CS 8395-03

Automatic Machine Learning

Slides adapted from the Advanced Course on Data Science & Machine Learning

Joaquin Vanschoren

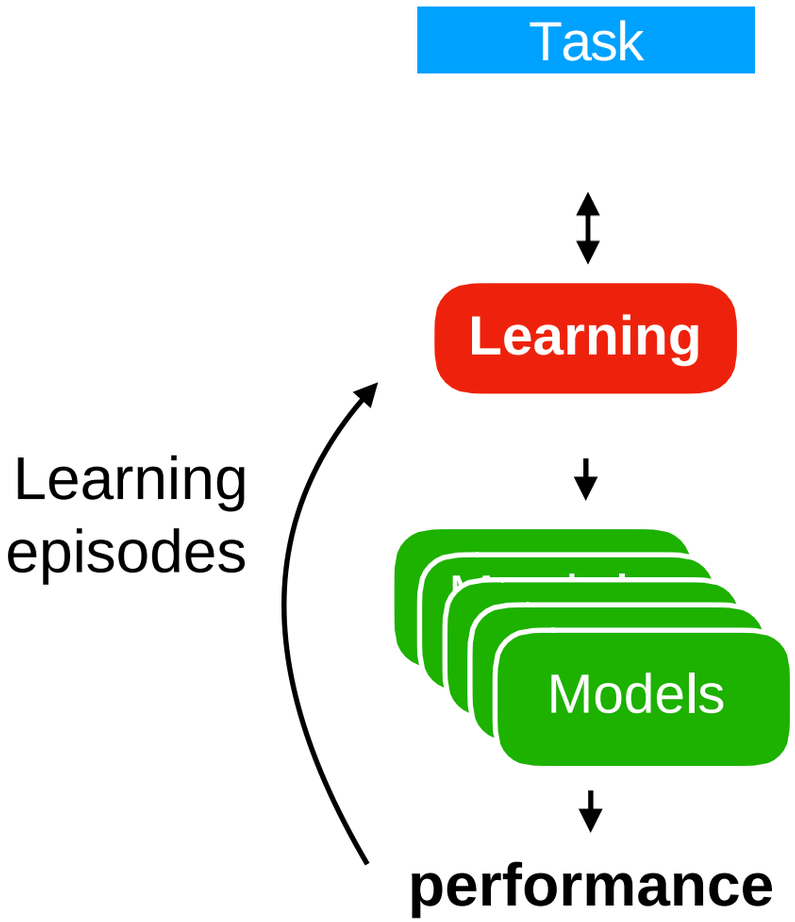
Siena, 2019

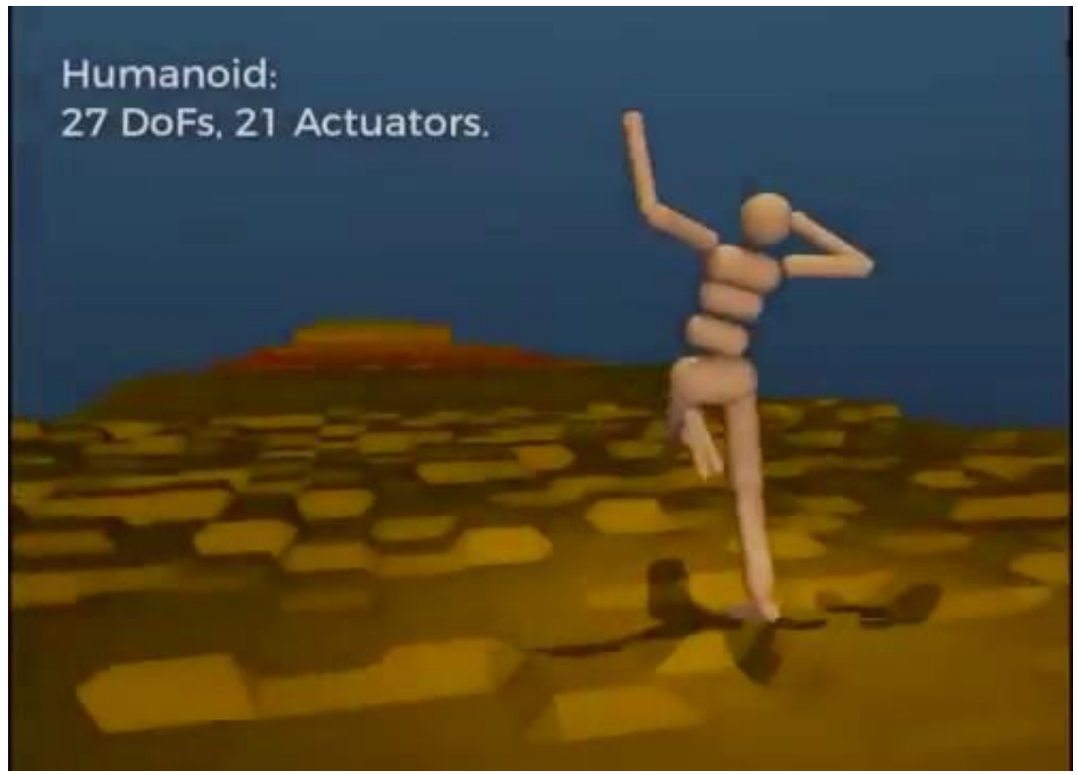
http://bit.ly/openml

AutoML

Learning takes experimentation

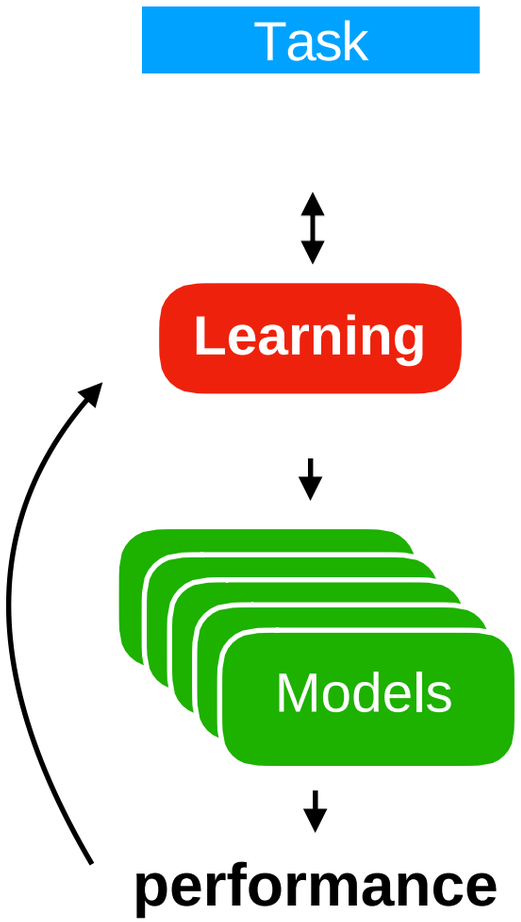
It can take a *lot* of trial and error



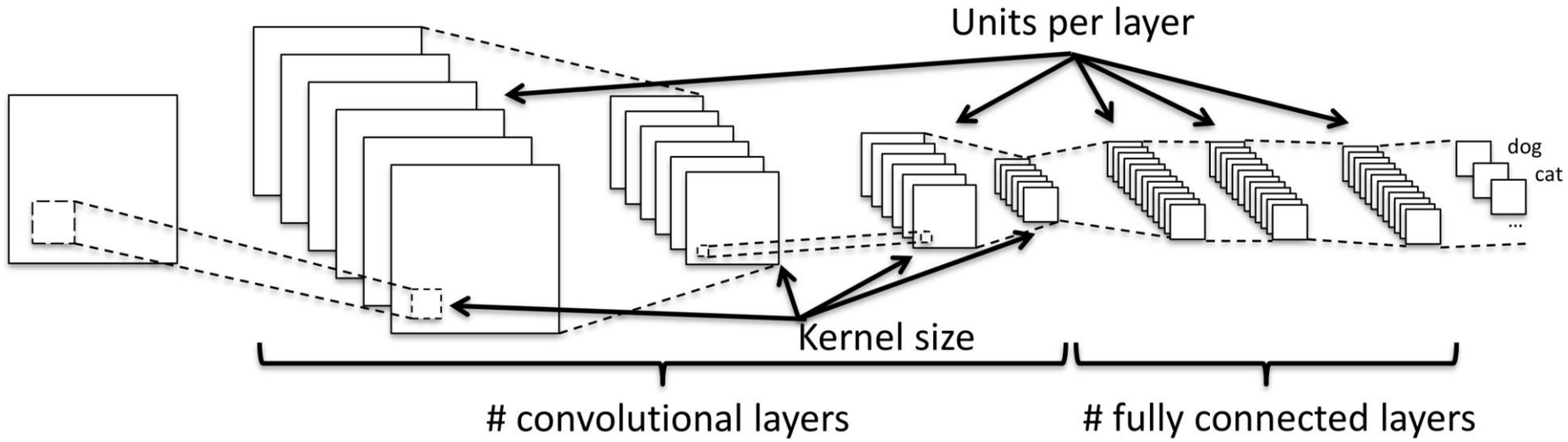


Learning to learn takes experimentation

Building machine learning systems requires a *lot* of expertise and trials



Neural networks



- design architecture (operators/layers)
- many sensitive hyperparameters
  - optimization algorithm, learning rate,…
  - regularization, dropout,…
  - batch size, epochs, early stopping, …
  - data augmentation, …

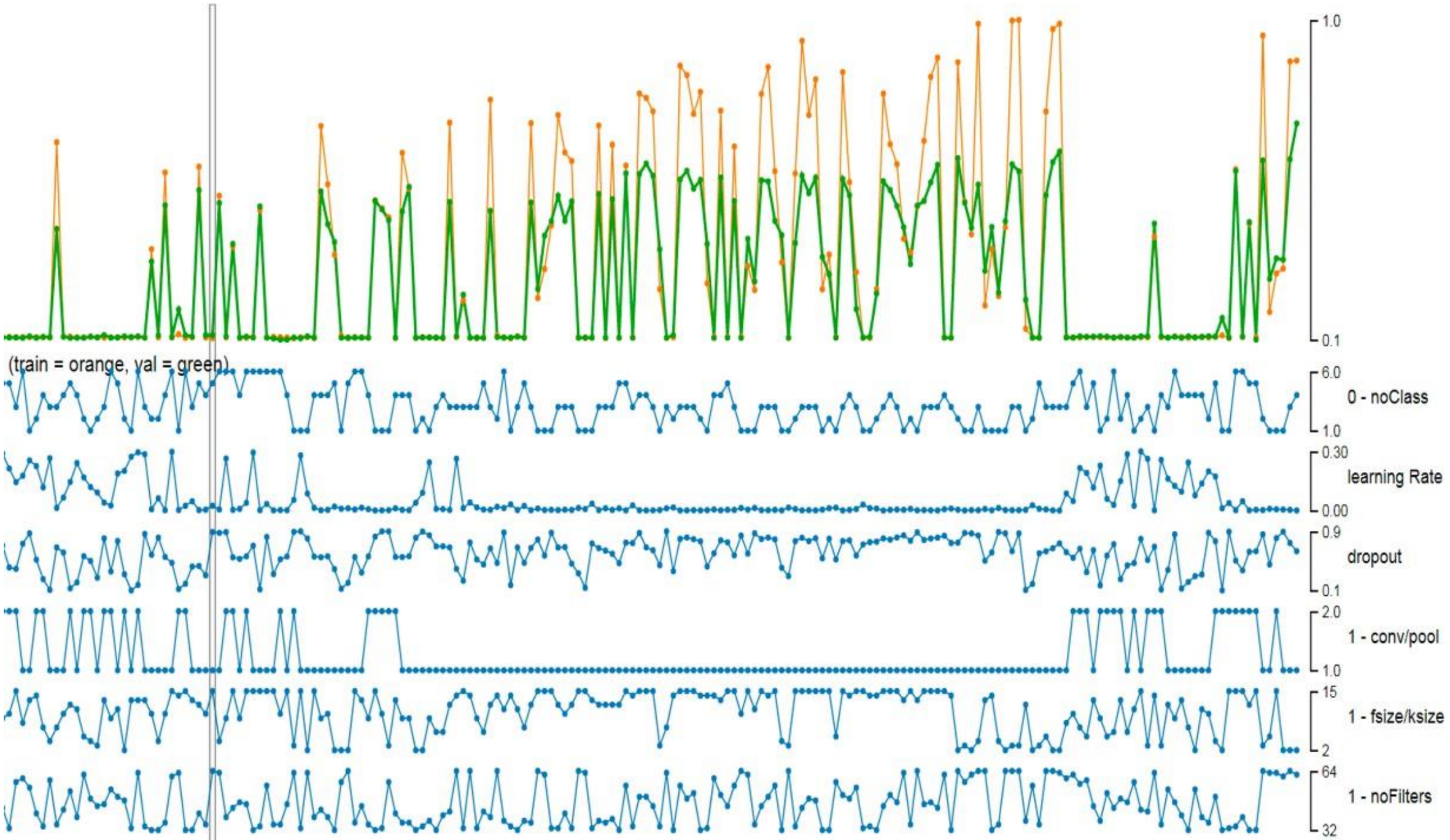
Hyperparameters

Every design decision made by the *user (architecture, operators, tuning, …)*

	Name	Range	Default	log scale	Type	Conditional
Network hyperparameters	batch size	$[32, 4096]$	32	✓	float	-
	number of updates	$[50, 2500]$	200	✓	int	-
	number of layers	$[1, 6]$	1	-	int	-
	learning rate	$[10^{-6}, 1.0]$	10^{-2}	✓	float	-
	L_2 regularization	$[10^{-7}, 10^{-2}]$	10^{-4}	✓	float	-
	dropout output layer	$[0.0, 0.99]$	0.5	✓	float	-
	solver type	{SGD, Momentum, Adam, Adadelta, Adagrad, smorm, Nesterov }	smorm3s	-	cat	-
	lr-policy	{Fixed, Inv, Exp, Step}	fixed	-	cat	-
Conditioned on solver type	β_1	$[10^{-4}, 10^{-1}]$	10^{-1}	✓	float	✓
	β_2	$[10^{-4}, 10^{-1}]$	10^{-1}	✓	float	✓
	ρ	$[0.05, 0.99]$	0.95	✓	float	✓
	momentum	$[0.3, 0.999]$	0.9	✓	float	✓
Conditioned on lr-policy	γ	$[10^{-3}, 10^{-1}]$	10^{-2}	✓	float	✓
	k	$[0.0, 1.0]$	0.5	-	float	✓
	s	$[2, 20]$	2	-	int	✓
Per-layer hyperparameters	activation-type	{Sigmoid, TanH, ScaledTanH, ELU, ReLU, Leaky, Linear}	ReLU	-	cat	✓
	number of units	$[64, 4096]$	128	✓	int	✓
	dropout in layer	$[0.0, 0.99]$	0.5	-	float	✓
	weight initialization	{Constant, Normal, Uniform, Glorot-Uniform, Glorot-Normal, He-Normal, He-Uniform, Orthogonal, Sparse}	He-Normal	-	cat	✓
	std. normal init.	$[10^{-7}, 0.1]$	0.0005	-	float	✓
	leakiness	$[0.01, 0.99]$	$\frac{1}{3}$	-	float	✓
	tanh scale in	$[0.5, 1.0]$	$2/3$	-	float	✓
	tanh scale out	$[1.1, 3.0]$	1.7159	✓	float	✓

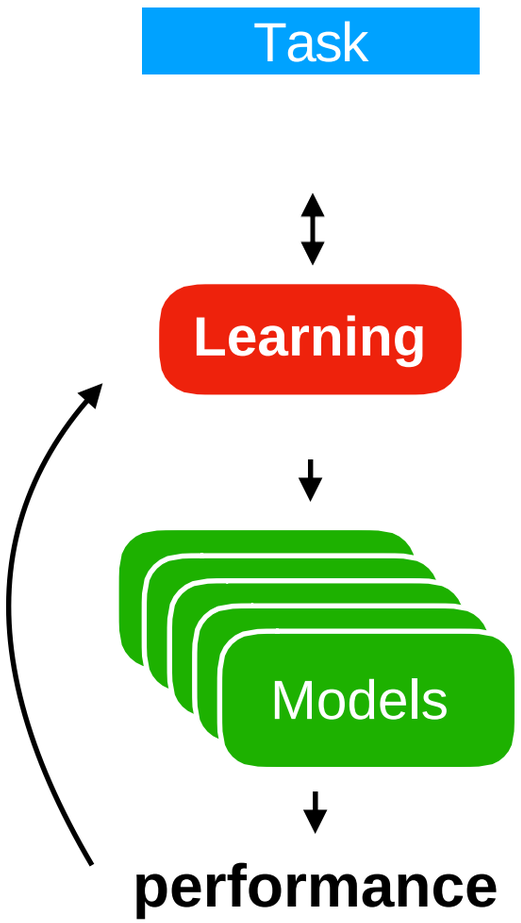
Hyperparameters

Can be very sensitive

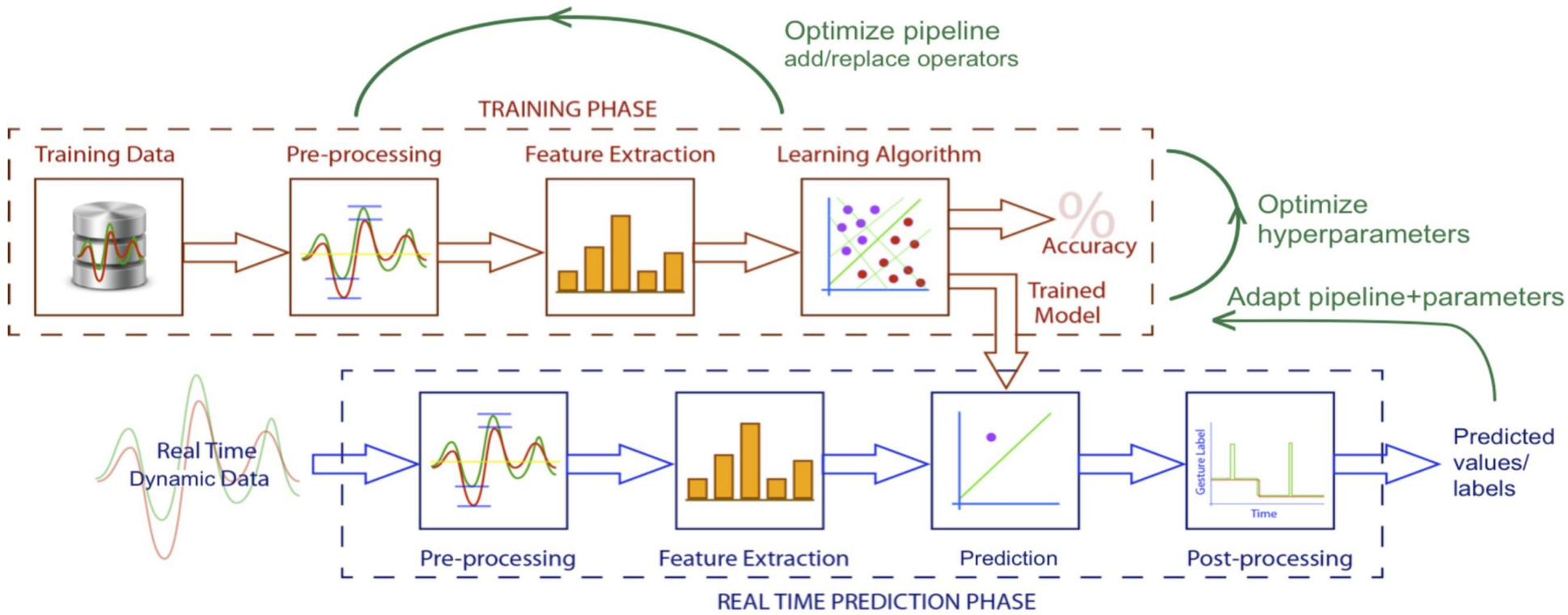


Learning to learn takes experimentation

Building machine learning systems requires a lot of expertise and trials



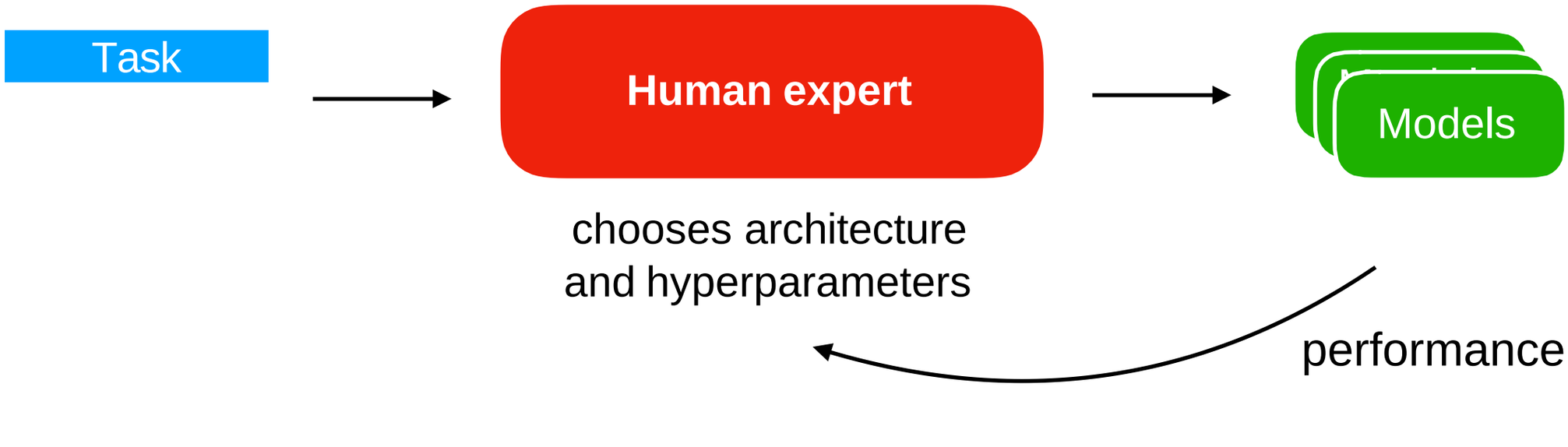
Machine learning pipelines



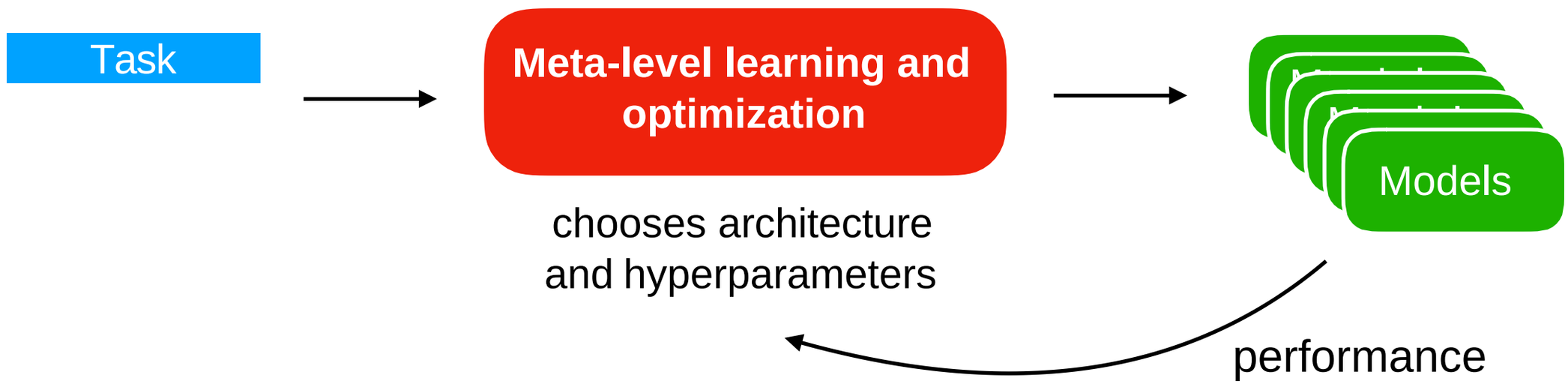
- design pipeline architecture (operators)
- clean, preprocess data (!)
- select and/or engineer features
- select and tune models
- adjust to evolving input data (concept drift),…

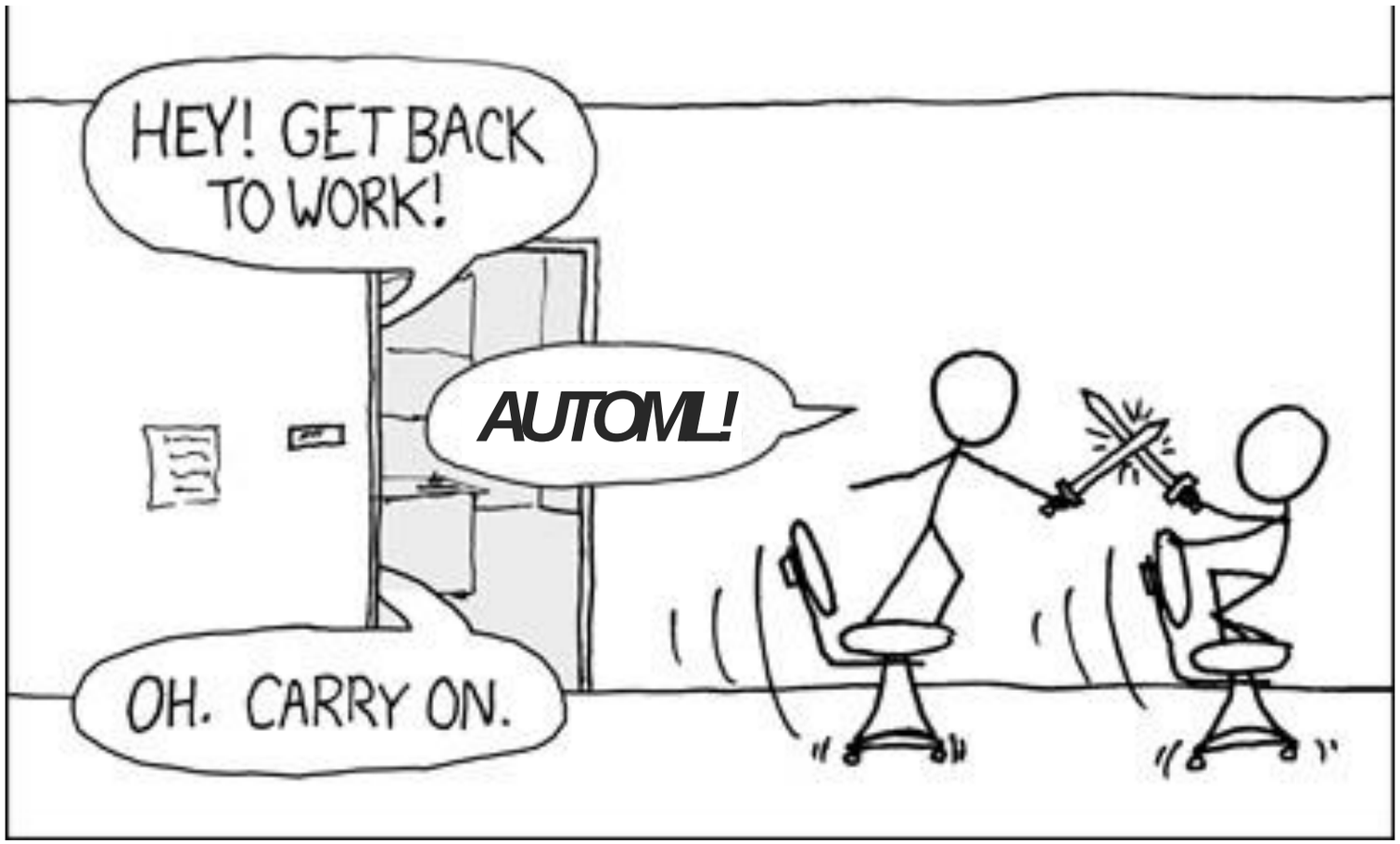
Automated machine learning

Current practice



Replace manual model building by automation

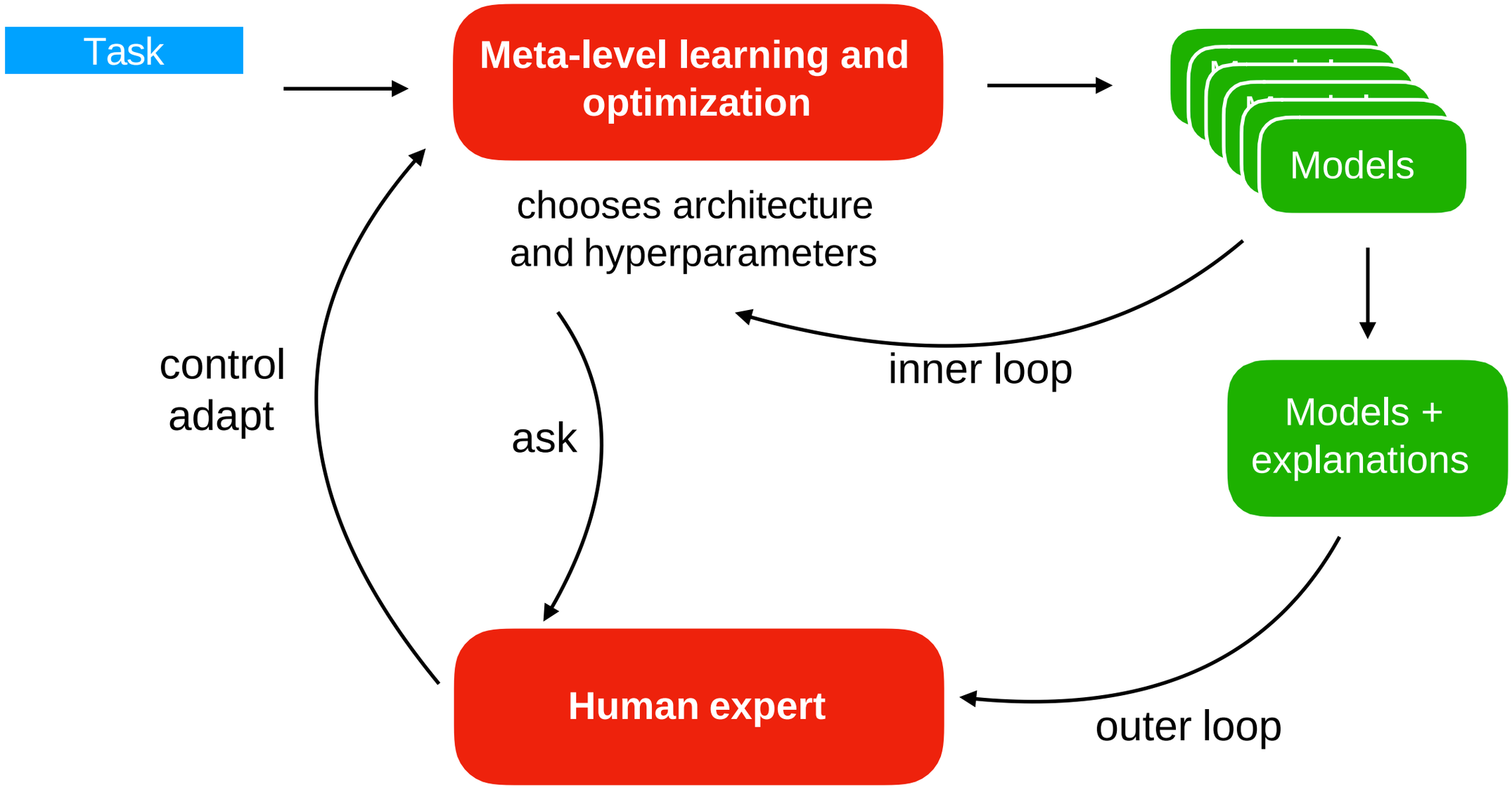



HEY! GET BACK TO WORK!
AUTOML!
OH. CARRY ON.

