Pipeline Combinators for Gradual AutoML

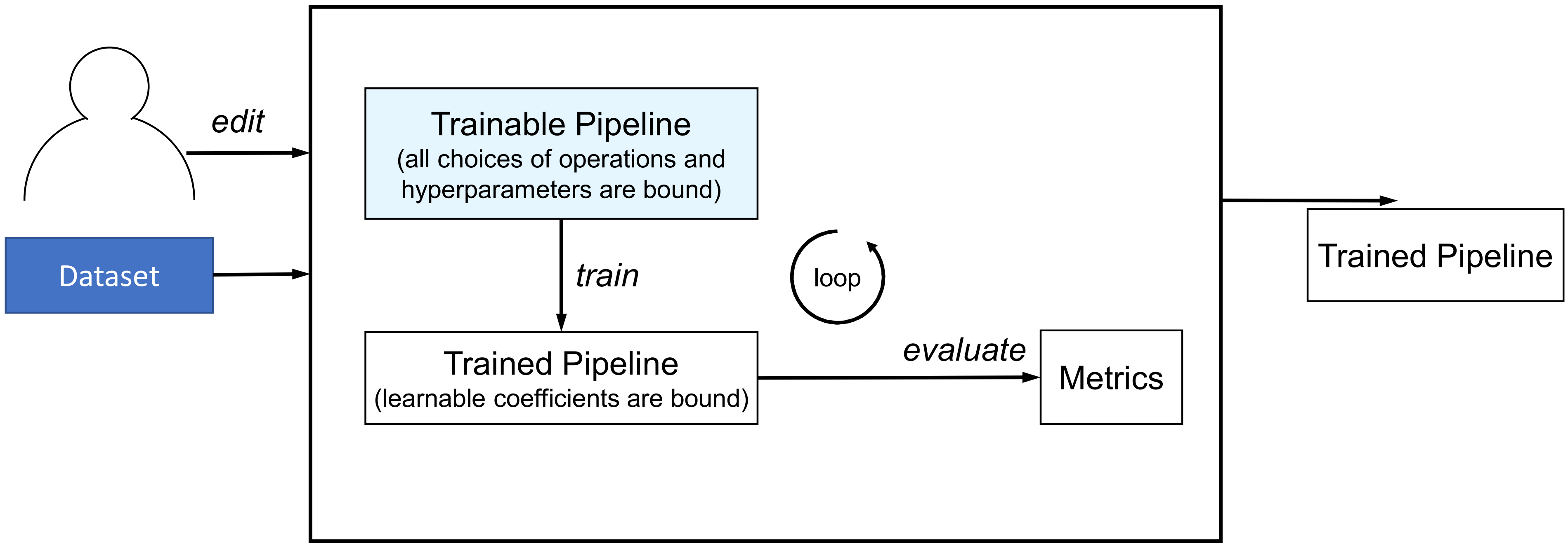
Guillaume Baudart, Martin Hirzel, Kiran Kate, Parikshit Ram, Avraham Shinnar, and Jason Tsay

