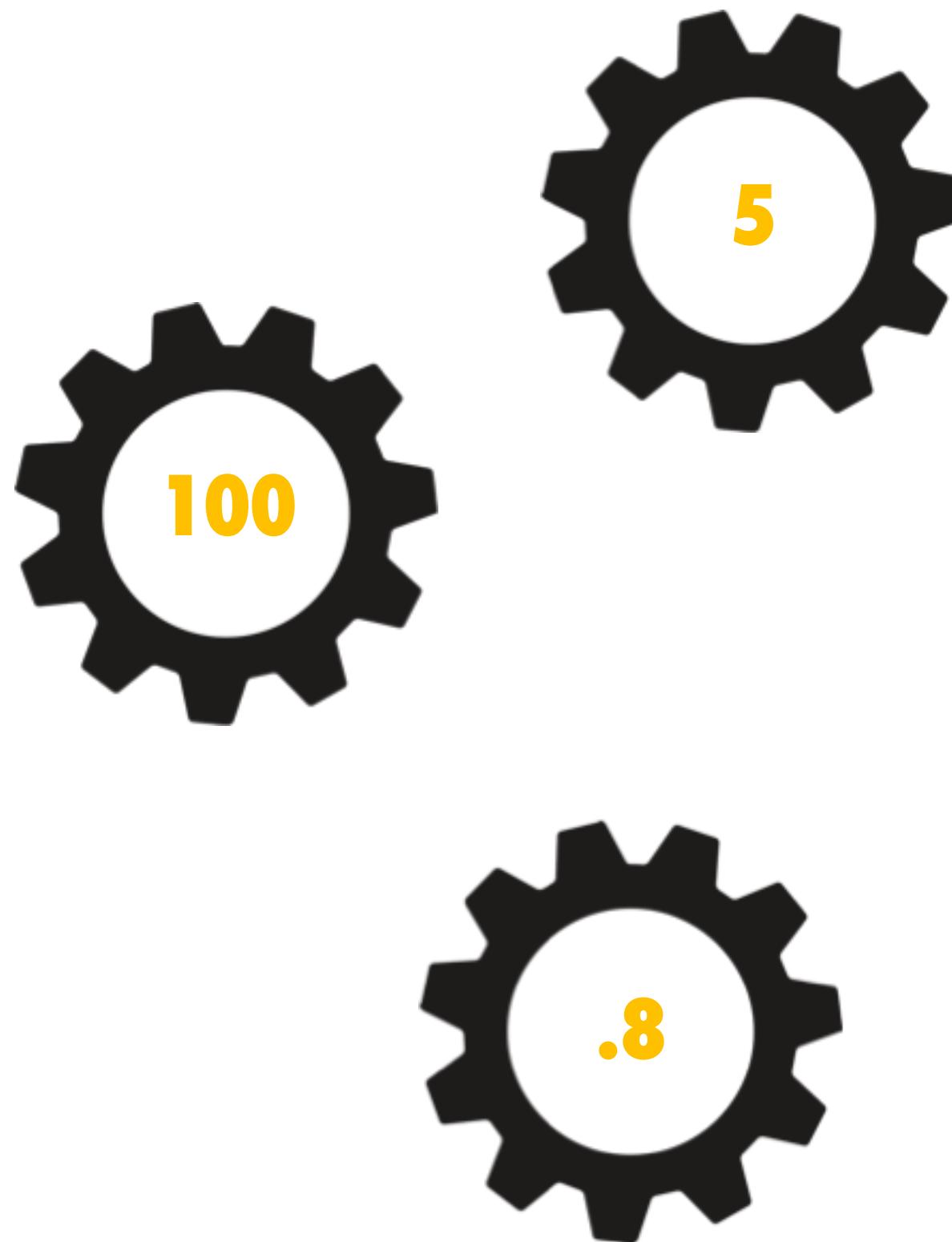
What are Hyperparameter?



Unlike regular model parameters, hyperparameters are not learned from the data

The user has to set the hyperparameters **values**

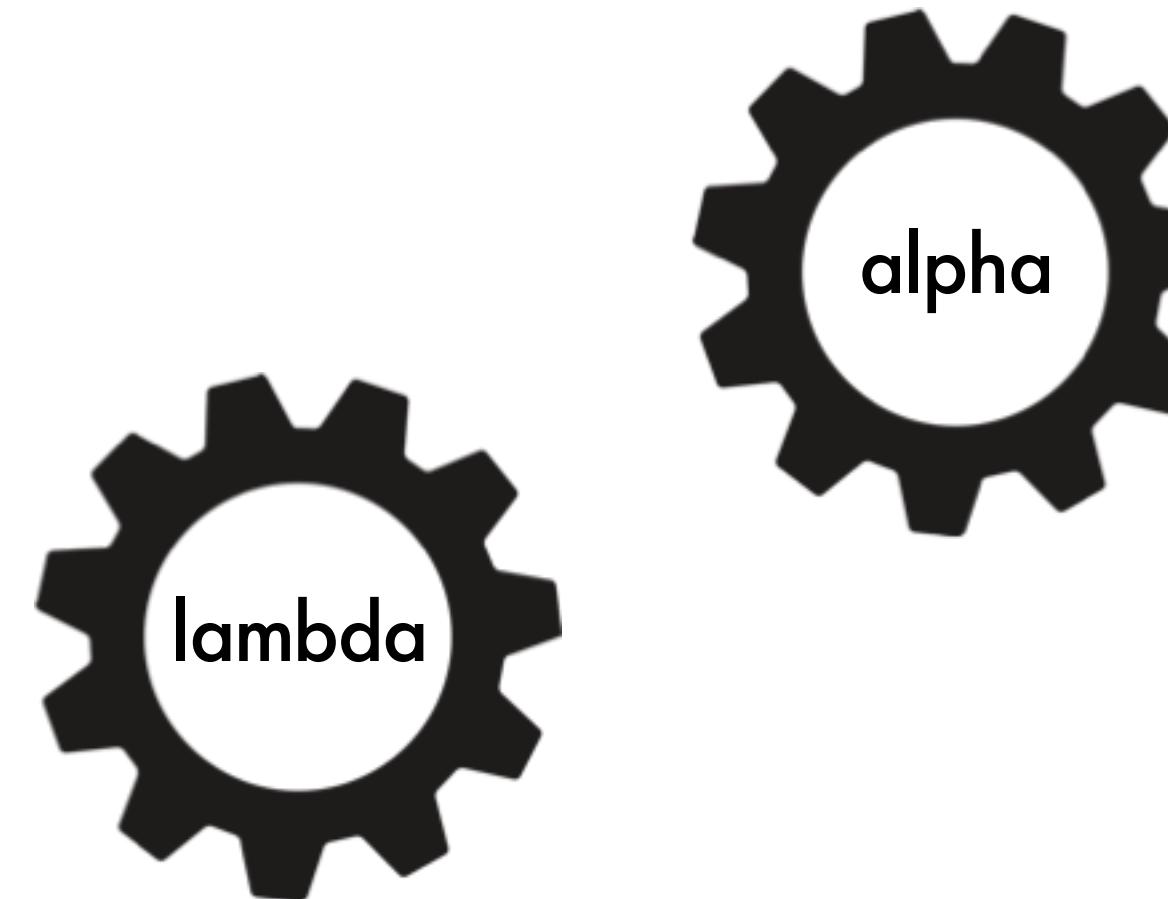
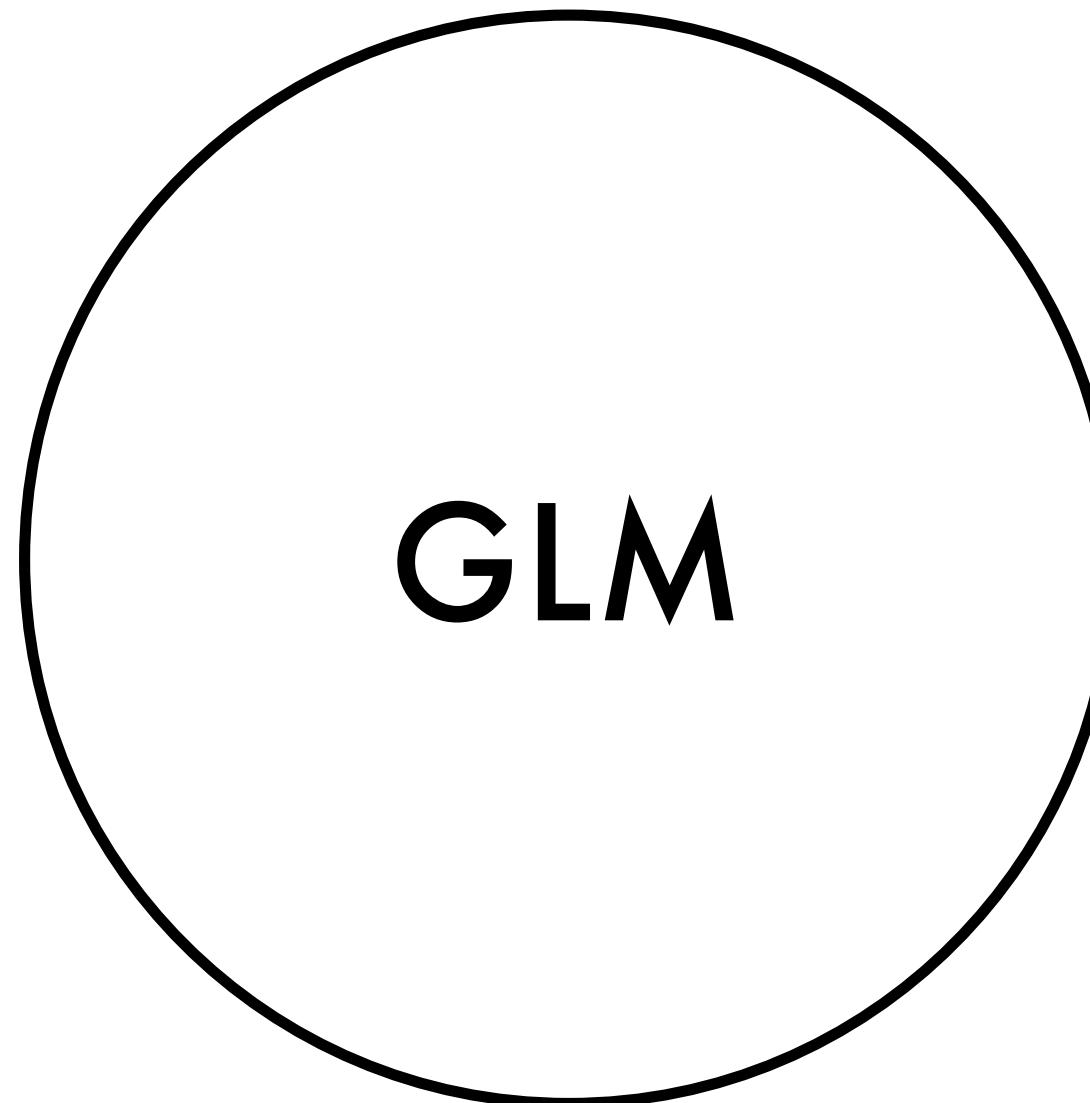
What are Hyperparameter?



Unlike regular model parameters, hyperparameters are not learned from the data

The user has to set the hyperparameters **values**

What are GLM's Hyperparameters?



GLM Model Parameters

$$\max_{\beta, \beta_0} \sum_{i=1}^N \log f(y_i; \beta, \beta_0) - \lambda \left(\alpha \|\beta\|_1 + \frac{1}{2}(1 - \alpha) \|\beta\|_2^2 \right)$$

GLM Model Parameters

$$\max_{\beta, \beta_0} \sum_{i=1}^N \log f(y_i; \beta, \beta_0) - \lambda \left(\alpha \|\beta\|_1 + \frac{1}{2}(1-\alpha) \|\beta\|_2^2 \right)$$

GLM Hyperparameter

$$\max_{\beta, \beta_0} \sum_{i=1}^N \log f(y_i; \beta, \beta_0) - \lambda \left(\alpha \|\beta\|_1 + \frac{1}{2} (1 - \alpha) \|\beta\|_2^2 \right)$$

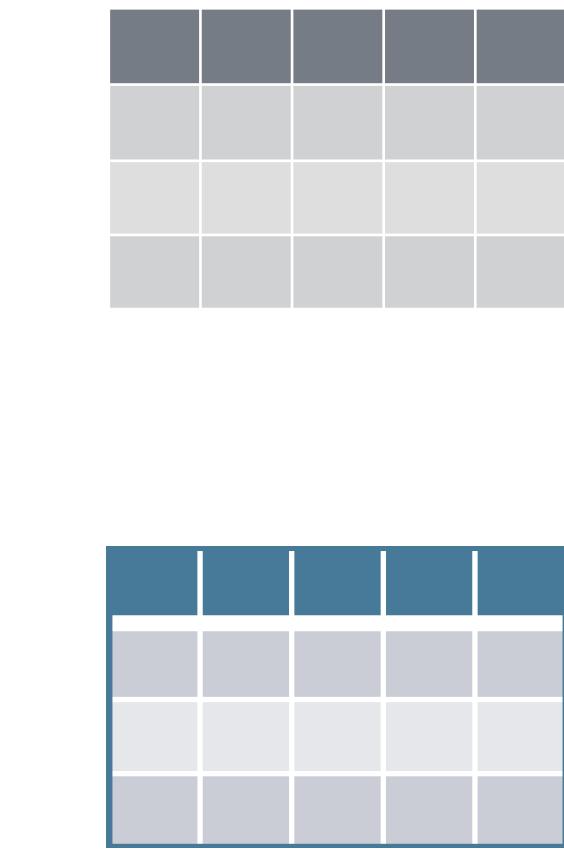
Model Building: Hyperparameter Search

3 Main Ways:

- Manual Model Tuning
- Cartesian Grid Search
- Random Grid Search*

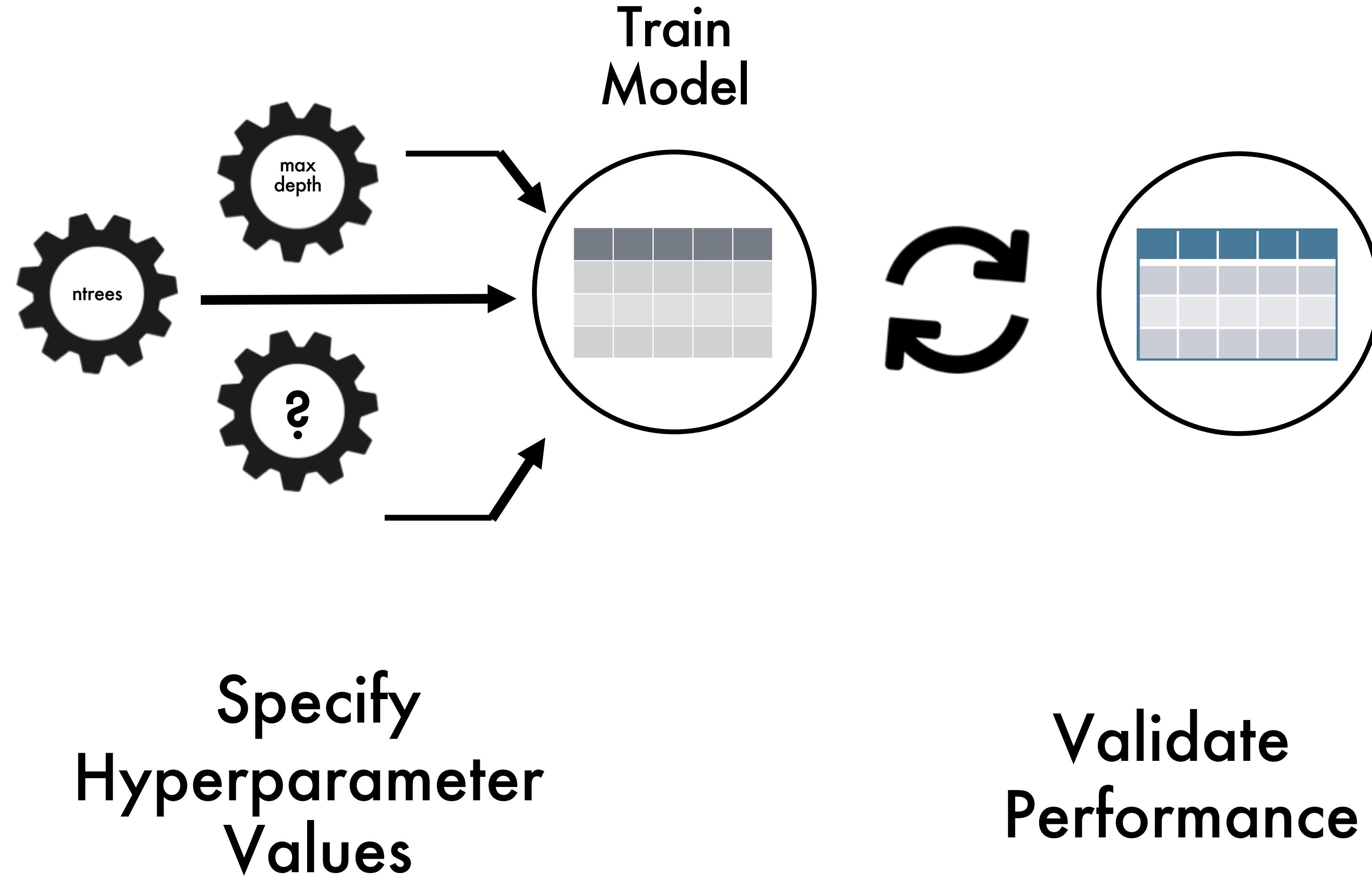
*What AutoML Uses

Manual Model Tuning



training
data

validation
data



Manual Model Tuning

How did you do?

