

# Type-Driven Automated Learning with LALE

---

Martin Hirzel, Kiran Kate, Avi Shinnar,  
Pari Ram, and Guillaume Baudart

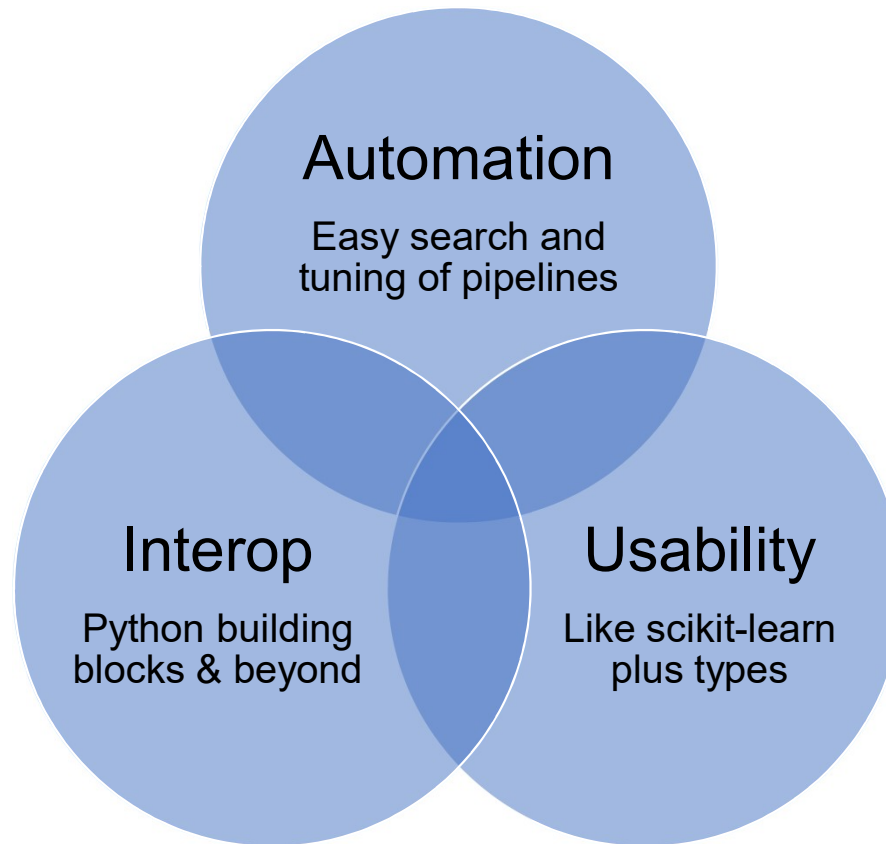
Friday 25 October 2019

<https://github.com/ibm/lale>



# Value Proposition

Augment, but  
don't replace, the  
data scientist.



# Categorical + Continuous Dataset

```
In [1]: 1 import lale.datasets.openml
2 import pandas as pd
3 (train_X, train_y), (test_X, test_y) = lale.datasets.openml.fetch(
4     'credit-g', 'classification', preprocess=False)
5 # print last five rows of labels in train_y and features in train_X
6 pd.concat([pd.DataFrame({'y': train_y}, index=train_X.index).tail(5),
7     train_X.tail(5)], axis=1)
```

Out[1]:

|     | y | checking_status | duration | credit_history                 | purpose             | credit_amount | savings_status   | employment | installment |
|-----|---|-----------------|----------|--------------------------------|---------------------|---------------|------------------|------------|-------------|
| 835 | 0 | <0              | 12.0     | no credits/all paid            | new car             | 1082.0        | <100             | 1<=X<4     |             |
| 192 | 0 | 0<=X<200        | 27.0     | existing paid                  | business            | 3915.0        | <100             | 1<=X<4     |             |
| 629 | 1 | no checking     | 9.0      | existing paid                  | education           | 3832.0        | no known savings | >=7        |             |
| 559 | 0 | 0<=X<200        | 18.0     | critical/other existing credit | furniture/equipment | 1928.0        | <100             | <1         |             |
| 684 | 1 | 0<=X<200        | 36.0     | delayed previously             | business            | 9857.0        | 100<=X<500       | 4<=X<7     |             |