[Sutton 2019]

Human-in-the-loop AutoML (semi-AutoML)

Domain knowledge and human expertise are very valuable
e.g. unknown unknowns, preference learning,…

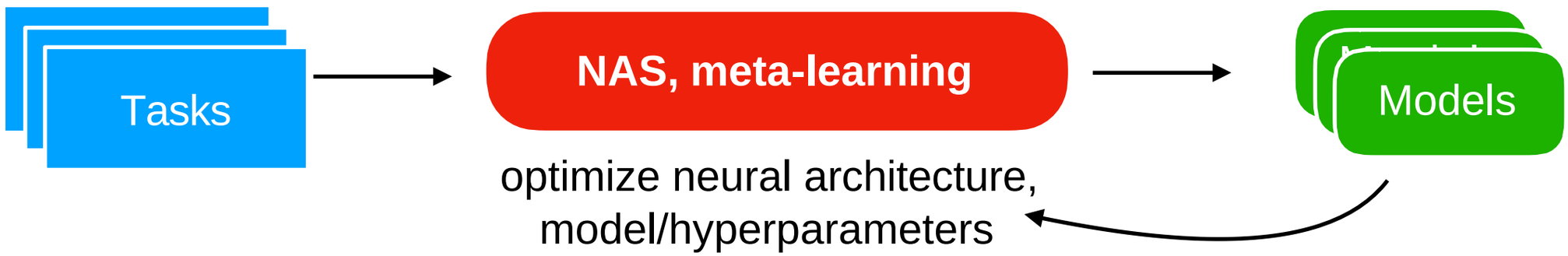


Overview

part 1 AutoML introduction, promising optimization techniques

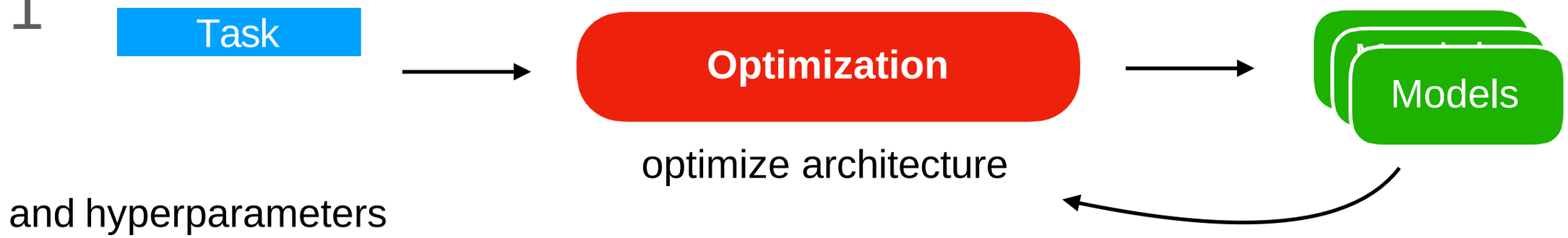
part 2 Meta-learning

part 3 Neural architecture search, meta-learning on neural nets



Overview

part 1 AutoML introduction, optimization techniques



1. Problem definition
2. (Pipeline) architecture search
3. Optimization techniques
4. Performance improvements
5. Benchmarks

AutoML Definition

Let:

$\{A^{(1)}, A^{(2)}, \ldots, A^{(n)}\}$ be a set of *algorithms* (operators)

$\Lambda^{(i)} = \lambda_1 \times \lambda_2 \times \ldots \times \lambda_m$ be the *hyperparameter space* for $A^{(i)}$

d be the space of possible *architectures* of one or more algorithms

$\Lambda_d = \Lambda^{(1)} \times \Lambda^{(2)} \times \ldots \times \Lambda^{(m)}$ its combined *configuration space*

$\boldsymbol{\lambda} \in \Lambda_d$ a specific *configuration* (of architecture and hyperparameters)

$\mathcal{L}(\boldsymbol{\lambda}, D_{train}, D_{valid})$ the loss of the model created by $\boldsymbol{\lambda}$, trained on data D_{train}, and validated on data D_{valid}

Find the configuration that minimizes the expected loss on a dataset fiJ :

$$\boldsymbol{\lambda}^* = \underset{\lambda \in \Lambda_d}{\operatorname{argmin}}\, \mathbb{E}_{(D_{train}, D_{valid}) \sim \text{fiJ}} \mathcal{L}(\boldsymbol{\lambda}, D_{train}, D_{test})$$

Types of hyperparameters

- Continuous (e.g. learning rate, SVM_C,…)
- Integer (e.g. number of hidden units, number of boosting iterations,…)
- Categorical
  - e.g. choice of algorithm (SVM, RandomForest, Neural Net,…)
  - e.g. choose of operator (Convolution, MaxPooling, DropOut,…)
  - e.g. activation function (ReLU, Leaky ReLU, tanh,…)
- Conditional
  - e.g. SVM kernel if SVM is selected, kernel width if RBF kernel is selected
  - e.g. Convolution kernel size if Convolution layer is selected

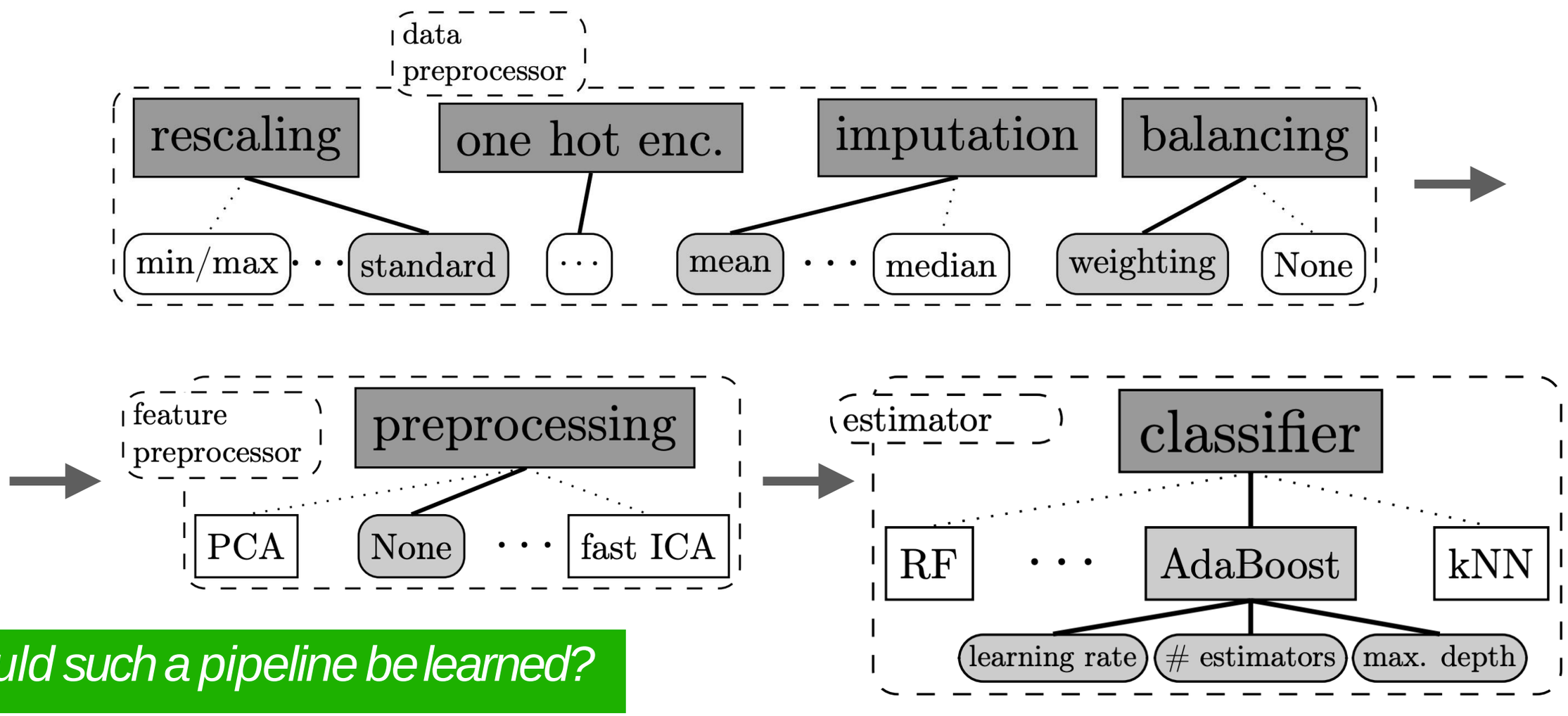
Architecture vs hyperparameters

- We can identify two subproblems:
  - **Architecture search**: search space of all possible architectures
    - ***Pipelines***: Fixed predefined pipeline, grammars, genetic programming, planning, Monte-Carlo Tree Search
    - ***Neural architectures***: See part 3
  - **Hyperparameter optimization**: optimize remaining hyperparameters
    - ***Optimization***: grid/random search, Bayesian optimization, evolution, multi-armed bandits, gradient descent (only NNs)
    - ***Meta-learning:*** See part 2
- Can be solved *consecutively*, *simultaneously* or *interleaved*
- ***Compositionality***: breaking down the learning process into smaller reusable tasks makes it easier, more transferable, more robust

Pipeline search: fixed pipelines

- Parameterize best-practice (linear) pipelines
  - Introduce conditional hyperparameters $\lambda_r \in \{A^{(1)}, \ldots, A^{(k)}, \emptyset\}$
  - Combined Algorithm Selection and Hyperparameter optimization (CASH):

$$\Lambda_d = \Lambda^{(1)} \times \Lambda^{(2)} \times \ldots \times \Lambda^{(m)} \times \lambda_r$$



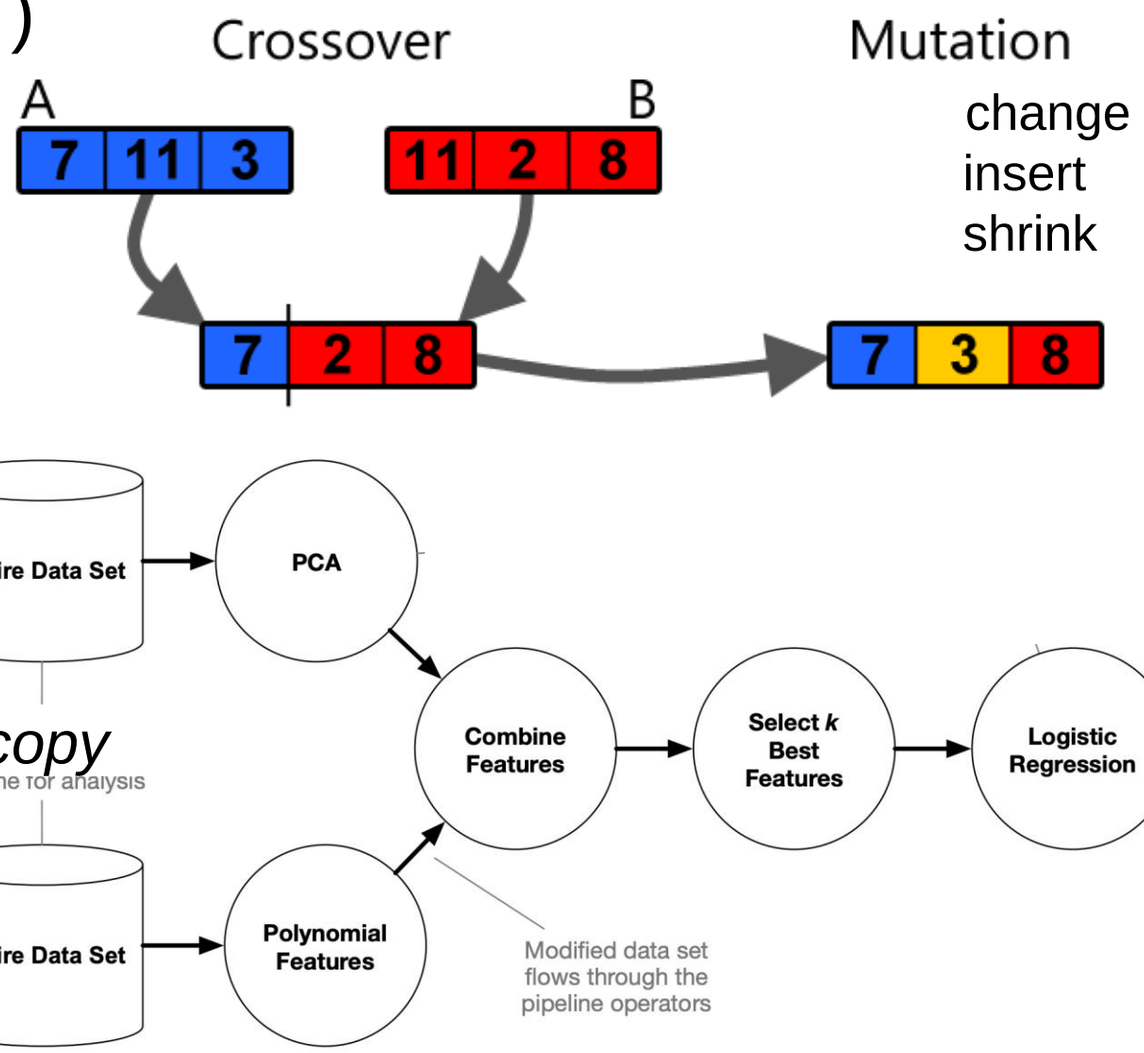
Could such a pipeline be learned?

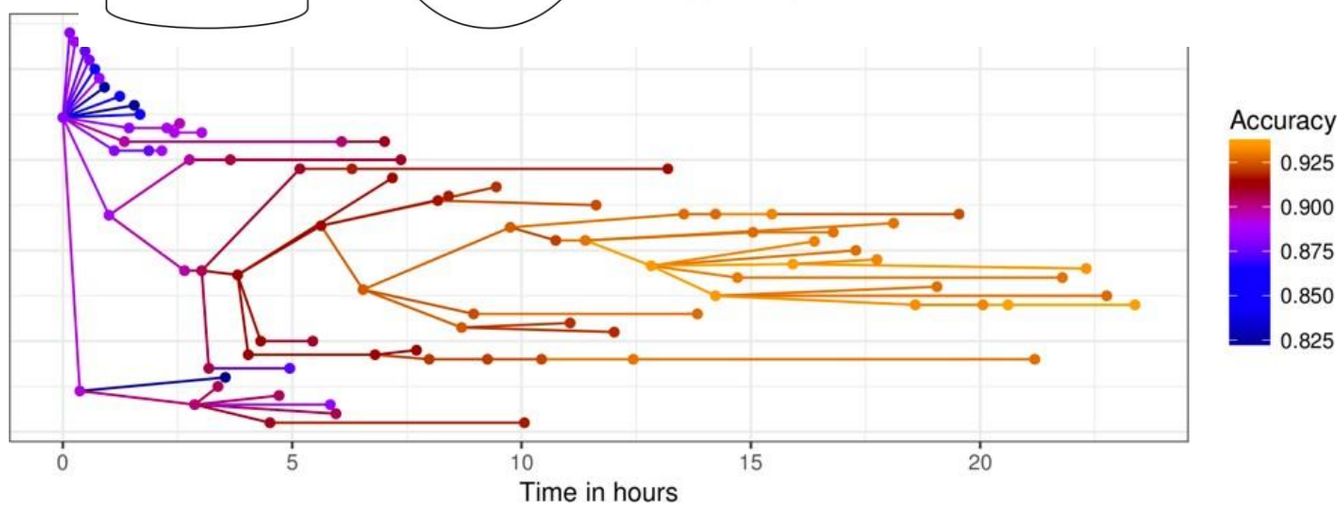
[Olson, Moore 2016, 2019]

Pipeline search: genetic programming

- Tree-based pipeline optimization (TPOT)
  - Start with random pipelines, best of every generation will cross-over or mutate
  - GP primitives include data copy and feature joins: trees
  - Multi-objective optimization: accurate but short
  - Easy to parallelize
- Asynchronous evolution (GAMA)

[Gijsbers, Vanschoren 2019]





Pipeline search: TPOT demo

```
from tpot import TPOTClassifier
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split

digits = load_digits()
X_train, X_test, y_train, y_test = train_test_split(digits.data, digits.target,
                                                    train_size=0.75, test_size=0.25)

tpot = TPOTClassifier(generations=5, population_size=50, verbosity=2, n_jobs=-1)
tpot.fit(X_train, y_train)
```

```
Optimization Progress:   0%|          | 0/300 [00:00<?, ?pipeline/s]
```

```
print(tpot.score(X_test, y_test))
```

Pipeline search: genetic programming

- Grammar-based genetic programming (RECIPE) *[de Sa et al. 2017]*
  - Encodes background knowledge, avoids invalid pipelines
  - Cross-over and mutation respect the grammar

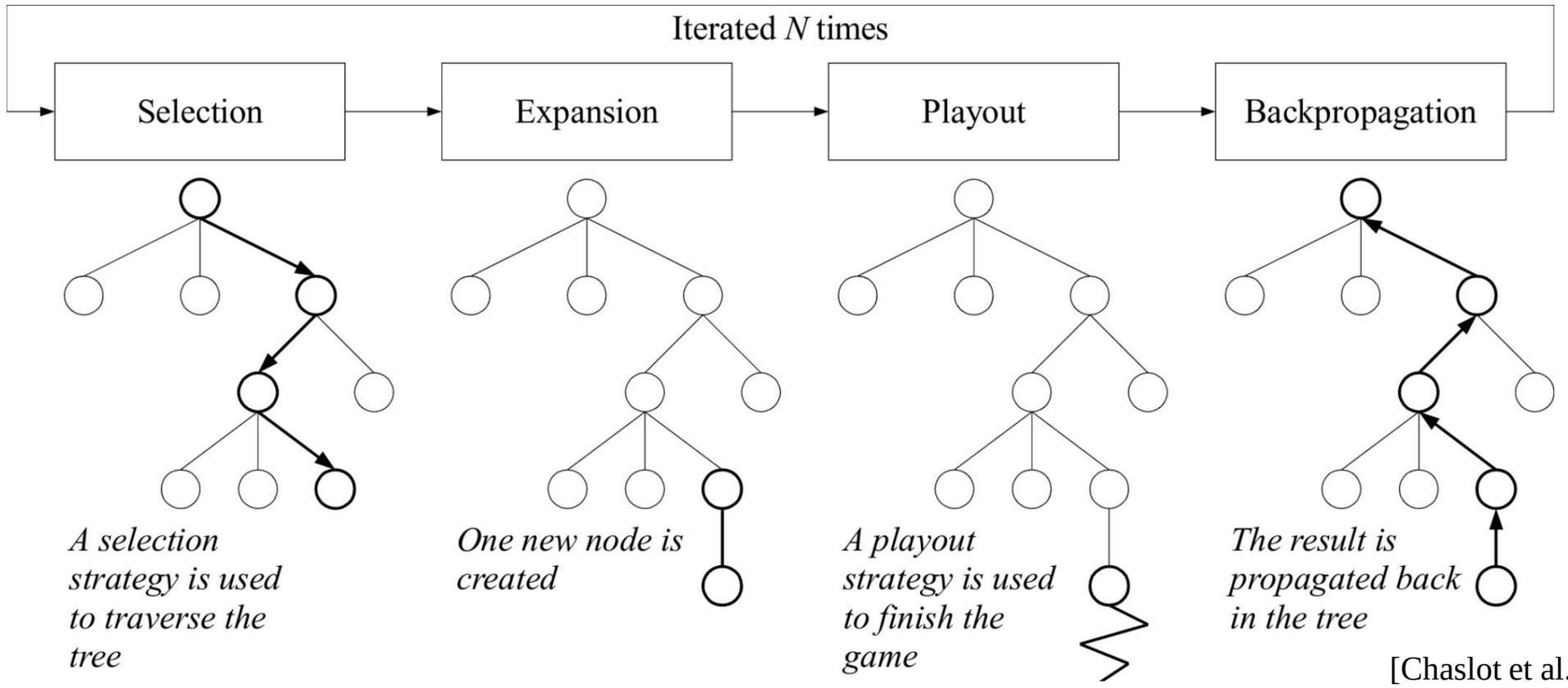
production rule *optional* *non-terminal*

```
<Start> ::= [<Pre-processing>] <Algorithm>
<Pre-processing> ::= [<Imputation>] <DimensionalityDefinition>
<Imputation> ::= Mean | Median | Max
<DimensionalityDefinition> ::= <FeatureSelection>   [<FeatureConstruction>]
                               [<FeatureSelection>] <FeatureConstruction>
<FeatureSelection> ::= <Supervised> | <Unsupervised>
<Supervised> ::= SelectKBest <K> <score> | VarianceThreshold | [...]
<score> ::= f-classification | chi2
<K> ::= 1 | 2 | 3 | [...] | NumberOfFeatures - 1
<perc> ::= 1 | 2 | 3 | [...] | 99
<Unsupervised> ::= PCA | FeatureAgglomeration <affinity> | [...]
<affinity> ::= Euclidian | L1 | L2 | Manhattan | Cosine
<FeatureConstruction> ::= PolynomialFeatures
<Algorithm> ::=  <NaiveBayes> | <Trees> | [...]
<NaiveBayes> ::= GaussianNB | MultinomialNB | BernoulliNB
```

terminal

Pipeline search: Monte Carlo Tree Search

- Use MCTS to search for optimal pipelines
- Optimize the structure and hyperparameters simultaneously by building a surrogate model to predict configuration performance
  - Bayesian surrogate model: MOSAIC *[Rakotoarison et al. 2019]*
  - Neural network: AlphaD3M *[Drori et al. 2018]*



[Chaslot et al., 2008]

AlphaD3M Example

NYU

VIDA VISUALIZATION IMAGING AND DATA ANALYSIS CENTER

In [2]:
```
from d3m_interface import AutoML
```

Generating pipelines using AlphaD3M

In [4]:
```
train_dataset_path = '/home/ubuntu/datasets/seed_datasets_current/JIDO_SOHR_Articles_1061/TRAIN'
test_dataset_path = '/home/ubuntu/datasets/seed_datasets_current/JIDO_SOHR_Articles_1061/TEST'
score_dataset_path = '/home/ubuntu/datasets/seed_datasets_current/JIDO_SOHR_Articles_1061/SCORE'
output_path = '/home/ubuntu/output/'
tmp_path = '/home/ubuntu/tmp'
```

In [5]:
```
automl = AutoML(output_path, 'AlphaD3M')
pipelines = automl.search_pipelines(train_dataset_path, time_bound=5)
```

```
automl_red.leaderboard
```

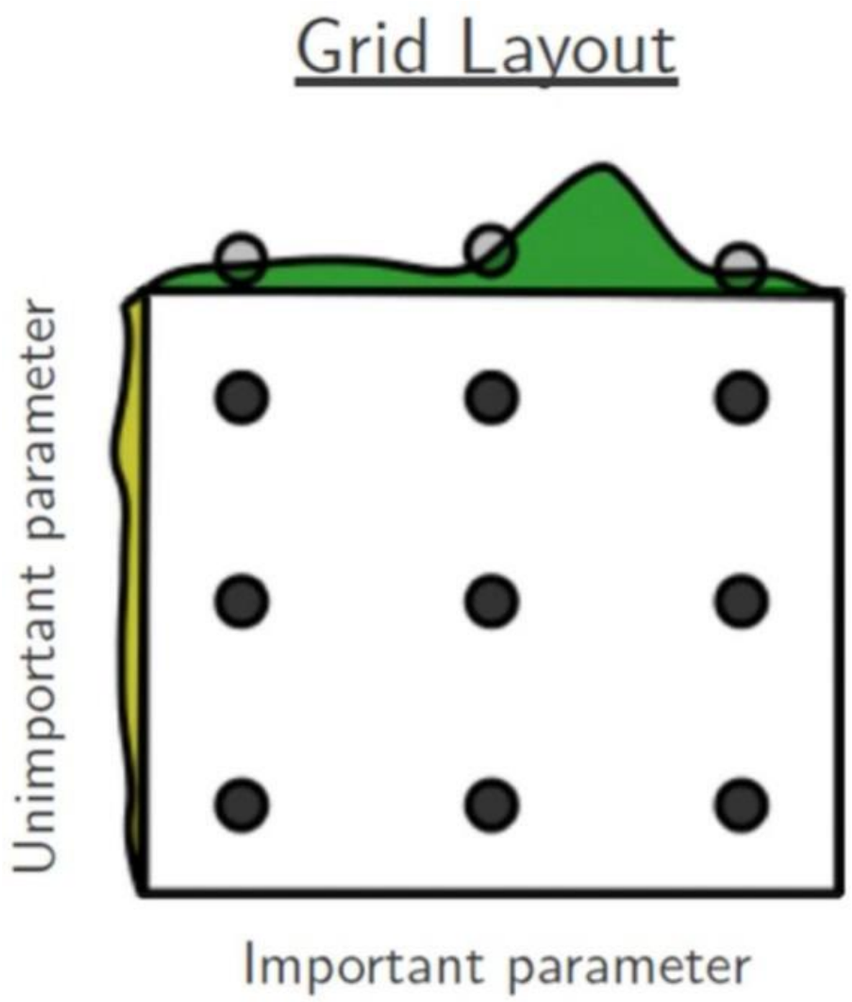
	ranking	id	summary	f1
0	1	c53f5b6e-872f-4ae0-9366-6b7cff85b5de	text_reader.common, add_semantic_types.common,...	0.83410
1	2	75f86ebc-37f1-446a-9baa-24ef474fd215	add_semantic_types.common, imputer.sklearn, en...	0.05021
2	3	bbfda3b0-6086-4a8a-8e9f-708ed586edef	add_semantic_types.common, imputer.sklearn, en...	0.04255
3	4	bb02b1a1-93fa-4df4-98d5-ee5574c65917	add_semantic_types.common, imputer.sklearn, en...	0.02542
4	5	343507f5-a408-4b8e-a1cc-91cde7135623	add_semantic_types.common, imputer.sklearn, en...	0.00000
5	6	f37da102-0663-4e9d-86b0-9e1de8488434	add_semantic_types.common, imputer.sklearn, en...	0.00000

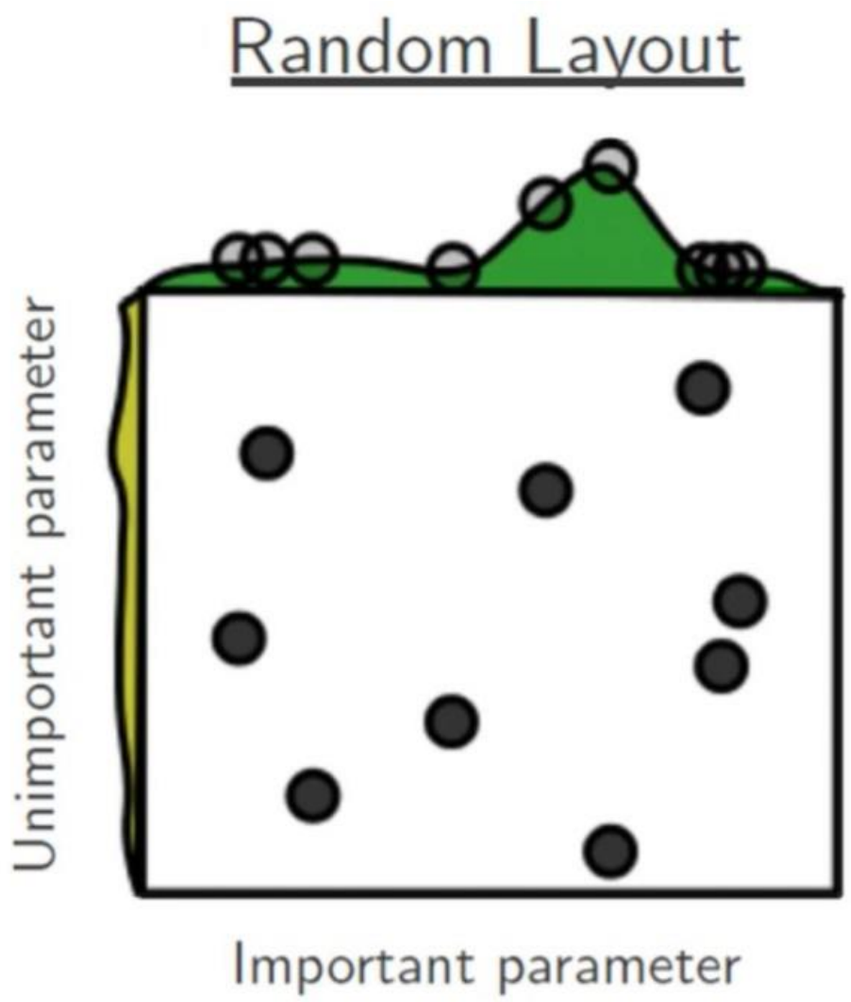
Hyperparameter optimization (HPO)

Black box optimization: expensive for machine learning.
Sample efficiency is important!