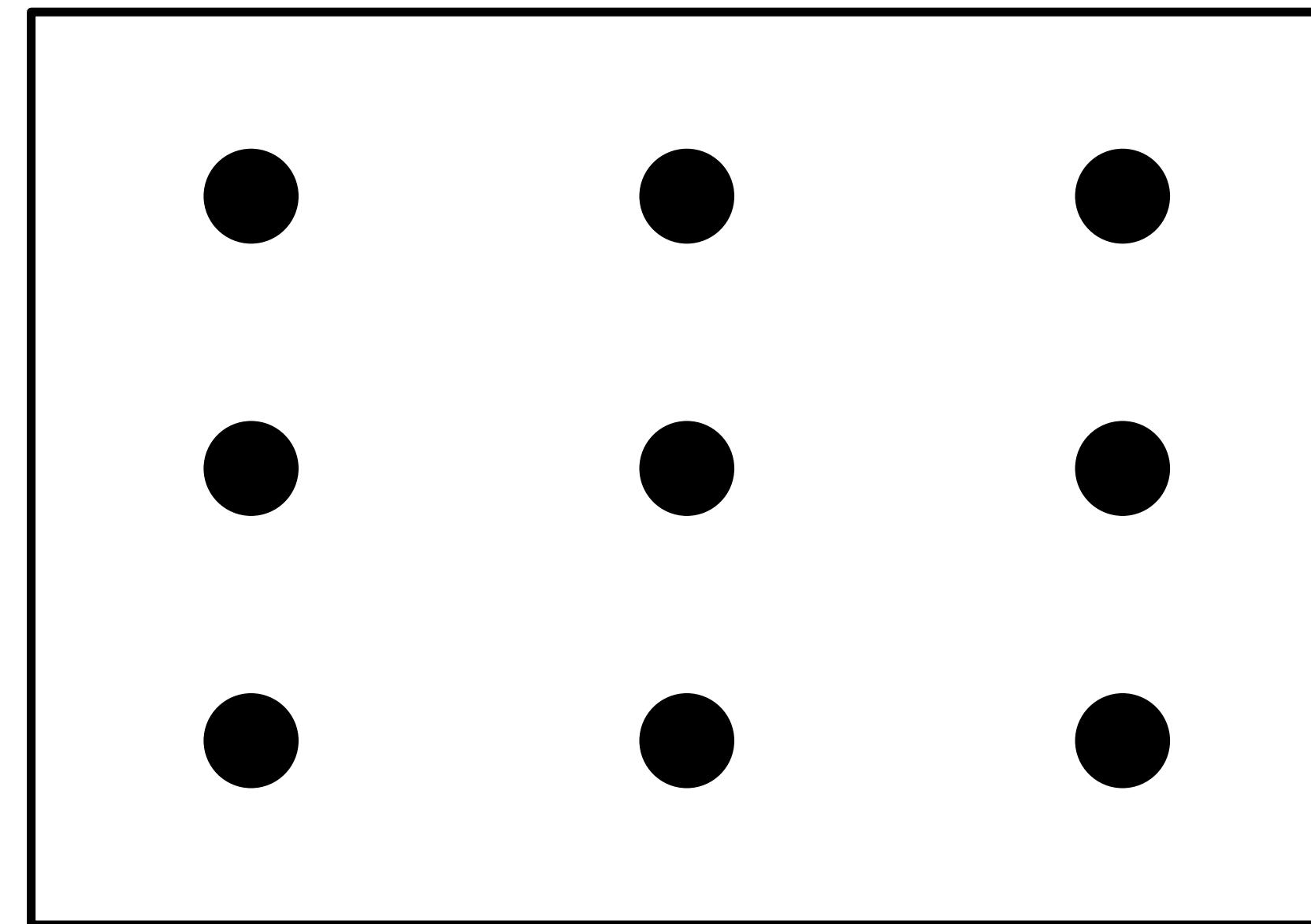
- Not so good: **Repeat** Process
- GOOD! : Stop

Cartesian Grid Search

AKA Exhaustive Search: Tries **Every Combination**

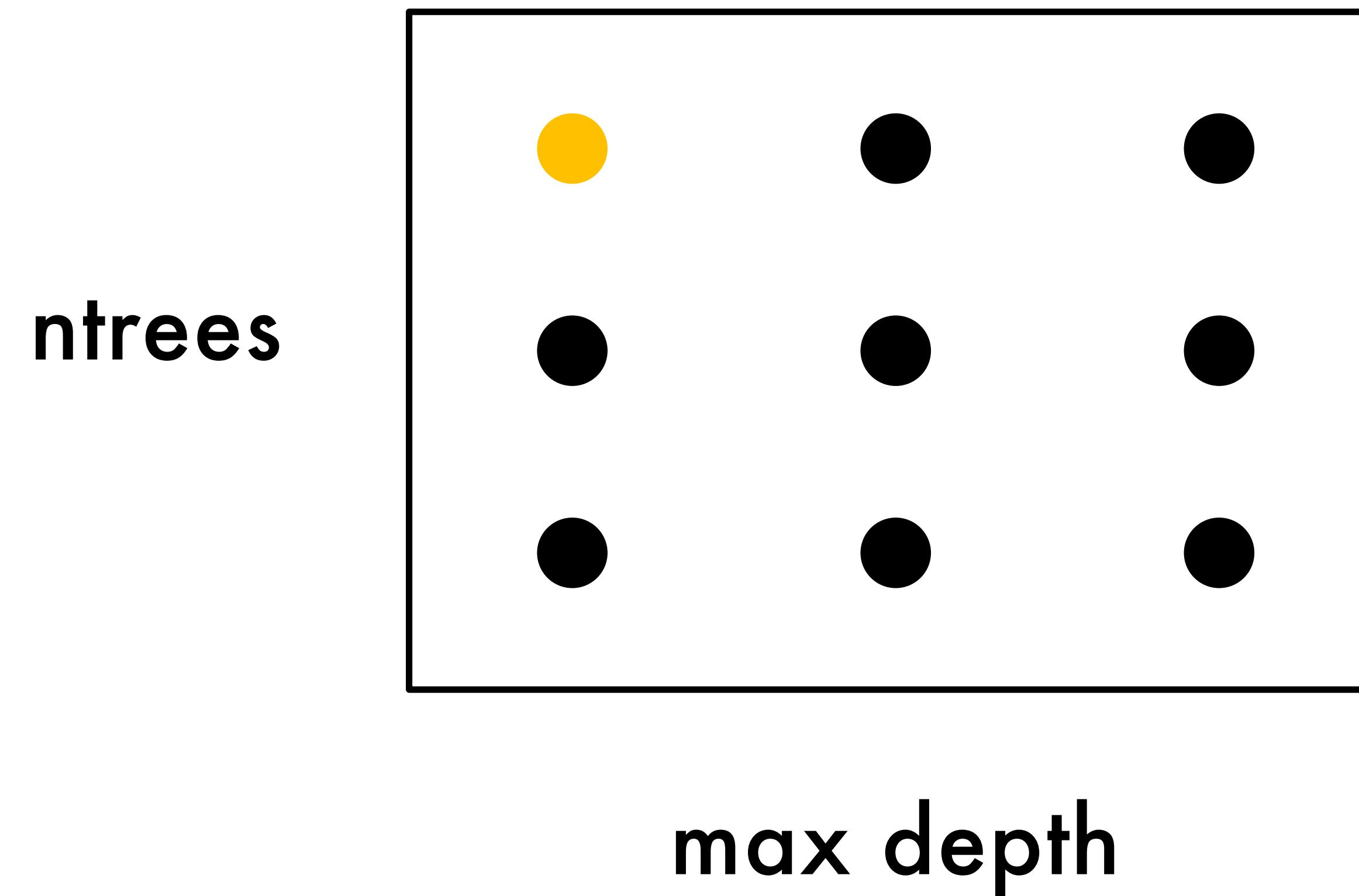
Cartesian Grid Search

ntrees



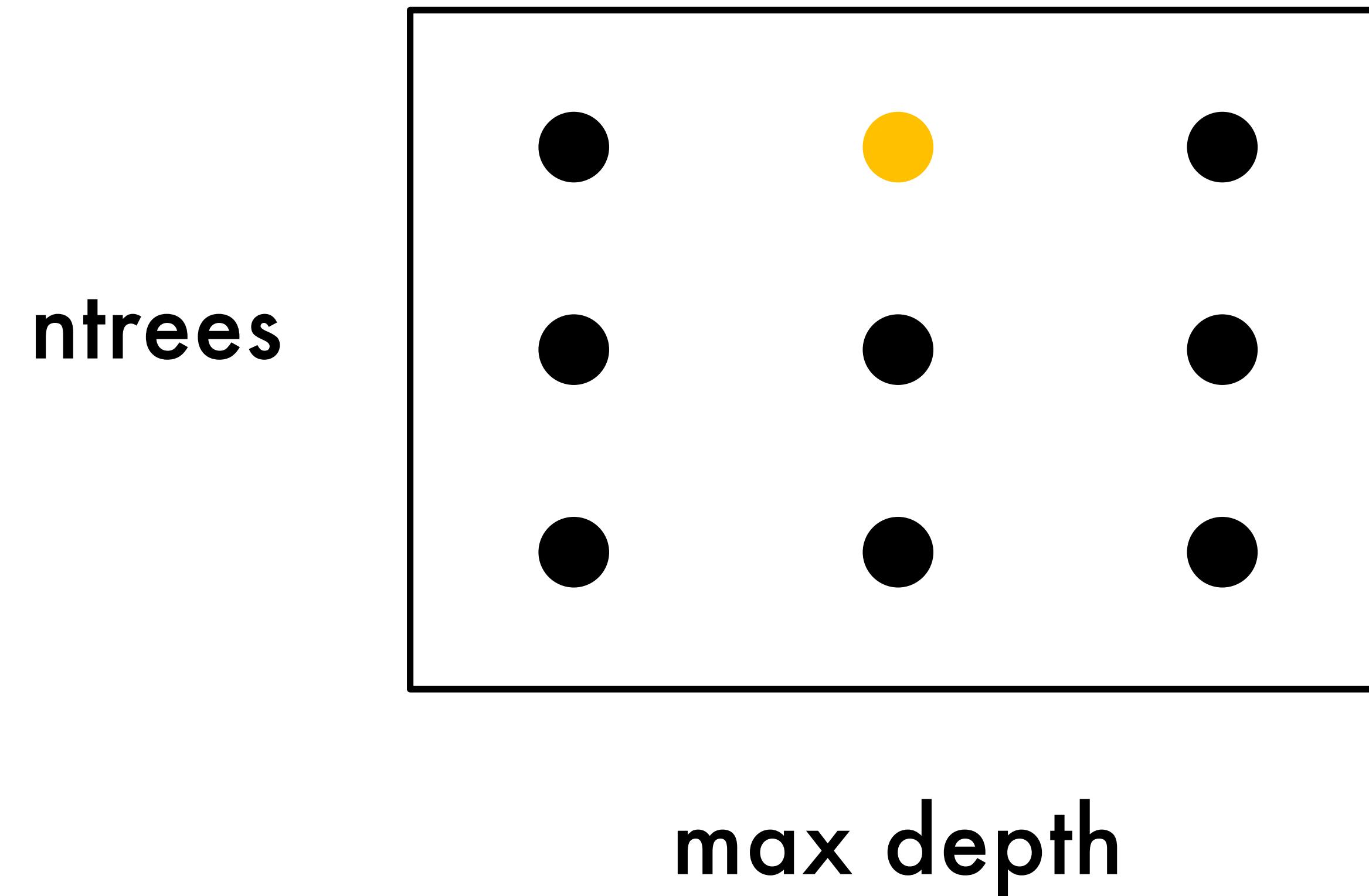
max depth

Cartesian Grid Search



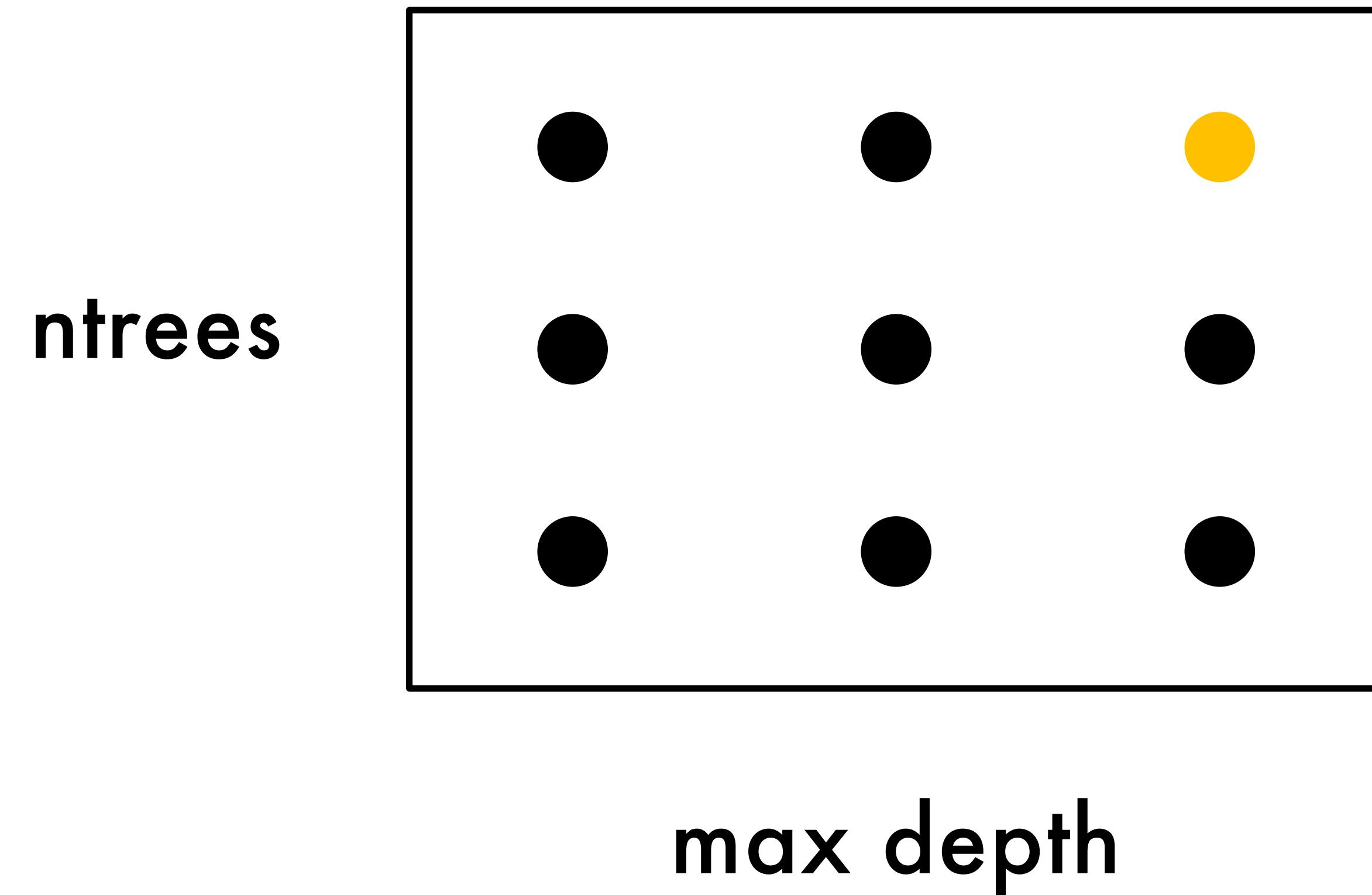
single trial
ntrees = 1000
max depth = 5