IBM Research

Presentation at NeurIPS on 9 December 2021

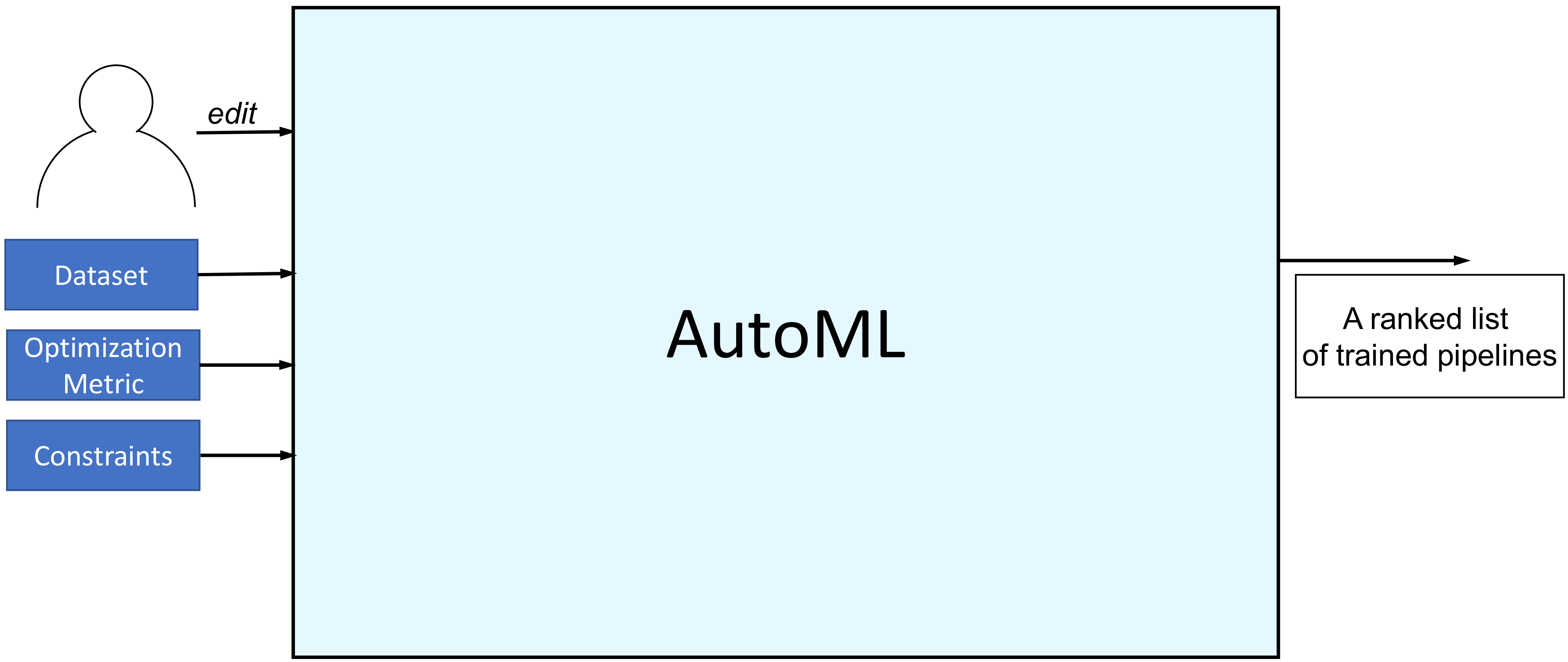
https://github.com/IBM/lale

Manual Machine Learning

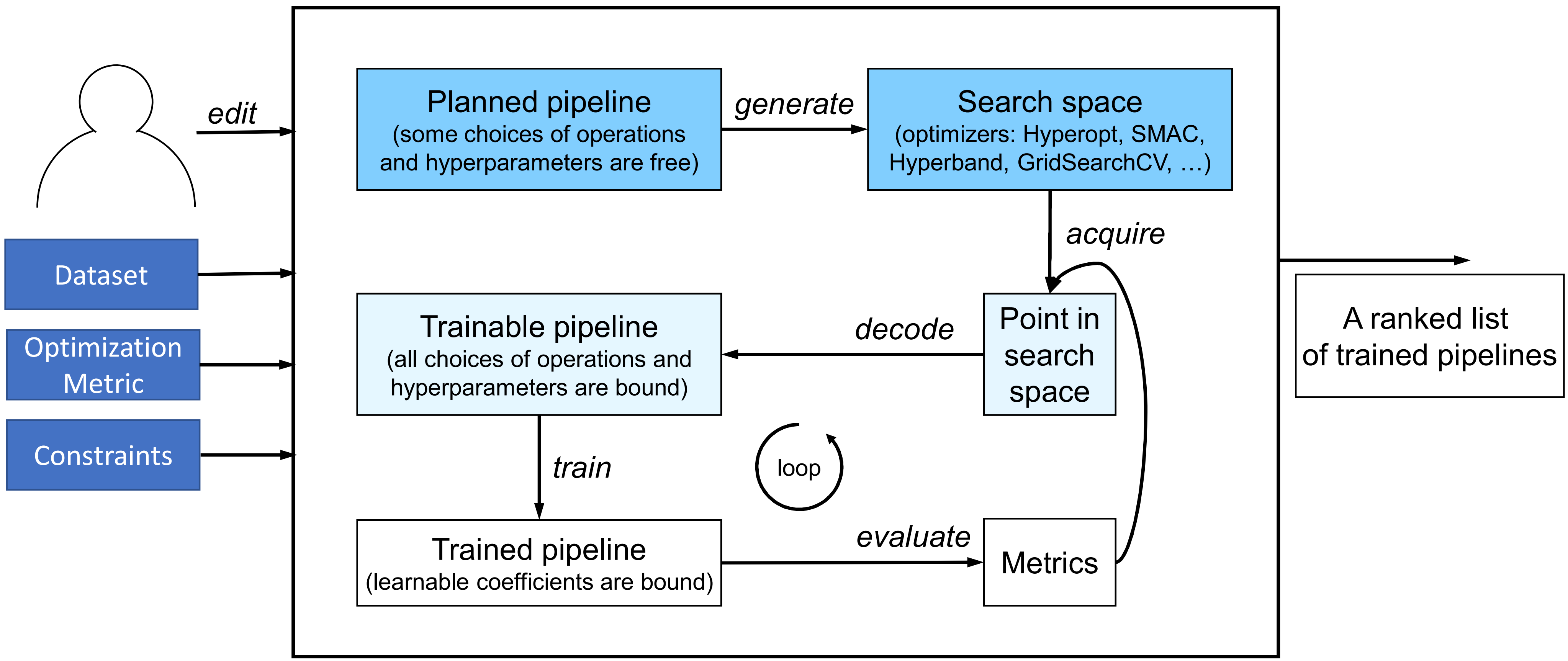


Pipeline = dataflow graph composed of multiple ML operations

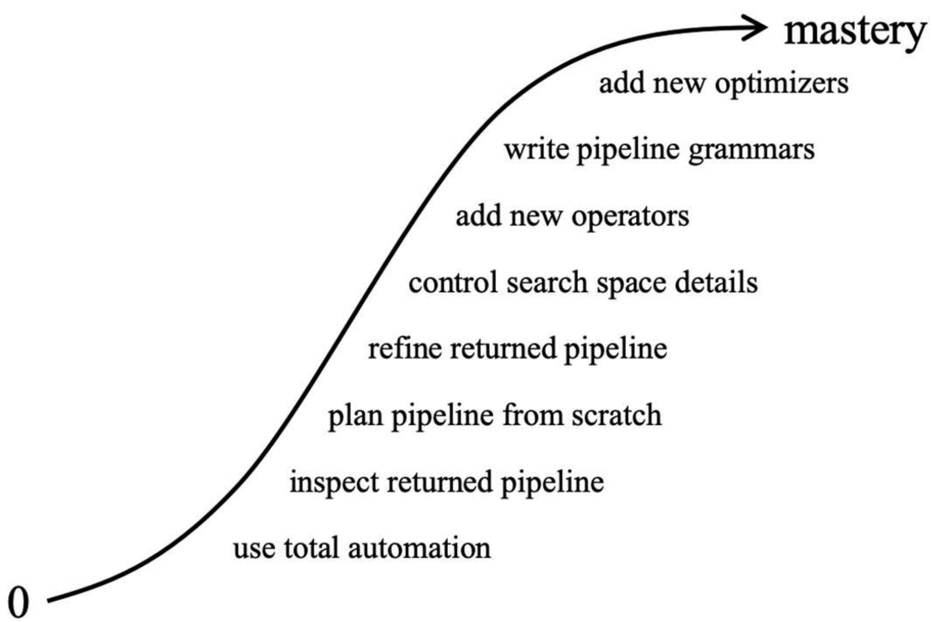
Automated Machine Learning (AutoML)



Automated Machine Learning (AutoML)

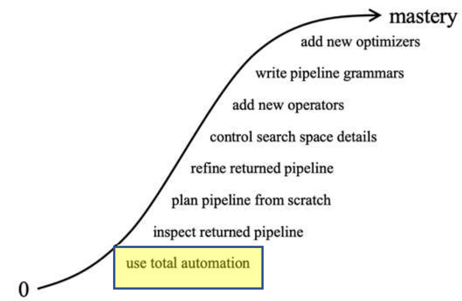


Programming Model for Gradual AutoML



- Progressive disclosure
    - Start with a small subset of constructs, learn more as required for higher levels of control
- Orthogonality