HPO: grid and random search

- Random search handles unimportant dimensions be4er
- Easily parallelizable, but uninformed (no learning)



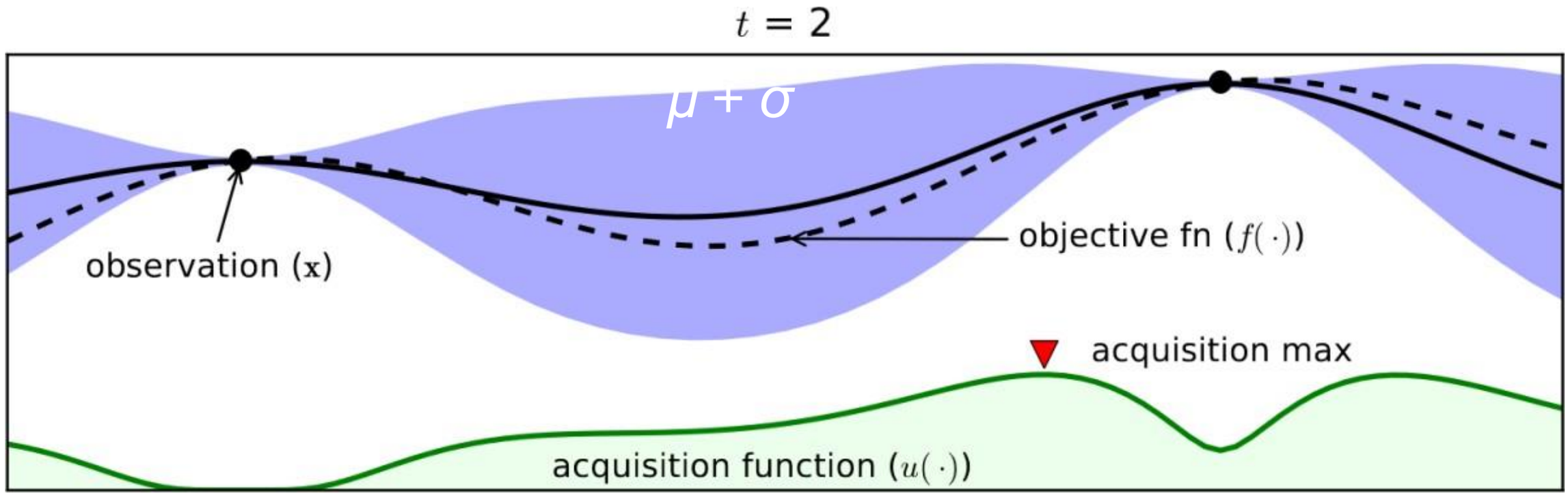


Bergstra & Bengio, JMLR 2012

Mockus [1974]

HPO: Bayesian Optimization

- Start with a few (random) hyperparameter configurations
- Build a ***surrogate model*** to predict how well other configurations will work: mean and standard deviation (blue band)
  - Any probabilistic regression model: e.g. Gaussian processes
- To avoid a greedy search, use an ***acquisition function*** to trade off exploration and exploitation, e.g. Expected Improvement (EI)
- Sample for the best configuration under that function

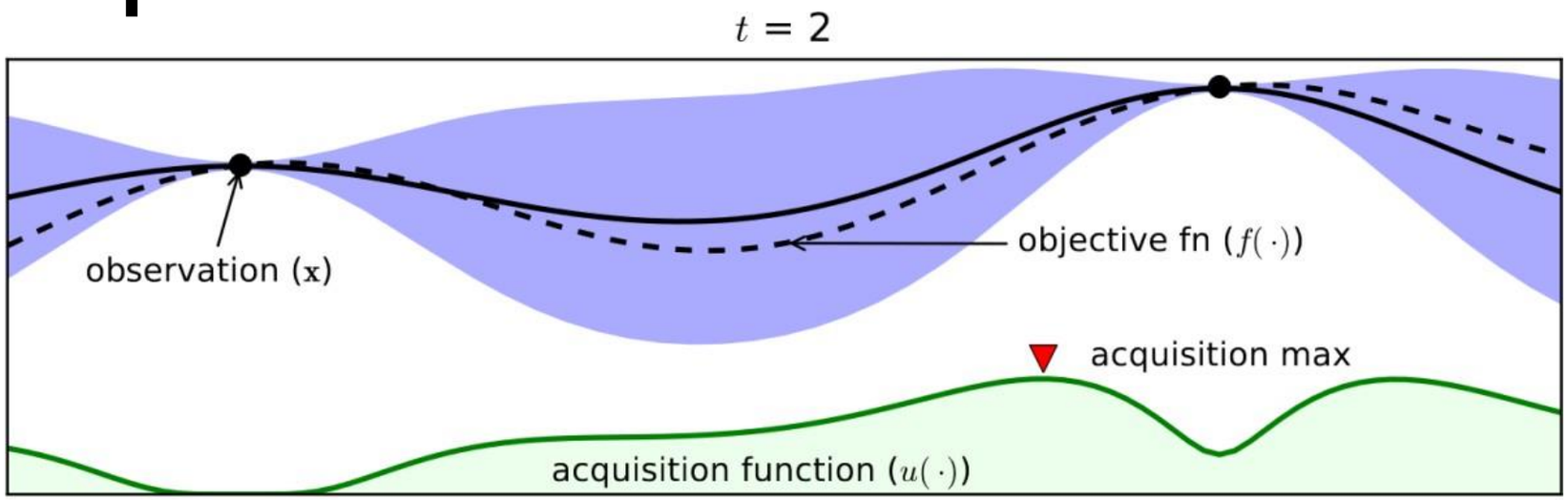


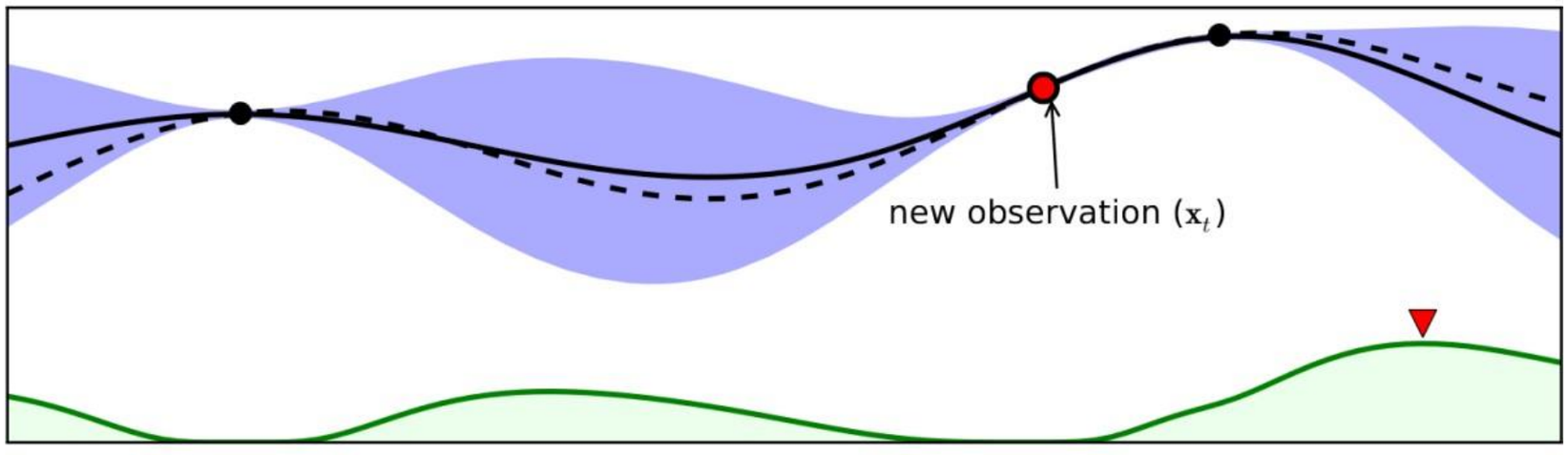
HPO: Bayesian Optimization

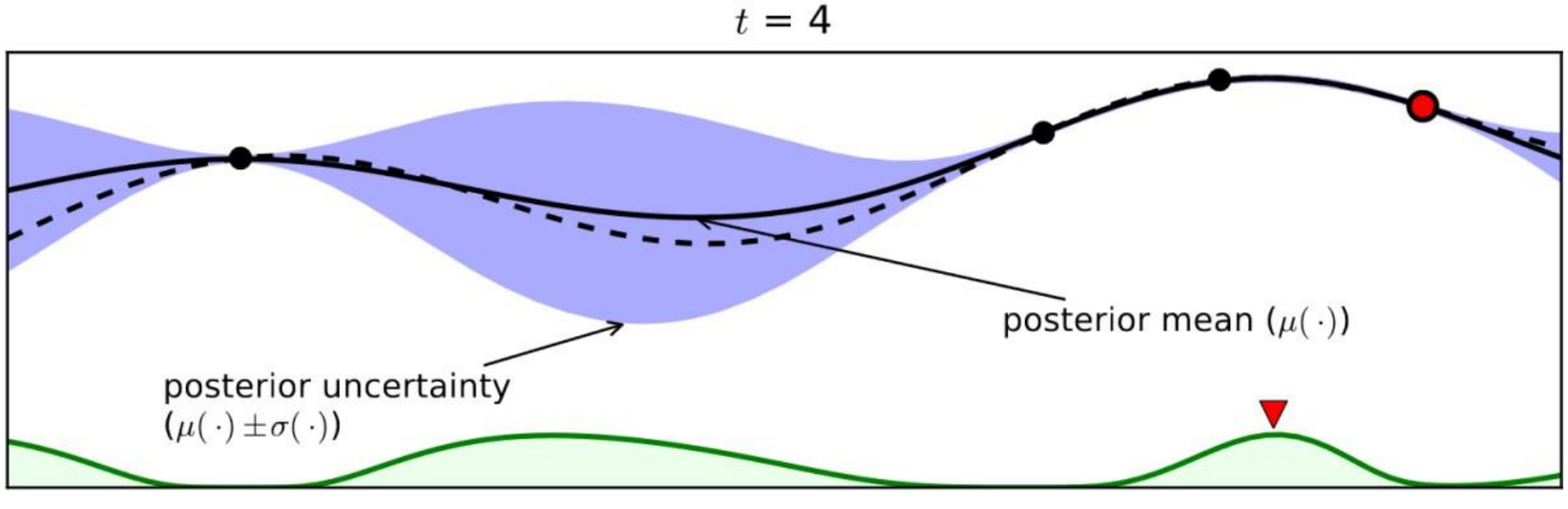
- Repeat
- Stopping criterion:
  - Fixed budget (time, evaluations)
  - Min. distance between configs
  - Threshold for acquisition function
  - Still overfits easily
- Convergence results

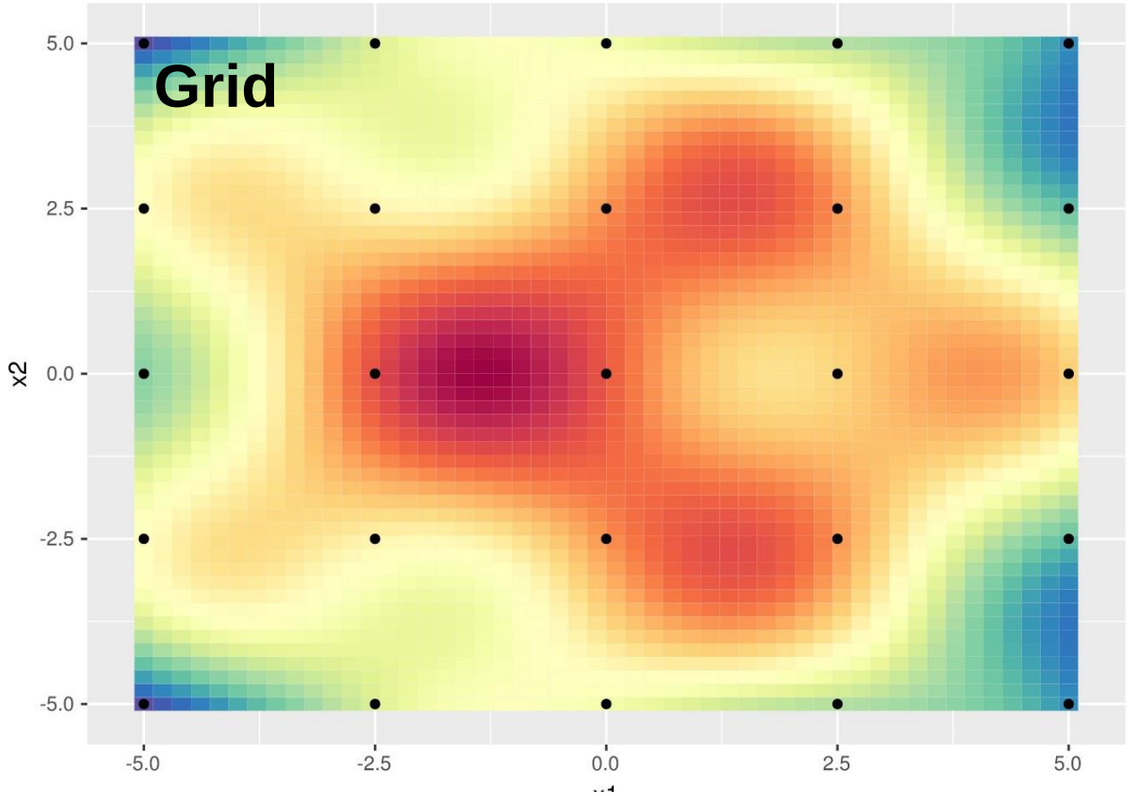
[Srinivas et al. 2010](), [Freitas et al. 2012](), [Kawaguchi et al. 2016]()

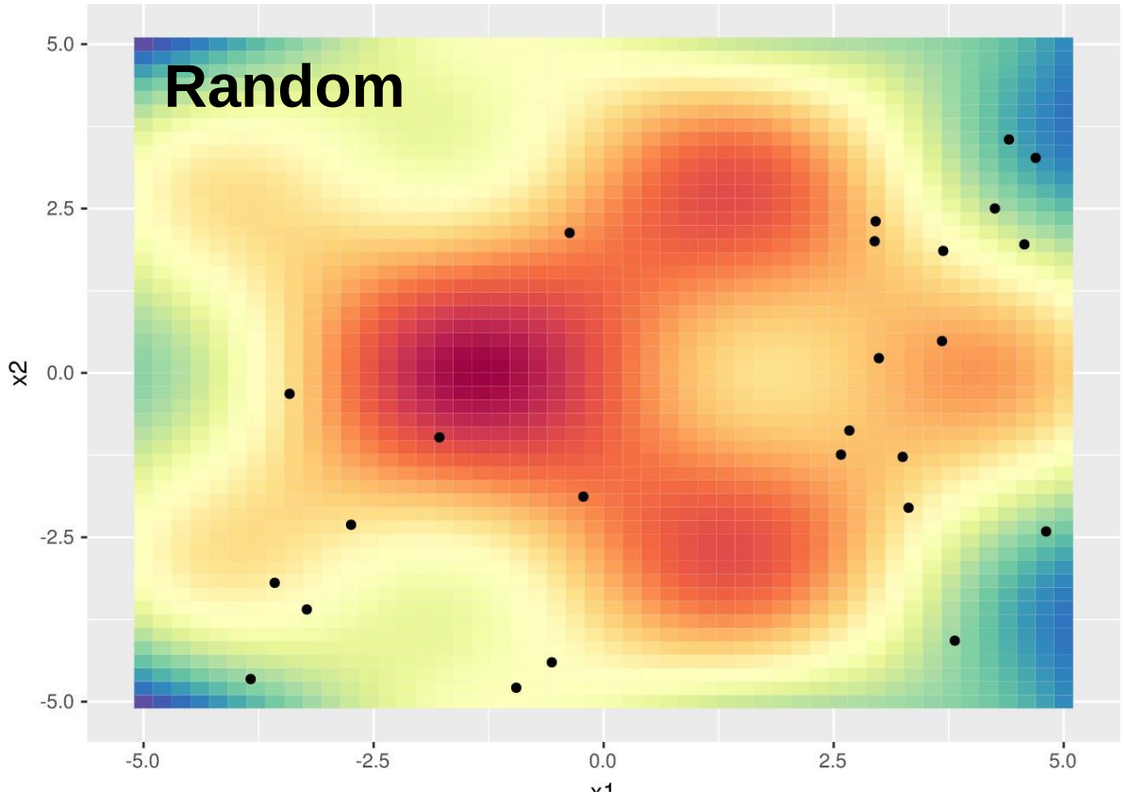
- Good for non-convex, noisy objectives
- Used in AlphaGo

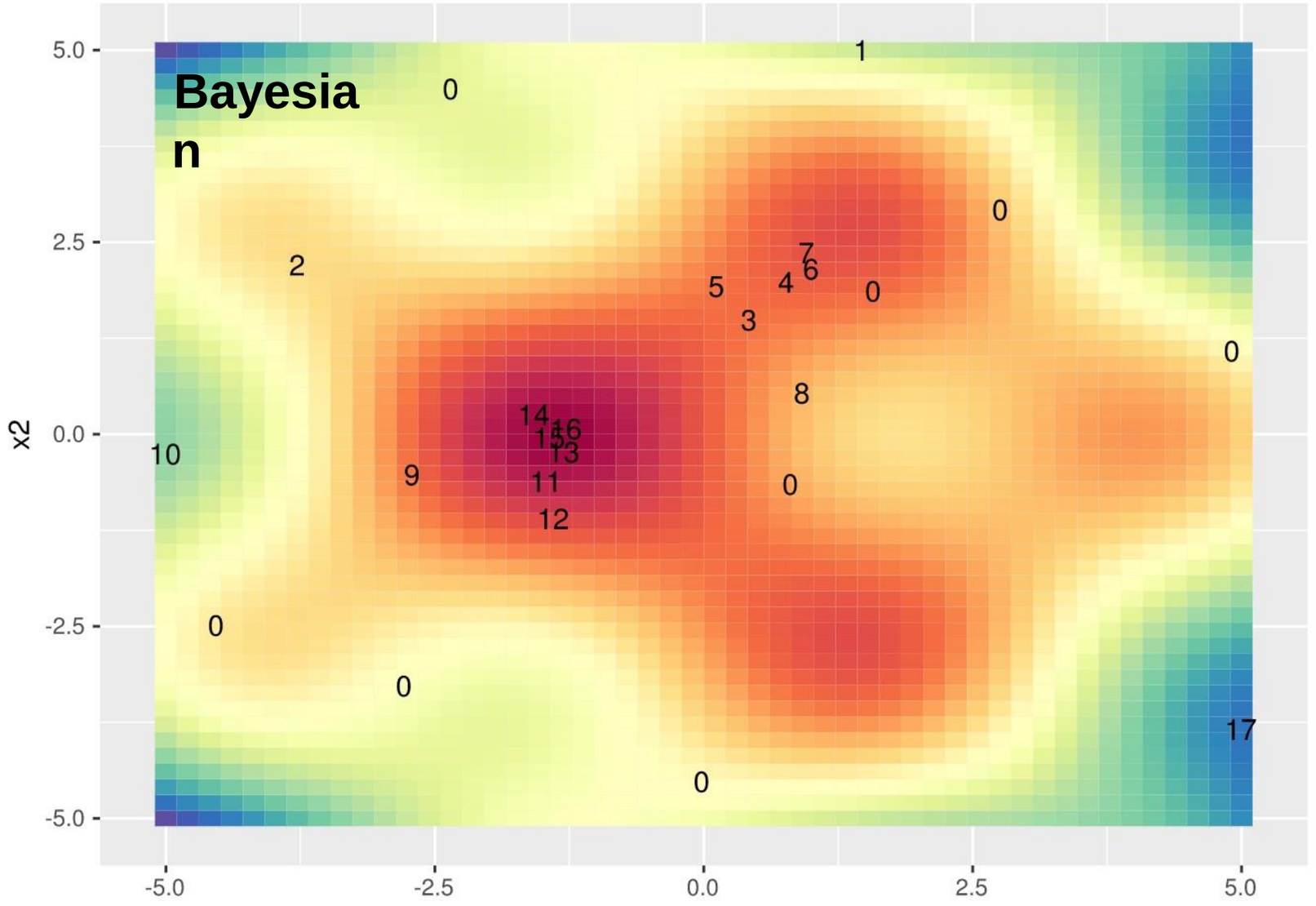







Grid
5.0
2.5
0.0
-2.5
-5.0
x2
-5.0
-2.5
0.0
2.5
5.0


Random
5.0
2.5
0.0
-2.5
-5.0
x2
-5.0
-2.5
0.0
2.5
5.0


Bayesian
5.0
2.5
0.0
-2.5
-5.0
x2
-5.0
-2.5
0.0
2.5
5.0
1
0
0
2
7
6
5
4
0
3
0
8
14
16
15
13
10
9
11
0
12
0
0
17
0

Meta-learning

Meta-learning can drastically speed up the architecture and hyperparameter search, in combination with the optimization techniques we just discussed

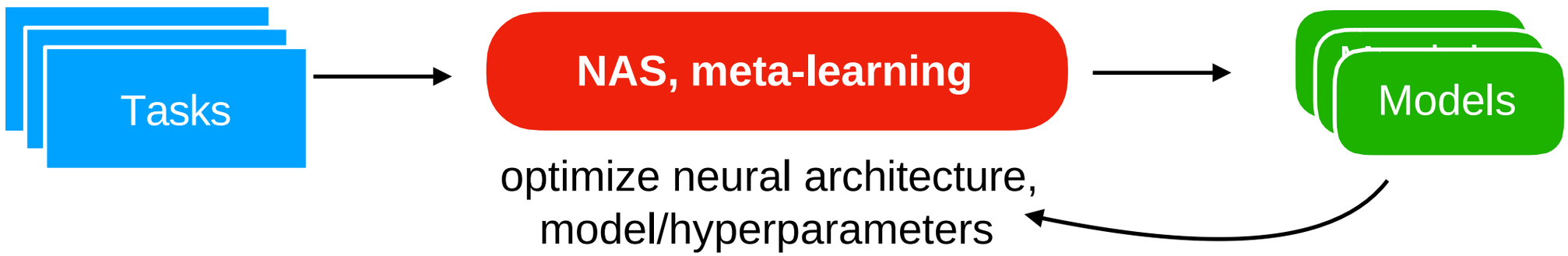
- Learn which hyperparameters are really important
- Learn which hyperparameter values should be tried first
- Learn which architectures will most likely work
- Learn how to clean and annotate data
- Learn which feature representations to use
- …

Overview

part 1 AutoML introduction, promising optimization techniques

part 2 Meta-learning

part 3 Neural architecture search, meta-learning on neural nets



Learning is a never-ending process

Humans don’t learn from scratch

Learning is a never-ending process

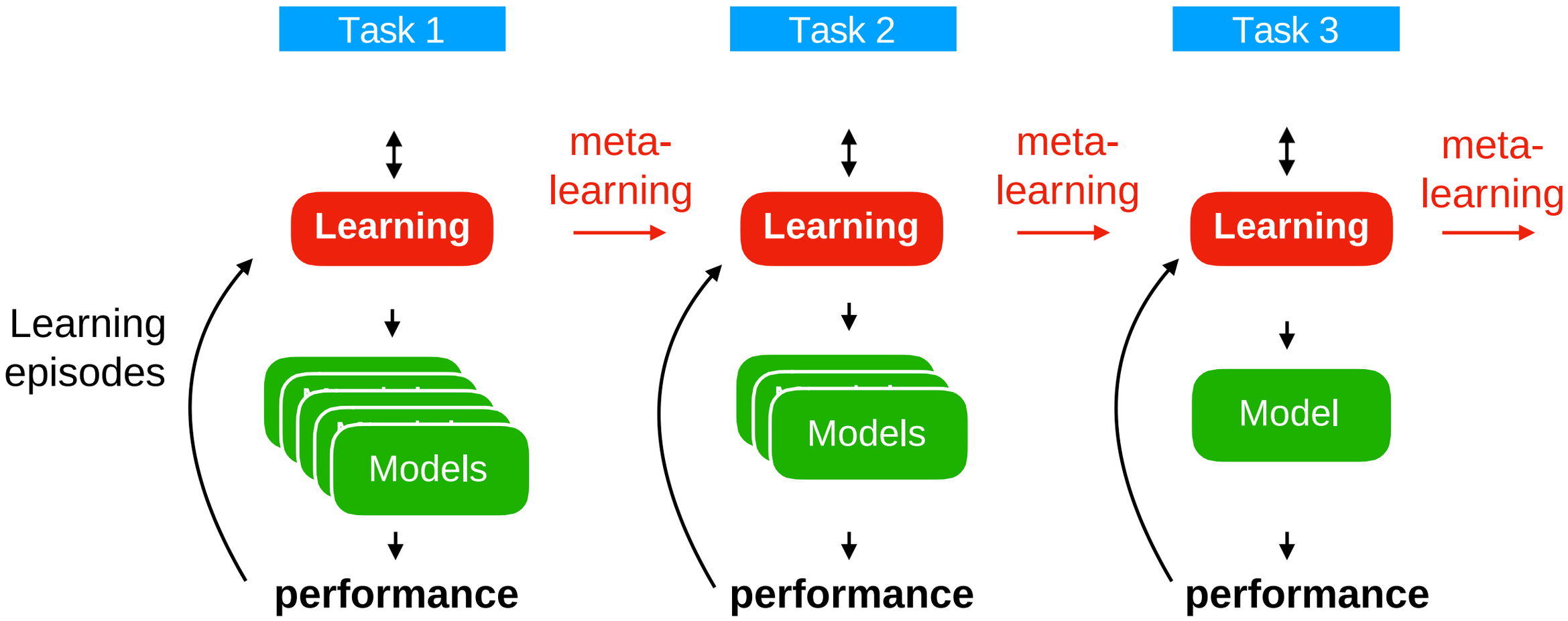
Learning humans also seek/create related tasks

E.g. find many similar puzzles, solve them in different ways,…

Learning is a never-ending process

Humans learn *across* tasks

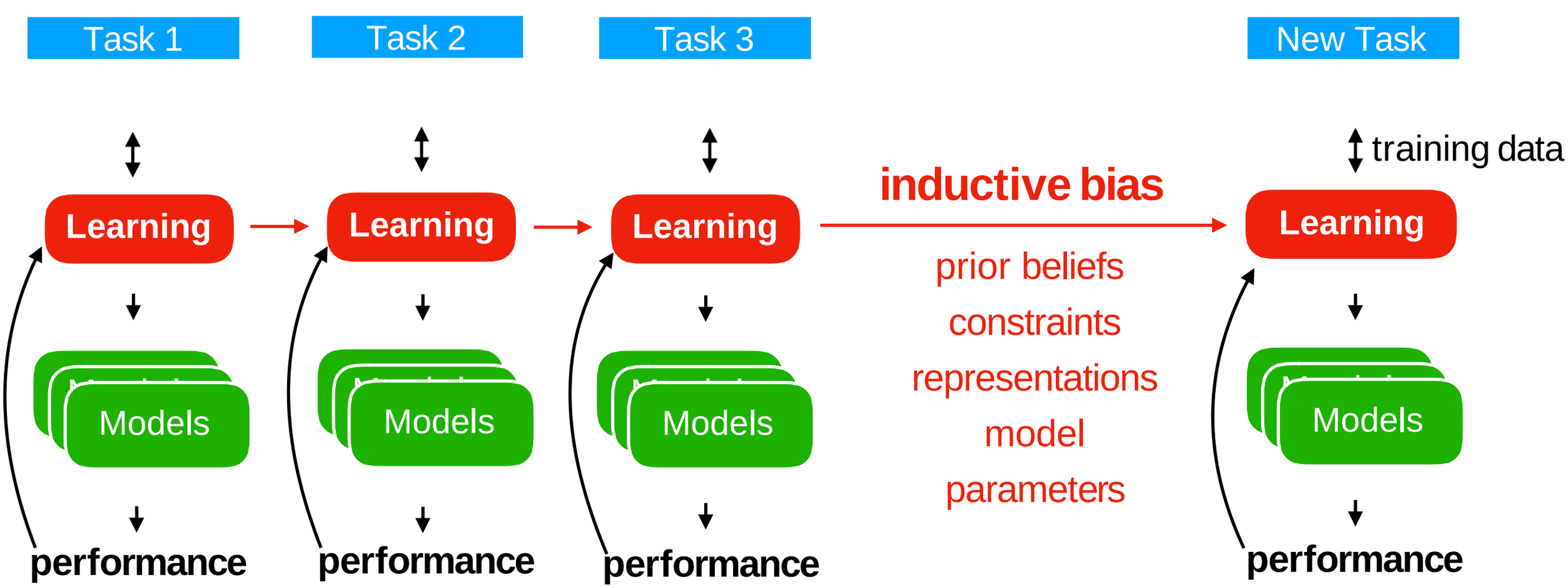
Why? Requires less trial-and-error, less data