Cartesian Search



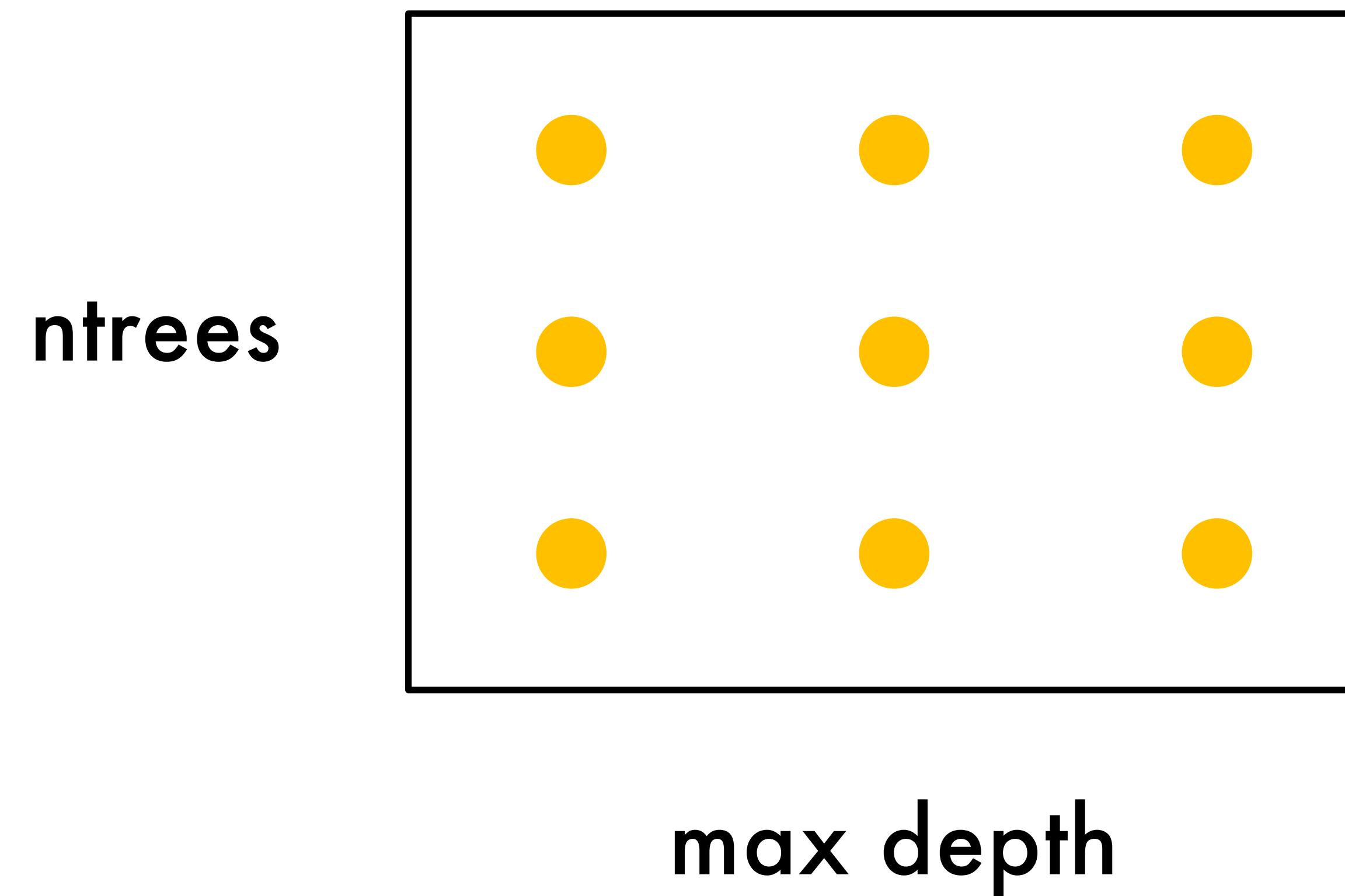
single trial
ntrees = 1000
max depth = 10