5 rows × 21 columns

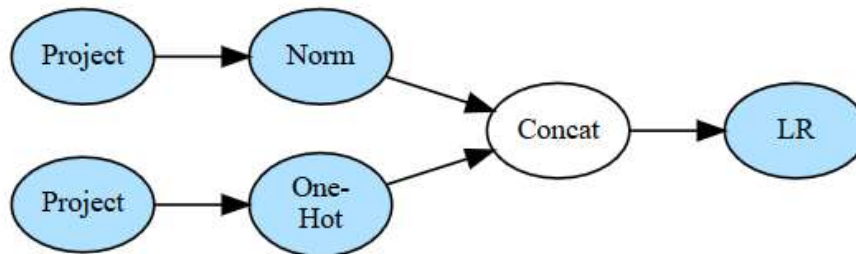
< >

[https://github.com/IBM/lale/blob/master/examples/talk\\_2019-1105-lale.ipynb](https://github.com/IBM/lale/blob/master/examples/talk_2019-1105-lale.ipynb)

# Manual Pipeline

```
In [4]: ► 1 manual_trainable = (  
2         ( Project(columns={'type': 'number'}) >> Norm()  
3           & Project(columns={'type': 'string'}) >> OneHot()  
4         >> Concat  
5         >> LR(LR.penalty.l1, C=0.001))  
6   lale.helpers.to_graphviz(manual_trainable)
```

Out[4]:



```
In [5]: ► 1 import sklearn.metrics  
2 manual_trained = manual_trainable.fit(train_X, train_y)  
3 manual_y = manual_trained.predict(test_X)  
4 print(f'accuracy {sklearn.metrics.accuracy_score(test_y, manual_y):.1%}')
```

accuracy 29.1%

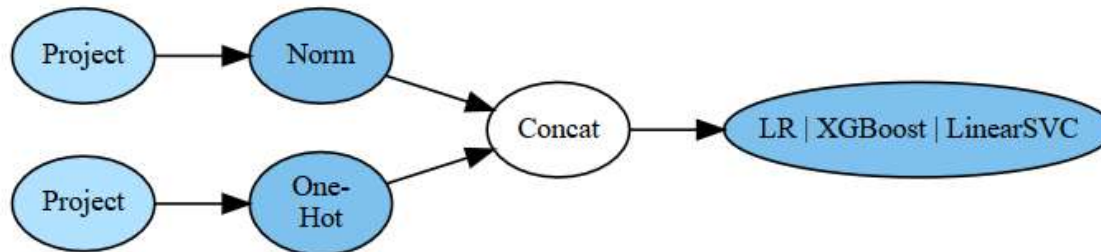
# Pipeline Combinators

| LAL features                                           | Name              | Description  | Scikit-learn features                                        |
|--------------------------------------------------------|-------------------|--------------|--------------------------------------------------------------|
| <code>&gt;&gt;</code> or<br><code>make_pipeline</code> | <code>pipe</code> | feed to next | <code>make_pipeline</code>                                   |
| <code>&amp;</code> or<br><code>make_union</code>       | <code>and</code>  | run both     | <code>make_union</code> or<br><code>ColumnTransformer</code> |
| <code> </code> or<br><code>make_choice</code>          | <code>or</code>   | choose one   | N/A (specific to given<br>Auto-ML tool)                      |

# Automated Pipeline

```
In [7]: ► 1 auto_planned = (  
2         ( Project(columns={'type': 'number'}) >> Norm  
3         & Project(columns={'type': 'string'}) >> OneHot)  
4         >> Concat  
5         >> (LR | XGBoost | LinearSVC))  
6     lale.helpers.to_graphviz(auto_planned)
```

Out[7]:



```
In [8]: ► 1 from lale.lib.lale.hyperopt_classifier import HyperoptClassifier  
2 auto_optimizer = HyperoptClassifier(auto_planned, cv=3, max_evals=10)  
3 auto_trained = auto_optimizer.fit(train_X, train_y)  
4 auto_y = auto_trained.predict(test_X)  
5 print(f'accuracy {sklearn.metrics.accuracy_score(test_y, auto_y):.1%}')
```