Learning to learn

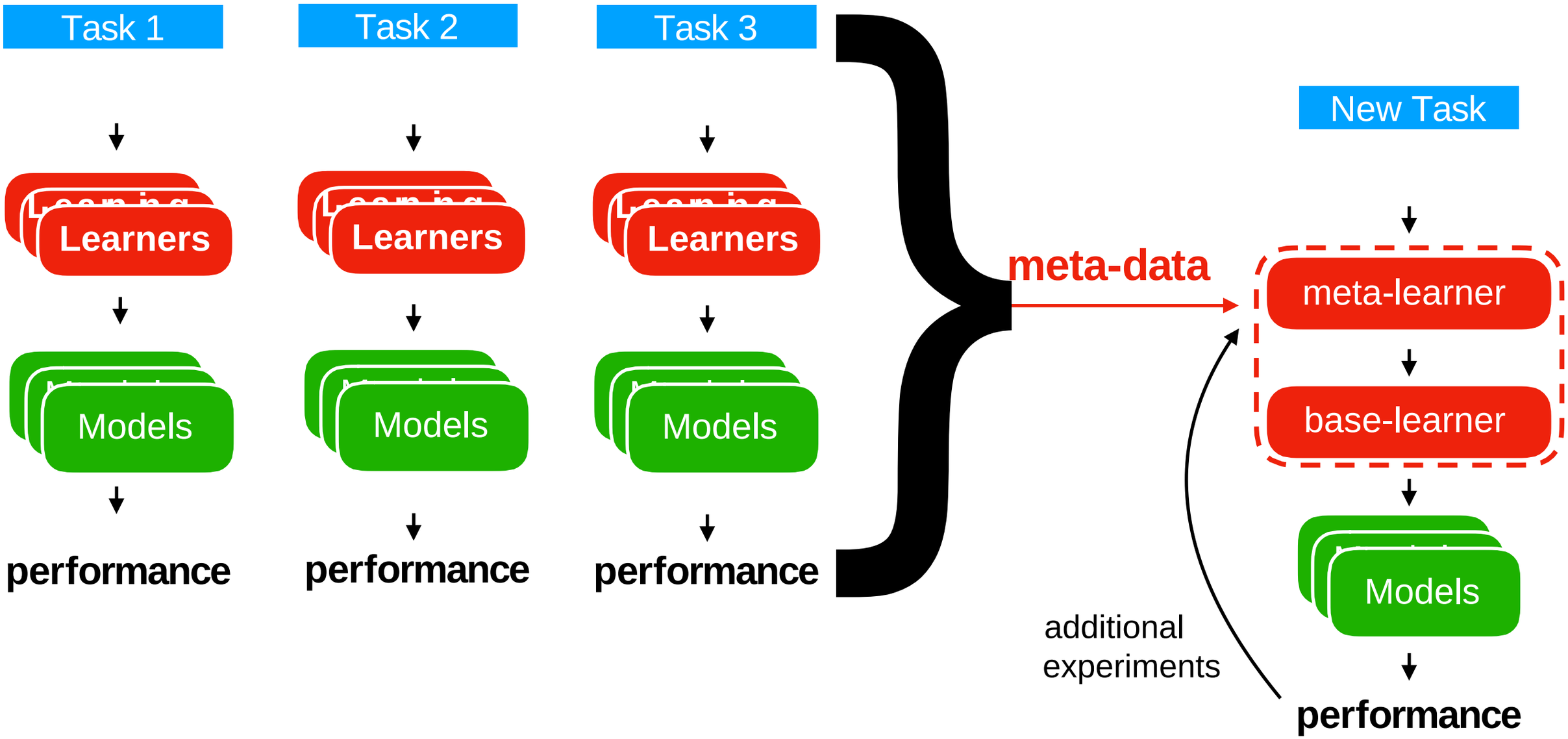
If prior tasks are *similar*, we can ***transfer*** prior knowledge to new tasks

Inductive bias: assumptions added to the training data to learn effectively



Meta-learning

Meta-learner *learns* a (base-)learning algorithm, based on *meta-data*



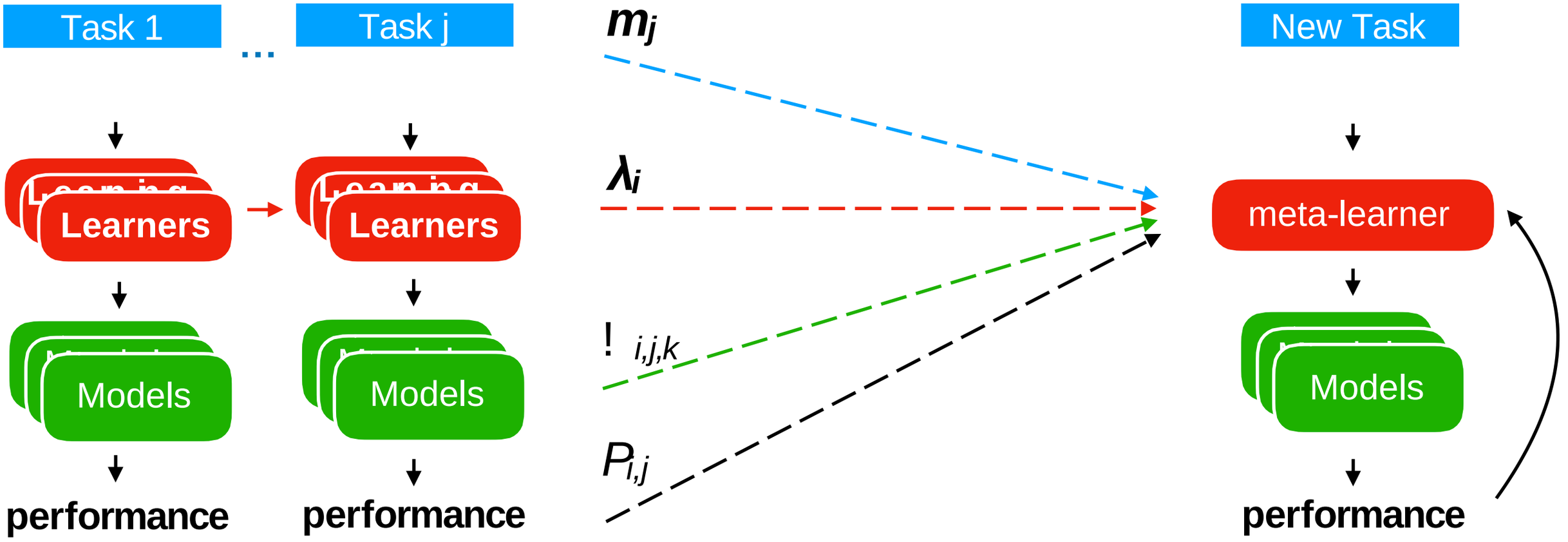
Meta-data

$\boldsymbol{m_j}$ Description of task j (meta-features)

$\boldsymbol{\lambda_i}$ Configuration i (architecture + hyperparameters)

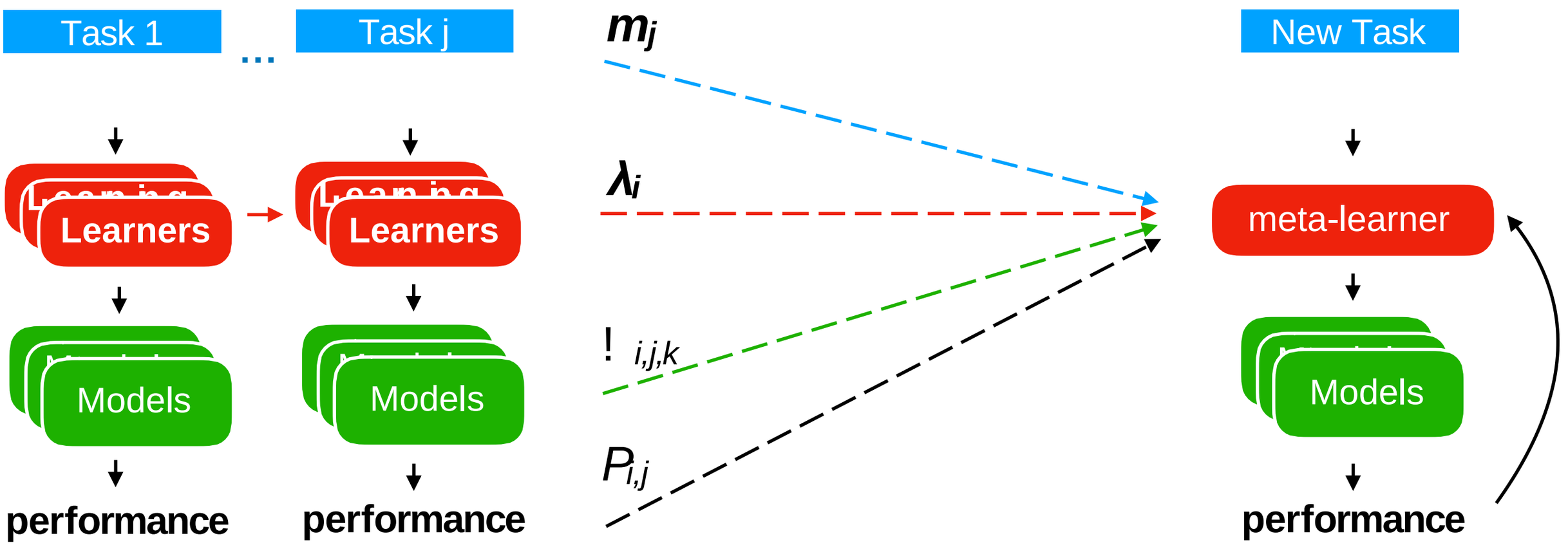
$!_{i,j,k}$ Model parameters (e.g. weights)

$P_{i,j}$ Performance estimate of model built with λ_i *on task j*



How?

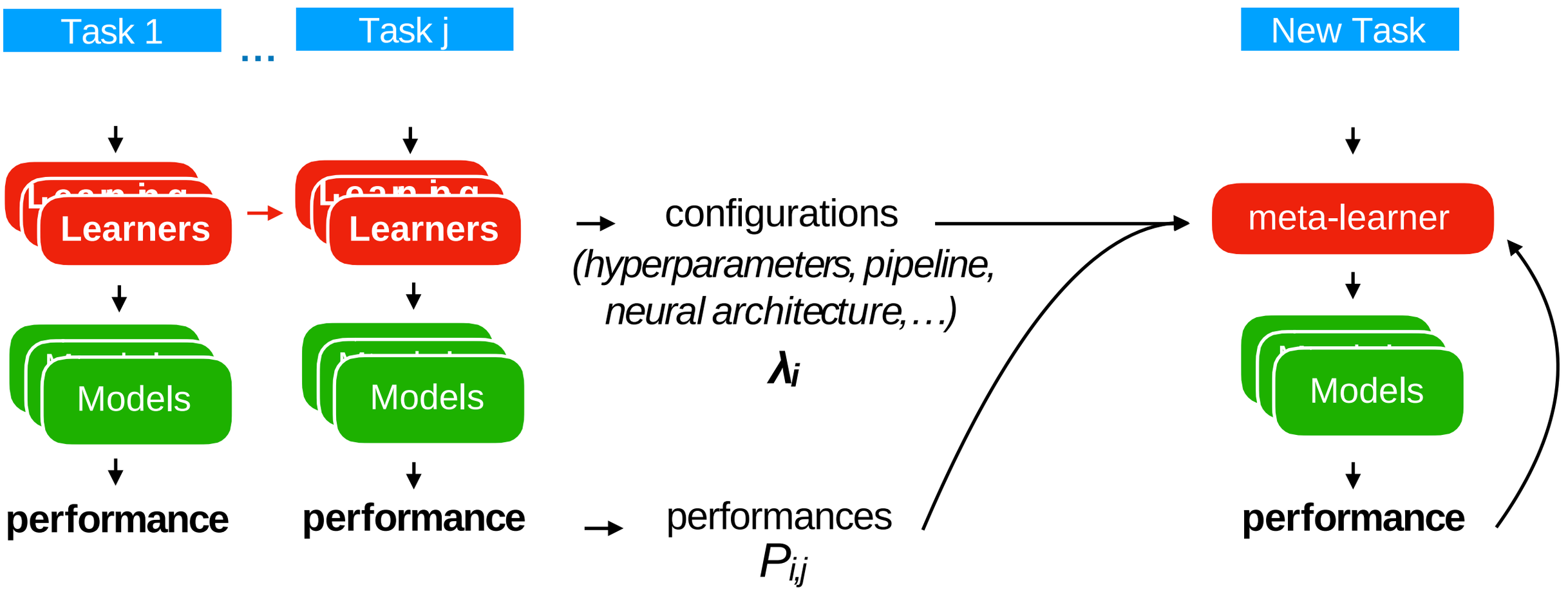
1. Learn from observing learning algorithms (for *any* task) $\boldsymbol{\lambda_i}$ $P_{i,j}$
2. Learn what may likely work (for *somewhat similar* tasks) $\boldsymbol{m_j}$ $\boldsymbol{\lambda_i}$ $P_{i,j}$
3. Learn from previous models (for *very similar* tasks) ! $_{i,j,k}$ $\boldsymbol{(m_j)}$ $\boldsymbol{\lambda_i}$ $P_{i,j}$



1. Learn from observing learning algorithms

Define, store and use meta-data:

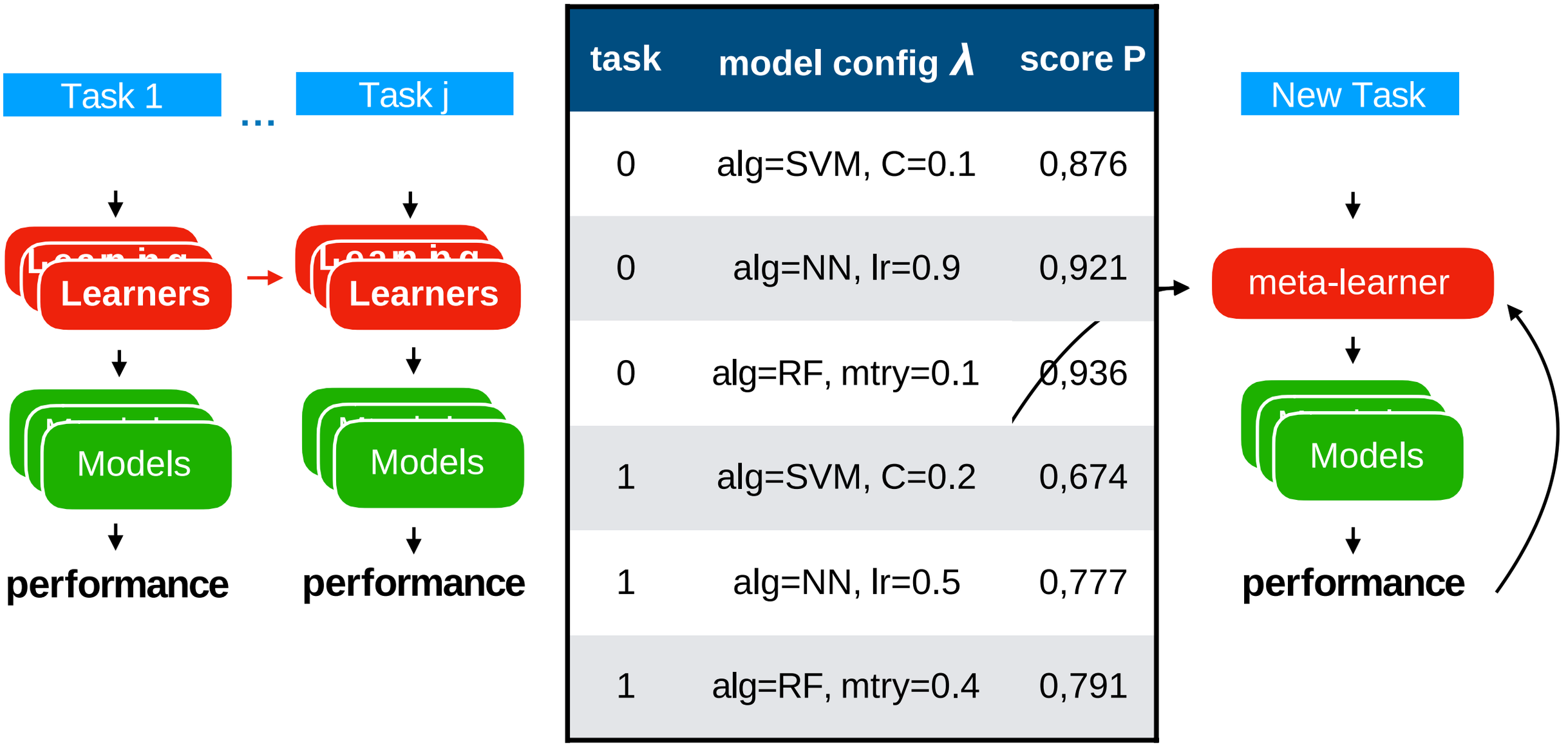
- ***configurations***: settings that uniquely define the model
- ***performance*** (e.g. accuracy) on specific tasks



1. Learn from observing learning algorithms

Define, store and use meta-data:

- ***configurations***: settings that uniquely define the model
- ***performance*** (e.g. accuracy) on specific tasks

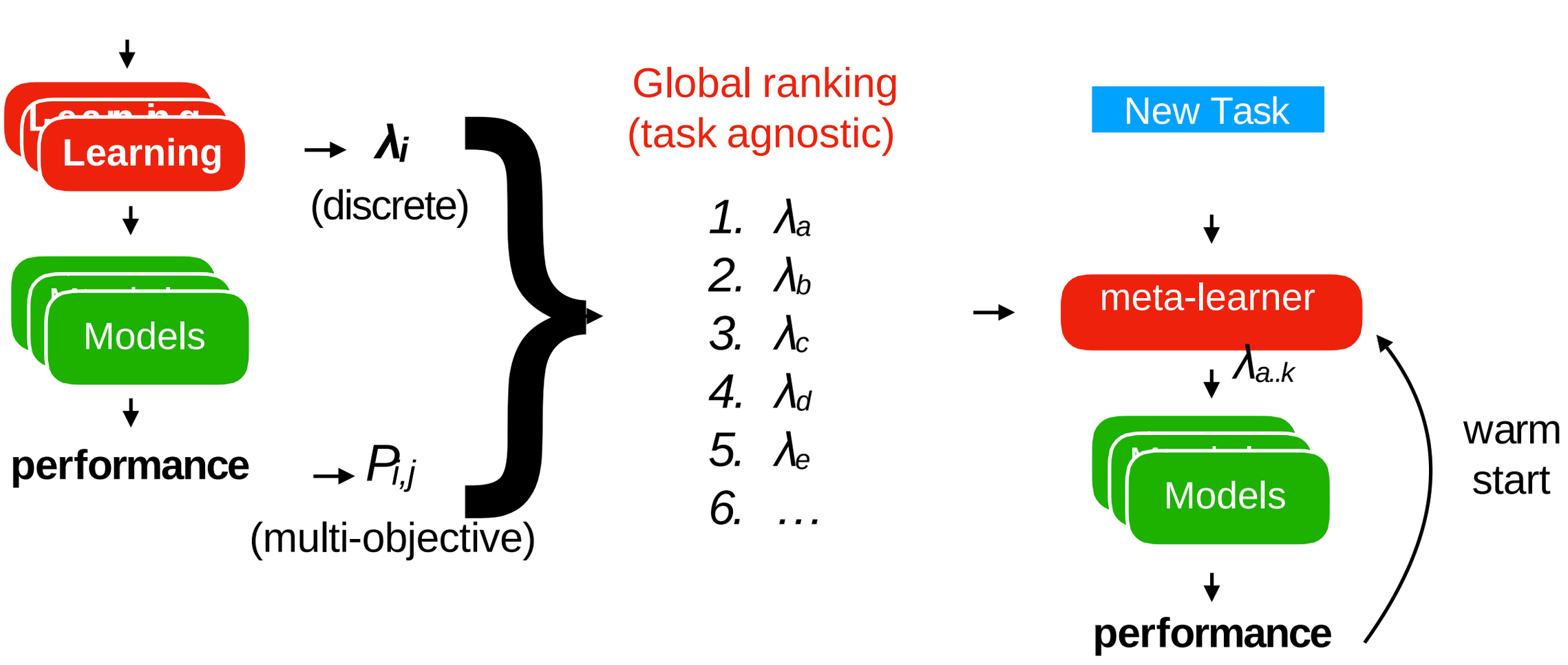


[Leite et al. 2012]

[Abdulrahman et al. 2018]

Rankings

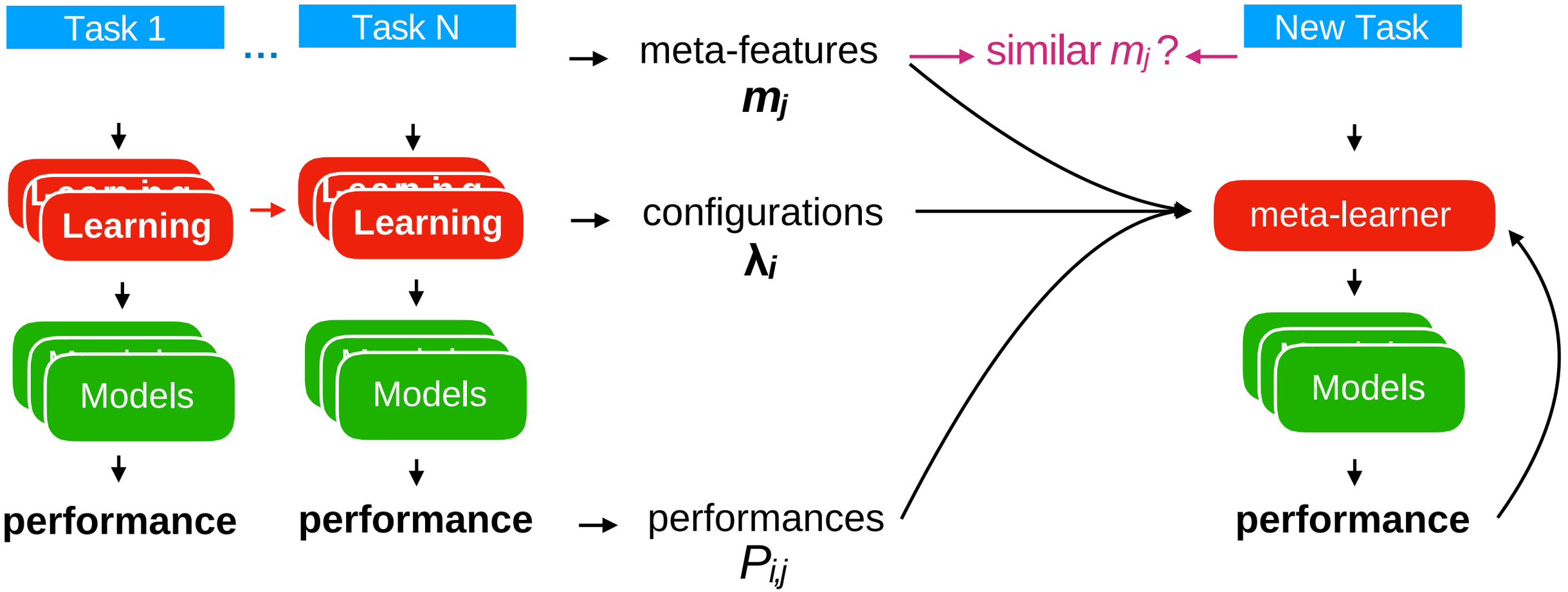
- Build a *global (multi-objective) ranking,* recommend the top-K
- Can be used as a *warm start* for optimization techniques
  - E.g. Bayesian optimization, evolutionary techniques,…



2. Learn what may likely work (for *partially similar* tasks)

Meta-features: measurable properties of the tasks

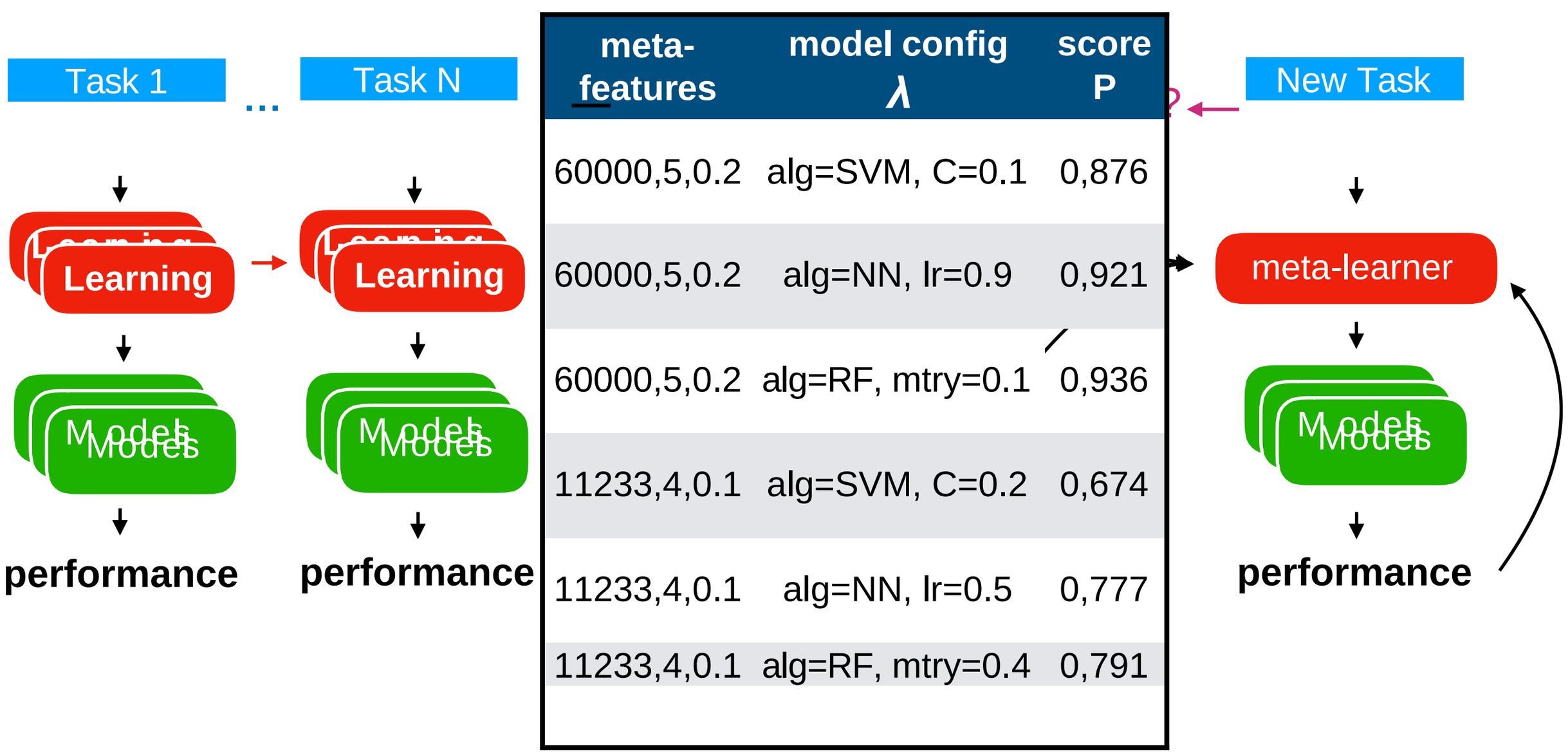
(number of instances and features, class imbalance, feature skewness,…)



2. Learn what may likely work (for *partially similar* tasks)

Meta-features: measurable properties of the tasks

(number of instances and features, class imbalance, feature skewness,…)



[1] Vanschoren 2018

[2] Rivolli et al 2019

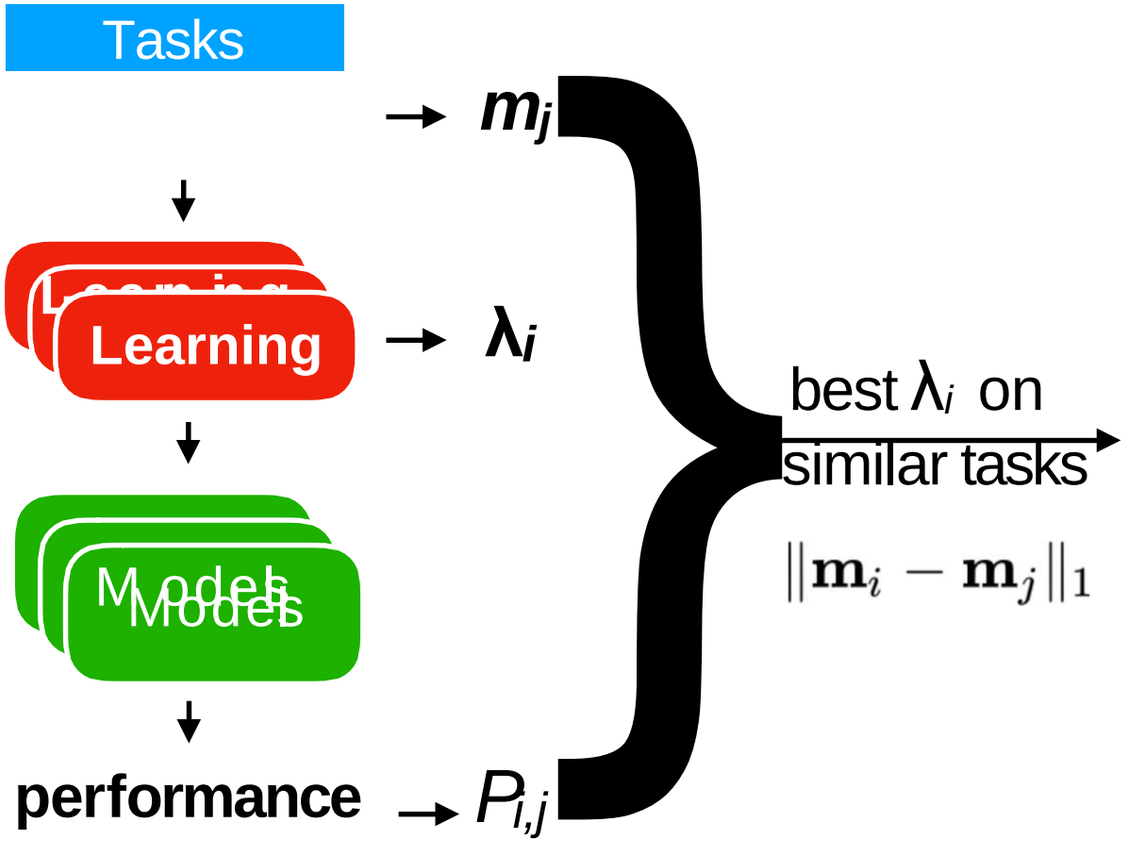
Meta-features

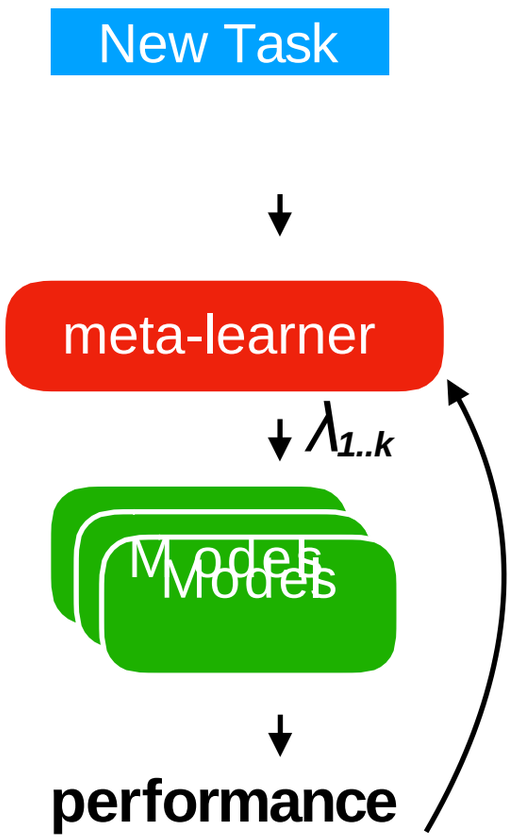
- **Hand-crafted (interpretable) meta-features**[1]
  - **Number of** instances, features, classes, missing values, outliers,…
  - **Statistical:** skewness, kurtosis, correlation, covariance, sparsity, variance,…
  - **Information-theoretic**: class entropy, mutual information, noise-signal ratio,…
  - **Model-based**: properties of simple models trained on the task
  - **Landmarkers**: performance of fast algorithms trained on the task
  - Domain specific task properties
- Optimized (recommended) representations exist [2]