Cartesian Search



single trial
ntrees = 1000
max depth = 15

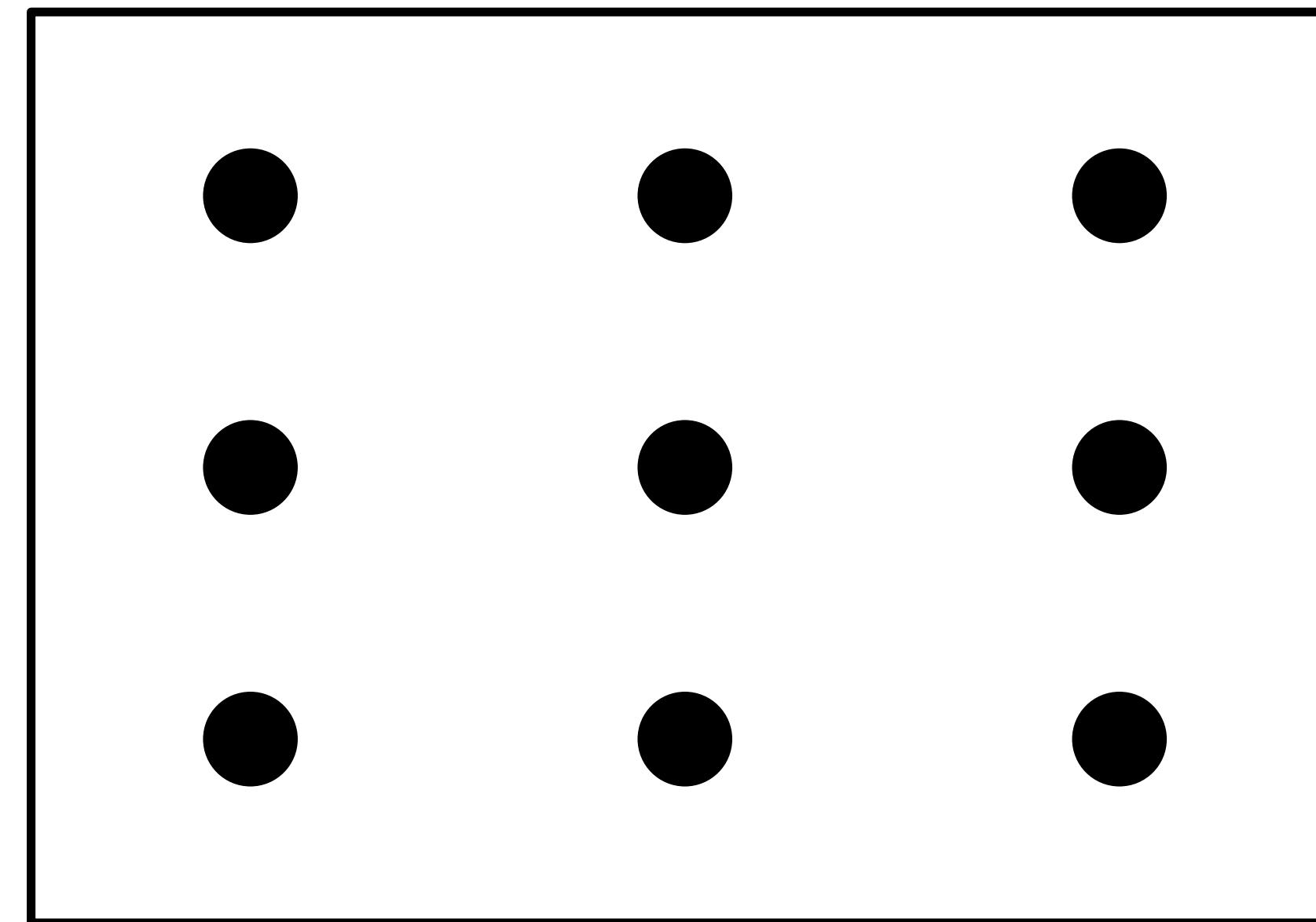
Cartesian Grid Search



**Cartesian Grid
Search**
Complete Exploration

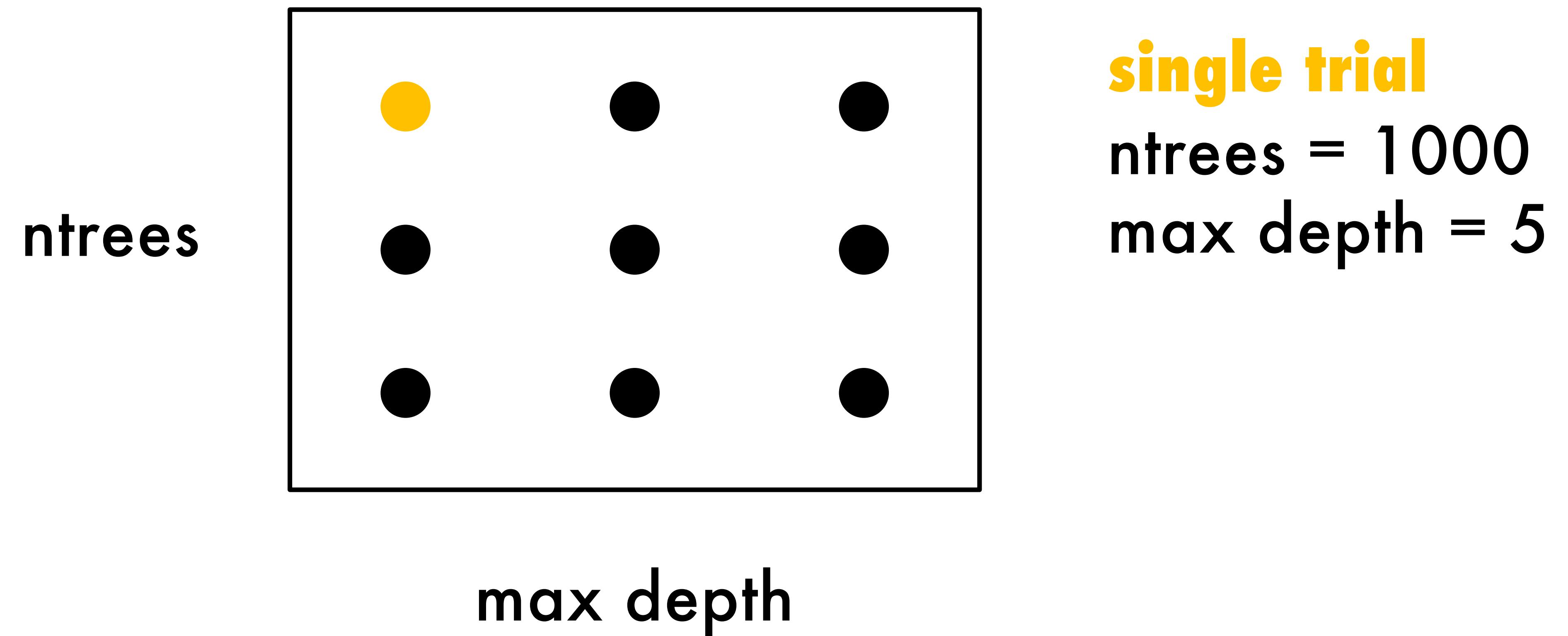
Random Grid Search

ntrees

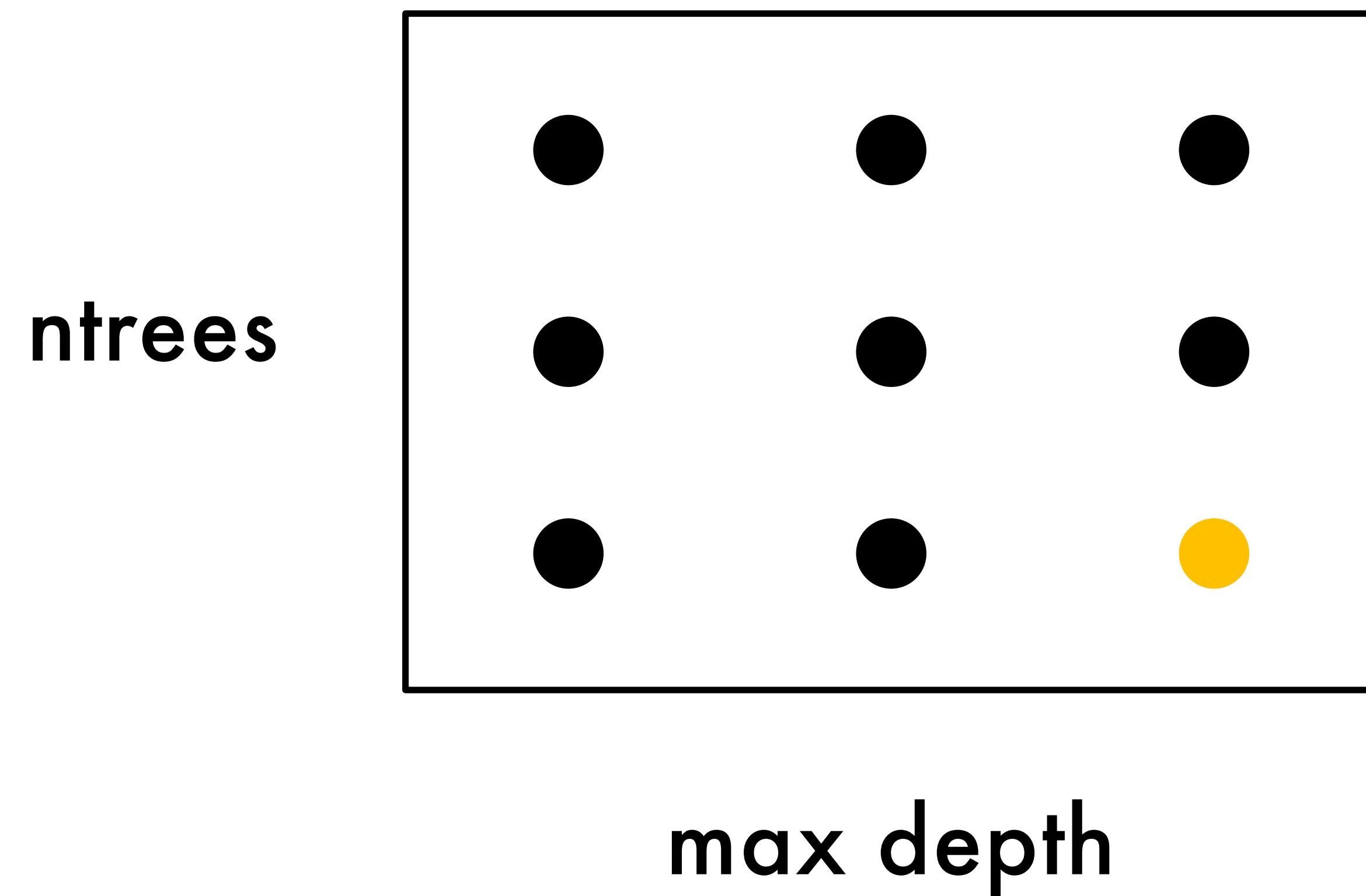


max depth

Random Grid Search

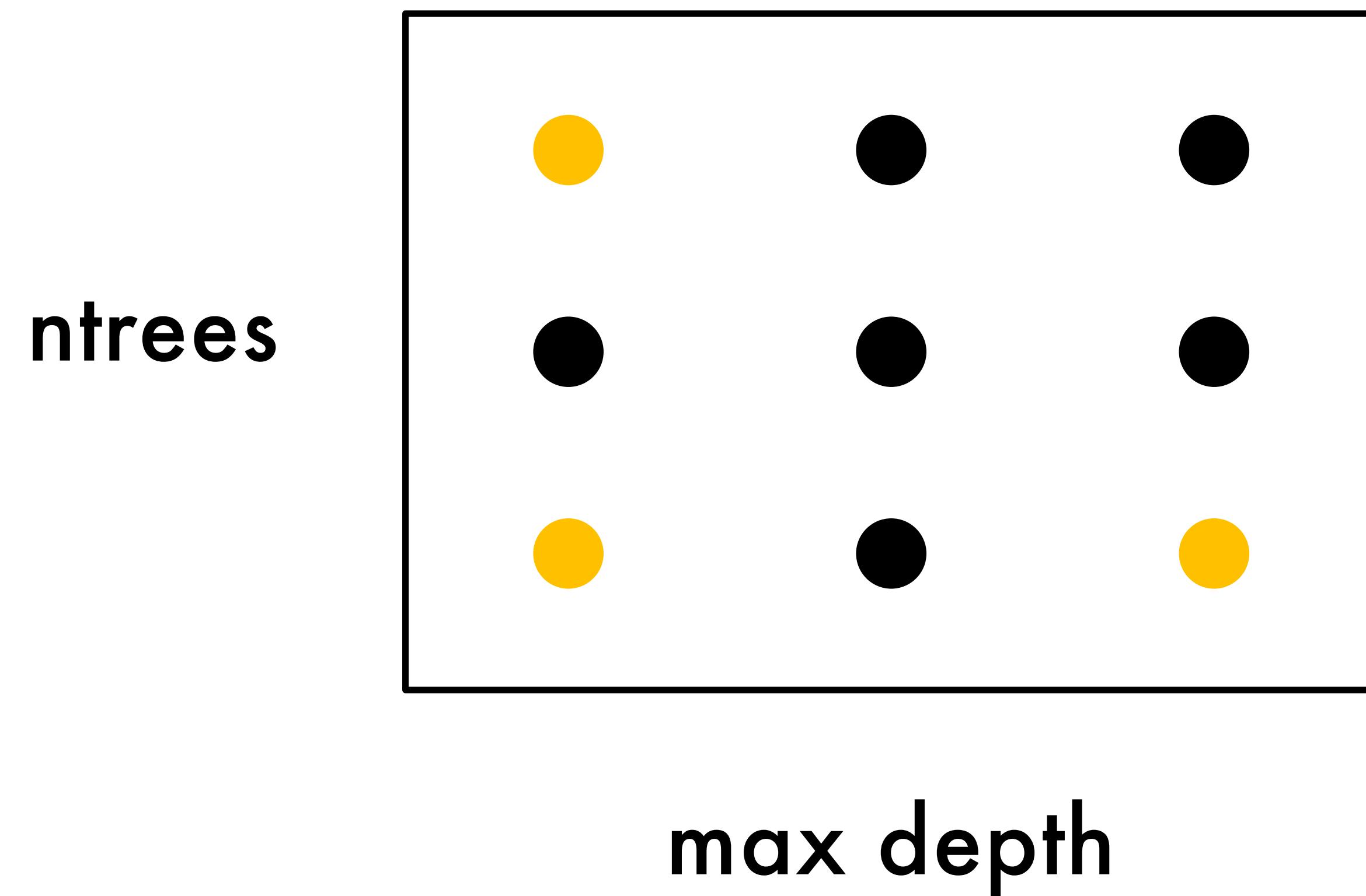


Random Grid Search



single trial
ntrees = 3000
max depth = 15

Random Grid Search



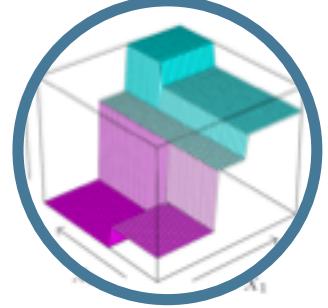
**Random Grid
Search**
explores more in
less time

Automatic ML Overview

- ~~Feature Preprocessing~~
- ~~Feature Engineering~~
- ~~Hyperparameter Search~~
- Ensembles

Ensembles

Vanilla Ensembles

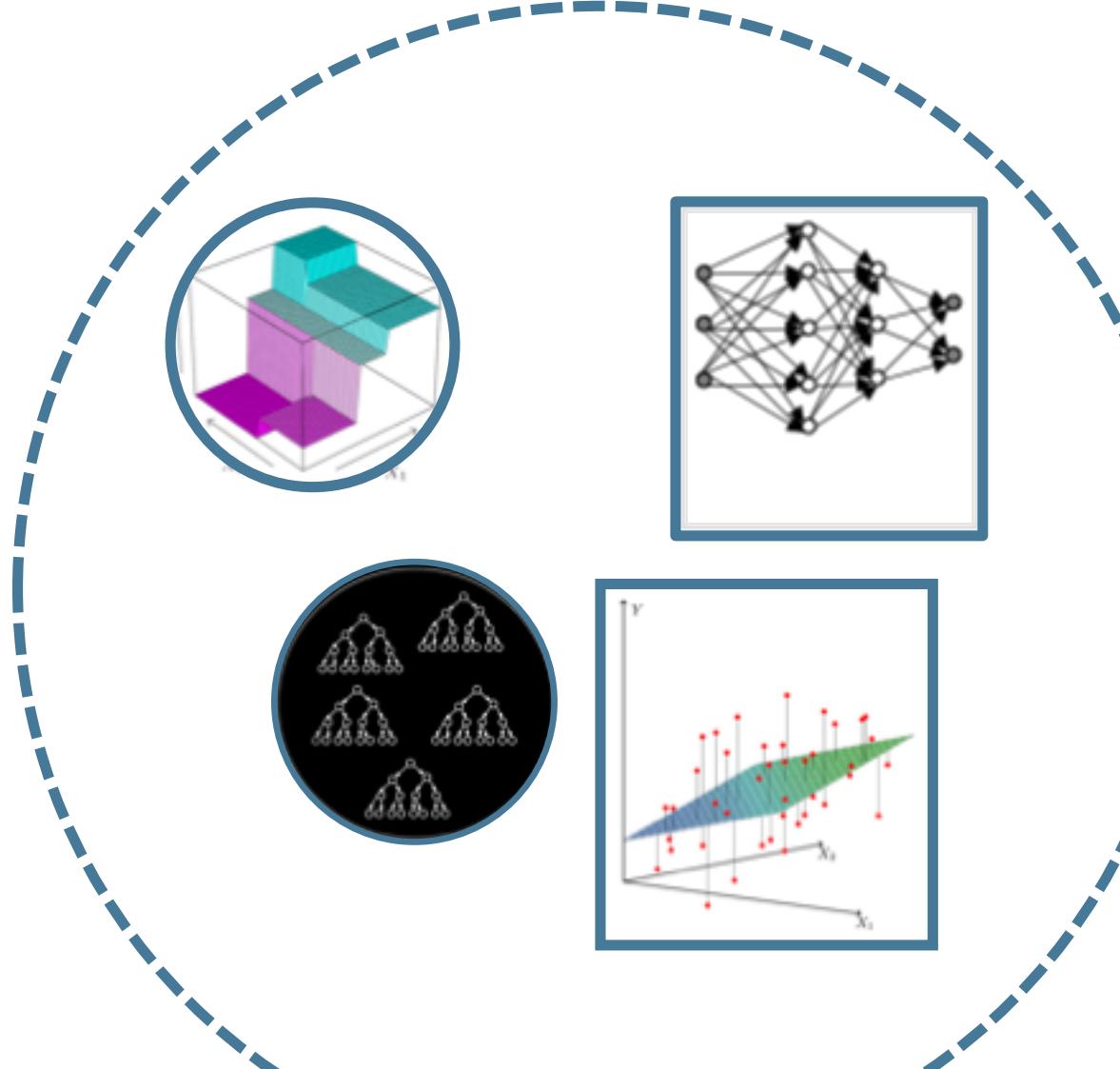


Gradient Boosting Machine



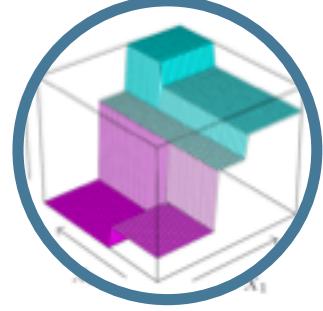
Random Forest

Super Learning Ensembles

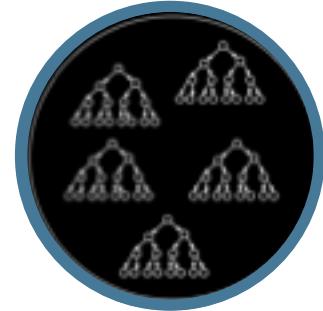


Ensembles

Vanilla Ensembles

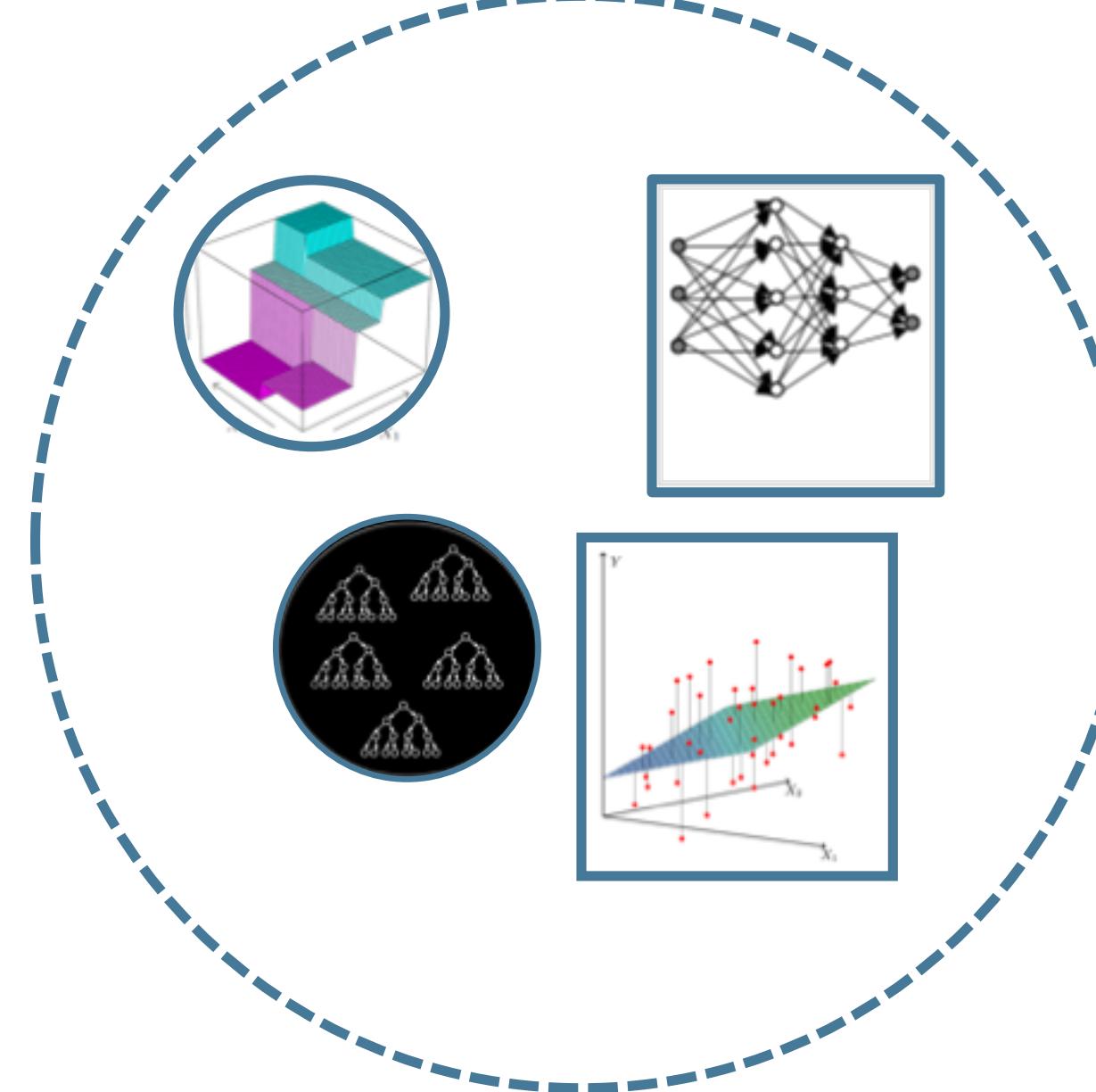


Gradient Boosting Machine



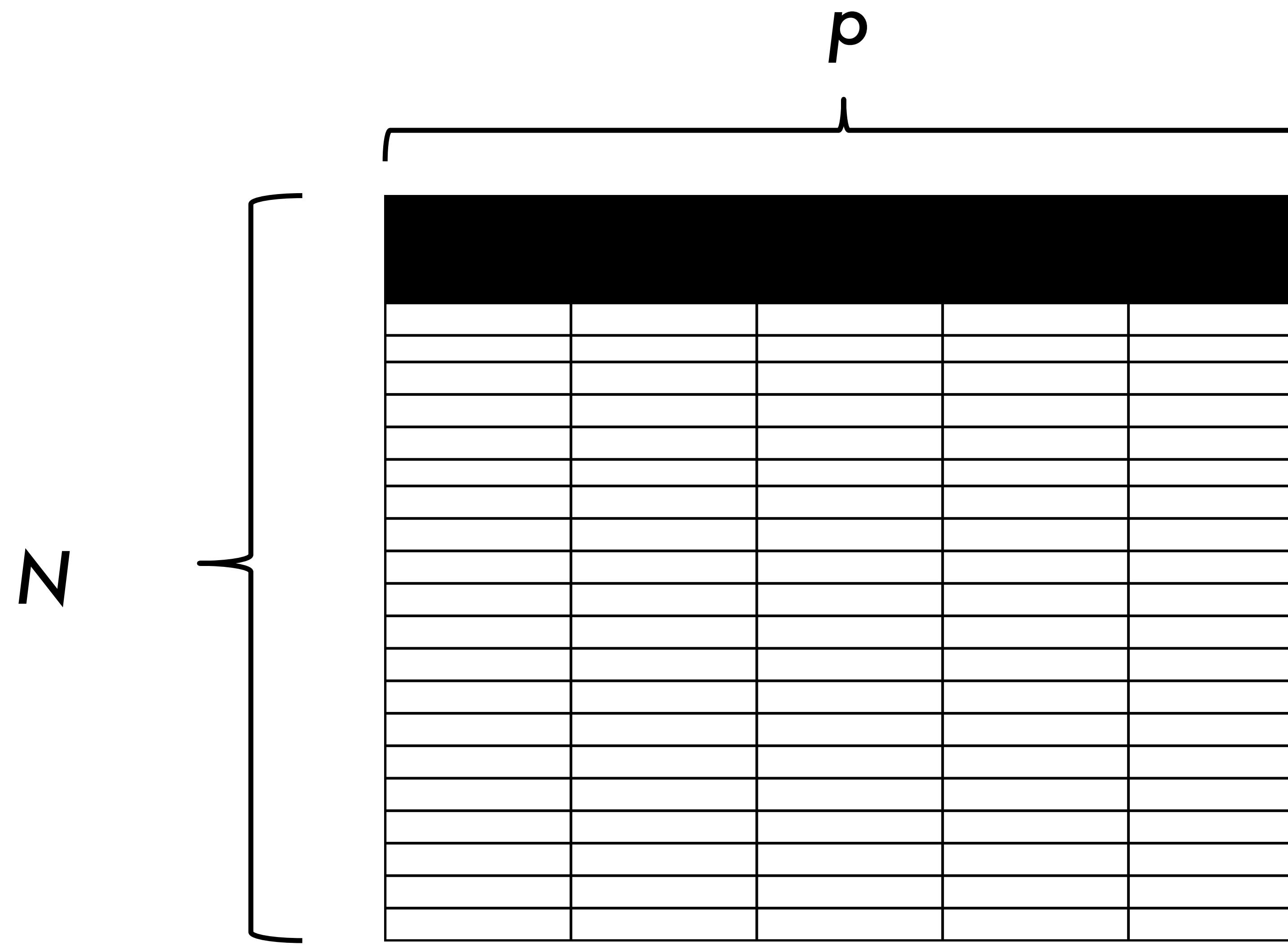
Random Forest

(AKA Stacking)



Stacked Ensemble Steps

- 1. Specify original dataset for training**
- 2. Specify algorithms with which to train**
- 3. Stack each algo's cross-validated predictions**
- 4. Train final model on cross-validated results**