    - Minimize the number of independent constructs
    - Maximize compositionality for expressiveness
- Principle of least surprise
    - Make each construct look and behave the way most users would expect

Total Automation



In [2]:

```python
from lale.lib.lale import AutoPipeline
trainable = AutoPipeline(prediction_type='classification',
                         scoring='accuracy', max_opt_time=90)
```

In [3]:

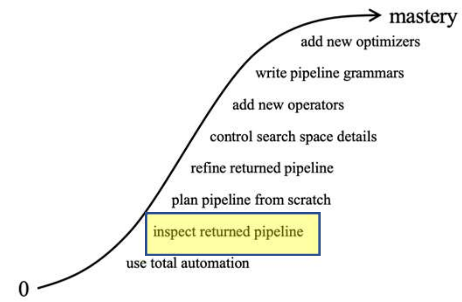
```python
trained = trainable.fit(train_X, train_y)
```

In [4]:

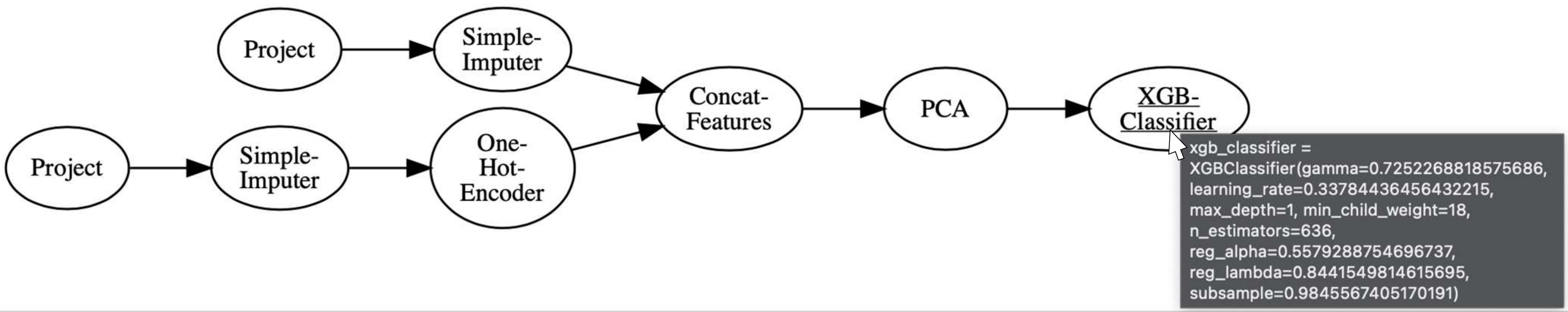
```python
predicted = trained.predict(test_X)
```