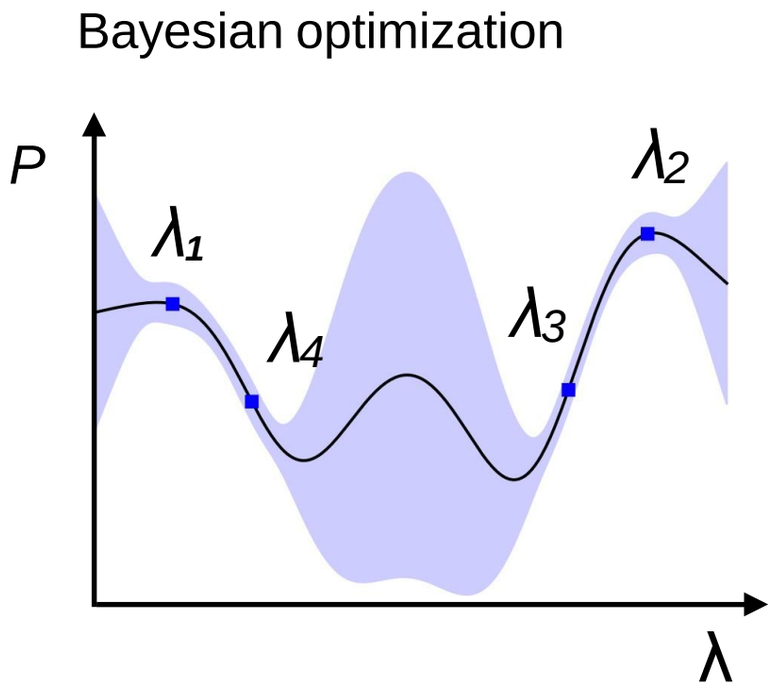
[Feurer et al. 2015]

Warm-starting from similar tasks

- Find k most similar tasks, warm-start search with best λ_i
  - Auto-sklearn: Bayesian optimization (SMAC)
    - Meta-learning yield better models, faster
    - Winner of AutoML Challenges







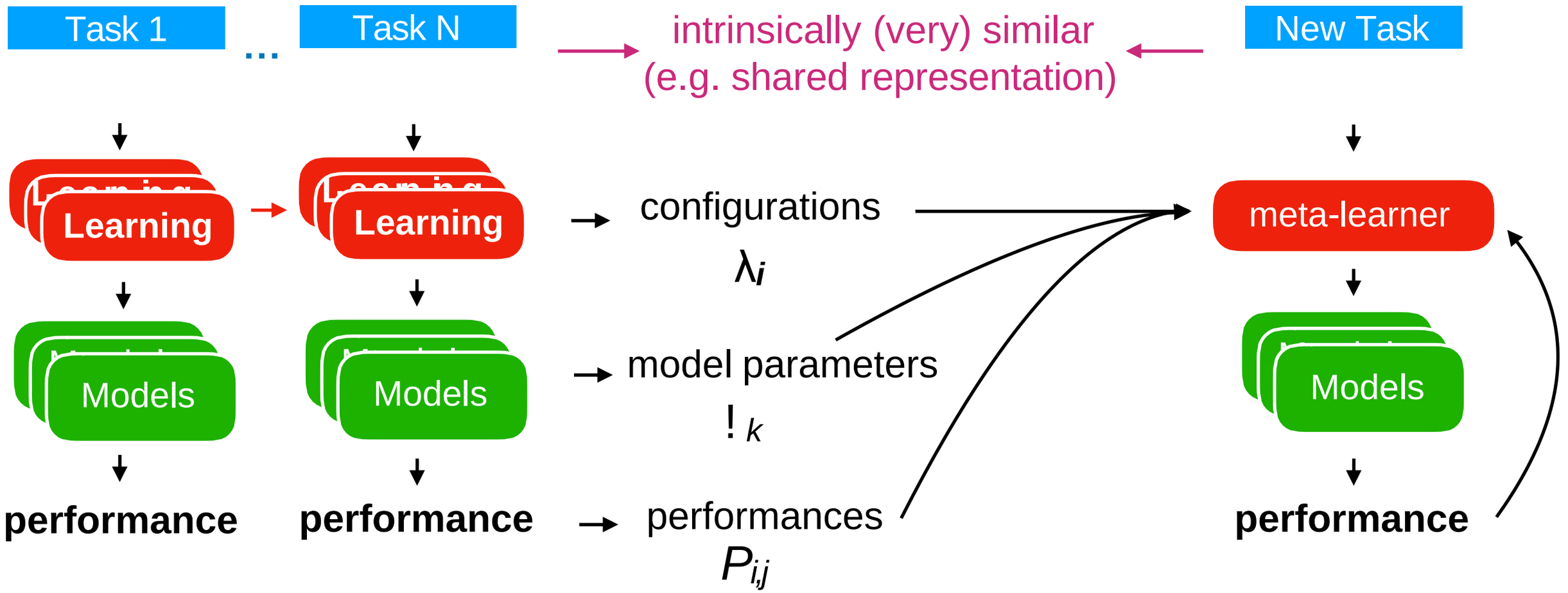
Auto-Sklearn Example

```
[ ]  import autosklearn.classification
     import sklearn.model_selection
     import sklearn.datasets
     import sklearn.metrics

     X, y = sklearn.datasets.load_digits(return_X_y=True)

     X_train, X_test, y_train, y_test = \
             sklearn.model_selection.train_test_split(X, y, random_state=1)

     automl = autosklearn.classification.AutoSklearnClassifier(time_left_for_this_task=60*5) #Auto-sklearn searches pipelines for 5 minutes

     automl.fit(X_train, y_train)

     y_hat = automl.predict(X_test)

     print("Accuracy score", sklearn.metrics.accuracy_score(y_test, y_hat))
```

3. Learn from previous *models* (for *very similar* tasks)

Models trained on intrinsically similar tasks

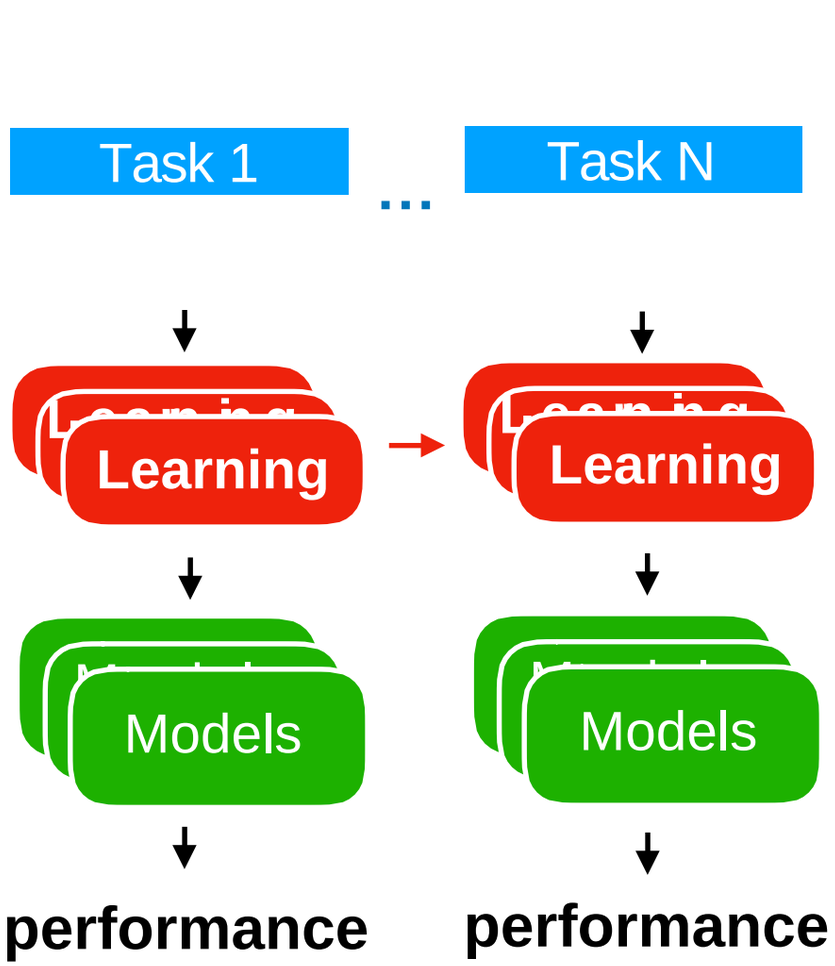
(model parameters, features,…)



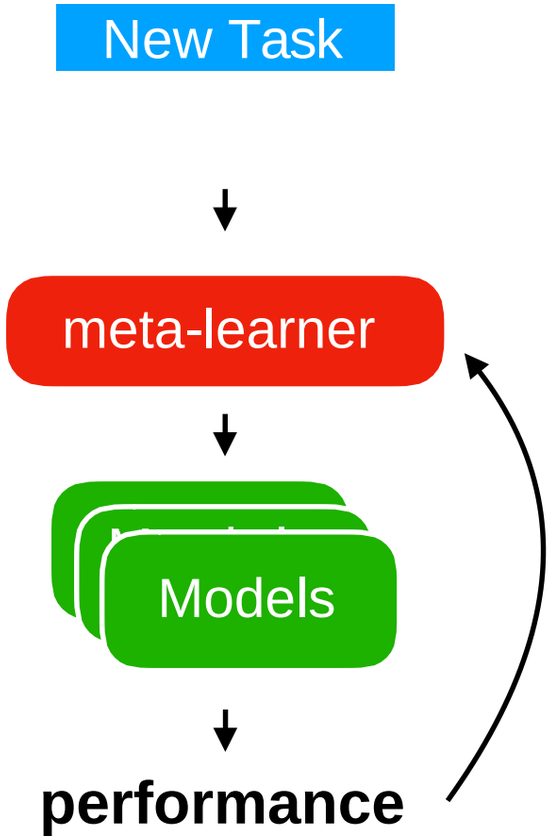
3. Learn from previous *models* (for *very similar* tasks)

Models trained on intrinsically similar tasks

(model parameters, features,…)



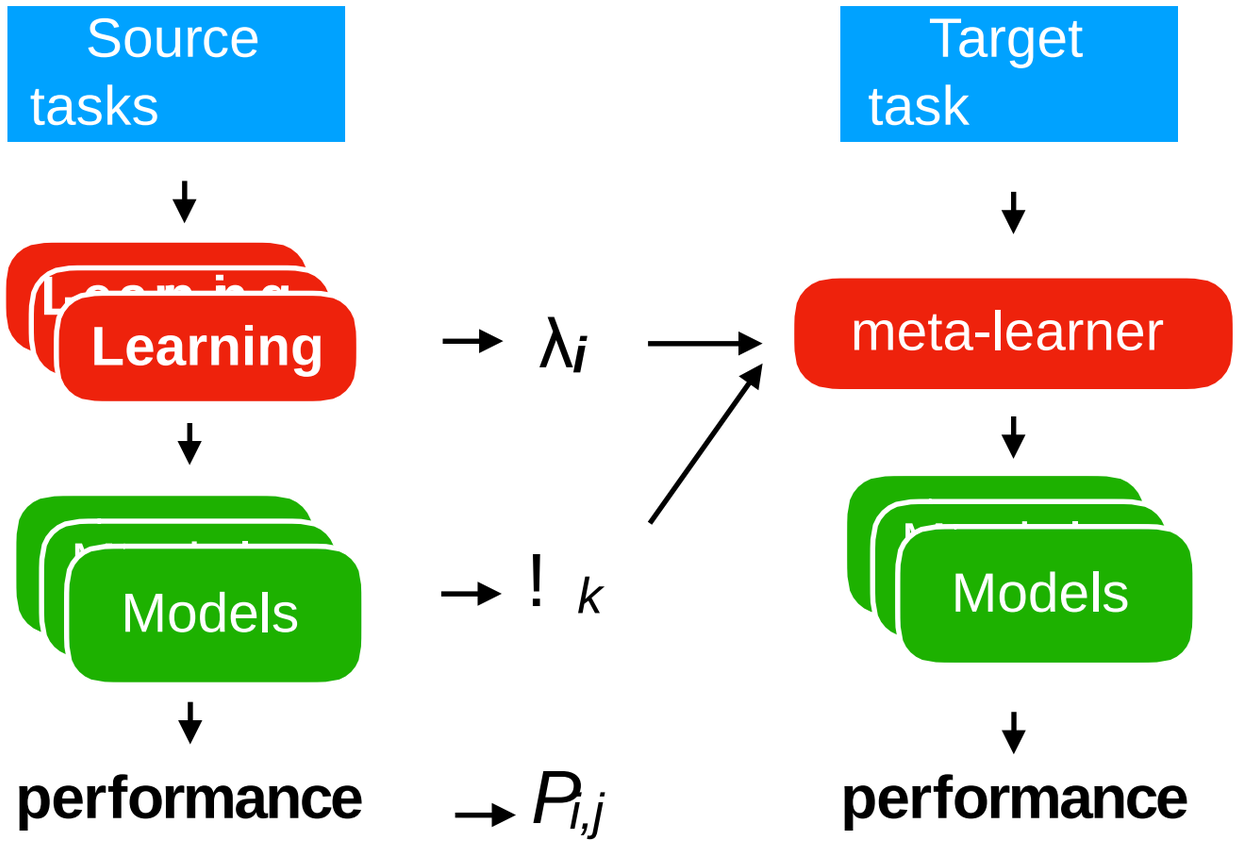
model param !	model config λ	score P
0.34,0.43,…	alg=NN, lr=0.9	o0,876
0.04,0.03,…	a alg=NN, lr=0.9	0,921
0.94,0.03,…	alg=NN, lr=0.9	0,936
0.34,0.23,…	alg=NN, lr=0.5	0,674
0.04,0.23,…	a alg=NN, lr=0.5	0,777
0.94,0.23,…	alg=NN, lr=0.5	0,791



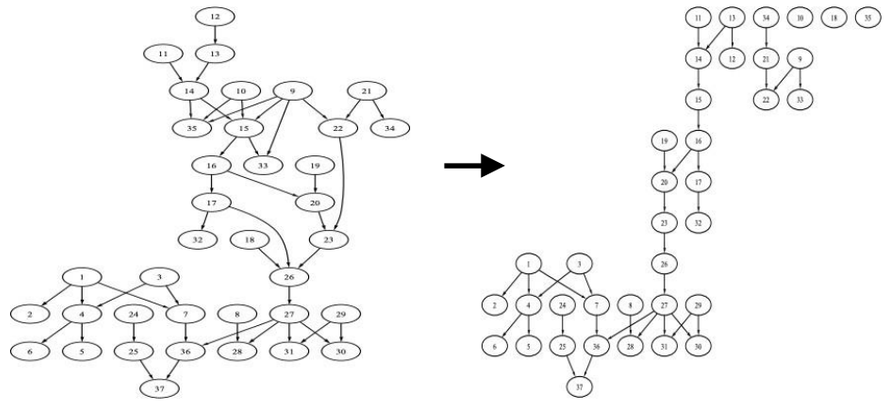
[1] Thrun and Pratt 1998
[2] Niculescu-Mizil and Caruana 2005
[3] Taylor and Stone 2009

Transfer Learning

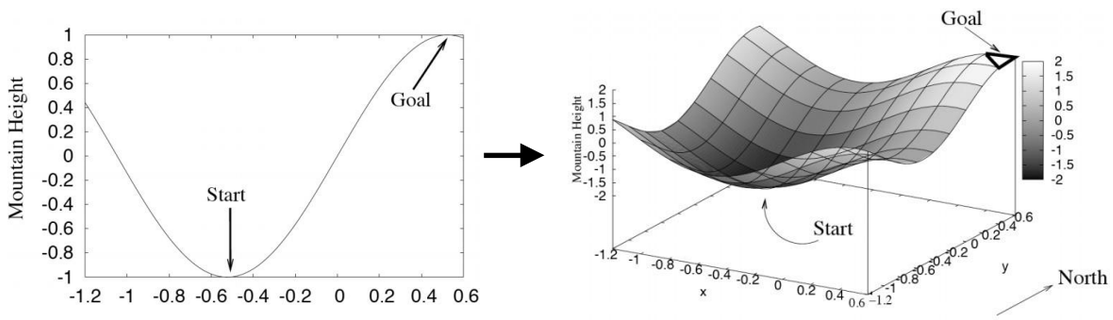
- Select source tasks, transfer trained models to similar target task [1]
- Use as starting point for tuning, or *freeze* certain aspects (e.g. structure)
  - Bayesian networks: start structure search from prior model [2]
  - Reinforcement learning: start policy search from prior policy [3]



Bayesian Network transfer

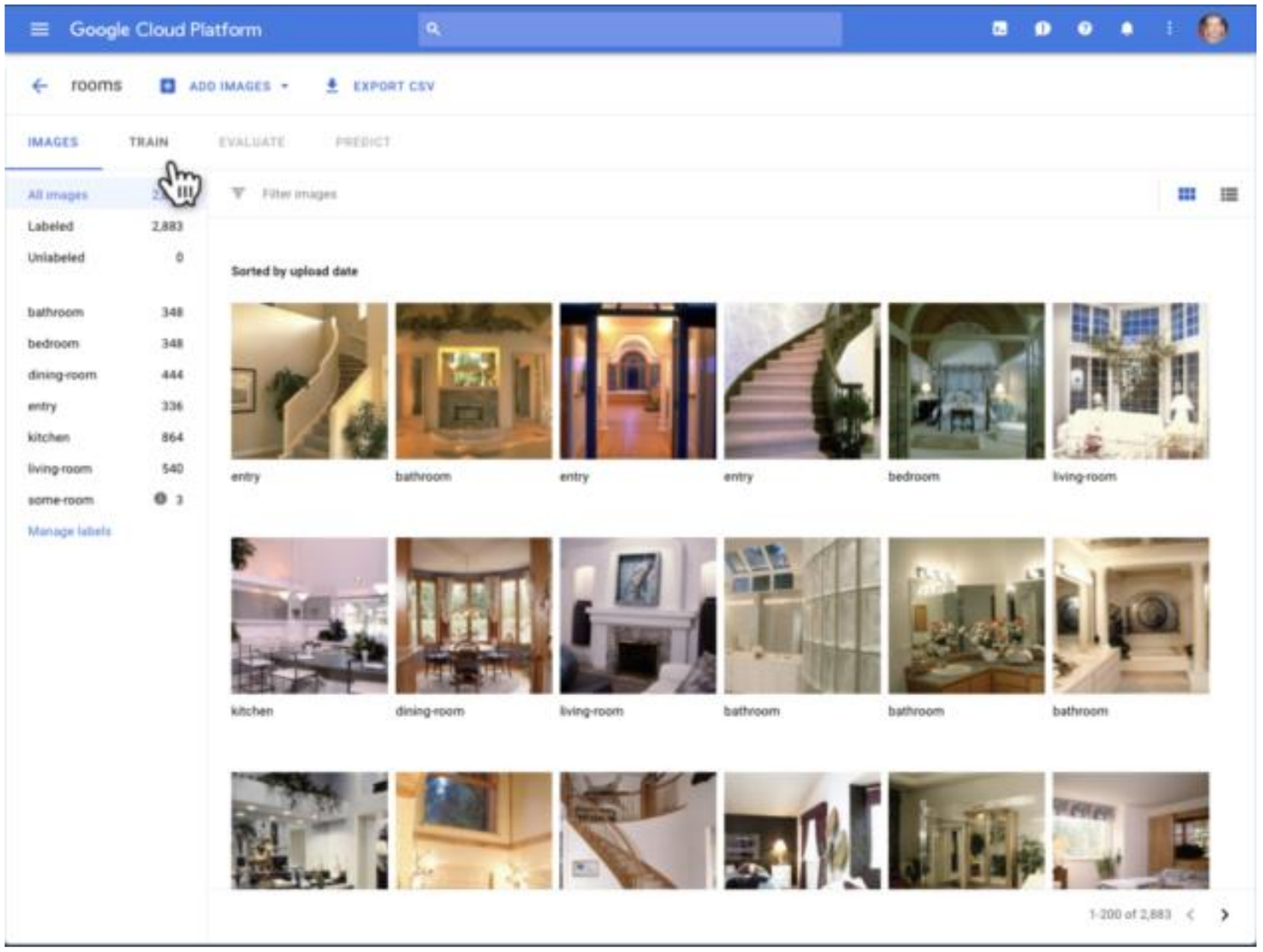


Reinforcement learning: 2D to 3D mountain car



Google Cloud AutoML

- Combines Neural Architecture Search with Transfer Learning



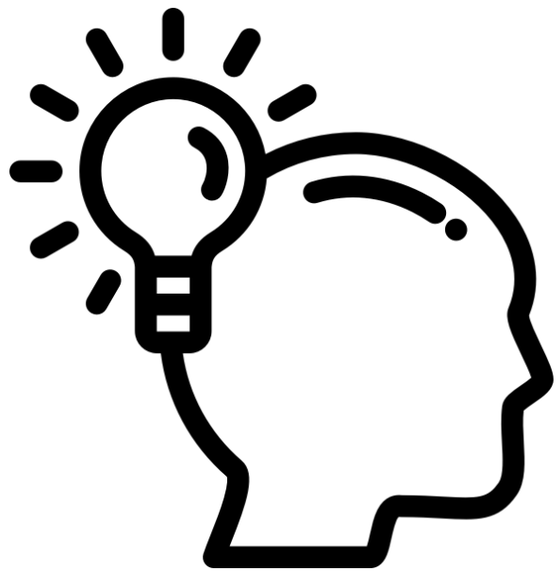
https://cloud.google.com/vision/automl/docs/samples/automl-vision-classification-create-model?authuser=1#automl_vision_classification_create_model-python

https://www.fast.ai/2018/07/23/auto-ml-3/

Human in the loop AutoML

AutoML challenges

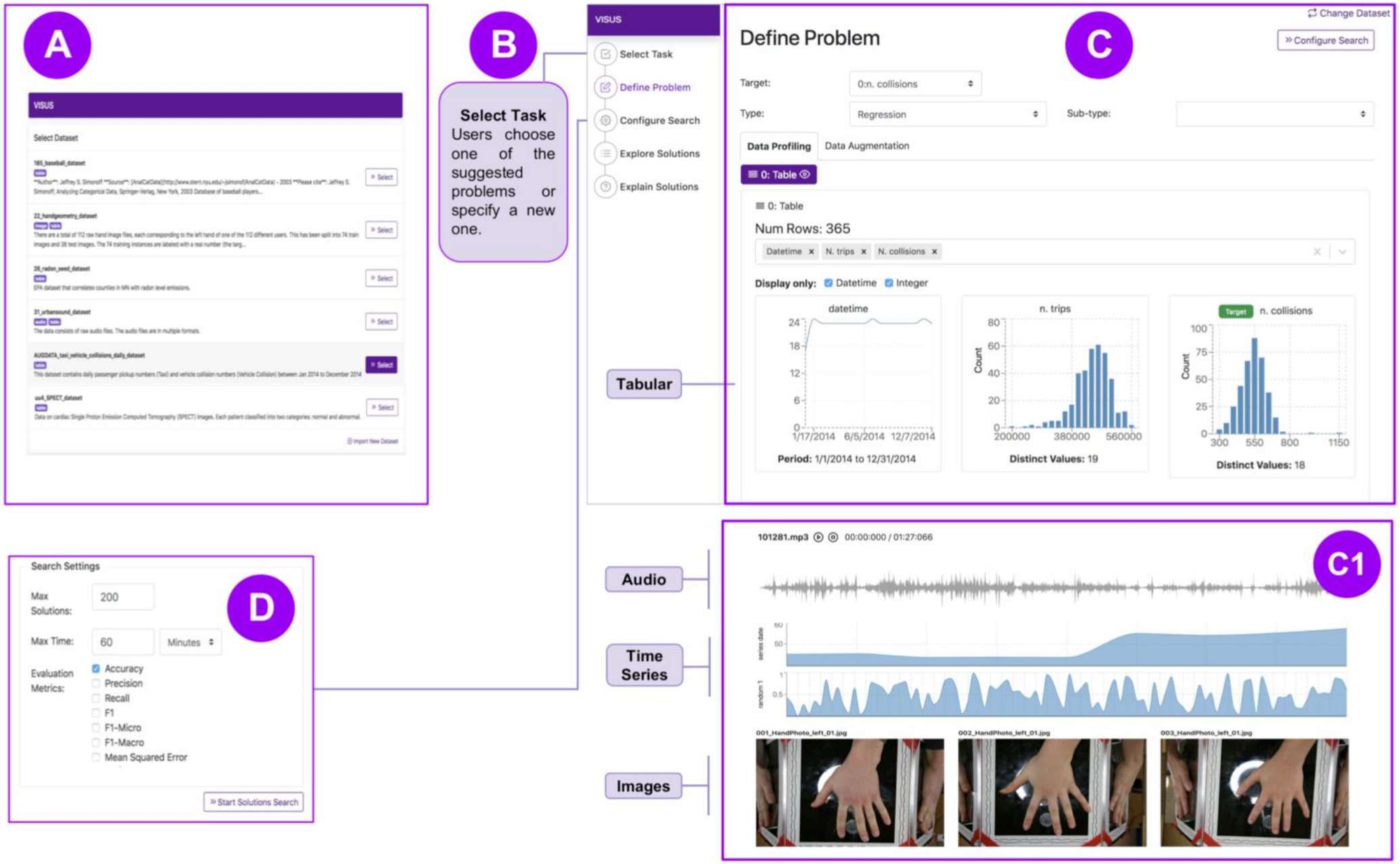
- Requires programming background
- Black box on top of Black box (hard to explain)
- No user feedback (how to adapt the models based on human knowledge)



Visual Analytics tools were created to tackle some of these problems

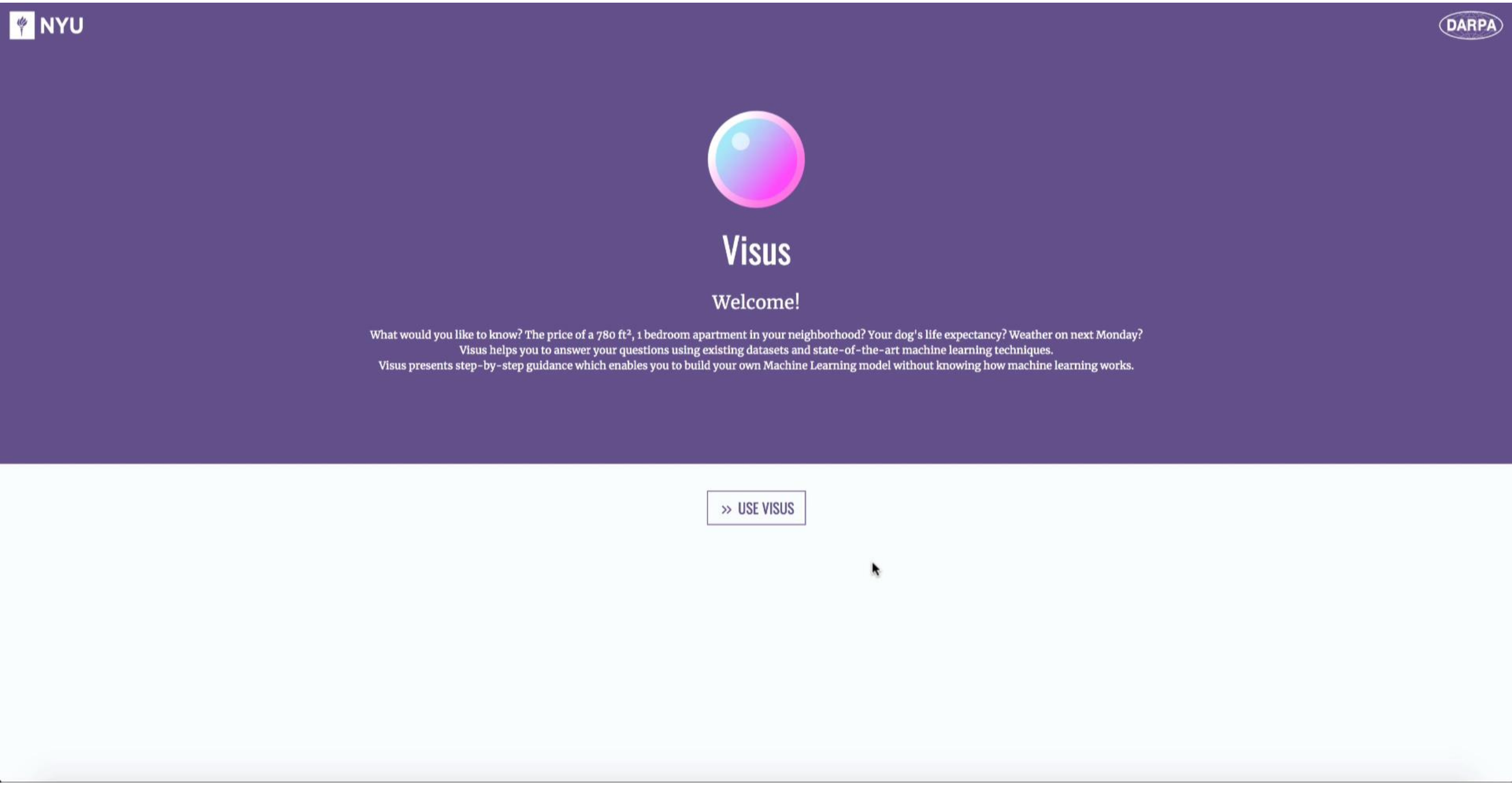
Visus

Interface between AutoML and user.
Model explanation / Score inspection



Santos, Aécio, et al. "Visus: An interactive system for automatic machine learning model building and curation."