**your original
training data**

“Level-zero” data

Stacked Ensemble Steps

1. Specify original dataset for training
2. **Specify algorithms with which to train**
3. Stack each algo's cross-validated predictions
4. Train final model on cross-validated results

supervised algos used to train

GLM

DRF

XRT

GBM

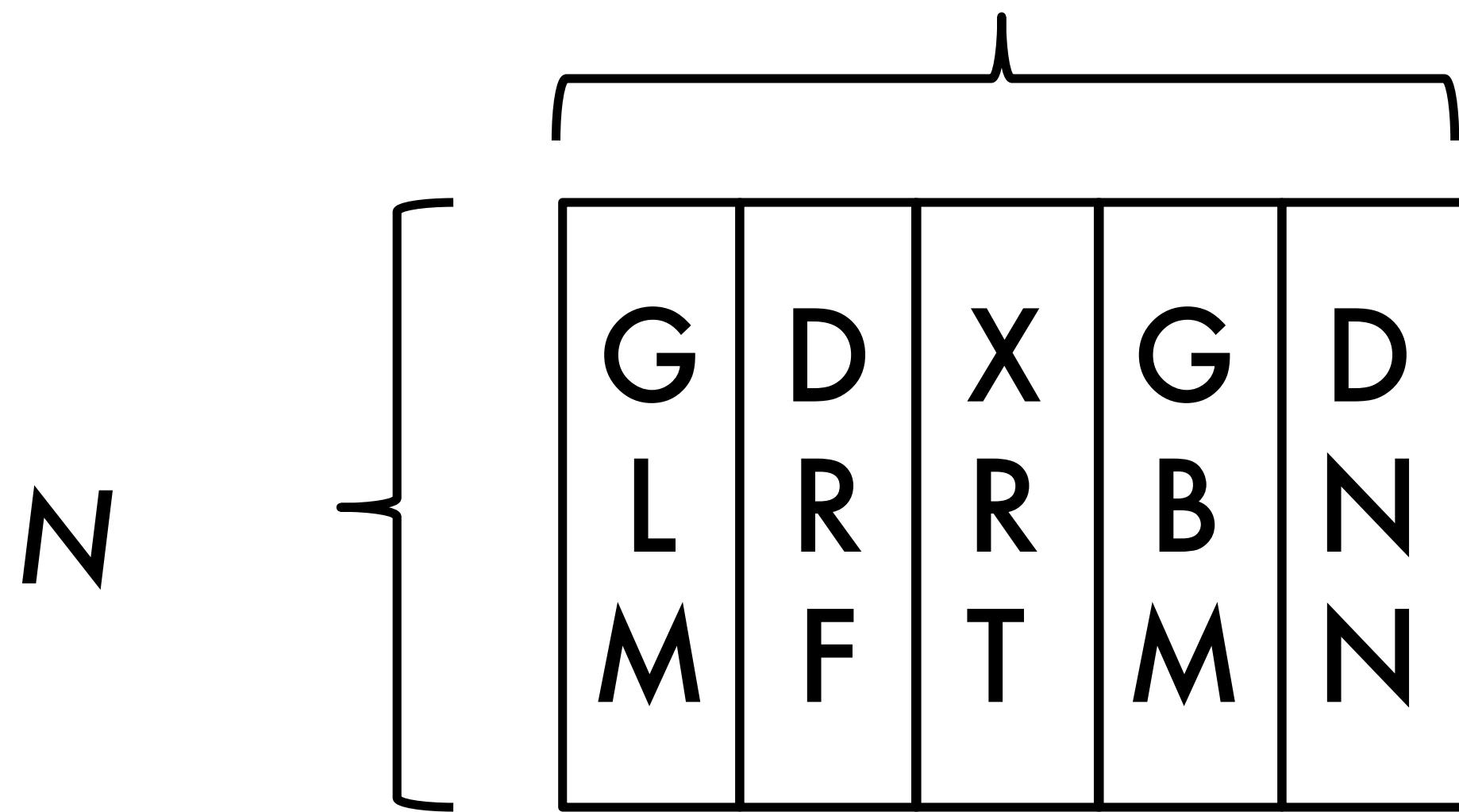
DNN

base learners

Stacked Ensemble Steps

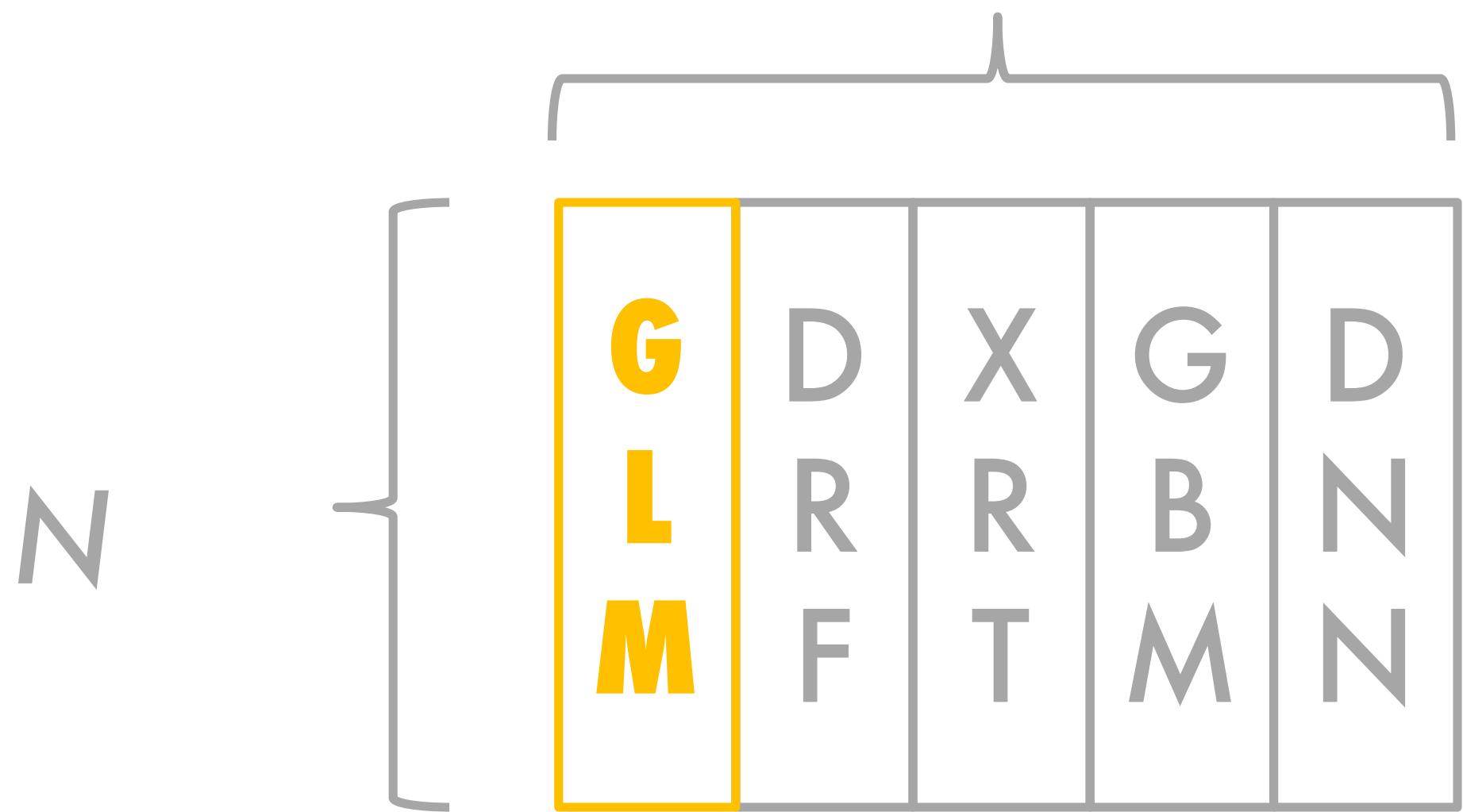
1. Specify original dataset for training
2. Specify algorithms with which to train
3. Stack each algo's cross-validated predictions
4. Train final model on cross-validated results

L base learners

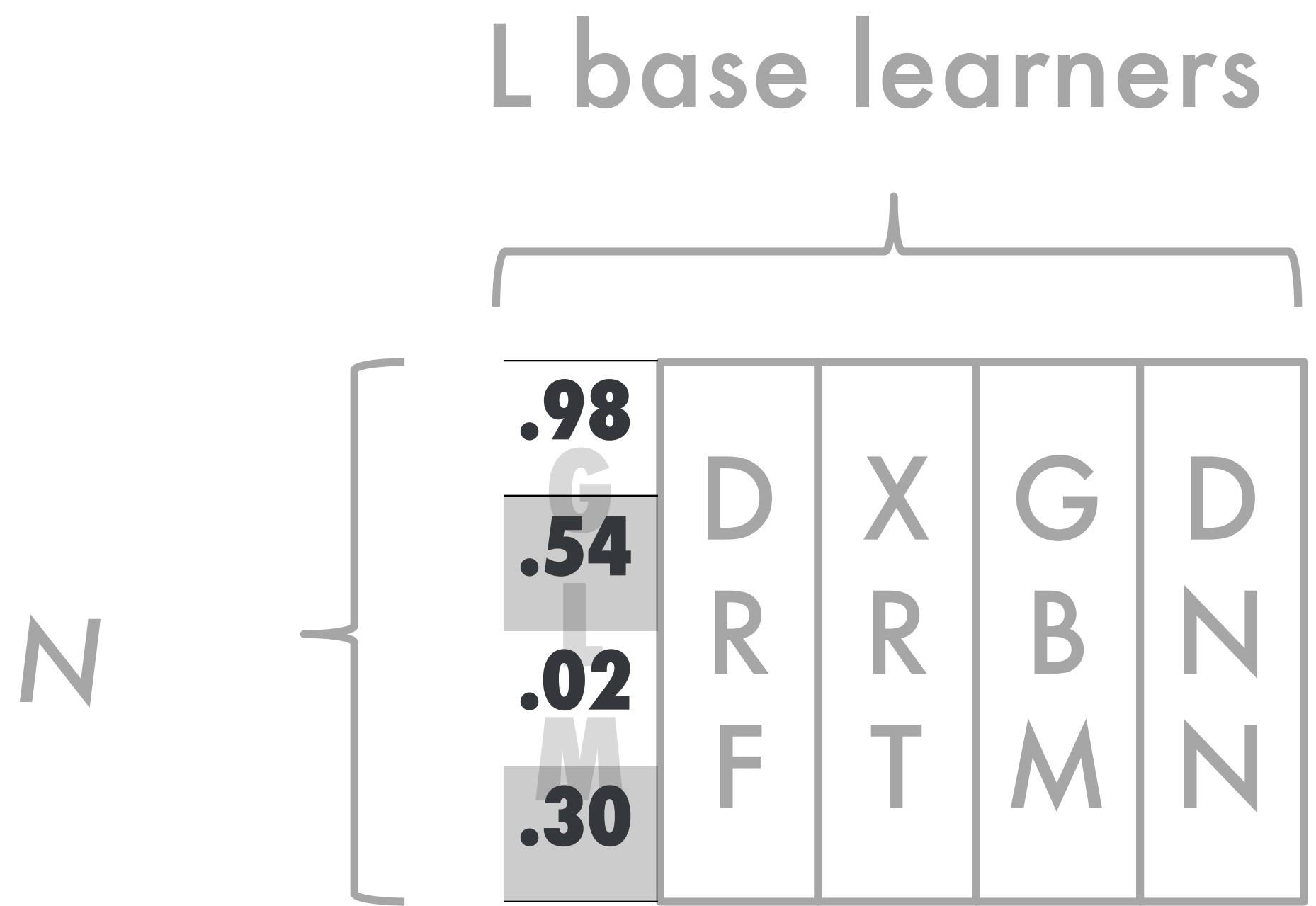


stacking your
base learners

L base learners



meaning:
each column is an algorithm's
cross-validated predictions



meaning:
each column is an algorithm's
cross-validated predictions

L base learners

N	10	D	X	G	D
G	5.5	R	R	B	N
R	7	T	T	M	N
F	M				
30					

meaning:
each column is an algorithm's
cross-validated predictions

How do we get the
cross-validated predictions?

How do we get the
cross-validated predictions?

`keep_cross_validation_predictions = True`

what is cross-validation?

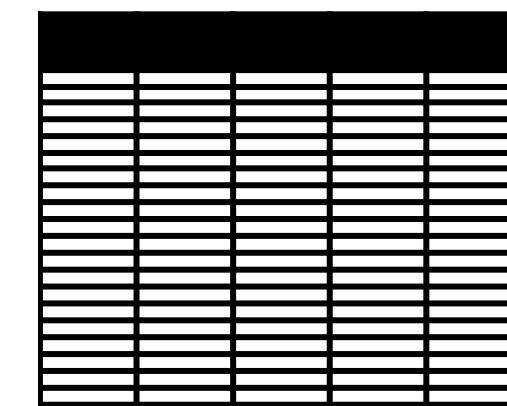
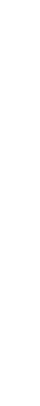
quick review

Without K-Fold Cross Validation:

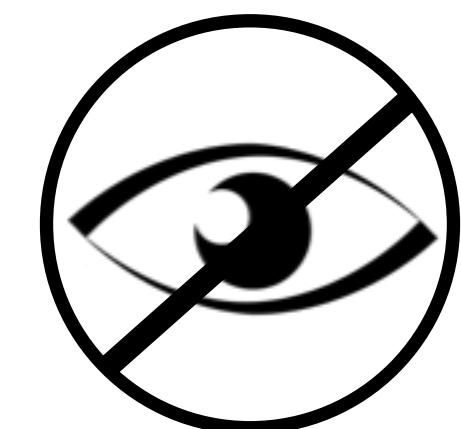
A)



Train



Test

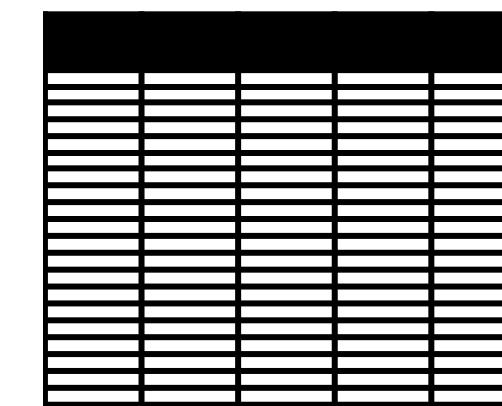


Without K-Fold Cross Validation:

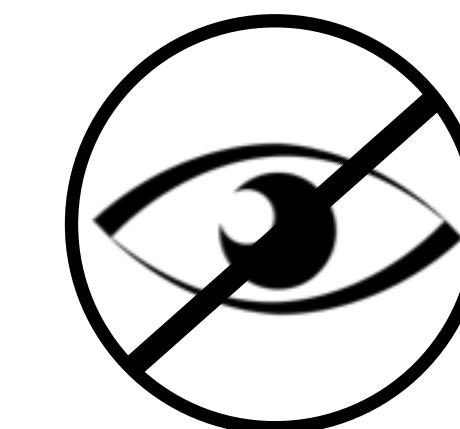
A)



Train



Test



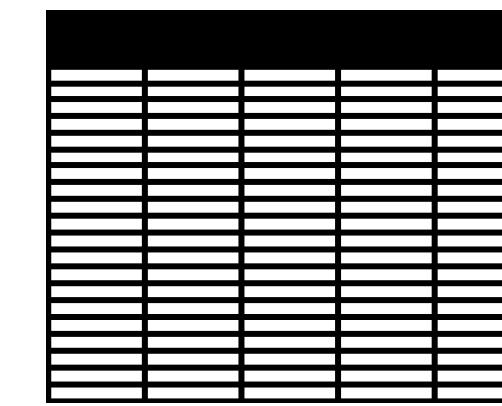
B)



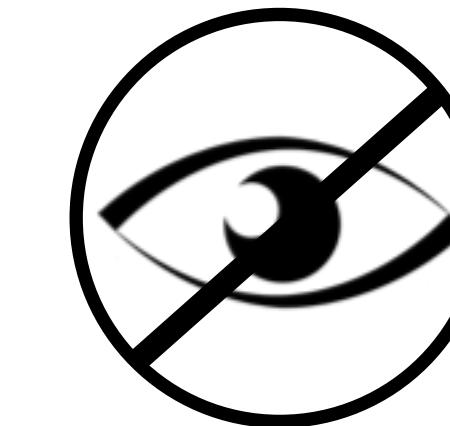
Train



Validate



Test

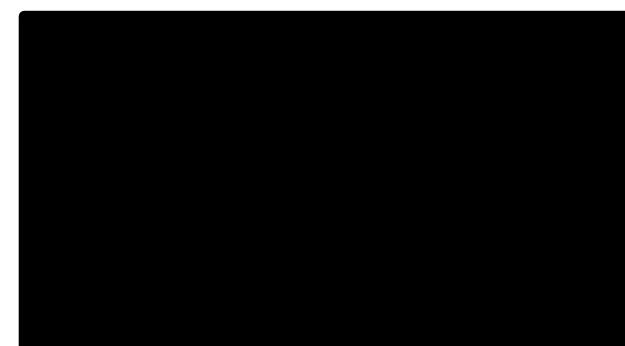
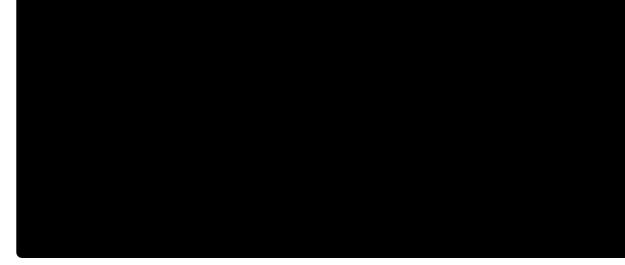