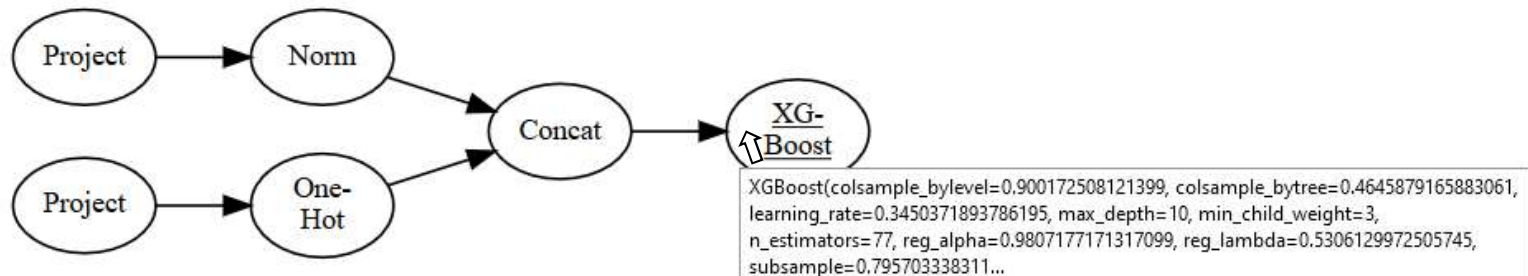
100%|██████████| 10/10 [00:30<00:00, 2.91s/it, best loss: -0.7373278347213325]  
accuracy 75.2%



# Displaying Automation Results

```
In [9]: 1 lale.helpers.to_graphviz(auto_trained)
```

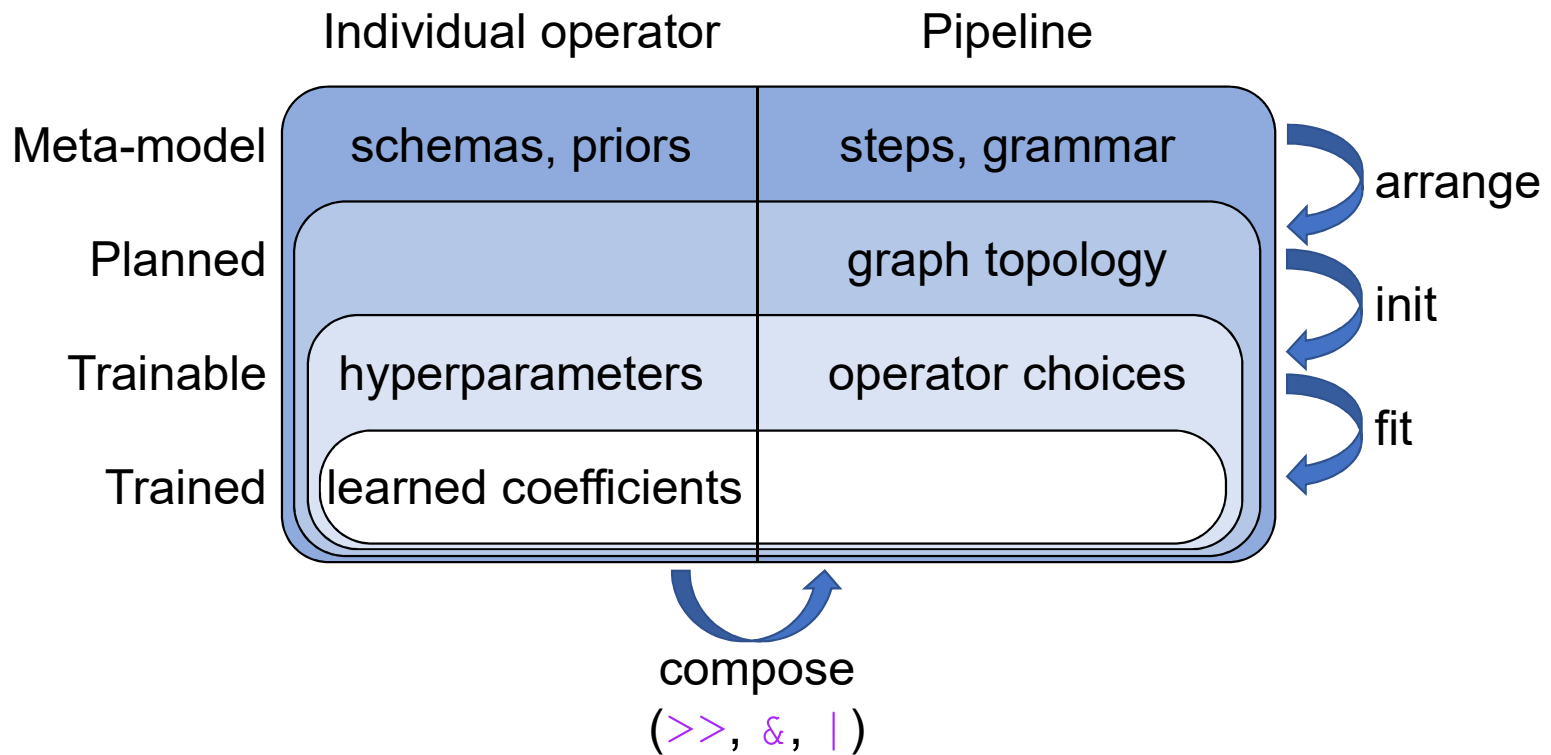
Out[9]:



```
In [10]: 1 import lale.pretty_print
2 lale.pretty_print.ipynon_display(auto_trained, show_imports=False)
```

```
project = Project(columns={'type': 'number'})
project_1 = Project_1(columns={'type': 'string'})
xgboost = XGBoost(colsample_bylevel=0.900172508121399, colsample_bytree=0.4645879165883061,
learning_rate=0.3450371893786195, max_depth=10, min_child_weight=3, n_estimators=77, reg_alpha=0.9807177171317099, reg_lambda=0.5306129972505745, subsample=0.7957033383112613)
pipeline = ((project >> Norm) & (project_1 >> OneHot)) >> Concat >> xgboost
```