Visus


NYU
DARPA
Visus
Welcome!
What would you like to know? The price of a 780 ft², 1 bedroom apartment in your neighborhood? Your dog's life expectancy? Weather on next Monday?
Visus helps you to answer your questions using existing datasets and state-of-the-art machine learning techniques.
Visus presents step-by-step guidance which enables you to build your own Machine Learning model without knowing how machine learning works.
» USE VISUS

Visus

Challenges Addressed

- Requires programming background
- Black box on top of Black box (hard to explain)
- No user feedback (how to adapt the models based on human knowledge)

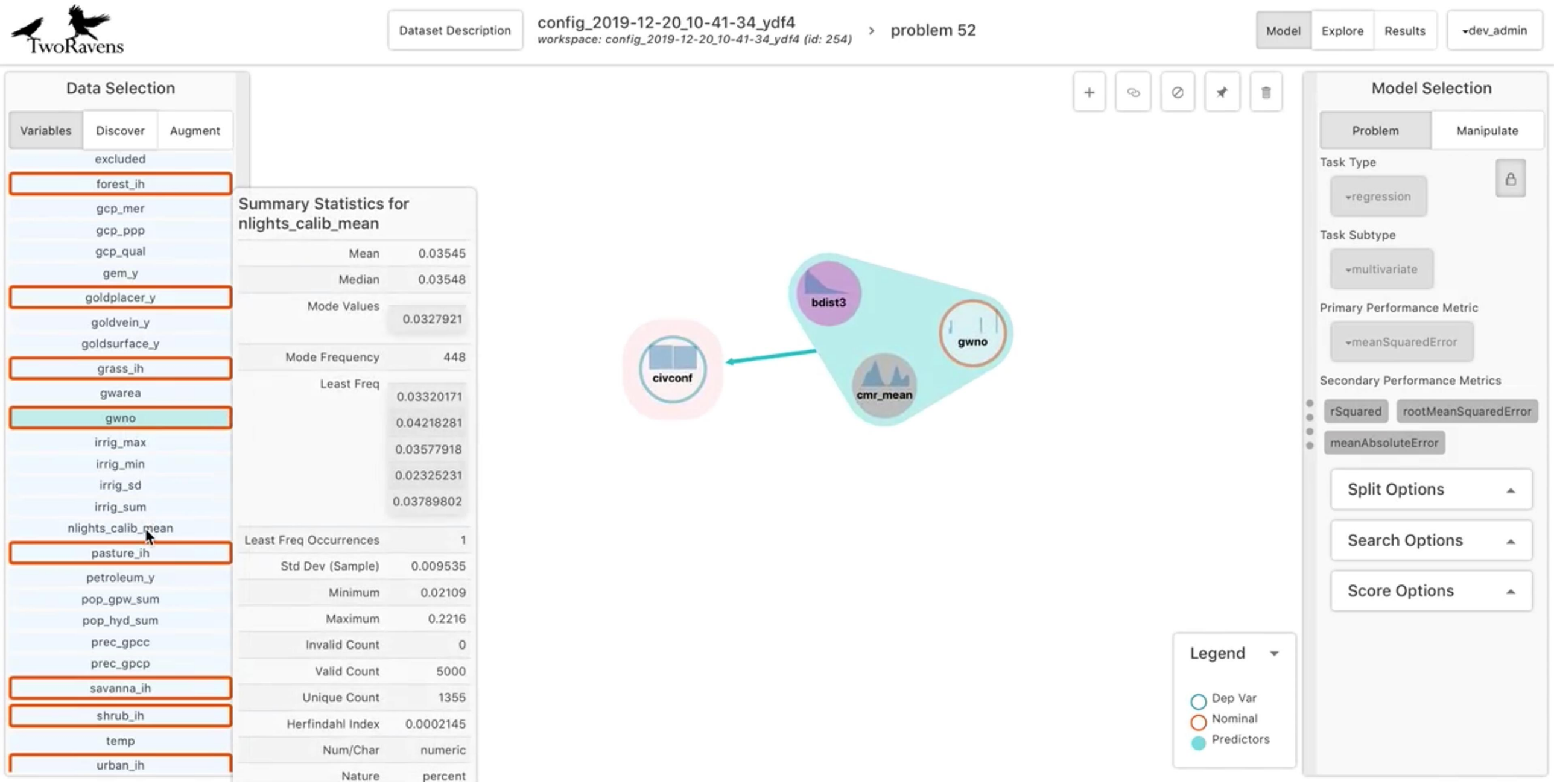
Limitations:

- User cannot edit the pipelines
- Linear flow

Santos, Aécio, et al. "Visus: An interactive system for automatic machine learning model building and curation."

Two Ravens

AutoML user interface
Feature selection



D'Orazio, Vito, Marcus Deng, and Michael Shoemate. "Tworavens for event data."

Two Ravens

Problem Specification and Model Construction

Two Ravens

Challenges Addressed:

- Requires programming background
- Black box on top of Black box (hard to explain)
- No user feedback (how to adapt the models based on human knowledge)

Limitations:

- No model explanation features
- Does not scale well for multiple features

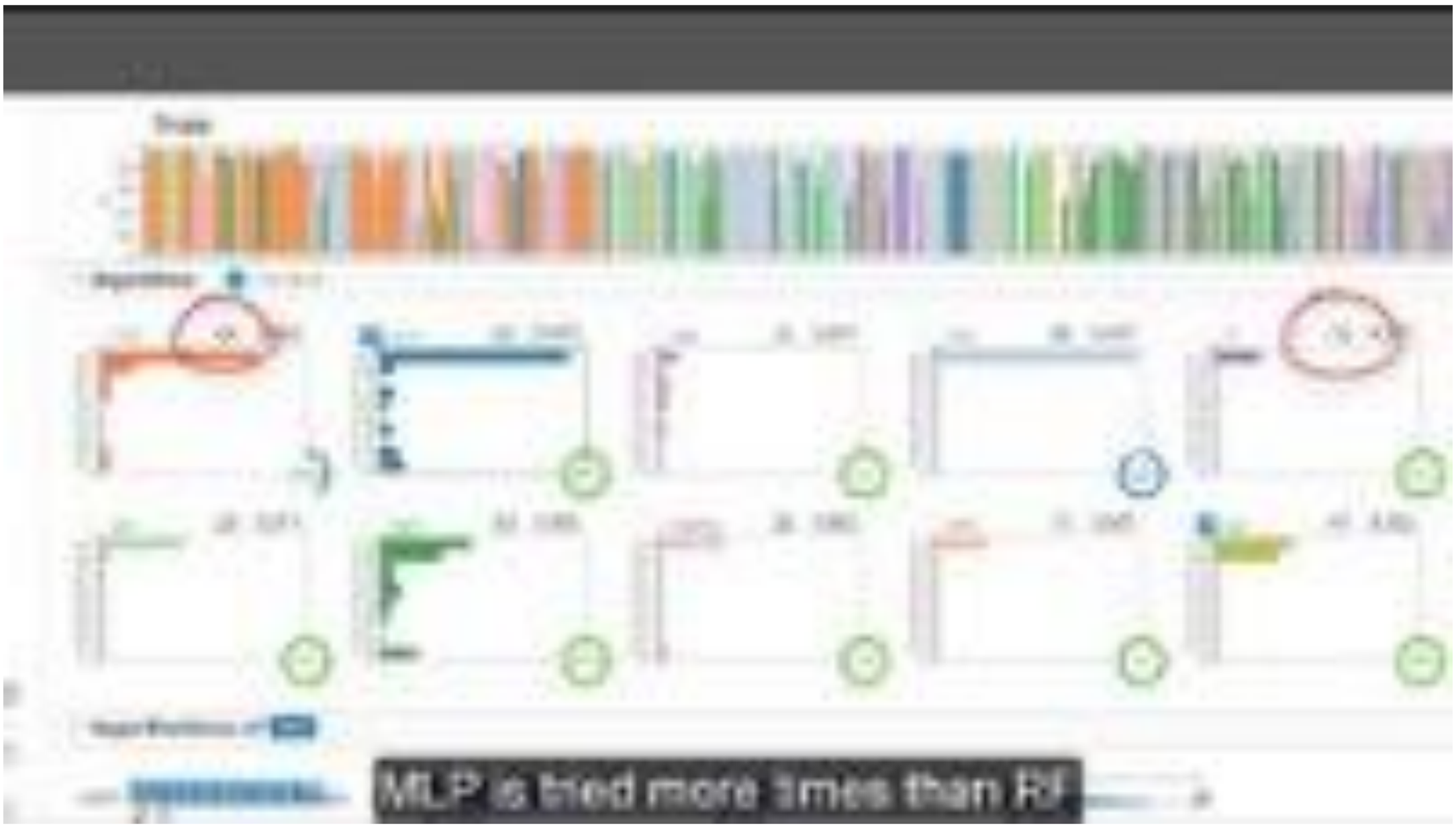
D'Orazio, Vito, Marcus Deng, and Michael Shoemate. "Tworavens for event data."

ATMSeer

Interface between AutoML and user.
Lets the user select new hyperparameters/algorithms to try

Wang, Qianwen, et al. "Atmseer: Increasing transparency and controllability in automated machine learning."

ATMSeer



ATMSeer

Challenges Addressed

- Requires programming background
- Black box on top of Black box (hard to explain)
- No user feedback (how to adapt the models based on human knowledge)

Limitations:

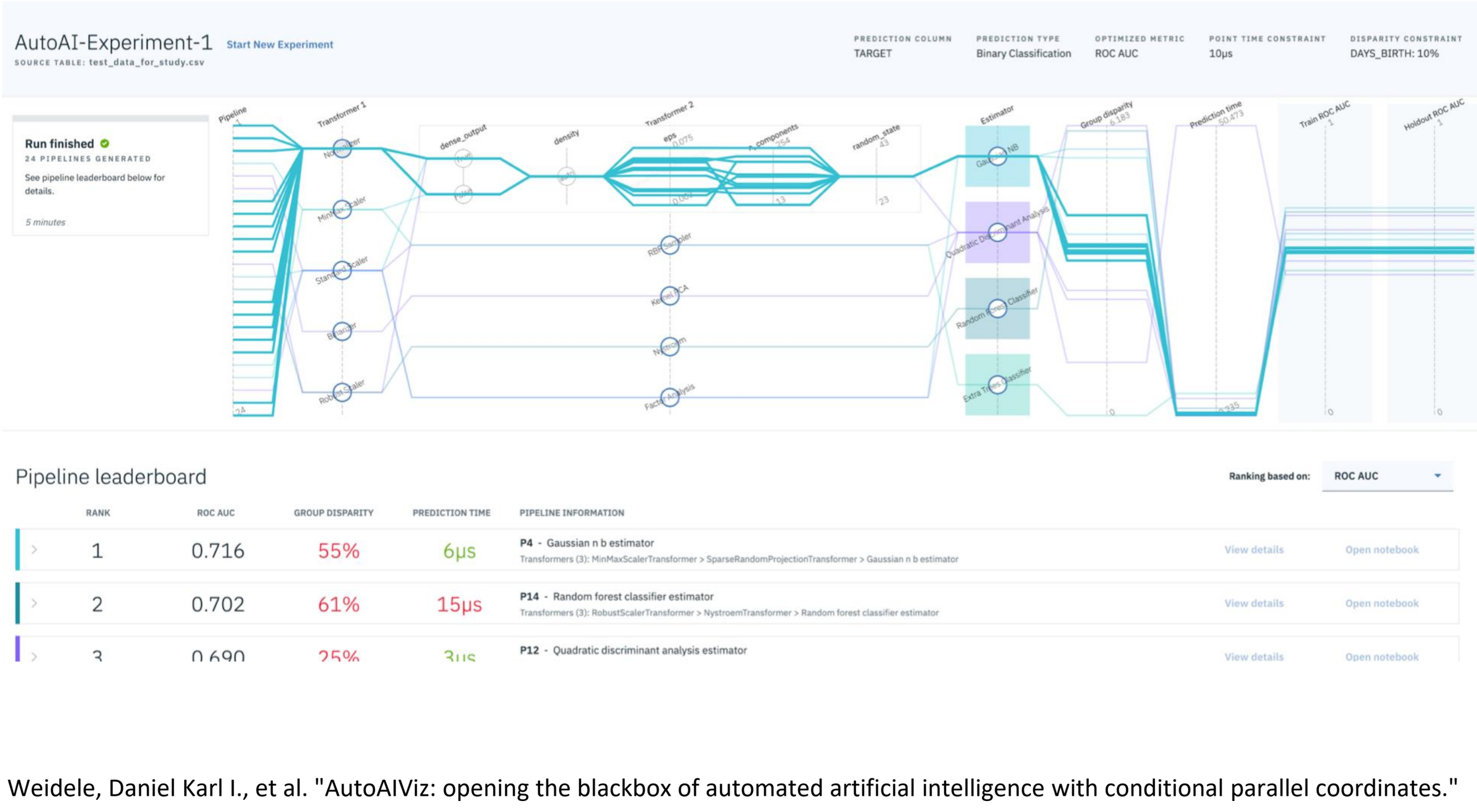
- Fixed pipeline

- Support for a single AutoML (ATM)

Wang, Qianwen, et al. "Atmseer: Increasing transparency and controllability in automated machine learning."

AutoAIViz

Debugging/Comparing AutoML Pipelines



Weidele, Daniel Karl I., et al. "AutoAIViz: opening the blackbox of automated artificial intelligence with conditional parallel coordinates."

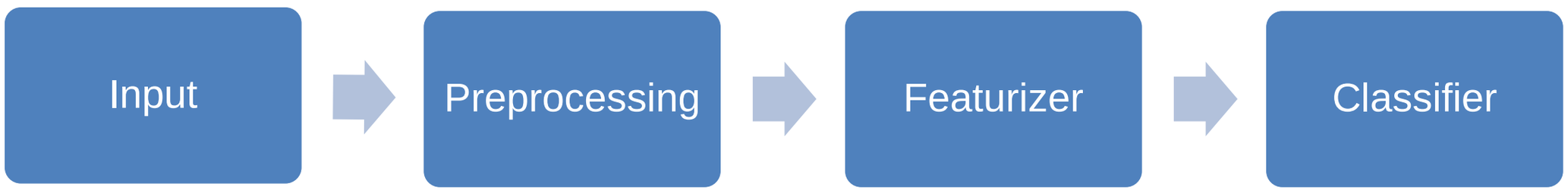
AutoAIViz

Challenges Addressed

- Requires programming background
- Black box on top of Black box (hard to explain)
- No user feedback (how to adapt the models based on human knowledge)

Limitations:

- Fixed pipeline



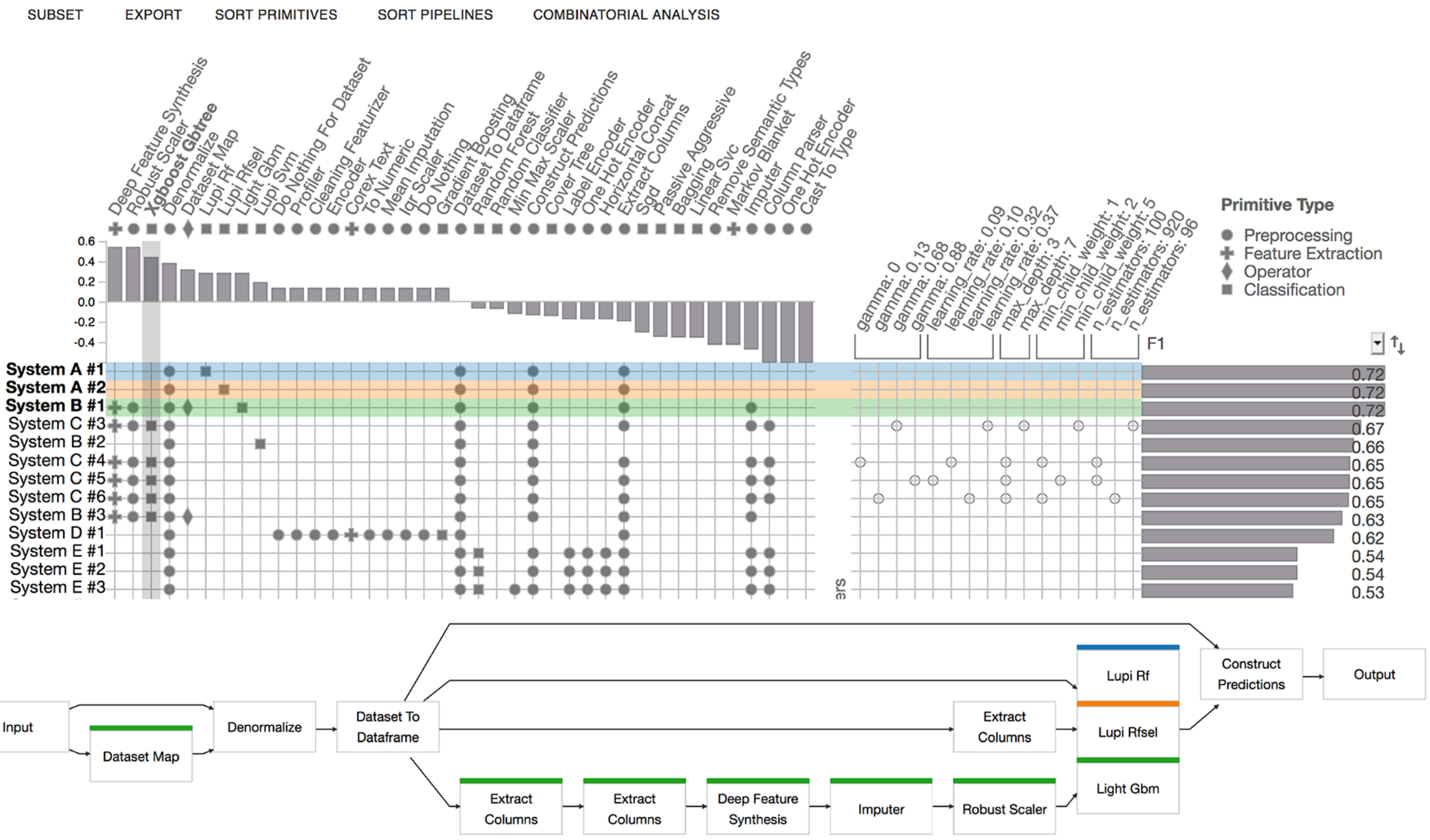
- Support for a single AutoML (ATM)

Weidele, Daniel Karl I., et al. "AutoAIViz: opening the blackbox of automated artificial intelligence with conditional parallel coordinates."

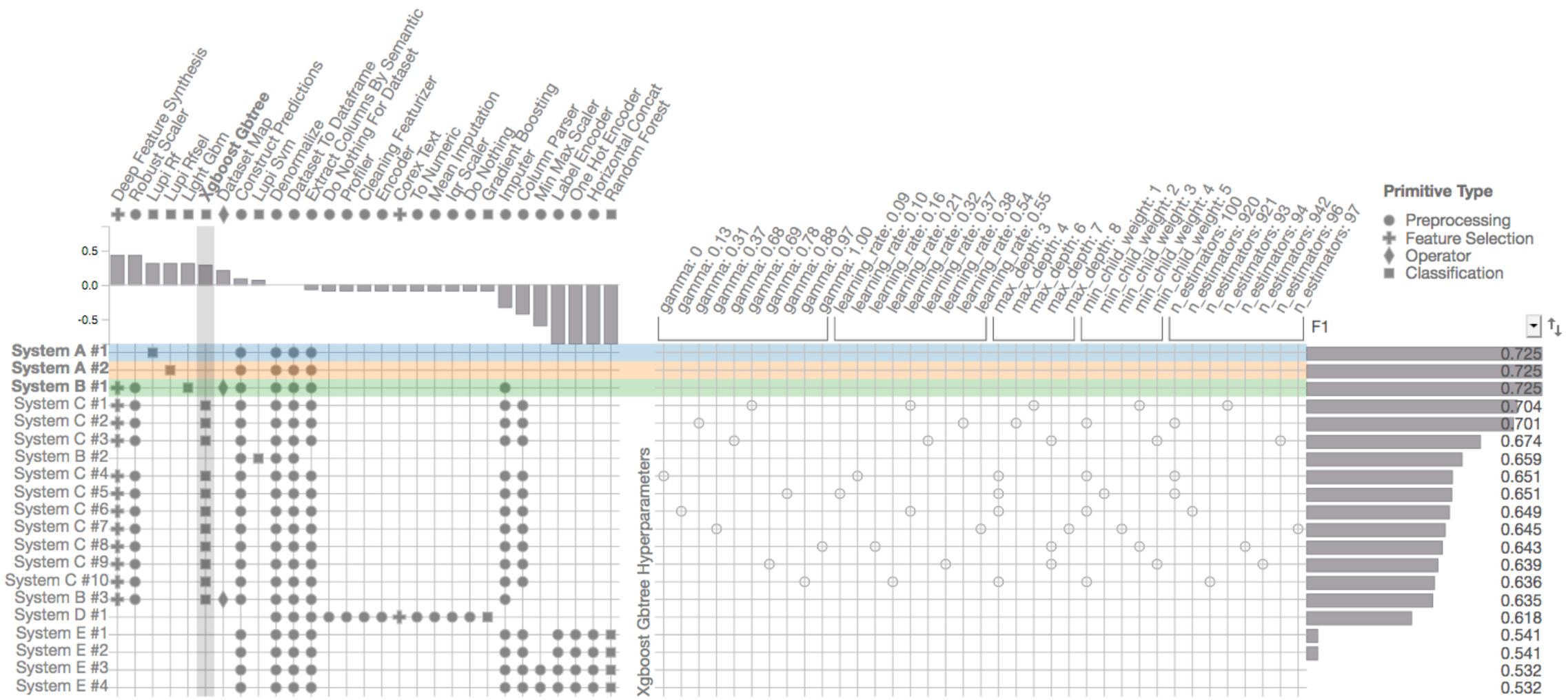
PipelineProfiler

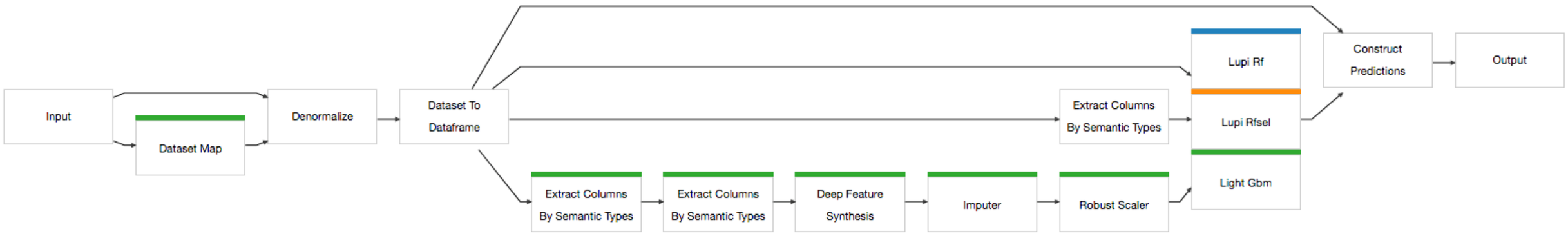
Debugging/comparing AutoML pipelines produced by multiple systems

```
PipelineProfiler.plot_pipeline_matrix(heartstatlog)
```

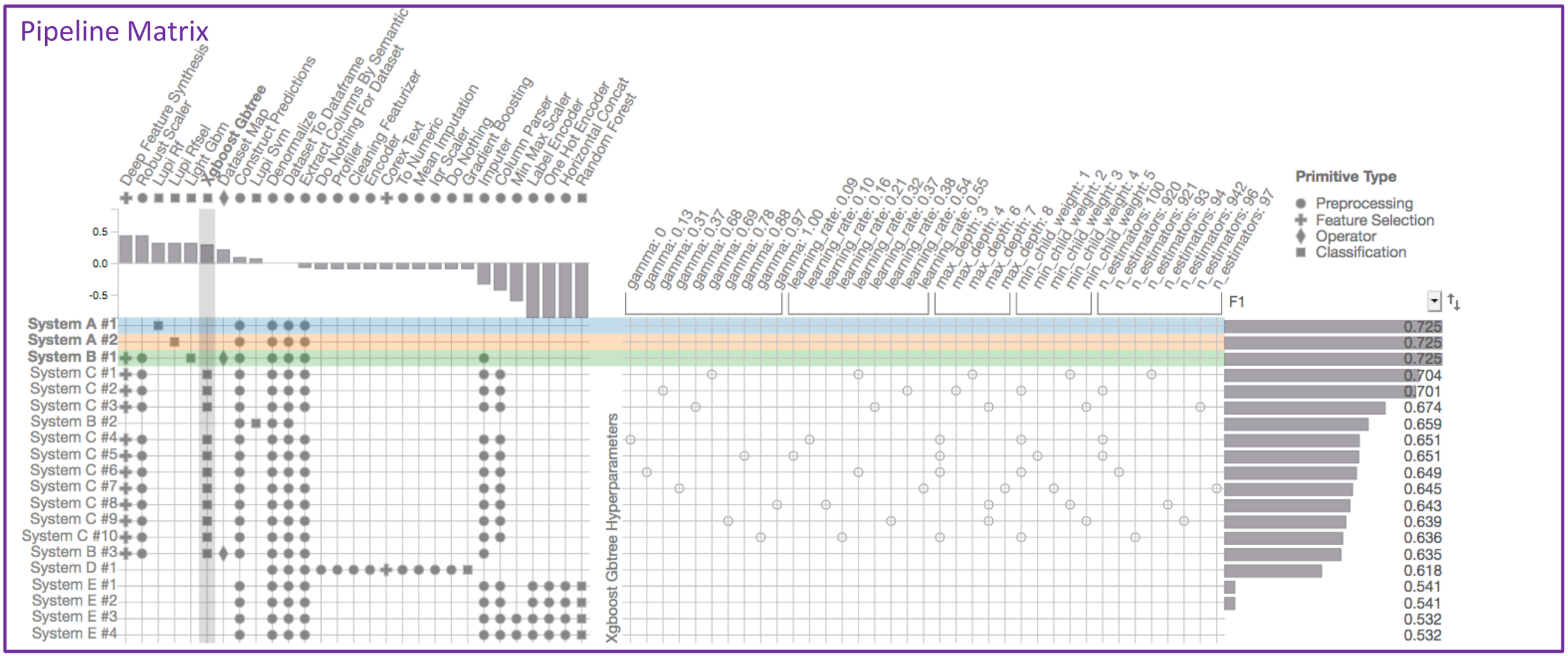


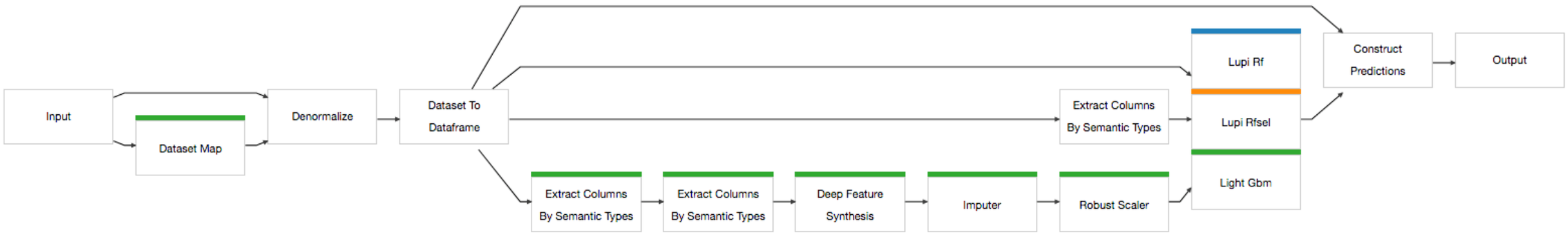
Ono, Jorge Piazentin, et al. "PipelineProfiler: A Visual Analytics Tool for the Exploration of AutoML Pipelines."