Model Selection

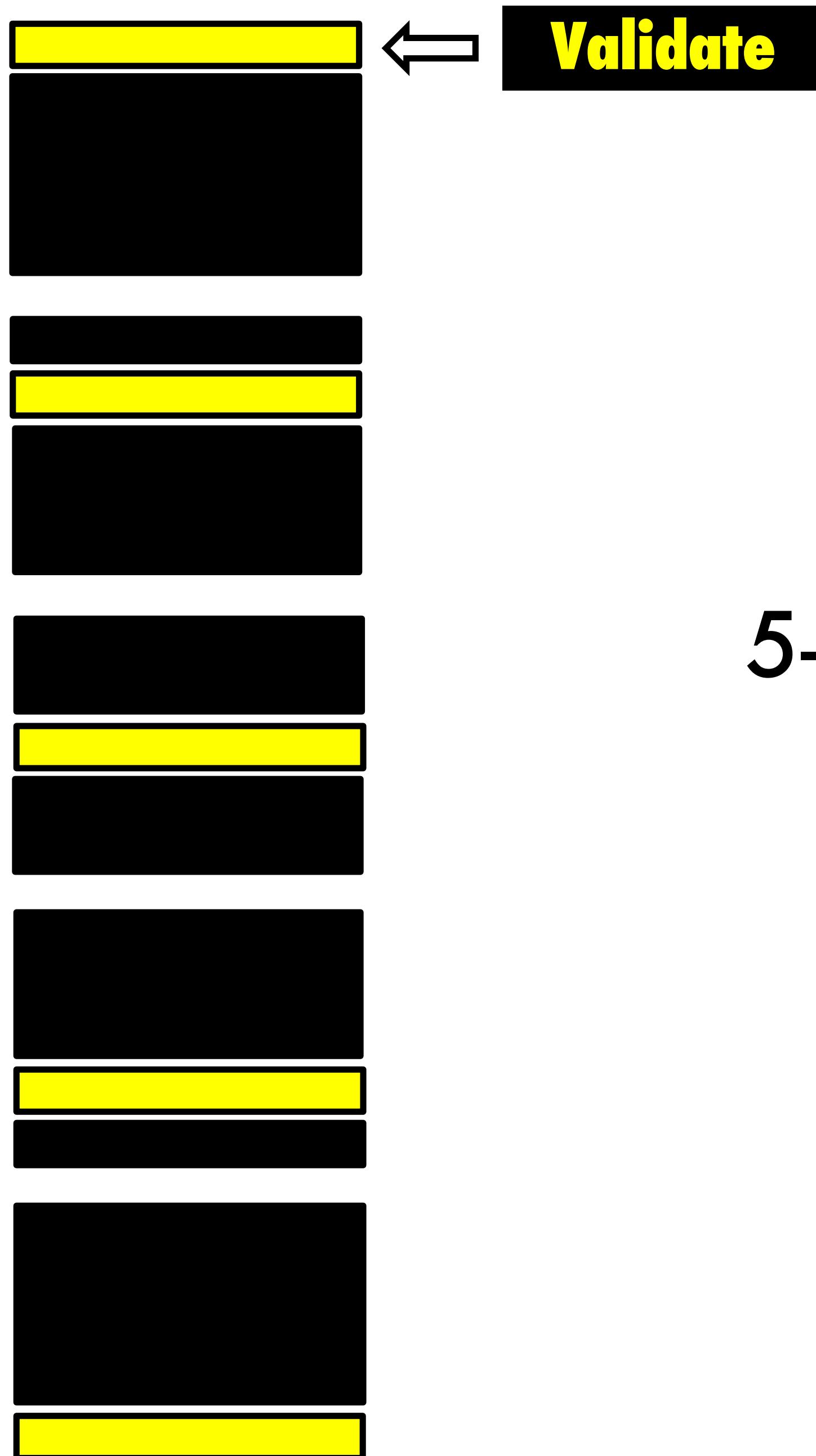


With K-Fold Cross Validation

**5-fold cross validation
for each algo**



**5-fold cross validation
for each algo**



**5-fold cross validation
for each algo**



**5-fold cross validation
for each algo**



Validate

**5-fold cross validation
for each algo**



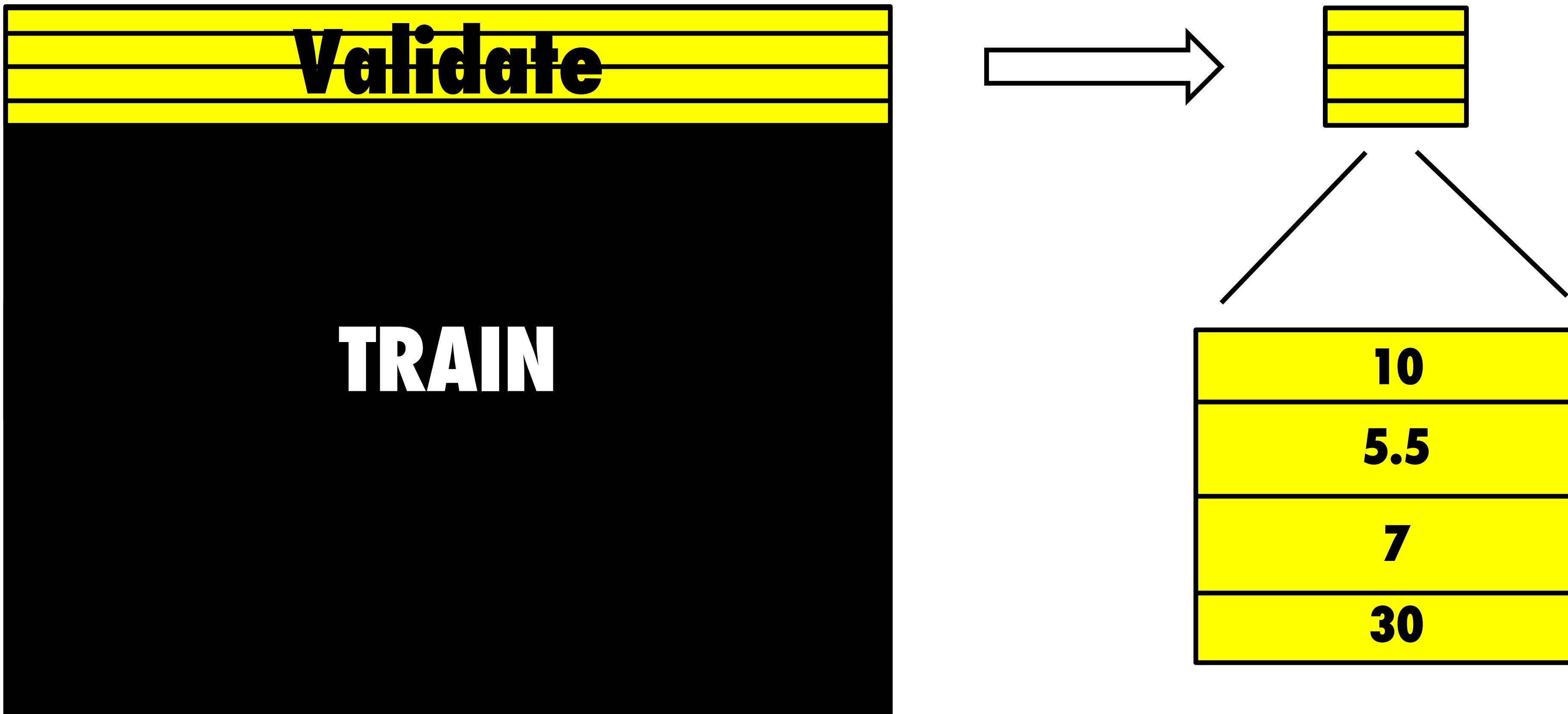
Continue

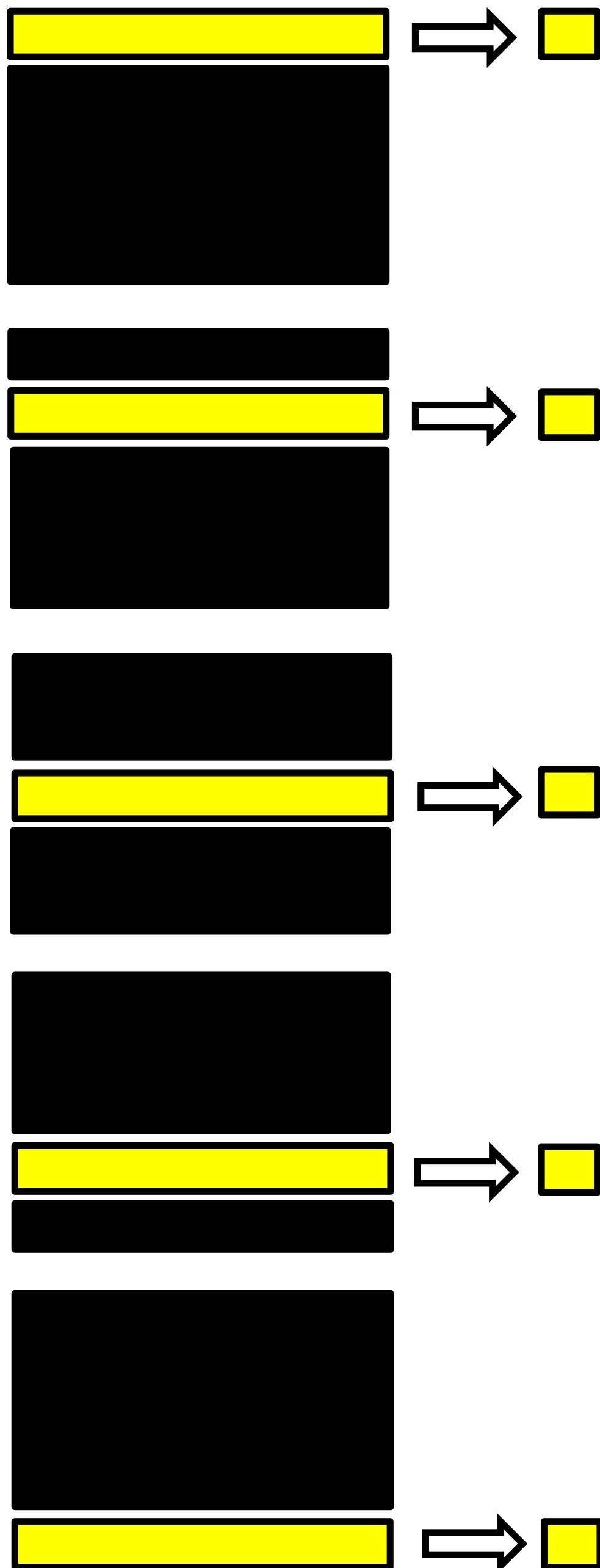
Process

5-fold cross validation
for each algo

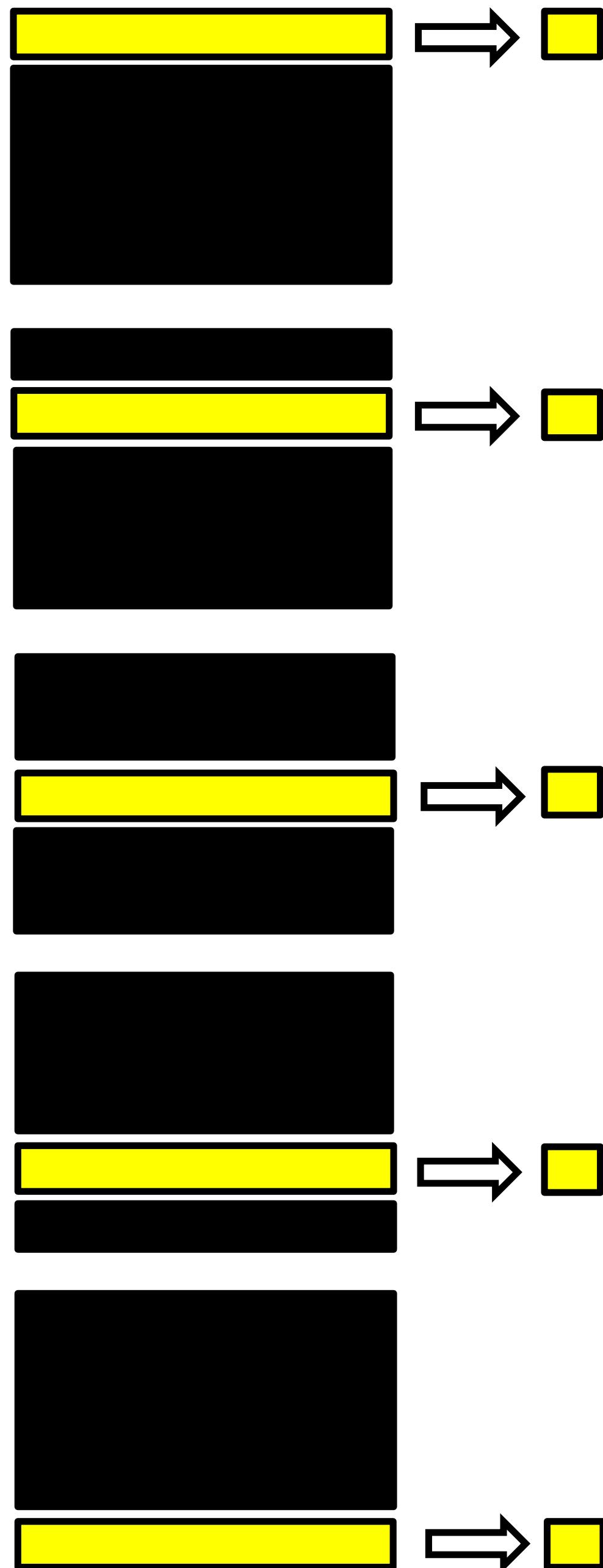
-
-
-
-
-
-
-
-
-

Cross-Validated Prediction Values

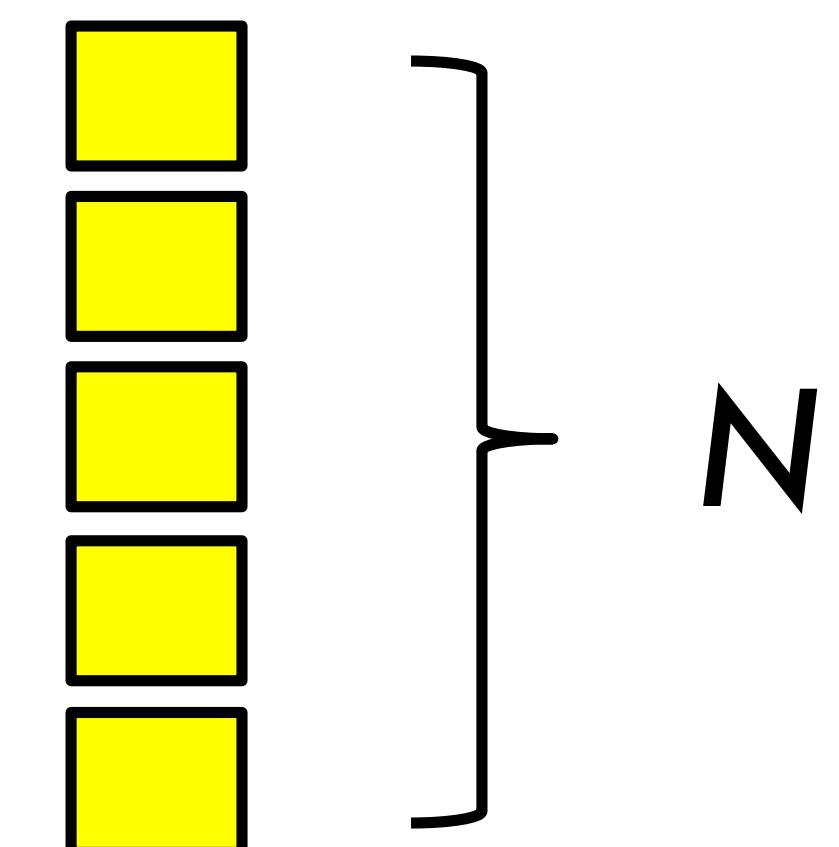
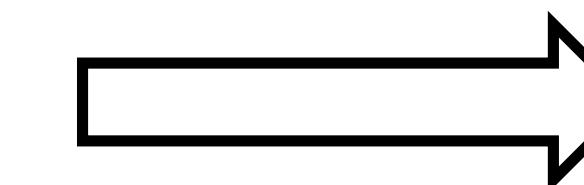