- Four lines of code to run AutoML and find the best pipeline for a given task
- The set of operators and the graph topology is pre-defined

Inspect Pipelines

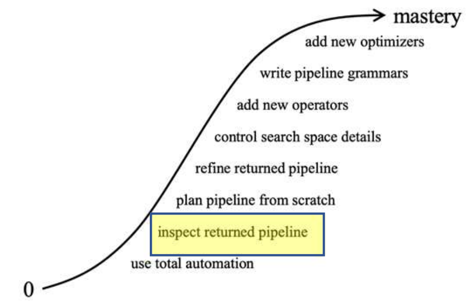


In [5]:
```
pipeline.visualize()
```



- Visualize pipelines and inspect the hyperparameter values

Inspect Pipelines

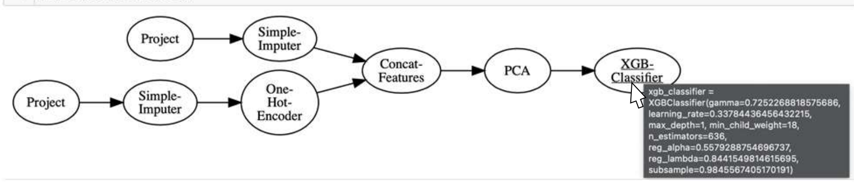


In [8]:
```
pipeline.pretty_print(ipython_display=True)
```

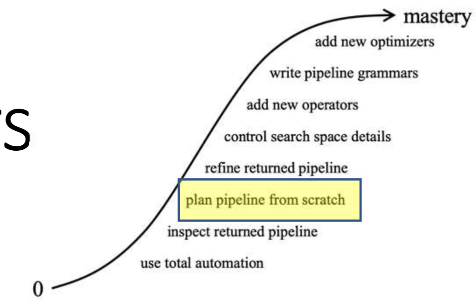
```
from lale.lib.lale import Project
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder
from lale.lib.lale import ConcatFeatures
from sklearn.decomposition import PCA
from xgboost import XGBClassifier
import lale

lale.wrap_imported_operators()
project_0 = Project(
    columns={"type": "number"},
    drop_columns=lale.lib.lale.categorical(max_values=5),
)
project_1 = Project(columns=lale.lib.lale.categorical(max_values=5))
simple_imputer_1 = SimpleImputer(strategy="most_frequent")
one_hot_encoder = OneHotEncoder(handle_unknown="ignore")
pca = PCA(svd_solver="full", whiten=True)
xgb_classifier = XGBClassifier(
    gamma=0.7252268818575686,
    learning_rate=0.33784436456432215,
    max_depth=1,
    min_child_weight=18,
    n_estimators=636,
    reg_alpha=0.5579288754696737,
    reg_lambda=0.8441549814615695,
    subsample=0.9845567405170191,
)
pipeline = (
    (
        (project_0 >> SimpleImputer())
        & (project_1 >> simple_imputer_1 >> one_hot_encoder)
    )
    >> ConcatFeatures()
    >> pca
    >> xgb_classifier
)
```

Get Python code as output by calling `pretty_print` on a pipeline object

In [5]:
```
pipeline.visualize()
```



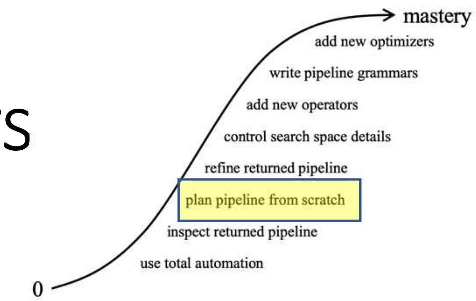
Create a Pipeline from Scratch: Combinators



- Functions that compose machine learning operators
- Imported from functional programming
- Enable a tacit programming style with function compositions independent of data
- Three main combinators in Lale:

Combinator	Description
`op3 = op1 >> op2`	pipe (add dataflow edge)
`op3 = op1 & op2`	union (no added edge)
`op3 = op1 | op2`	choice (AutoML picks one)

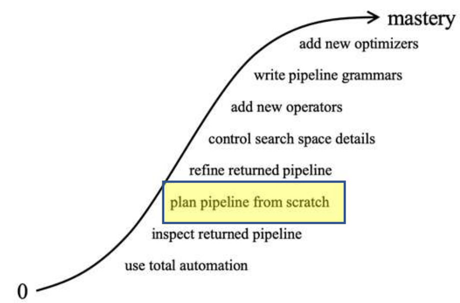
Create a Pipeline from Scratch: Combinators



In [2]:

```
pre_n = (Project(columns={"type":"number"})
```

Execution Modes



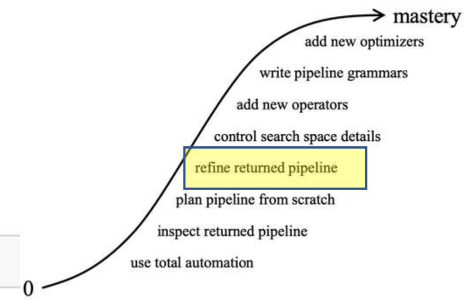
- Fit and predict similar to Scikit-learn
- Third execution mode for AutoML search:

In [6]:
```
best_found = pipeline.auto_configure(
    train_X, train_y, optimizer=Hyperopt, cv=3,
    scoring='accuracy', max_opt_time=300)
```

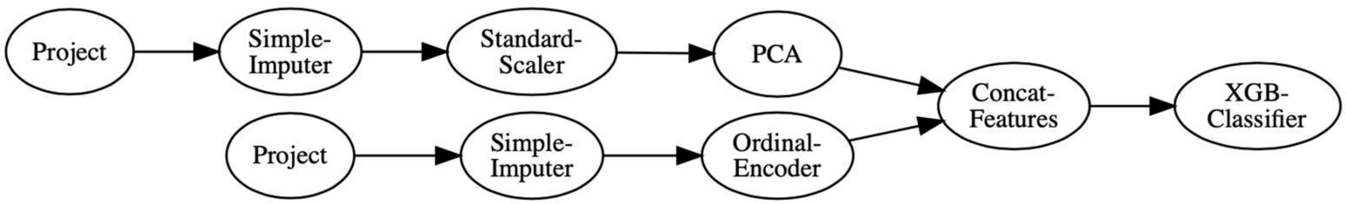
In [7]:
```
best_found.predict(test_X)
```

In [8]:
```
best_found.visualize()
```

Refine a Pipeline



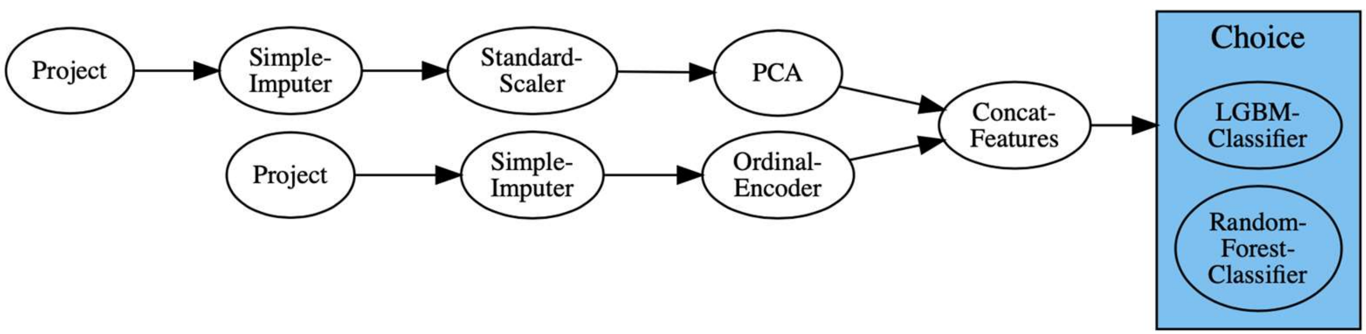
In [8]:
```
best_found.visualize()
```



In [9]:
```
pipeline_prefix = best_found.remove_last()
```

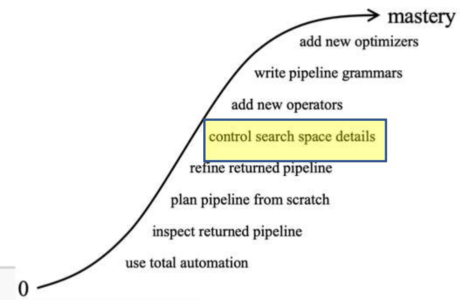
In [10]:
```
new_pipeline = pipeline_prefix >> (LGBMClassifier | RandomForestClassifier)
```