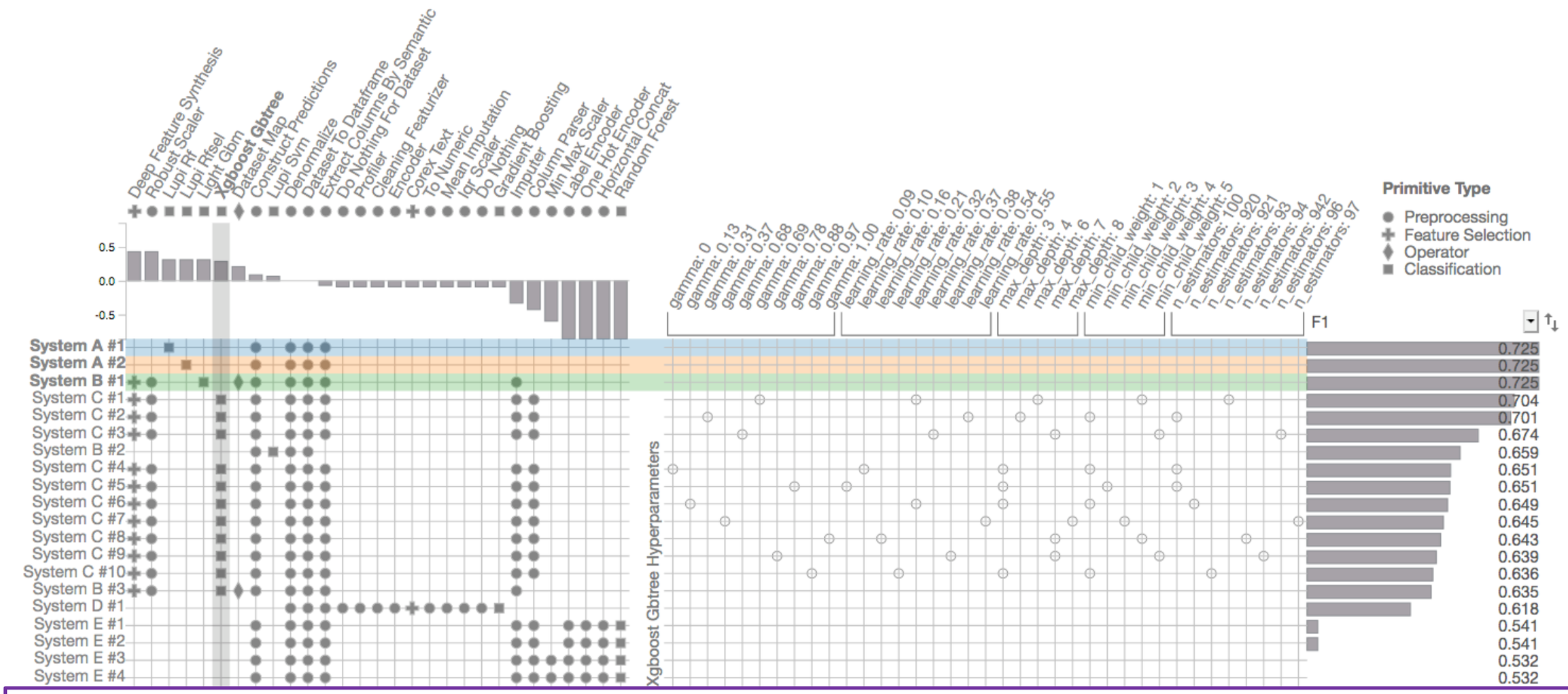

Deep Feature Synthesis
Robust Scaler
Lupi Rf
Lupi Rfsel
Light Gbm
Xgboost Gbtree
Dataset Map
Construct Predictions
Lupi Svm
Denormalize
Dataset To Dataframe
Extract Columns By Semantic
Do Nothing For Dataset
Profiler
Cleaning Featurizer
Encoder
Corex Text
To Numeric
Mean Imputation
Iqr Scaler
Do Nothing
Gradient Boosting
Imputer
Column Parser
Min Max Scaler
Label Encoder
One Hot Encoder
Horizontal Concat
Random Forest
0.5
0.0
-0.5
System A #1
System A #2
System B #1
System C #1
System C #2
System C #3
System B #2
System C #4
System C #5
System C #6
System C #7
System C #8
System C #9
System C #10
System B #3
System D #1
System E #1
System E #2
System E #3
System E #4
gamma: 0
gamma: 0.13
gamma: 0.31
gamma: 0.37
gamma: 0.68
gamma: 0.69
gamma: 0.78
gamma: 0.88
gamma: 0.97
gamma: 1.00
learning_rate: 0.09
learning_rate: 0.10
learning_rate: 0.16
learning_rate: 0.21
learning_rate: 0.32
learning_rate: 0.37
learning_rate: 0.38
learning_rate: 0.54
learning_rate: 0.55
max_depth: 3
max_depth: 4
max_depth: 6
max_depth: 7
max_depth: 8
min_child_weight: 1
min_child_weight: 2
min_child_weight: 3
min_child_weight: 4
min_child_weight: 5
n_estimators: 100
n_estimators: 920
n_estimators: 921
n_estimators: 93
n_estimators: 94
n_estimators: 942
n_estimators: 96
n_estimators: 97
F1
Xgboost Gbtree Hyperparameters
Primitive Type
Preprocessing
Feature Selection
Operator
Classification
0.725
0.725
0.725
0.704
0.701
0.674
0.659
0.651
0.651
0.649
0.645
0.643
0.639
0.636
0.635
0.618
0.541
0.541
0.532
0.532


Input
Dataset Map
Denormalize
Dataset To Dataframe
Extract Columns By Semantic Types
Extract Columns By Semantic Types
Deep Feature Synthesis
Imputer
Robust Scaler
Extract Columns By Semantic Types
Lupi Rf
Lupi Rfsel
Light Gbm
Construct Predictions
Output

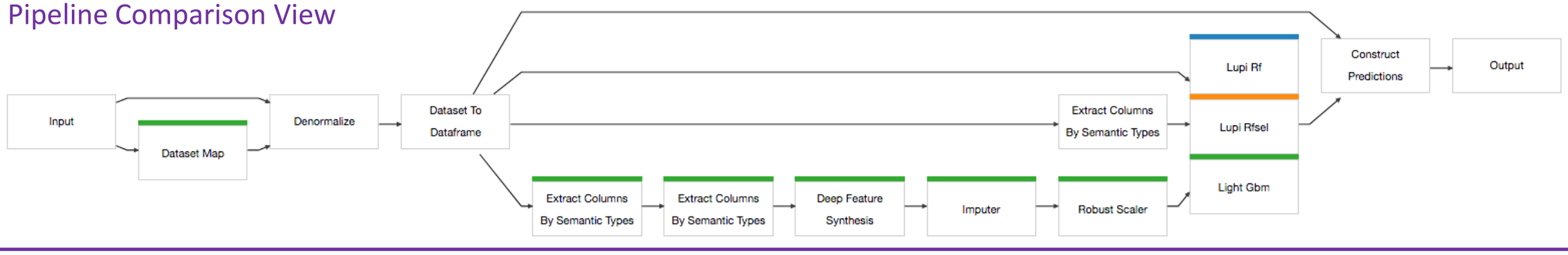
Pipeline Matrix

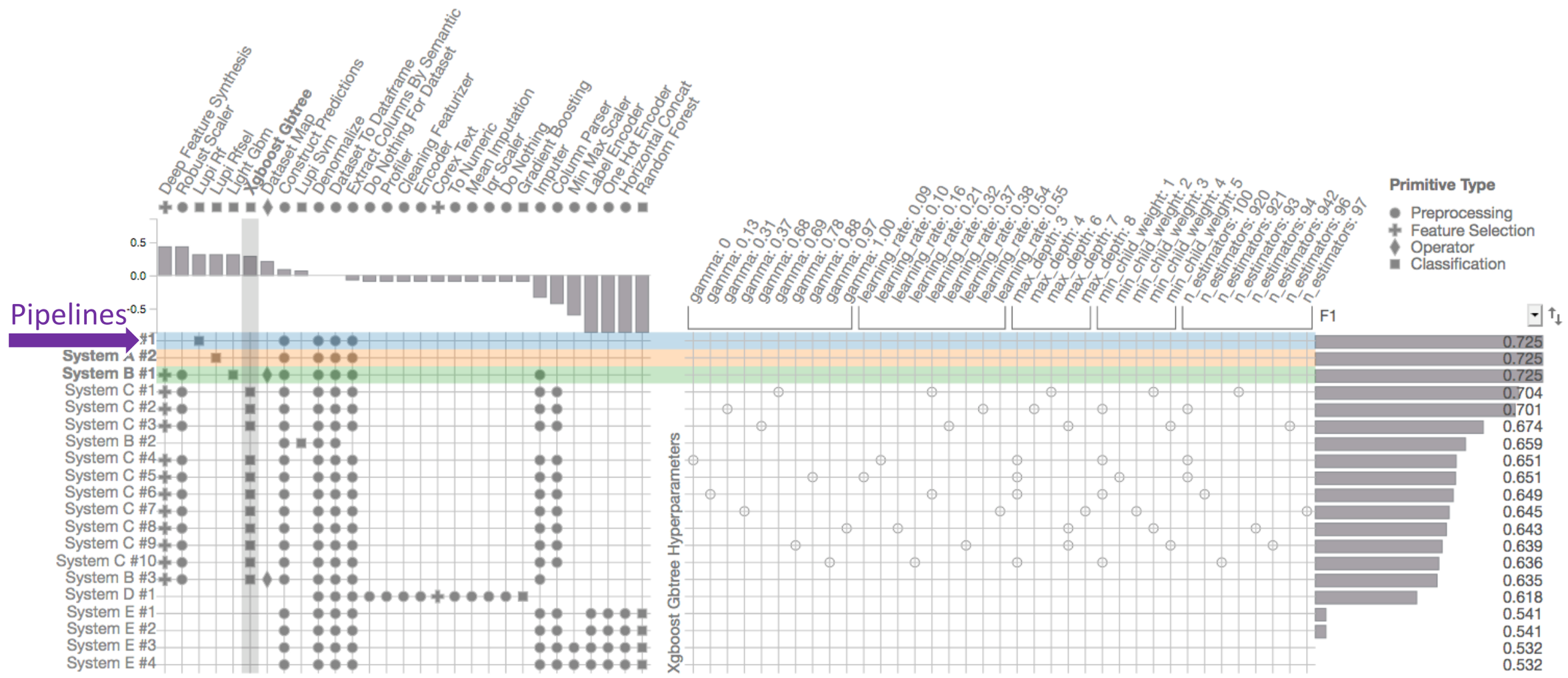


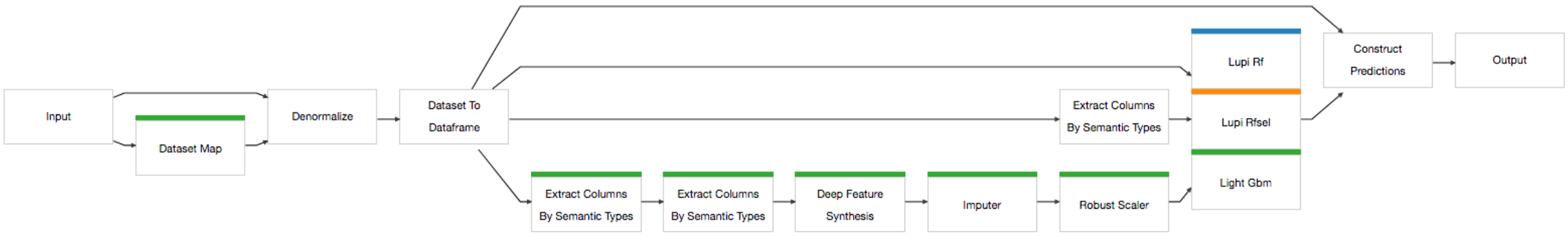


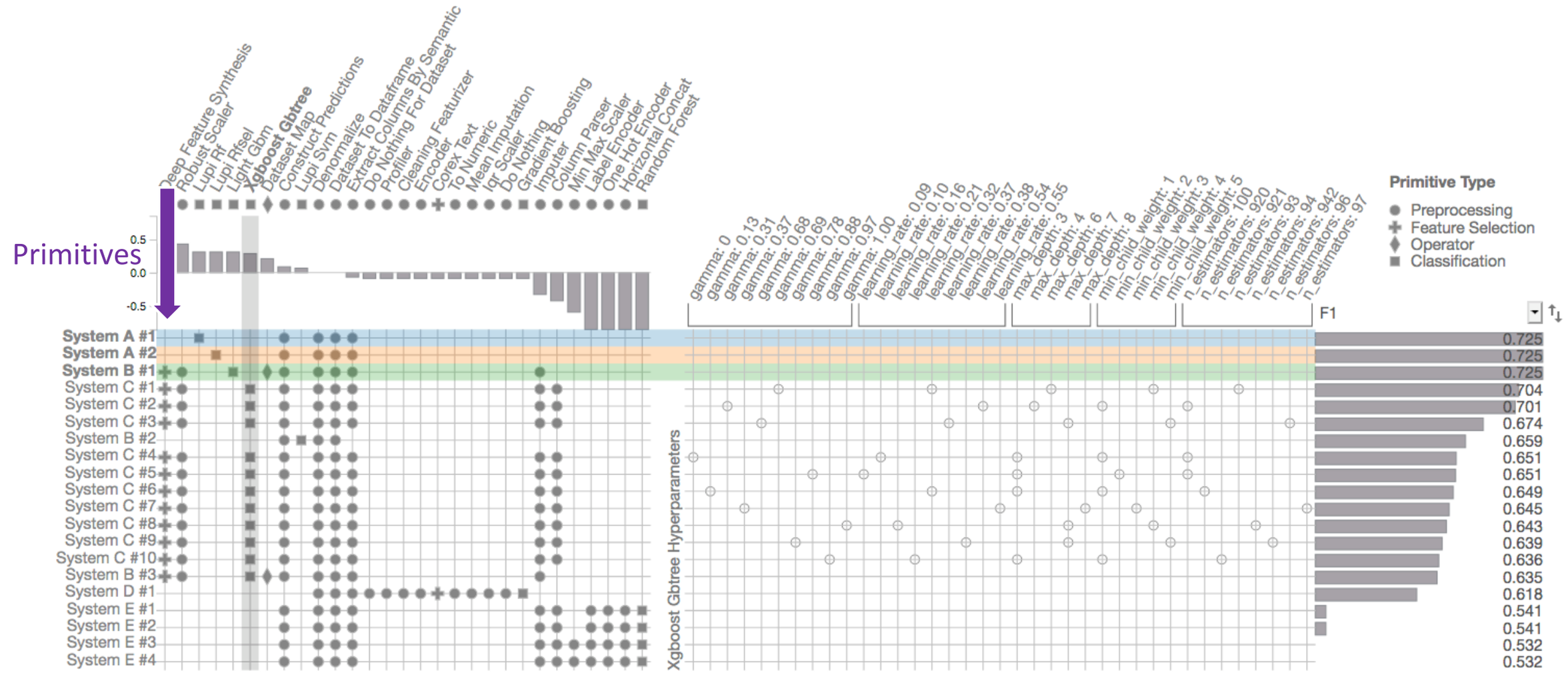


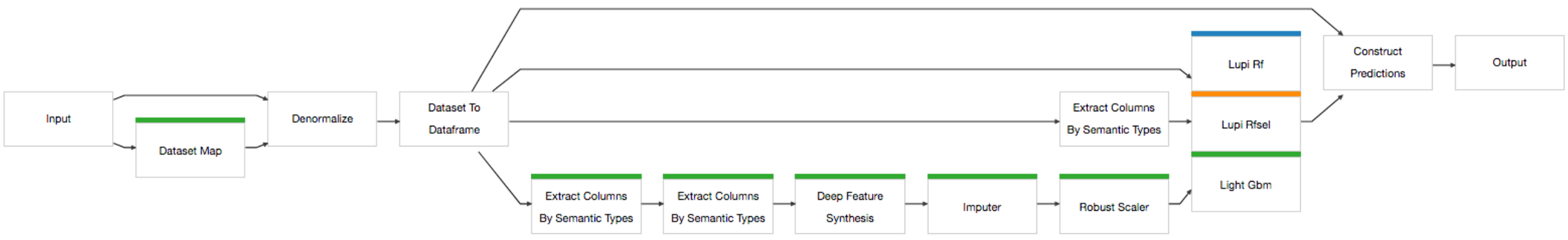
Pipeline Comparison View

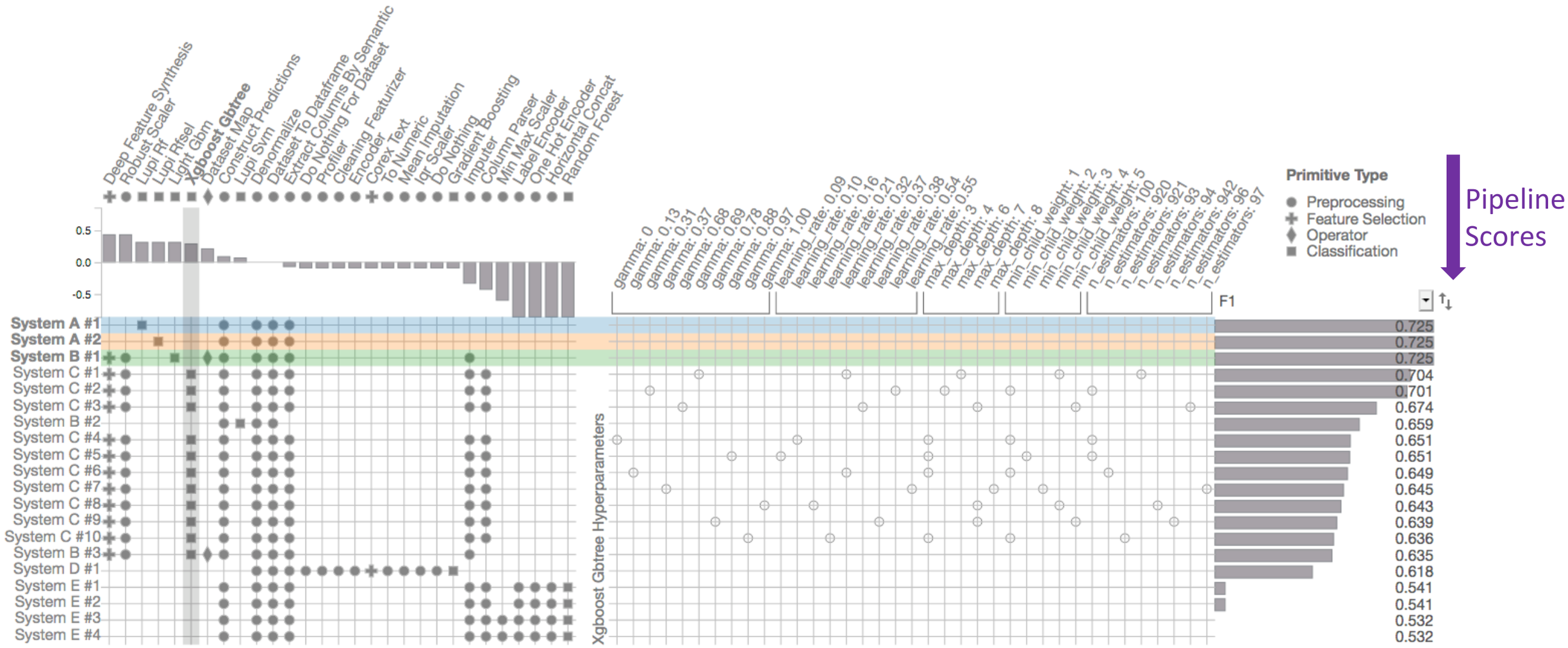


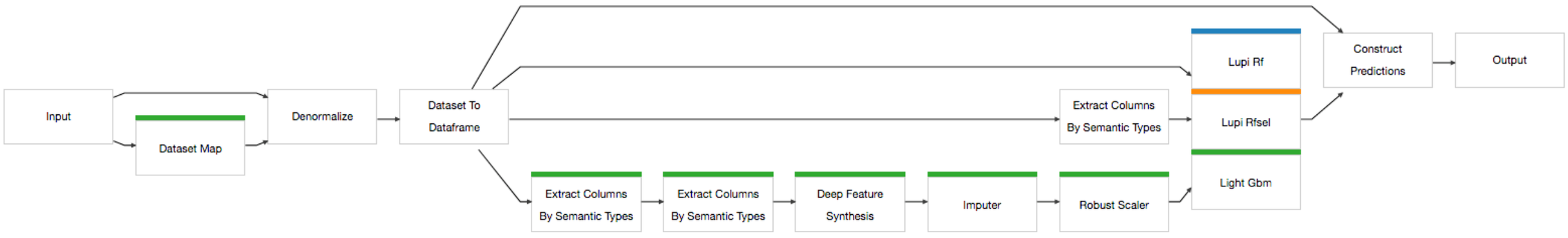

Deep Feature Synthesis
Robust Scaler
Lupi Rf
Lupi Rfsel
Light Gbm
Xgboost Gbtree
Dataset Map
Construct Predictions
Lupi Svm
Denormalize
Dataset To Dataframe
Extract Columns By Semantic
Do Nothing For Dataset
Profiler
Cleaning Featurizer
Encoder
Corex Text
To Numeric
Mean Imputation
Iqr Scaler
Do Nothing
Gradient Boosting
Imputer
Column Parser
Min Max Scaler
Label Encoder
One Hot Encoder
Horizontal Concat
Random Forest
0.5
0.0
-0.5
Pipelines
System A #1
System A #2
System B #1
System C #1
System C #2
System C #3
System B #2
System C #4
System C #5
System C #6
System C #7
System C #8
System C #9
System C #10
System B #3
System D #1
System E #1
System E #2
System E #3
System E #4
gamma: 0
gamma: 0.13
gamma: 0.31
gamma: 0.37
gamma: 0.68
gamma: 0.69
gamma: 0.78
gamma: 0.88
gamma: 0.97
gamma: 1.00
learning_rate: 0.09
learning_rate: 0.10
learning_rate: 0.16
learning_rate: 0.21
learning_rate: 0.32
learning_rate: 0.37
learning_rate: 0.38
learning_rate: 0.54
learning_rate: 0.55
max_depth: 3
max_depth: 4
max_depth: 6
max_depth: 7
max_depth: 8
min_child_weight: 1
min_child_weight: 2
min_child_weight: 3
min_child_weight: 4
min_child_weight: 5
n_estimators: 100
n_estimators: 920
n_estimators: 921
n_estimators: 93
n_estimators: 94
n_estimators: 942
n_estimators: 96
n_estimators: 97
F1
Primitive Type
Preprocessing
Feature Selection
Operator
Classification
Xgboost Gbtree Hyperparameters
0.725
0.725
0.725
0.704
0.701
0.674
0.659
0.651
0.651
0.649
0.645
0.643
0.639
0.636
0.635
0.618
0.541
0.541
0.532
0.532


Input
Dataset Map
Denormalize
Dataset To Dataframe
Extract Columns By Semantic Types
Extract Columns By Semantic Types
Deep Feature Synthesis
Imputer
Robust Scaler
Extract Columns By Semantic Types
Lupi Rf
Lupi Rfsel
Light Gbm
Construct Predictions
Output

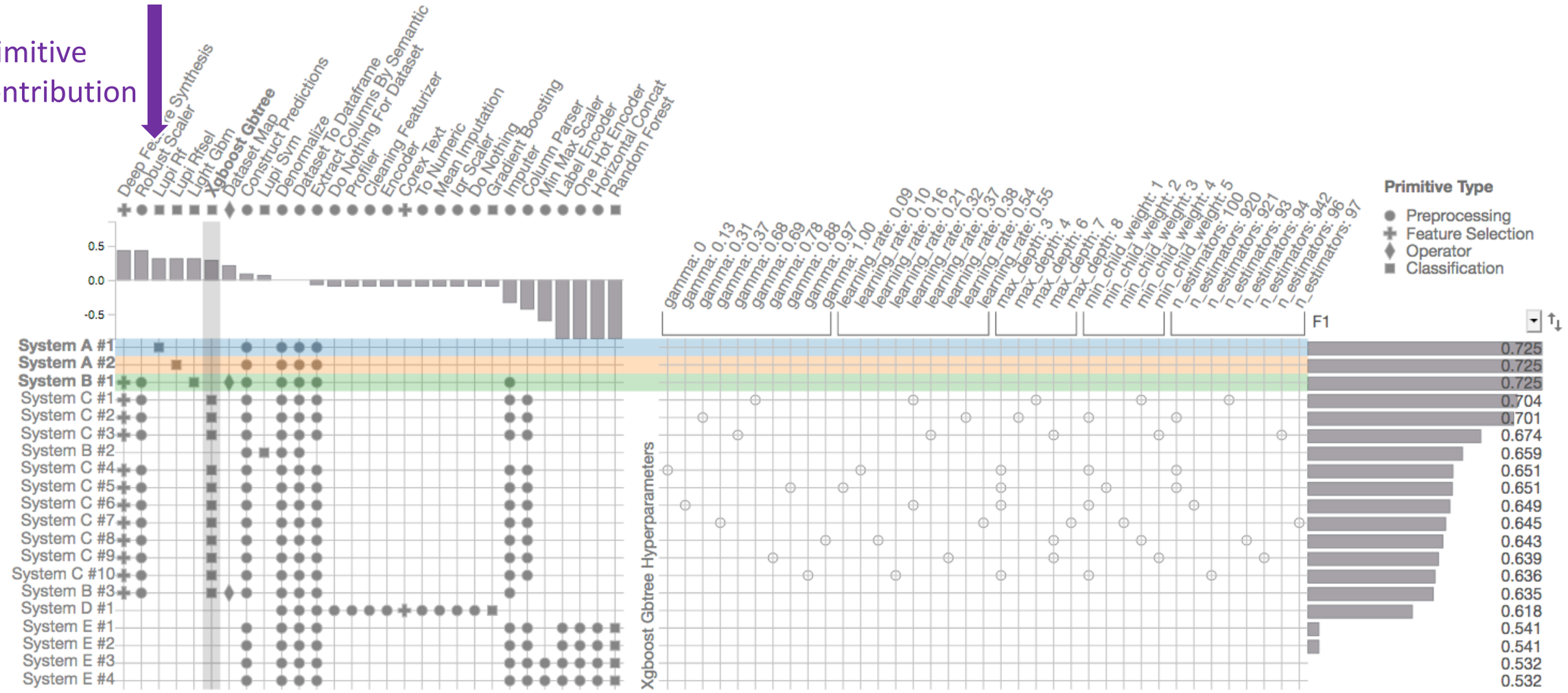

Primitives
Deep Feature Synthesis
Robust Scaler
Lupi Rf
Lupi Rfsel
Light Gbm
Xgboost Gbtree
Dataset Map
Construct Predictions
Lupi Svm
Denormalize
Dataset To Dataframe
Extract Columns By Semantic
Do Nothing For Dataset
Profiler
Cleaning Featurizer
Encoder
Corex Text
To Numeric
Mean Imputation
Iqr Scaler
Do Nothing
Gradient Boosting
Imputer
Column Parser
Min Max Scaler
Label Encoder
One Hot Encoder
Horizontal Concat
Random Forest
0.5
0.0
-0.5
gamma: 0
gamma: 0.13
gamma: 0.31
gamma: 0.37
gamma: 0.68
gamma: 0.69
gamma: 0.78
gamma: 0.88
gamma: 0.97
gamma: 1.00
learning_rate: 0.09
learning_rate: 0.10
learning_rate: 0.16
learning_rate: 0.21
learning_rate: 0.32
learning_rate: 0.37
learning_rate: 0.38
learning_rate: 0.54
learning_rate: 0.55
max_depth: 3
max_depth: 4
max_depth: 6
max_depth: 7
max_depth: 8
min_child_weight: 1
min_child_weight: 2
min_child_weight: 3
min_child_weight: 4
min_child_weight: 5
n_estimators: 100
n_estimators: 920
n_estimators: 921
n_estimators: 93
n_estimators: 94
n_estimators: 942
n_estimators: 96
n_estimators: 97
Primitive Type
Preprocessing
Feature Selection
Operator
Classification
F1
System A #1
System A #2
System B #1
System C #1
System C #2
System C #3
System B #2
System C #4
System C #5
System C #6
System C #7
System C #8
System C #9
System C #10
System B #3
System D #1
System E #1
System E #2
System E #3
System E #4
Xgboost Gbtree Hyperparameters
0.725
0.725
0.725
0.704
0.701
0.674
0.659
0.651
0.651
0.649
0.645
0.643
0.639
0.636
0.635
0.618
0.541
0.541
0.532
0.532

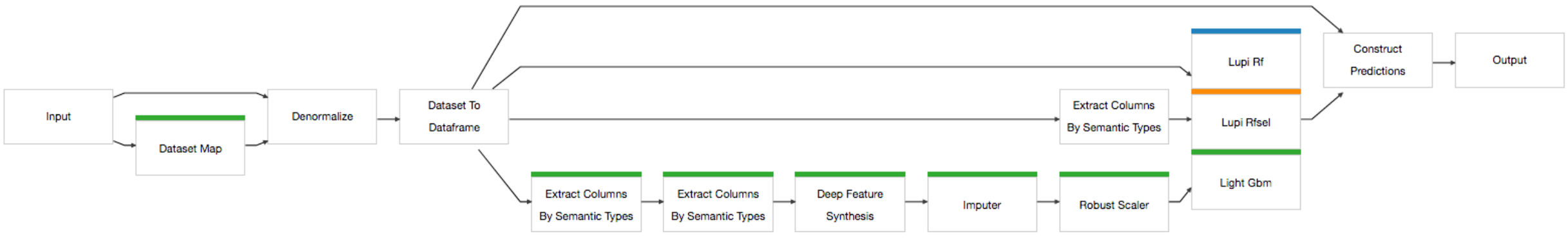

Input
Dataset Map
Denormalize
Dataset To Dataframe
Extract Columns By Semantic Types
Extract Columns By Semantic Types
Deep Feature Synthesis
Imputer
Robust Scaler
Extract Columns By Semantic Types
Lupi Rf
Lupi Rfsel
Light Gbm
Construct Predictions
Output