# Bindings as Lifecycle: Venn Diagram

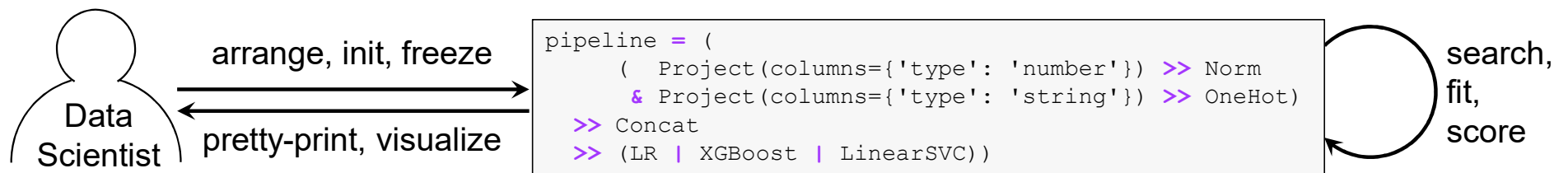


“Type-Driven Automated Learning with Lale”, <https://arxiv.org/pdf/1906.03957.pdf>



# Semi-Automated Data Science

| Manual control over automation      | Examples                                                                                                                        |
|-------------------------------------|---------------------------------------------------------------------------------------------------------------------------------|
| Restrict available operator choices | <ul style="list-style-type: none"><li>• Interpretable</li><li>• Based on licenses</li><li>• Based on GPU requirements</li></ul> |
| Tweak graph topology                | <ul style="list-style-type: none"><li>• Custom preprocessing</li><li>• Multi-modal data</li><li>• Fairness mitigation</li></ul> |
| Tweak hyperparameter schemas        | <ul style="list-style-type: none"><li>• Adjust range for continuous</li><li>• Restrict choices for categorical</li></ul>        |
| Expand available operator choices   | <ul style="list-style-type: none"><li>• Wrap existing library</li><li>• Write your own operators</li></ul>                      |



# Constraints in Scikit-learn

```
In [13]: ► 1 sklearn_misconfigured = sklearn.pipeline.make_pipeline(  
2         sklearn.feature_extraction.text.TfidfVectorizer(),  
3         sklearn.linear_model.LogisticRegression(solver='sag', penalty='l1'))  
4 print('no error detected yet')
```

no error detected yet

```
In [14]: ► 1 %%time  
2 import sys  
3 try:  
4     sklearn_misconfigured.fit(news_X, news_y)  
5 except ValueError as e:  
6     print(e, file=sys.stderr)
```

CPU times: user 3.61 s, sys: 172 ms, total: 3.78 s

Wall time: 3.96 s

Solver sag supports only l2 penalties, got l1 penalty.