In [11]:
```
new_pipeline.visualize()
```

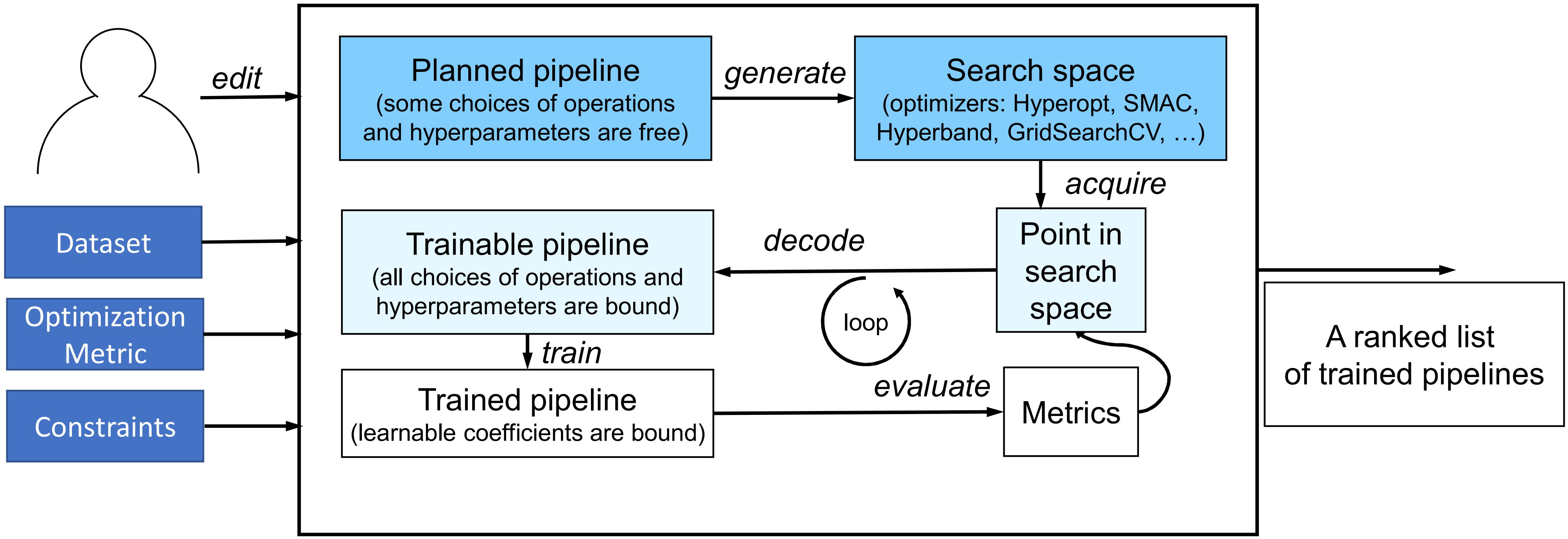


- Functions which decompose a pipeline
- Can be used with the combinators to refine pipelines

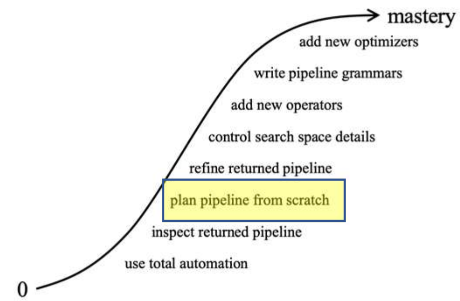
Search Spaces and Optimizers



In [6]:
```
best_found = pipeline.auto_configure(
    train_X, train_y, optimizer=Hyperopt, cv=3,
    scoring='accuracy', max_opt_time=300)
```

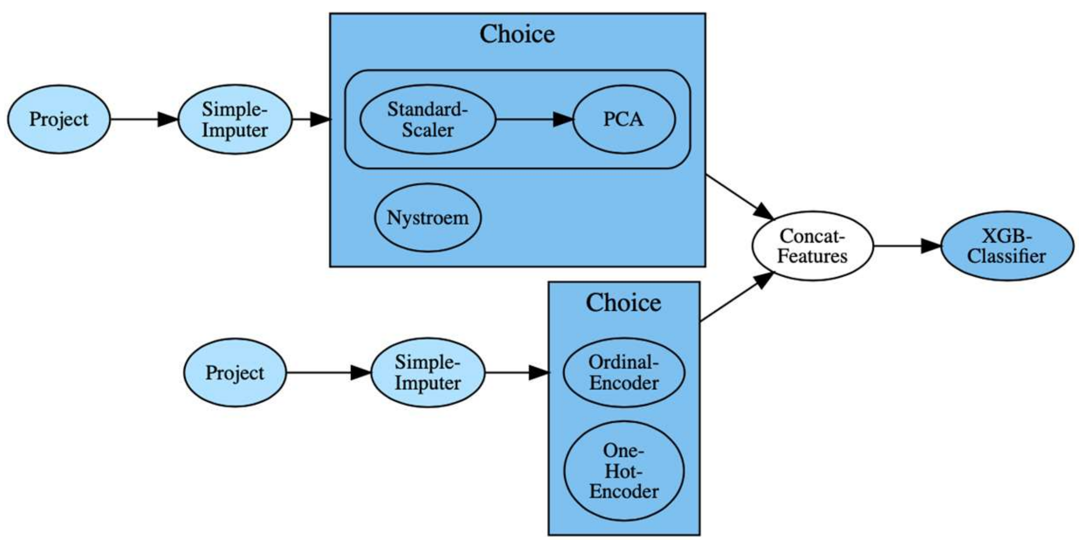


Pipeline -> Search Space

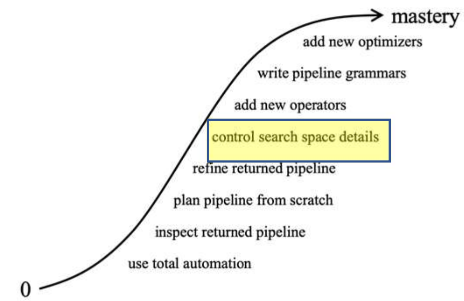


In [2]:

```
pre_n = (Project(columns={"type":"number"})
              >> SimpleImputer(strategy="mean")
              >> ((StandardScaler >> PCA) | Nystroem))
pre_s = (Project(columns={"type":"string"})
              >> SimpleImputer(strategy="most_frequent")
              >> (OrdinalEncoder|OneHotEncoder))
pipeline = ((pre_n & pre_s)
              >> ConcatFeatures
              >> XGBClassifier)
```