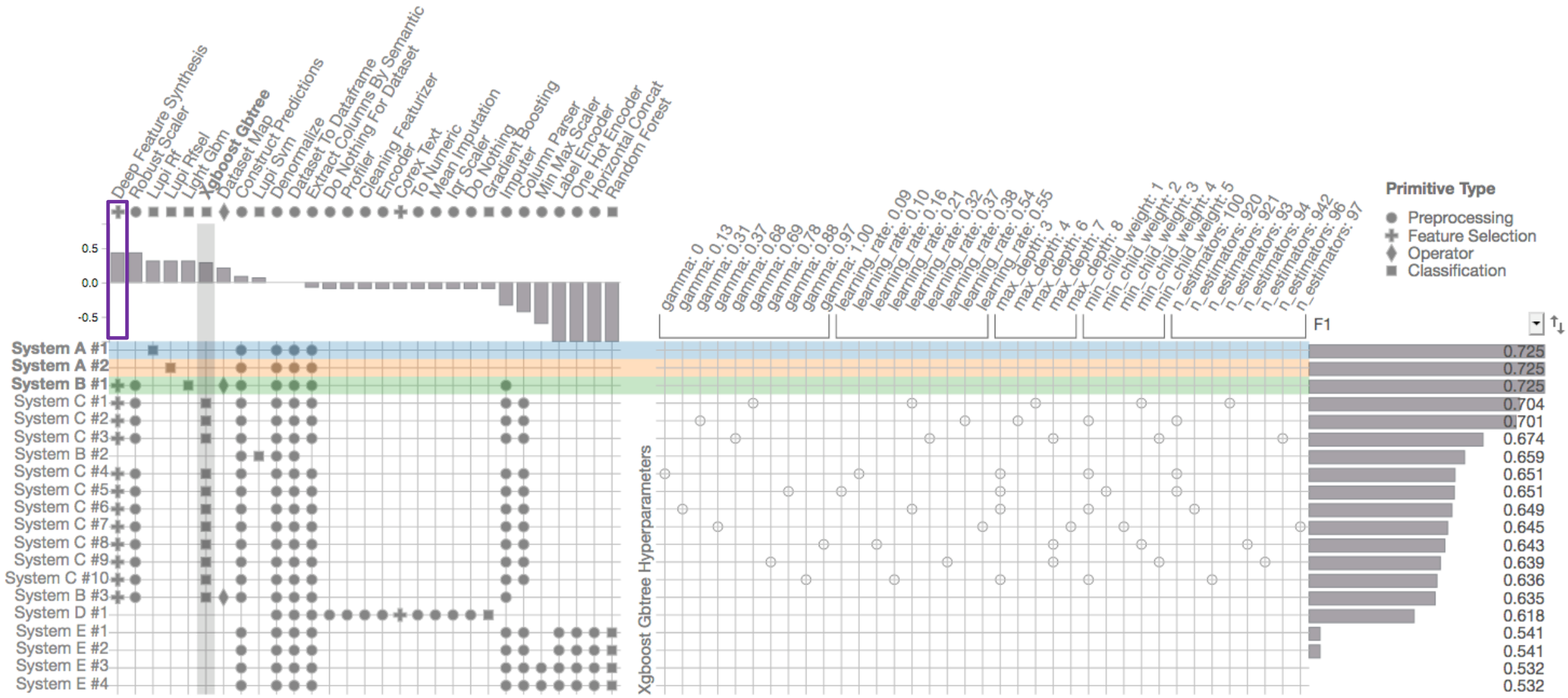

Deep Feature Synthesis
Robust Scaler
Lupi Rf
Lupi Rfsel
Light Gbm
Xgboost Gbtree
Dataset Map
Construct Predictions
Lupi Svm
Denormalize
Dataset To Dataframe
Extract Columns By Semantic
Do Nothing For Dataset
Profiler
Cleaning Featurizer
Encoder
Corex Text
To Numeric
Mean Imputation
Iqr Scaler
Do Nothing
Gradient Boosting
Imputer
Column Parser
Min Max Scaler
Label Encoder
One Hot Encoder
Horizontal Concat
Random Forest
0.5
0.0
-0.5
System A #1
System A #2
System B #1
System C #1
System C #2
System C #3
System B #2
System C #4
System C #5
System C #6
System C #7
System C #8
System C #9
System C #10
System B #3
System D #1
System E #1
System E #2
System E #3
System E #4
gamma: 0
gamma: 0.13
gamma: 0.31
gamma: 0.37
gamma: 0.68
gamma: 0.69
gamma: 0.78
gamma: 0.88
gamma: 0.97
gamma: 1.00
learning_rate: 0.09
learning_rate: 0.10
learning_rate: 0.16
learning_rate: 0.21
learning_rate: 0.32
learning_rate: 0.37
learning_rate: 0.38
learning_rate: 0.54
learning_rate: 0.55
max_depth: 3
max_depth: 4
max_depth: 6
max_depth: 7
max_depth: 8
min_child_weight: 1
min_child_weight: 2
min_child_weight: 3
min_child_weight: 4
min_child_weight: 5
n_estimators: 100
n_estimators: 920
n_estimators: 921
n_estimators: 93
n_estimators: 94
n_estimators: 942
n_estimators: 96
n_estimators: 97
F1
Xgboost Gbtree Hyperparameters
Primitive Type
Preprocessing
Feature Selection
Operator
Classification
Pipeline Scores
0.725
0.725
0.725
0.704
0.701
0.674
0.659
0.651
0.651
0.649
0.645
0.643
0.639
0.636
0.635
0.618
0.541
0.541
0.532
0.532

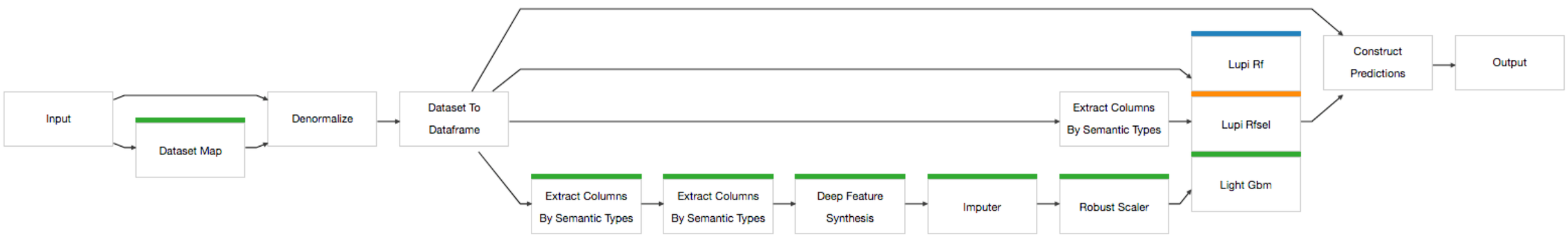

Input
Dataset Map
Denormalize
Dataset To Dataframe
Extract Columns By Semantic Types
Extract Columns By Semantic Types
Deep Feature Synthesis
Imputer
Robust Scaler
Extract Columns By Semantic Types
Lupi Rf
Lupi Rfsel
Light Gbm
Construct Predictions
Output

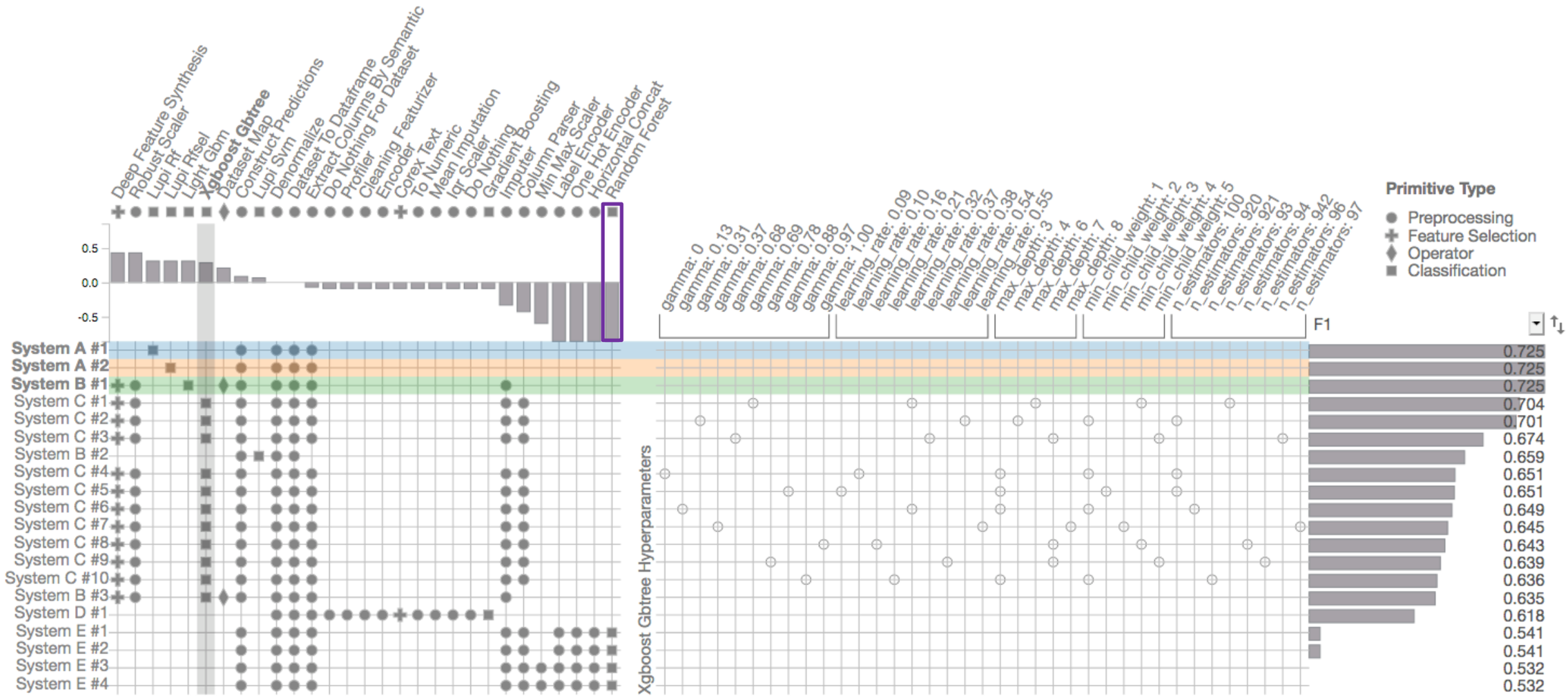
Primitive Contribution

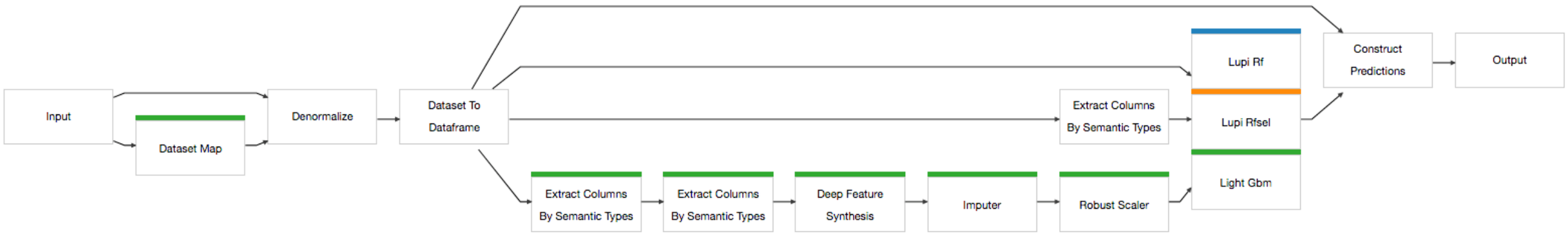





Deep Feature Synthesis
Robust Scaler
Lupi Rf
Lupi Rfsel
Light Gbm
Xgboost Gbtree
Dataset Map
Construct Predictions
Lupi Svm
Denormalize
Dataset To Dataframe
Extract Columns By Semantic
Do Nothing For Dataset
Profiler
Cleaning Featurizer
Encoder
Corex Text
To Numeric
Mean Imputation
Iqr Scaler
Do Nothing
Gradient Boosting
Imputer
Column Parser
Min Max Scaler
Label Encoder
One Hot Encoder
Horizontal Concat
Random Forest
0.5
0.0
-0.5
gamma: 0
gamma: 0.13
gamma: 0.31
gamma: 0.37
gamma: 0.68
gamma: 0.69
gamma: 0.78
gamma: 0.88
gamma: 0.97
gamma: 1.00
learning_rate: 0.09
learning_rate: 0.10
learning_rate: 0.16
learning_rate: 0.21
learning_rate: 0.32
learning_rate: 0.37
learning_rate: 0.38
learning_rate: 0.54
learning_rate: 0.55
max_depth: 3
max_depth: 4
max_depth: 6
max_depth: 7
max_depth: 8
min_child_weight: 1
min_child_weight: 2
min_child_weight: 3
min_child_weight: 4
min_child_weight: 5
n_estimators: 100
n_estimators: 920
n_estimators: 921
n_estimators: 93
n_estimators: 94
n_estimators: 942
n_estimators: 96
n_estimators: 97
Primitive Type
Preprocessing
Feature Selection
Operator
Classification
F1
System A #1
System A #2
System B #1
System C #1
System C #2
System C #3
System B #2
System C #4
System C #5
System C #6
System C #7
System C #8
System C #9
System C #10
System B #3
System D #1
System E #1
System E #2
System E #3
System E #4
Xgboost Gbtree Hyperparameters
0.725
0.725
0.725
0.704
0.701
0.674
0.659
0.651
0.651
0.649
0.645
0.643
0.639
0.636
0.635
0.618
0.541
0.541
0.532
0.532


Input
Dataset Map
Denormalize
Dataset To Dataframe
Extract Columns By Semantic Types
Extract Columns By Semantic Types
Deep Feature Synthesis
Imputer
Robust Scaler
Extract Columns By Semantic Types
Lupi Rf
Lupi Rfsel
Light Gbm
Construct Predictions
Output


Deep Feature Synthesis
Robust Scaler
Lupi Rf
Lupi Rfsel
Light Gbm
Xgboost Gbtree
Dataset Map
Construct Predictions
Lupi Svm
Denormalize
Dataset To Dataframe
Extract Columns By Semantic
Do Nothing For Dataset
Profiler
Cleaning Featurizer
Encoder
Corex Text
To Numeric
Mean Imputation
Iqr Scaler
Do Nothing
Gradient Boosting
Imputer
Column Parser
Min Max Scaler
Label Encoder
One Hot Encoder
Horizontal Concat
Random Forest
0.5
0.0
-0.5
gamma: 0
gamma: 0.13
gamma: 0.31
gamma: 0.37
gamma: 0.68
gamma: 0.69
gamma: 0.78
gamma: 0.88
gamma: 0.97
gamma: 1.00
learning_rate: 0.09
learning_rate: 0.10
learning_rate: 0.16
learning_rate: 0.21
learning_rate: 0.32
learning_rate: 0.37
learning_rate: 0.38
learning_rate: 0.54
learning_rate: 0.55
max_depth: 3
max_depth: 4
max_depth: 6
max_depth: 7
max_depth: 8
min_child_weight: 1
min_child_weight: 2
min_child_weight: 3
min_child_weight: 4
min_child_weight: 5
n_estimators: 100
n_estimators: 920
n_estimators: 921
n_estimators: 93
n_estimators: 94
n_estimators: 942
n_estimators: 96
n_estimators: 97
Primitive Type
Preprocessing
Feature Selection
Operator
Classification
F1
System A #1
System A #2
System B #1
System C #1
System C #2
System C #3
System B #2
System C #4
System C #5
System C #6
System C #7
System C #8
System C #9
System C #10
System B #3
System D #1
System E #1
System E #2
System E #3
System E #4
Xgboost Gbtree Hyperparameters
0.725
0.725
0.725
0.704
0.701
0.674
0.659
0.651
0.651
0.649
0.645
0.643
0.639
0.636
0.635
0.618
0.541
0.541
0.532
0.532


Input
Dataset Map
Denormalize
Dataset To Dataframe
Extract Columns By Semantic Types
Extract Columns By Semantic Types
Deep Feature Synthesis
Imputer
Robust Scaler
Extract Columns By Semantic Types
Lupi Rf
Lupi Rfsel
Light Gbm
Construct Predictions
Output

Primitive Contribution

Given a primitive *p*, and the pipeline scores *S*:

1. We compute the primitive indicator vector *P*, where $P_i = 1$ if *p* is used in pipeline *i,* and 0 otherwise.
2. The contribution of primitive *p* to the set of pipelines is given by

$$Contribution = Corr(P, S)$$

Primitive Contribution is useful for guiding the pipeline analysis. However, it does not convey primitive interactions.

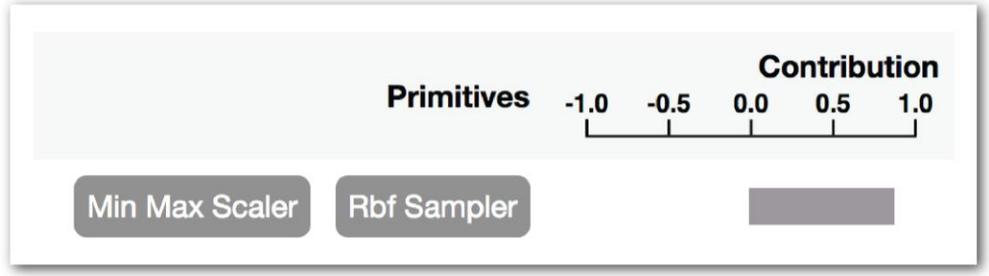
Combined Primitive Contribution

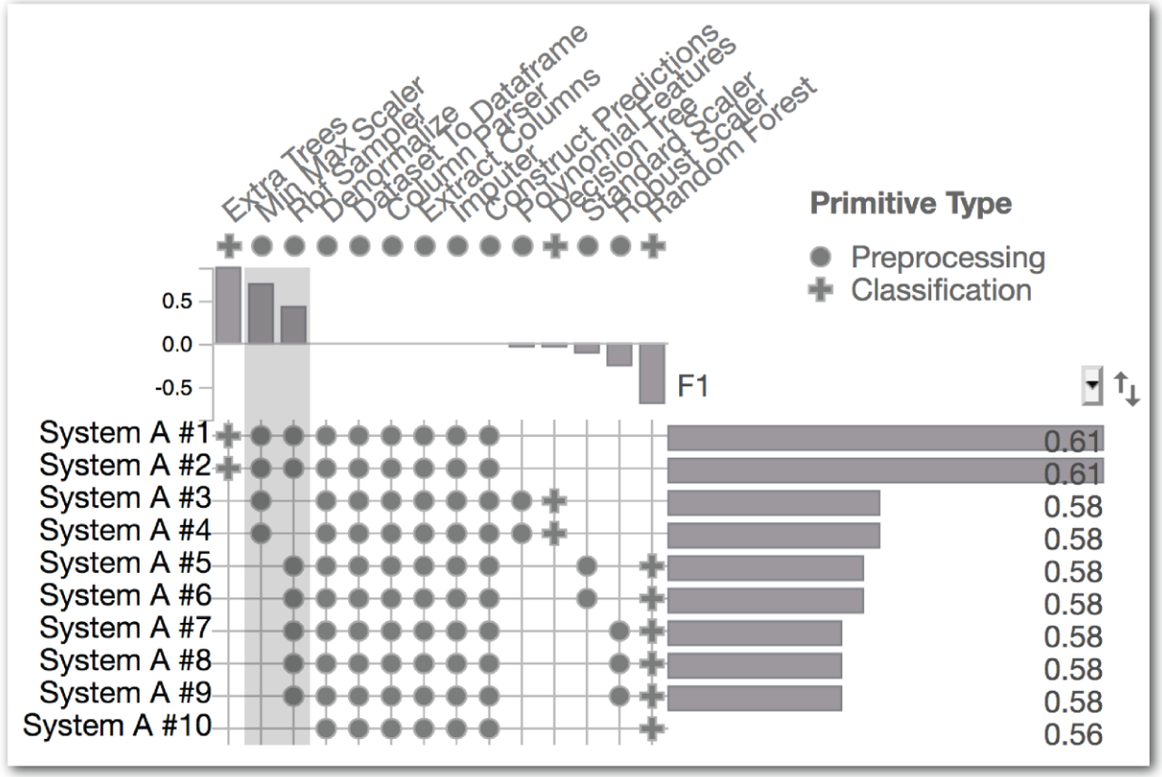
Idea: Investigate correlations of scores with the powerset of primitives

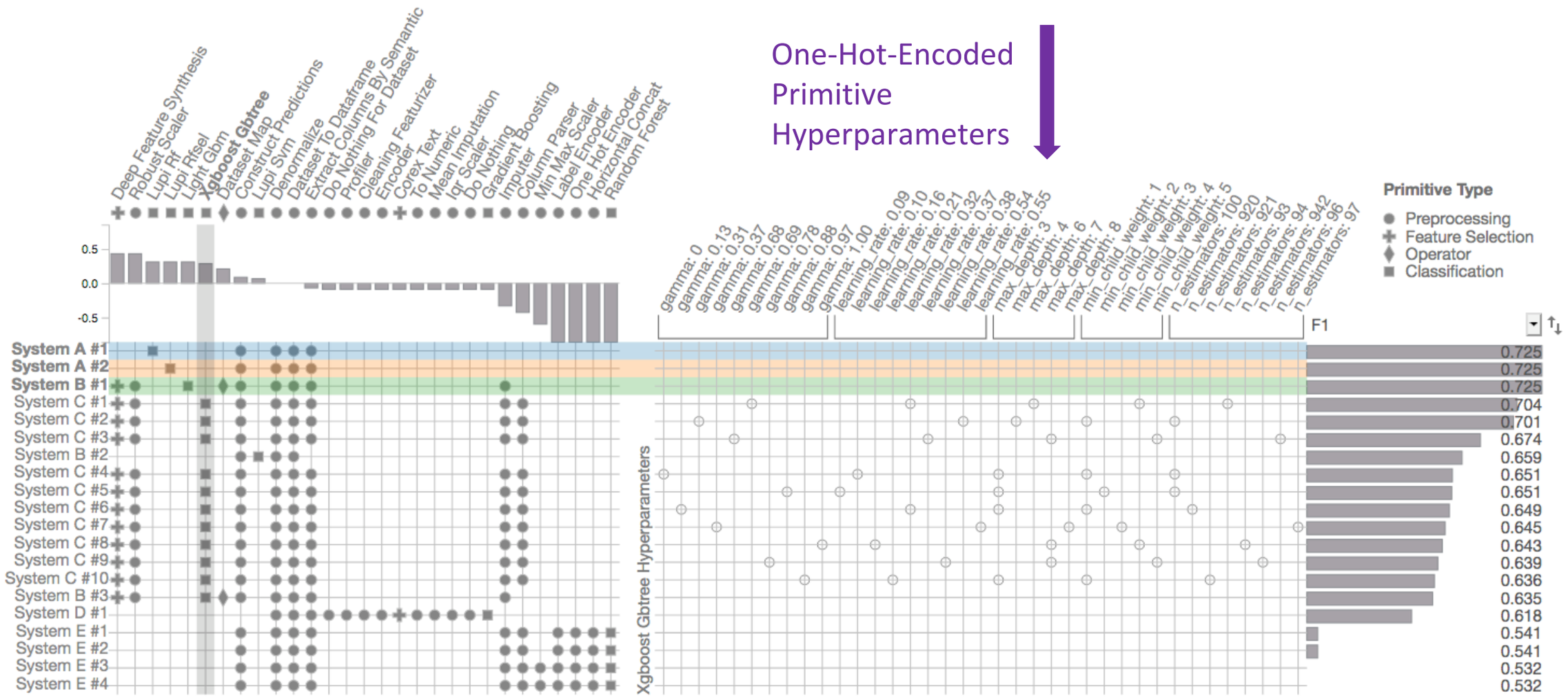
Only show surprising results.

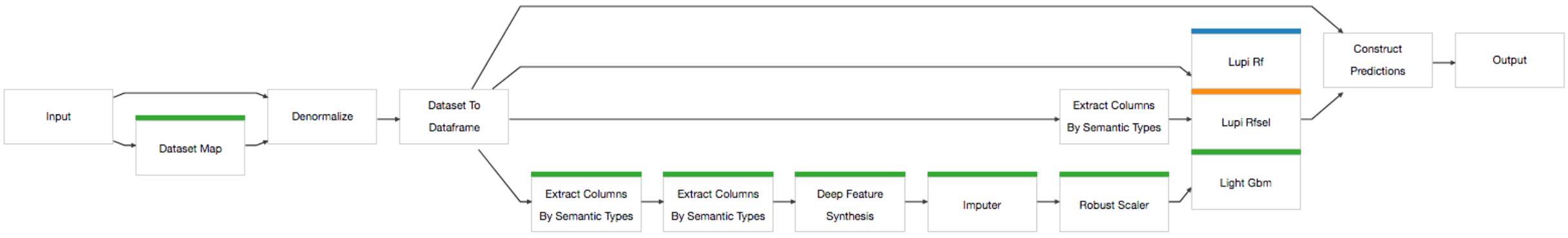
Given $g \in 2^P$

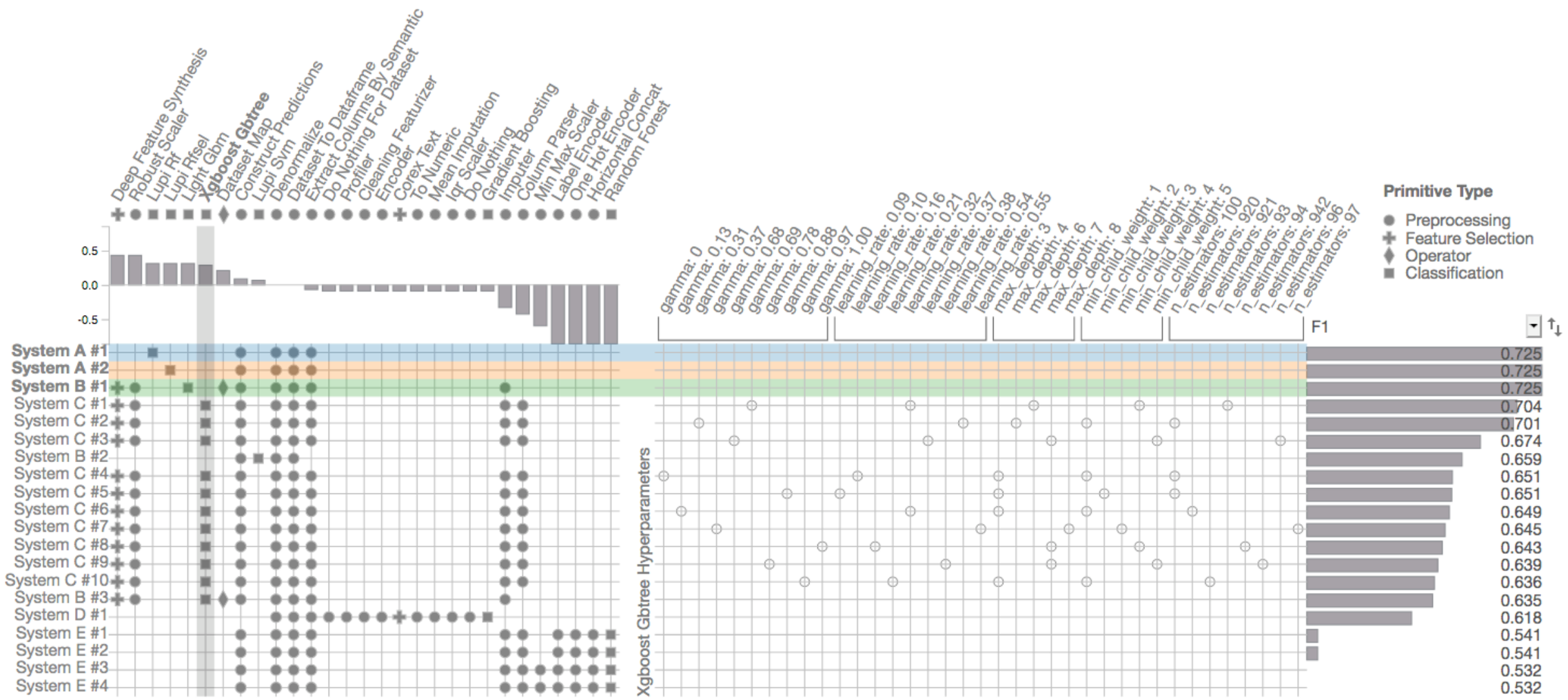
$$Corr(g, s) > Corr(g', s) \forall g' \in 2^g$$



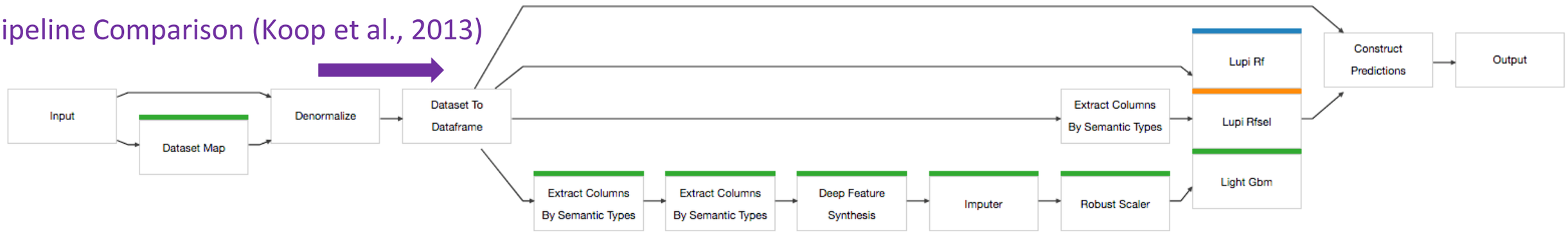



One-Hot-Encoded
Primitive
Hyperparameters
Deep Feature Synthesis
Robust Scaler
Lupi Rf
Lupi Rfsel
Light Gbm
Xgboost Gbtree
Dataset Map
Construct Predictions
Lupi Svm
Denormalize
Dataset To Dataframe
Extract Columns By Semantic
Do Nothing For Dataset
Profiler
Cleaning Featurizer
Encoder
Corex Text
To Numeric
Mean Imputation
Iqr Scaler
Do Nothing
Gradient Boosting
Imputer
Column Parser
Min Max Scaler
Label Encoder
One Hot Encoder
Horizontal Concat
Random Forest
gamma: 0
gamma: 0.13
gamma: 0.31
gamma: 0.37
gamma: 0.68
gamma: 0.69
gamma: 0.78
gamma: 0.88
gamma: 0.97
gamma: 1.00
learning_rate: 0.09
learning_rate: 0.10
learning_rate: 0.16
learning_rate: 0.21
learning_rate: 0.32
learning_rate: 0.37
learning_rate: 0.38
learning_rate: 0.54
learning_rate: 0.55
max_depth: 3
max_depth: 4
max_depth: 6
max_depth: 7
max_depth: 8
min_child_weight: 1
min_child_weight: 2
min_child_weight: 3
min_child_weight: 4
min_child_weight: 5
n_estimators: 100
n_estimators: 920
n_estimators: 921
n_estimators: 93
n_estimators: 94
n_estimators: 942
n_estimators: 96
n_estimators: 97
Primitive Type
Preprocessing
Feature Selection
Operator
Classification
F1
0.5
0.0
-0.5
System A #1
System A #2
System B #1
System C #1
System C #2
System C #3
System B #2
System C #4
System C #5
System C #6
System C #7
System C #8
System C #9
System C #10
System B #3
System D #1
System E #1
System E #2
System E #3
System E #4
Xgboost Gbtree Hyperparameters
0.725
0.725
0.725
0.704
0.701
0.674
0.659
0.651
0.651
0.649
0.645
0.643
0.639
0.636
0.635
0.618
0.541
0.541
0.532
0.532


Input
Dataset Map
Denormalize
Dataset To Dataframe
Extract Columns By Semantic Types
Extract Columns By Semantic Types
Deep Feature Synthesis
Imputer
Robust Scaler
Extract Columns By Semantic Types
Lupi Rf
Lupi Rfsel
Light Gbm
Construct Predictions
Output



Pipeline Comparison (Koop et al., 2013)



PipelineProfiler

Challenges Addressed:

- Requires programming background
- Black box on top of Black box (hard to explain)
- No user feedback (how to adapt the models based on human knowledge)

Limitation:

- User needs Python background

Ono, Jorge Piazentin, et al. "PipelineProfiler: A Visual Analytics Tool for the Exploration of AutoML Pipelines."

Lab – Pipeline Profiler & Auto-Sklearn

Material

https://colab.research.google.com/drive/1_2FRIkHNFGOiIJt-n_3zuh8vpSMLhwzx?usp=sharing