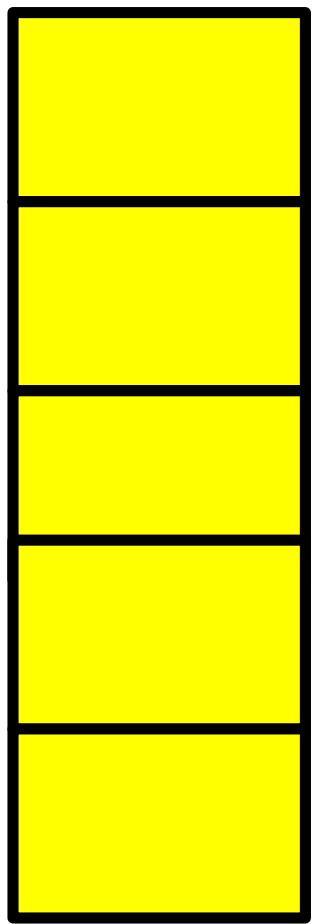
collect the
cross-validated
predictions



collect the
cross-validated
predictions

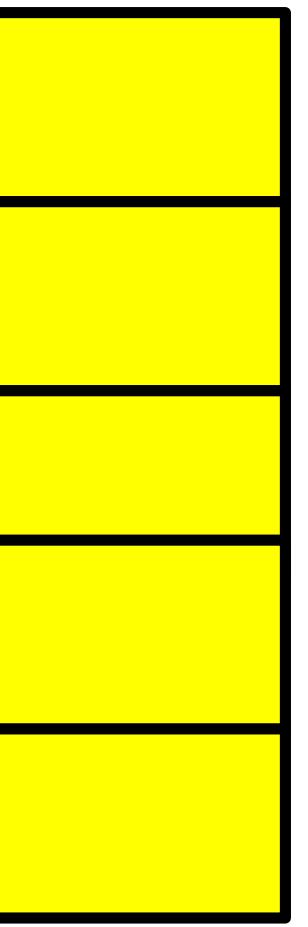


GLM



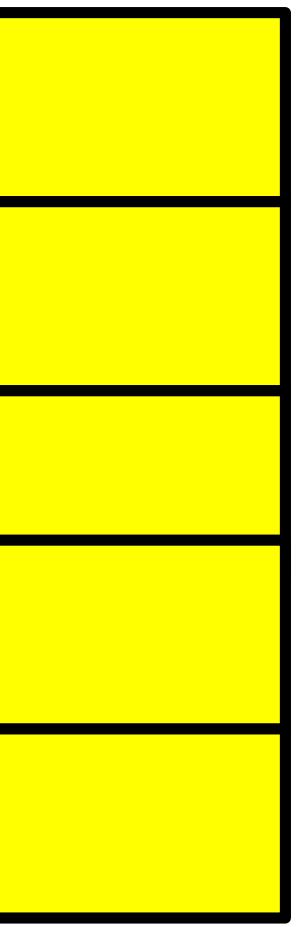
**results for each
algorithm**

DRF



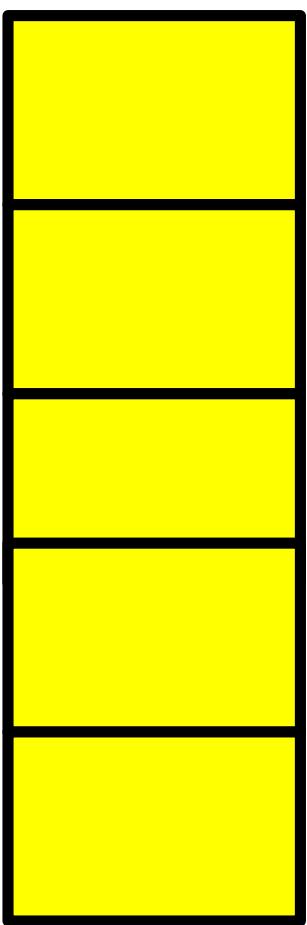
results for each
algorithm

XRT

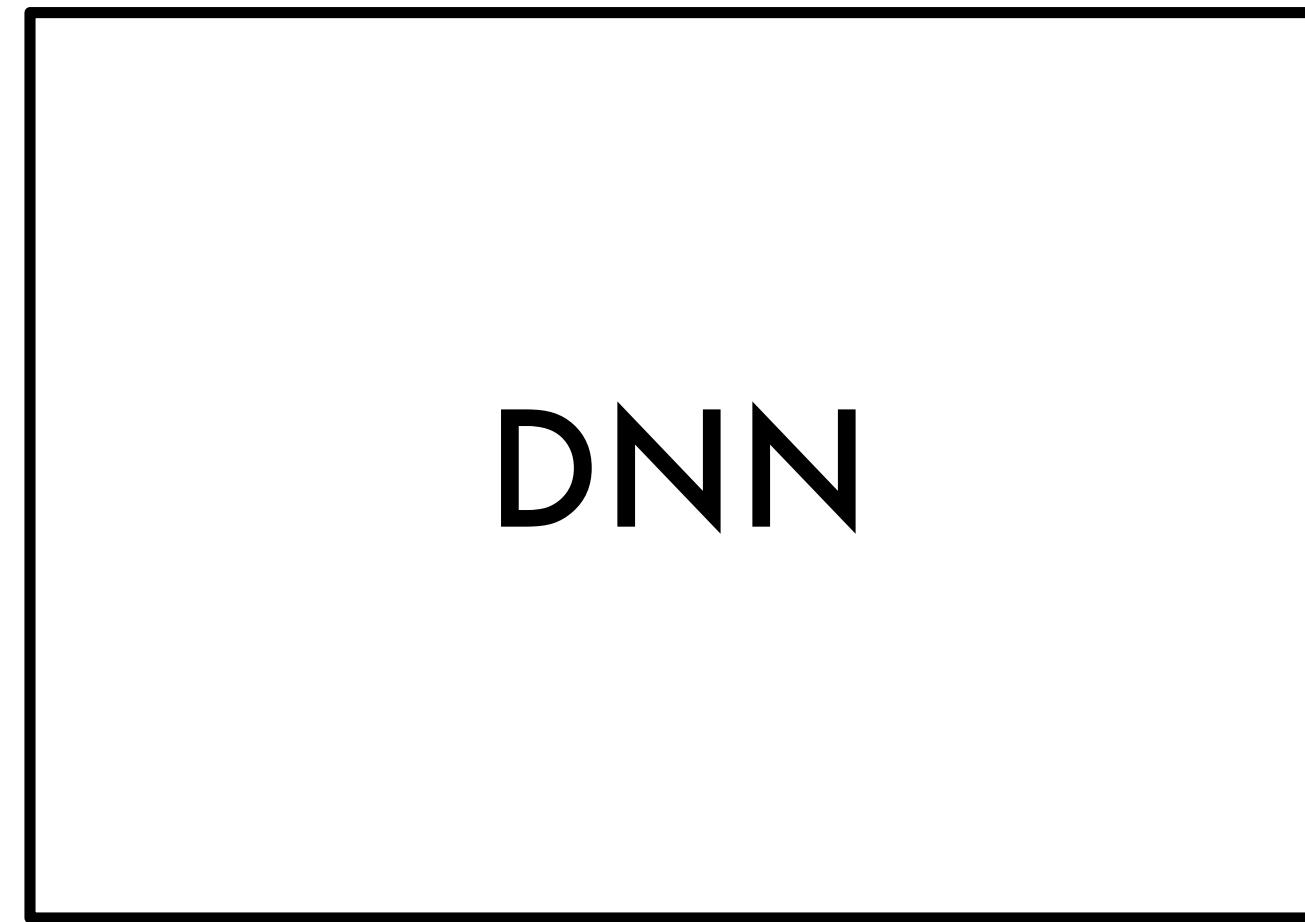


results for each
algorithm

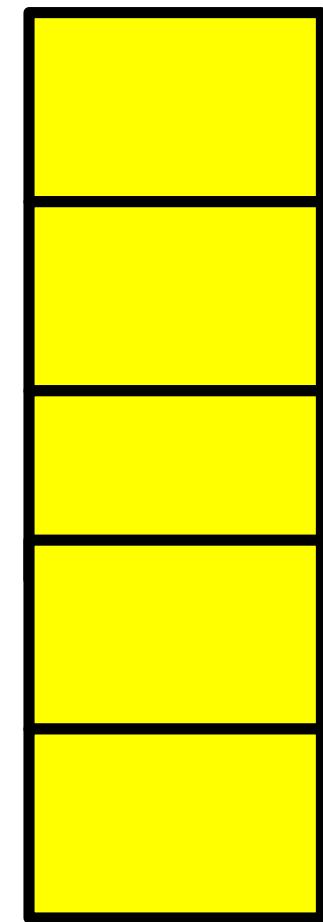
GBM



**results for each
algorithm**



DNN

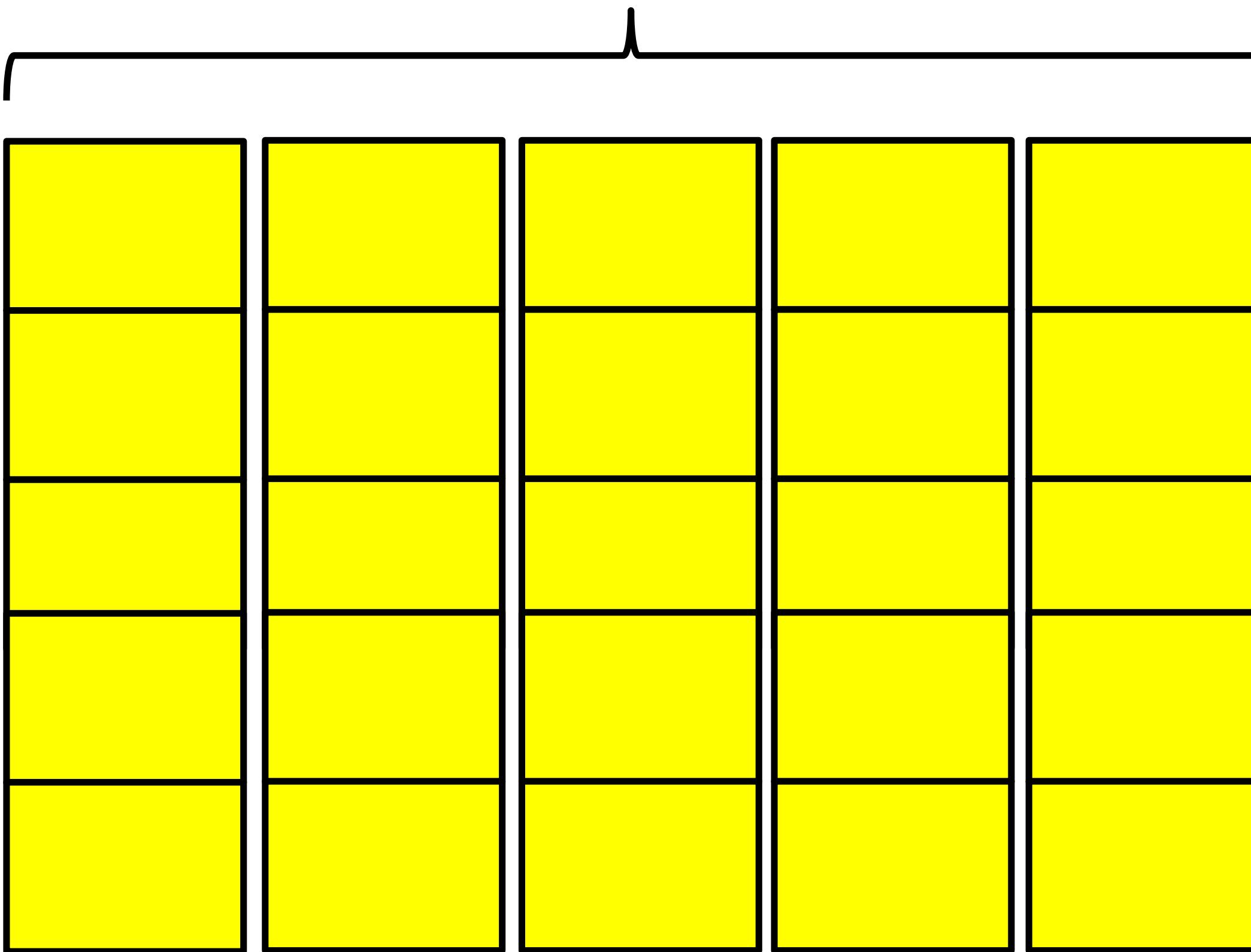


results for each
algorithm

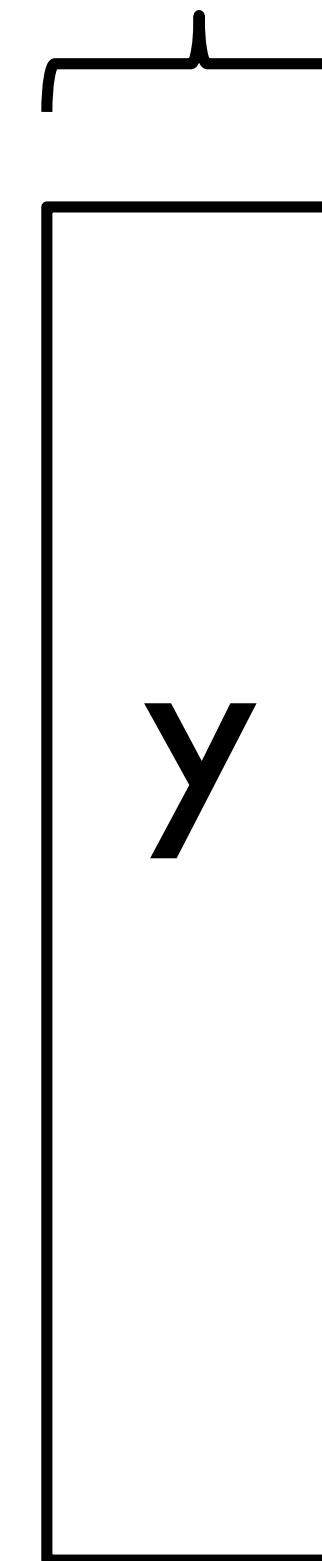
Stacked Ensemble Steps

1. Specify original dataset for training
2. Specify algorithms with which to train
3. Stack each algorithm's cross-validated predictions
4. Train final model on cross-validated results