Pipeline -> Search Space: Combinators



In [2]:

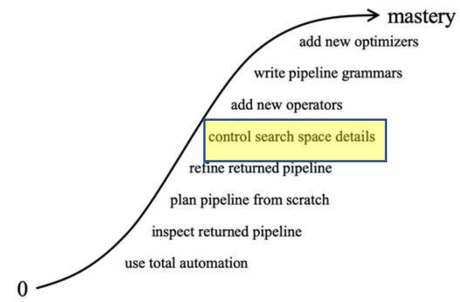
```
pre_n = (Project(columns={"type":"number"})
            >> SimpleImputer(strategy="mean")
            >> ((StandardScaler >> PCA) | Nystroem))
pre_s = (Project(columns={"type":"string"})
            >> SimpleImputer(strategy="most_frequent")
            >> (OrdinalEncoder|OneHotEncoder))
pipeline = ((pre_n & pre_s)
            >> ConcatFeatures
            >> XGBClassifier)
```

- Combinators “glue” together operators.
- We automatically ”glue” together individual search spaces appropriately.

Combinator	Description
`op3 = op1 >> op2`	pipe (add dataflow edge)
`op3 = op1 & op2`	union (no added edge)
`op3 = op1 | op2`	choice (AutoML picks one)

More details in the paper

Hyperparameter (JSON) Schemas

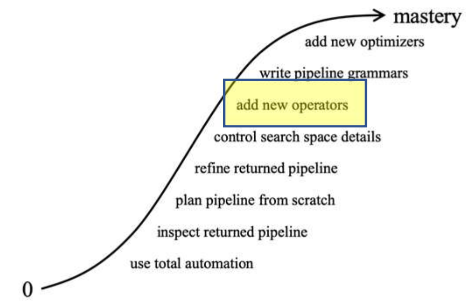



```
SimpleImputer: {"type" : "object",
"properties": {
 "strategy":  {
    "enum": ["constant ", "mean",
                 "median", "most_frequent"],
    "default": "mean”},
 "fill_value": {
   "anyOf": [
       {"type": "number"},
       {"type": "string"},
       {"enum": [None]}],
   "default": None,
}
```

```
PCA: {"type" : "object",
"properties”
   "n_components": {
     "anyOf": [
       { "enum": [None, ["mle"]] },
       {  "type": "number",
          "minimum": 0.0,
          "exclusiveMinimum":
True,
          "maximum": 1.0,
          "exclusiveMaximum":
True,
       },
       { "type": "integer",
          "minimum": 1

        },
     ],
     "default": None,
   }
}
```

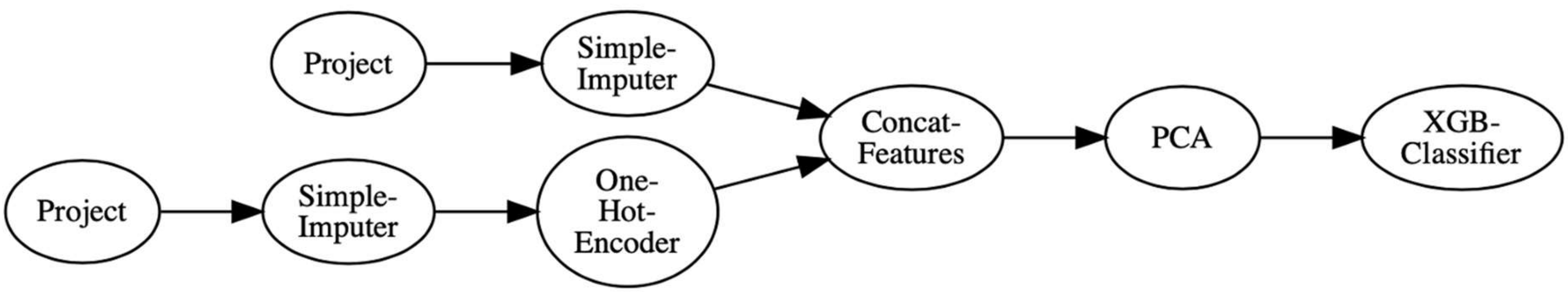
We also support:
- Description fields
- Validation only specifications
- Inter-parameter constraints
- Data dependent specifications
- Lists, tuples, and arrays
- Nested operators
- “Any”
- …

Operator Libraries

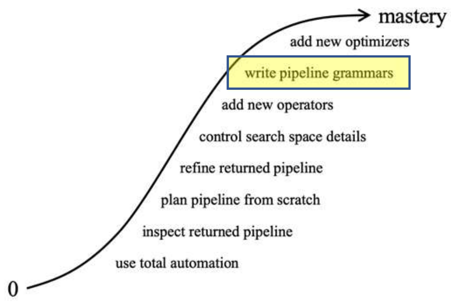


make_operator(implementation, schemas)

- Lale currently has 216 operators from sklearn, imblearn, aif360, lightgbm, xgboost etc.
- The operators can also be deep neural networks.
- Papers at AutoML@ICML'20 and SecML@ICLR'21 on inferring schemas from documentation and static analysis of code.



Operator Grammars



- Extends combinators to allow for recursion.
- Enables pipeline topology search with a simple compact representation

Combinator	Description
`op3 = op1 >> op2`	pipe (add dataflow edge)
`op3 = op1 & op2`	union (no added edge)
`op3 = op1 | op2`	choice (AutoML picks one)
`g.op3 = … g.op3 …`	Recursive reference

Experiments

- Does the translation scheme work for diverse planned pipelines?

Dataset	Absolute accuracy mean (and stddev) over 5 runs						100 ∗ (LALE/AUTOSKL − 1)			
	AUTOSKL	LALE-AUTO	LALE-TPOT	LALE-AD3M	LALE-ADB	ASKL-ADB	AUTO	TPOT	AD3M	ADB
australian	85.1(0.4)	86.2 (0.0)	85.9 (0.6)	86.8 (0.0)	86.0 (1.6)	84.7 (3.1)	1.3	0.9	2.0	1.1
blood	77.9(1.4)	75.3 (0.0)	77.5 (2.5)	74.7 (0.7)	77.1 (0.7)	74.7 (0.8)	-3.3	-0.5	-4.0	-1.0
breast-cancer	73.0(0.6)	73.0 (0.0)	71.4 (1.1)	69.5 (3.3)	70.9 (2.0)	72.4 (0.5)	0.0	-2.3	-4.9	-2.9
car	99.4(0.1)	97.7 (0.0)	99.1 (0.1)	92.7 (0.6)	98.3 (0.3)	98.2 (0.2)	-1.6	-0.2	-6.7	-1.1
credit-g	76.6(1.2)	75.7 (0.0)	74.1 (0.5)	74.8 (0.4)	76.1 (1.3)	76.2 (1.0)	-1.1	-3.2	-2.4	-0.7
diabetes	77.0(1.3)	76.3 (0.0)	76.4 (1.1)	77.9 (0.2)	76.0 (0.5)	75.0 (1.0)	-0.9	-0.8	1.1	-1.3
jungle-chess	88.1(0.2)	92.4 (0.0)	88.9 (2.0)	74.1 (2.0)	89.4 (2.3)	86.9 (0.2)	4.9	0.9	-15.8	1.5
kc1	83.8(0.3)	83.4 (0.0)	83.5 (0.5)	83.6 (0.2)	83.3 (0.4)	84.0 (0.3)	-0.5	-0.4	-0.2	-0.6
kr-vs-kp	99.7(0.0)	99.5 (0.0)	99.4 (0.0)	96.8 (0.1)	99.5 (0.1)	99.5 (0.2)	-0.2	-0.3	-2.9	-0.2
mfeat-factors	98.7(0.1)	97.1 (0.0)	97.9 (0.5)	97.5 (0.1)	97.5 (0.4)	97.9 (0.1)	-1.6	-1.5	-1.2	-1.2
phoneme	90.3(0.4)	89.5 (0.0)	89.6 (0.4)	76.6 (0.0)	90.1 (0.4)	91.4 (0.2)	-0.8	-0.8	-15.2	-0.2
shuttle	87.3(11.6)	100.0 (0.0)	99.9 (0.0)	99.9 (0.0)	100.0 (0.0)	100.0 (0.0)	14.5	14.5	14.4	14.6
spectf	87.9(0.9)	87.7 (0.0)	88.4 (2.2)	83.6 (6.9)	88.4 (2.6)	89.7 (2.9)	-0.2	0.6	-4.9	0.6
sylvine	95.4(0.2)	95.0 (0.0)	94.4 (0.7)	91.3 (0.1)	95.1 (0.2)	95.1 (0.1)	-0.4	-1.1	-4.3	-0.3

- Does the translation scheme work for diverse optimizer backends?

TPE	Hyperopt Anneal	Rand	Hyperband	RND	ADMM BOBa	GPRND	GridSearchCV	Halving-GridSearchCV
88.75 (0.2)	88.94 (0.3)	88.75 (0.5)	87.94 (0.2)	89.00 (0.2)	89.14 (0.2)	88.61 (0.3)	Timed out	Timed out

User Study

- Between-subjects user study with 18 participants
- Four tasks to compare Lale to Scikit-learn
- The tasks covered basics of manual ML and AutoML

VERSION	COUNT	T1 CORRECT	T4 CORRECT	T4 LoC MEDIAN (ALL)	TOTAL TIME MEAN (STDDEV)
Lale	9	89%	100%	10 (7, 8, 9, 9, 10, 10, 10, 14, 75)	17:36 (5:03)
Sklearn	9	56%	78%	23 (12, 15, 16, 22, 24, 43, 100, 100)	19:26 (6:34)
Total	18	72%	89%	14	18:54 (5:45)

Lale

Library for semi-automated data science

https://github.com/IBM/lale