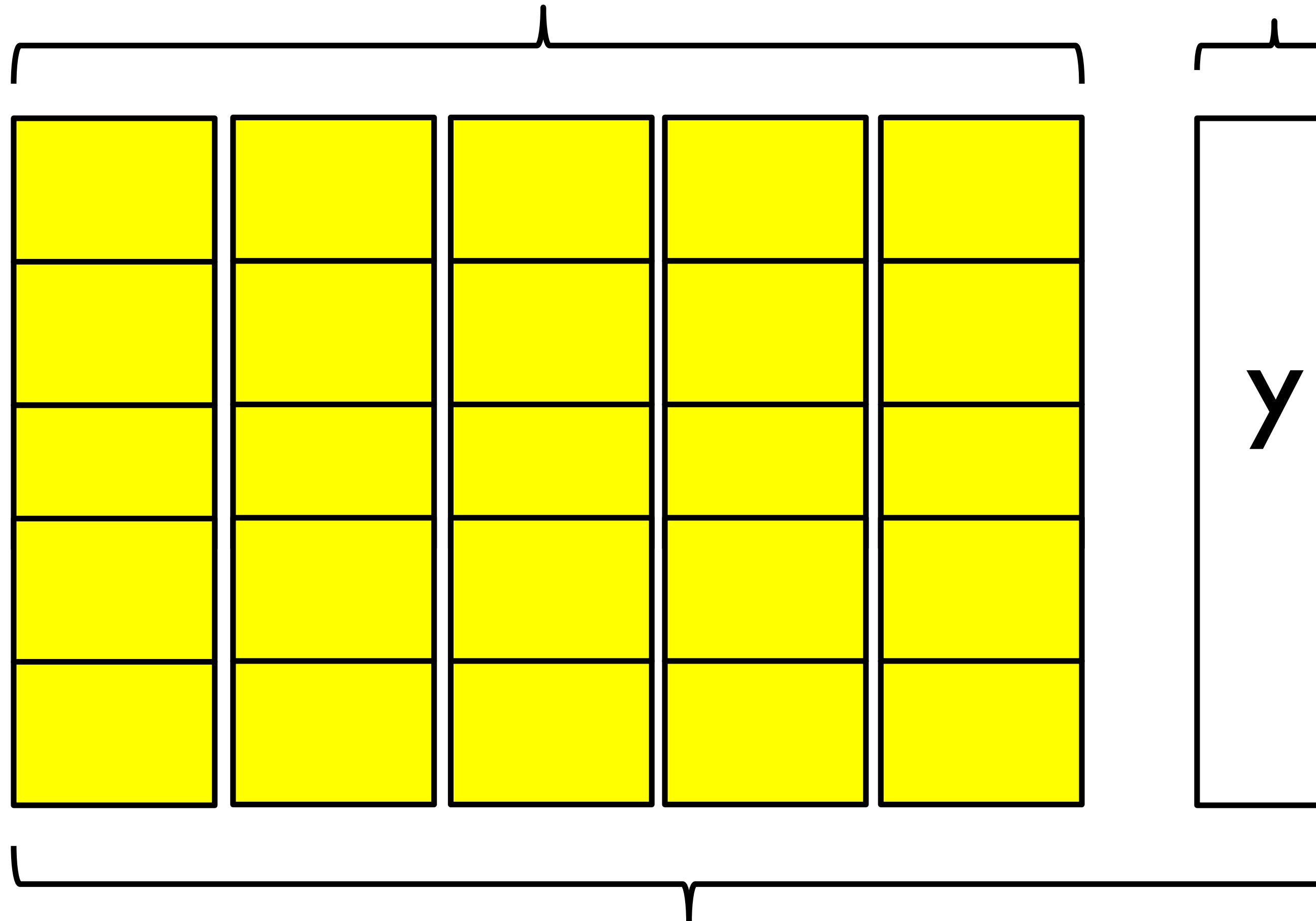
Base Learners' CV Predictions



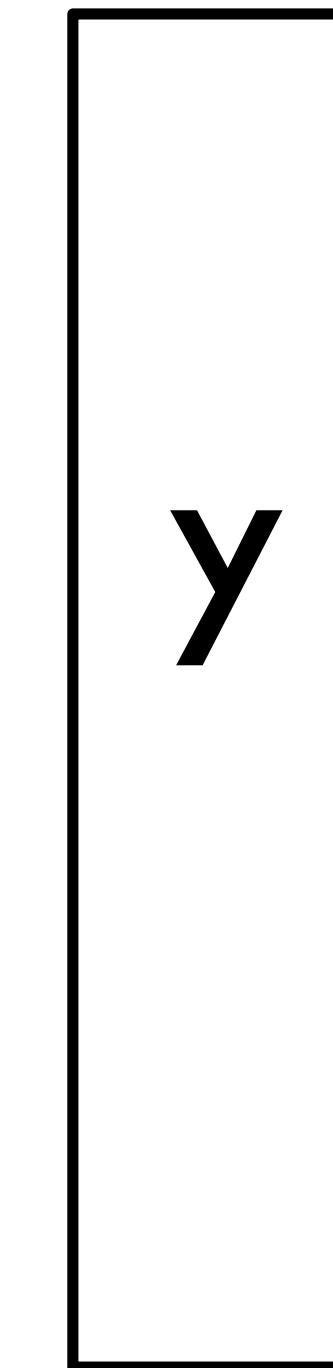
Original Target



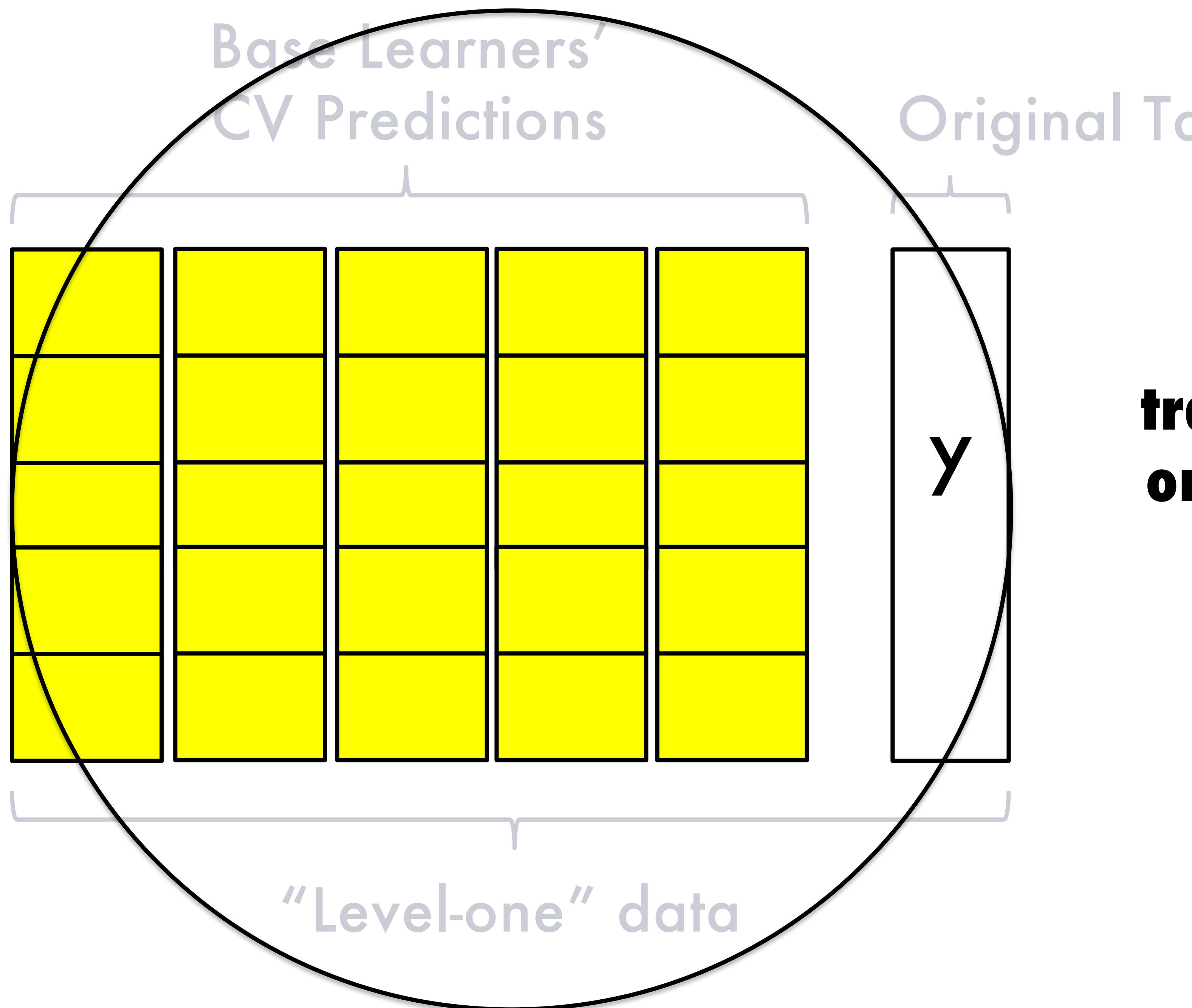
Base Learners' CV Predictions



Original Target



“Level-one” data



**train metalearner
on level-one data**



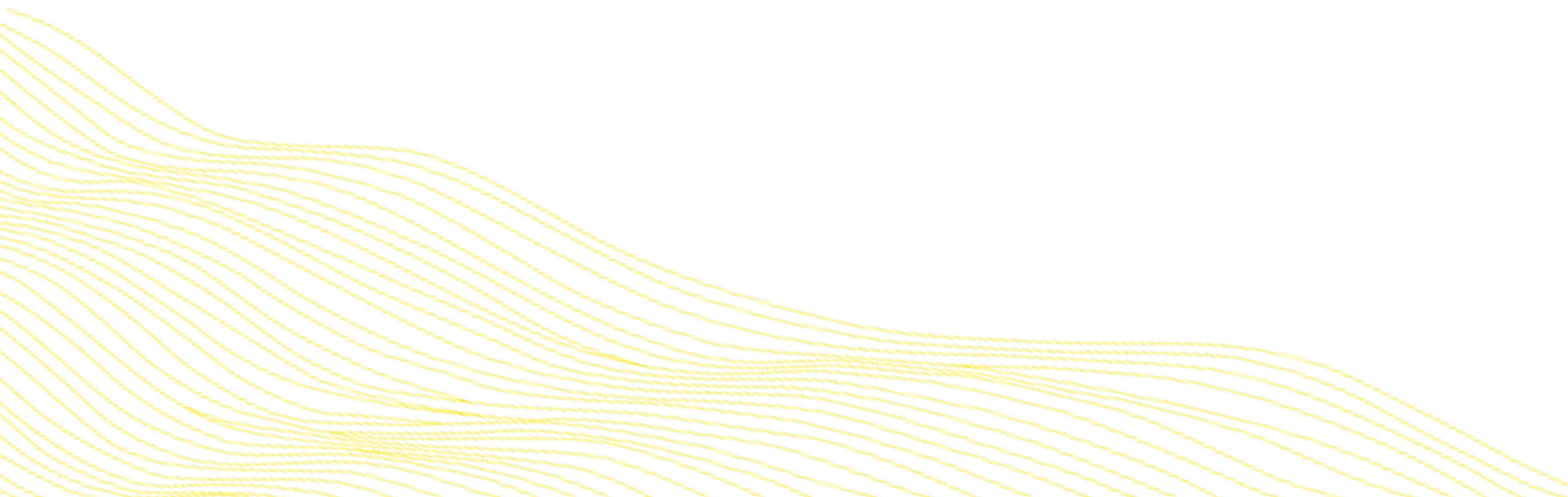
GLM

train **metalearner**
on level-one data

Summary: H2O's AutoML

1. Imputation, Categorical Encoding & Standardization
2. Random Grid Search
3. Early Stopping for Individual Models
4. 2 Stacked Ensembles
5. Auto-selects Winning Model

The AutoML Experience



The Interface: Flow

Simplify Machine Learning

2 Required parameters
training frame & response

CS | runAutoML

Run AutoML

Project Name:

Training Frame:

Seed:

Max models to build:

Max Run Time (sec):

Early stopping metric:

Early stopping rounds:

Stopping Tolerance:

nfold:

The Interface: Python

```
from h2o.automl import H2OAutoML  
  
automl = H2OAutoML(max_runtime_secs = 60, seed = 12345)  
  
automl.train(x = predictor_columns, y = target, training_frame = loan_stats)
```

```
automl.leaderboard
```

model_id	auc	logloss
StackedEnsemble_AllModels_0_AutoML_20180113_105834	0.720685	0.385048
StackedEnsemble_BestOfFamily_0_AutoML_20180113_105834	0.720319	0.385191
GLM_grid_0_AutoML_20180113_105834_model_0	0.71506	0.384576
GBM_grid_0_AutoML_20180113_105834_model_0	0.709894	0.387613
GBM_grid_0_AutoML_20180113_105834_model_1	0.703851	0.388938
GBM_grid_0_AutoML_20180113_105834_model_2	0.699872	0.391217
DRF_0_AutoML_20180113_105834	0.690366	0.406982
XRT_0_AutoML_20180113_105834	0.685729	0.39767

Additional Resources

All Documentation @ <http://docs.h2o.ai/h2o/latest-stable/index.html>

H₂O Open Source Software
[Documentation](#) | [H₂O Commercial](#)
[Software Documentation](#)

Open Source Software Documentation

[Getting Started & User Guides](#) | [Q & A](#) | [Algorithms](#) | [Languages](#) | [Tutorials, Examples, & Presentations](#) | [API & Developer Docs](#) | [For the Enterprise](#)

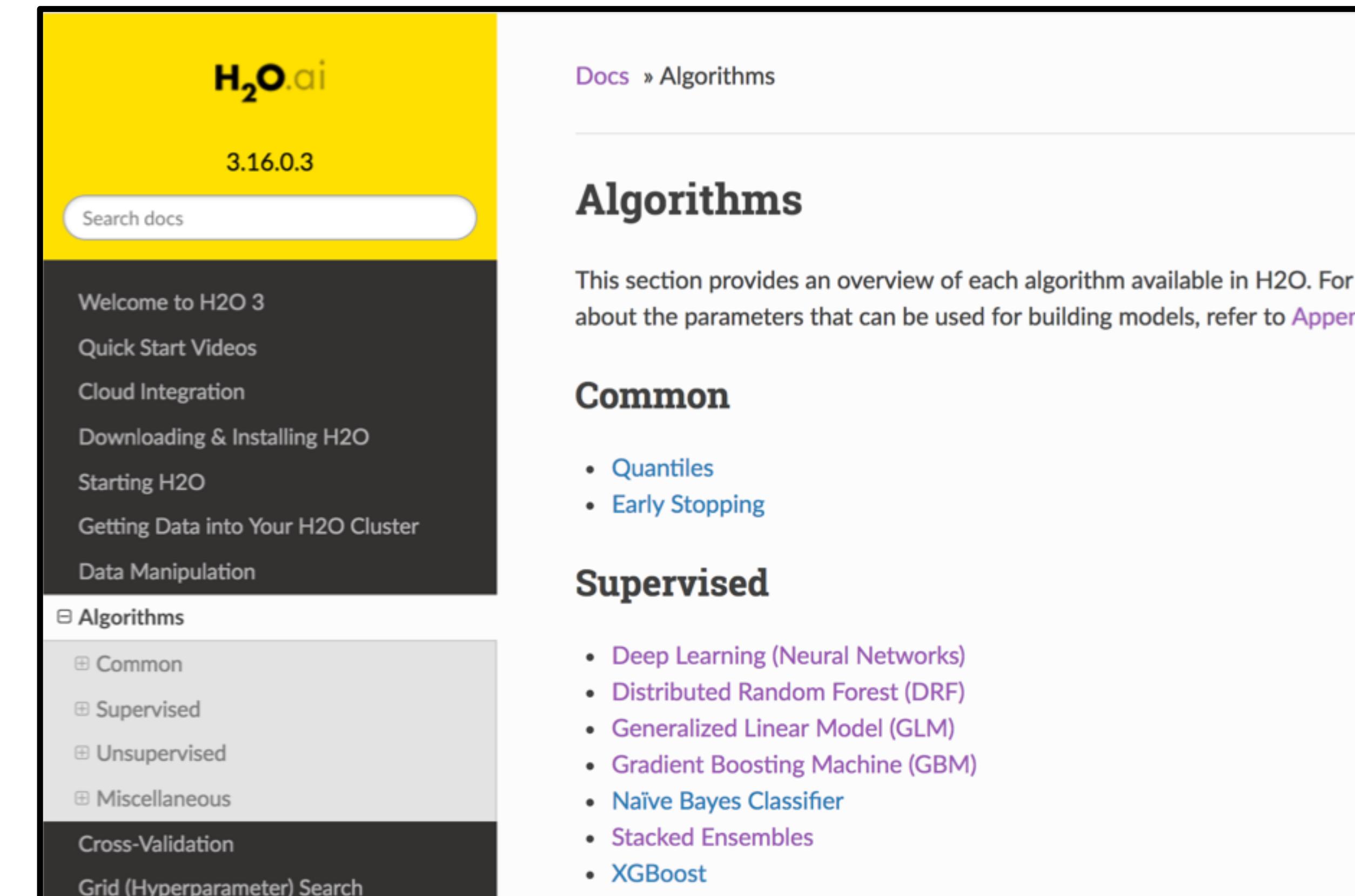
Getting Started & User Guides

H₂O

What is H₂O?
[H₂O User Guide](#) (Main docs)
H₂O Book (O'Reilly)
Recent Changes
Open Source License (Apache V2)

Quick Start Video - Flow Web UI
Quick Start Video - R
Quick Start Video - Python

[Download H₂O](#)



The screenshot shows the H₂O.ai documentation interface. At the top, there's a yellow header bar with the H₂O.ai logo and the version 3.16.0.3. Below it is a search bar labeled "Search docs". The main content area has a dark sidebar on the left containing links like "Welcome to H₂O 3", "Quick Start Videos", "Cloud Integration", "Downloading & Installing H₂O", "Starting H₂O", "Getting Data into Your H₂O Cluster", and "Data Manipulation". To the right of the sidebar, the page title is "Algorithms". Under "Algorithms", there's a heading "Common" with two items: "Quantiles" and "Early Stopping". Below that is a heading "Supervised" with a list of nine items: "Deep Learning (Neural Networks)", "Distributed Random Forest (DRF)", "Generalized Linear Model (GLM)", "Gradient Boosting Machine (GBM)", "Naïve Bayes Classifier", "Stacked Ensembles", and "XGBoost". There are also collapsed sections for "Algorithms", "Common", "Supervised", "Unsupervised", and "Miscellaneous".

post questions to: <https://stackoverflow.com/questions/tagged/h2o>

Send questions to me: laurend@h2o.ai

H₂O.ai

Thanks!