# Constraints in Auto-ML

**Problem:** Some automated trials raise exceptions

**Solution 1:** Unconstrained search space

- $\{solver: [linear, sag, lbfgs], penalty: [l1, l2]\}$
- Catch exception (after some time)
- Return made-up loss `np.float.max`

**Solution 2:** Constrained search space

- $\{solver: [linear, sag, lbfgs], penalty: [l1, l2]\}$  **and** (if  $solver: [sag, lbfgs]$  **then**  $penalty: [l2]$ )
- No exceptions (no time wasted)
- No made-up loss

# Constraints in LALE

In [16]:

```
1 %%time
2 import jsonschema
3 try:
4     lale_misconfigured = Tfidf >> LR(LR.solver.sag, LR.penalty.l1)
5 except jsonschema.ValidationError as e:
6     print(e.message, file=sys.stderr)
```

CPU times: user 46.9 ms, sys: 15.6 ms, total: 62.5 ms

Wall time: 36.7 ms

Invalid configuration for LR(solver='sag', penalty='l1') due to constraint the newton-cg, sag, and lbfgs solvers support only l2 penalties.

Schema of constraint 1: {

'description': 'The newton-cg, sag, and lbfgs solvers support only l2 penalties.',

'anyOf': [{

'type': 'object',

'properties': {

'solver': {

'not': {

'enum': ['newton-cg', 'sag', 'lbfgs']}}}}, {

'type': 'object',

'properties': {

'penalty': {

'enum': ['l2']}}}},

}

Value: {'solver': 'sag', 'penalty': 'l1', 'dual': False, 'C': 1.0, 'tol': 0.0001, 'fit\_intercept': True, 'intercept\_scaling': 1.0, 'class\_weight': None, 'random\_state': None, 'max\_iter': 100, 'multi\_class': 'ovr', 'verbose': 0, 'warm\_start': False, 'n\_jobs': None}

# Schemas as Documentation

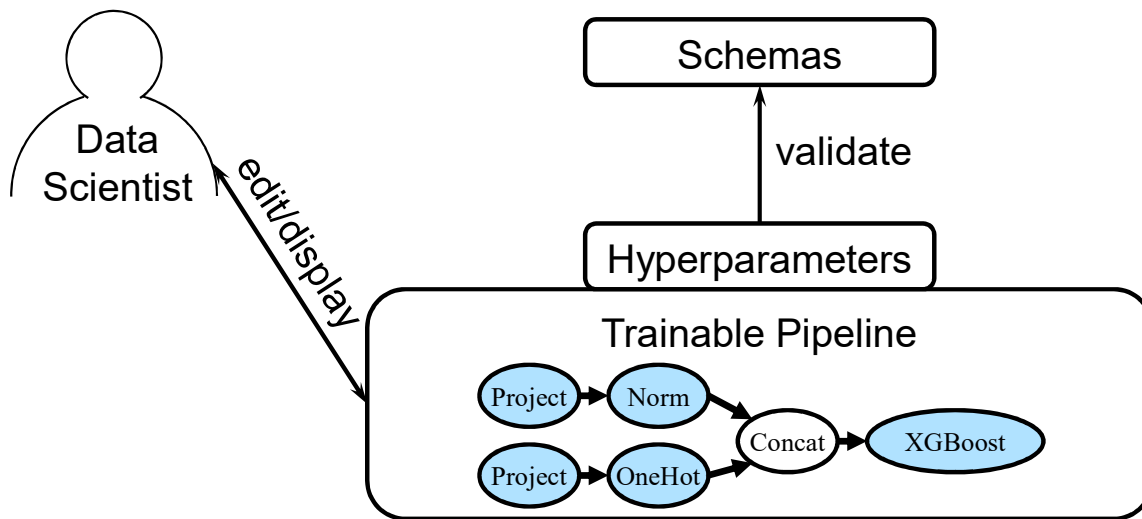
```
In [17]: 1 XGBoost.hyperparam_schema('n_estimators')
```

```
Out[17]: {'description': 'Number of trees to fit.',  
          'type': 'integer',  
          'default': 100,  
          'minimumForOptimizer': 10,  
          'maximumForOptimizer': 1500}
```

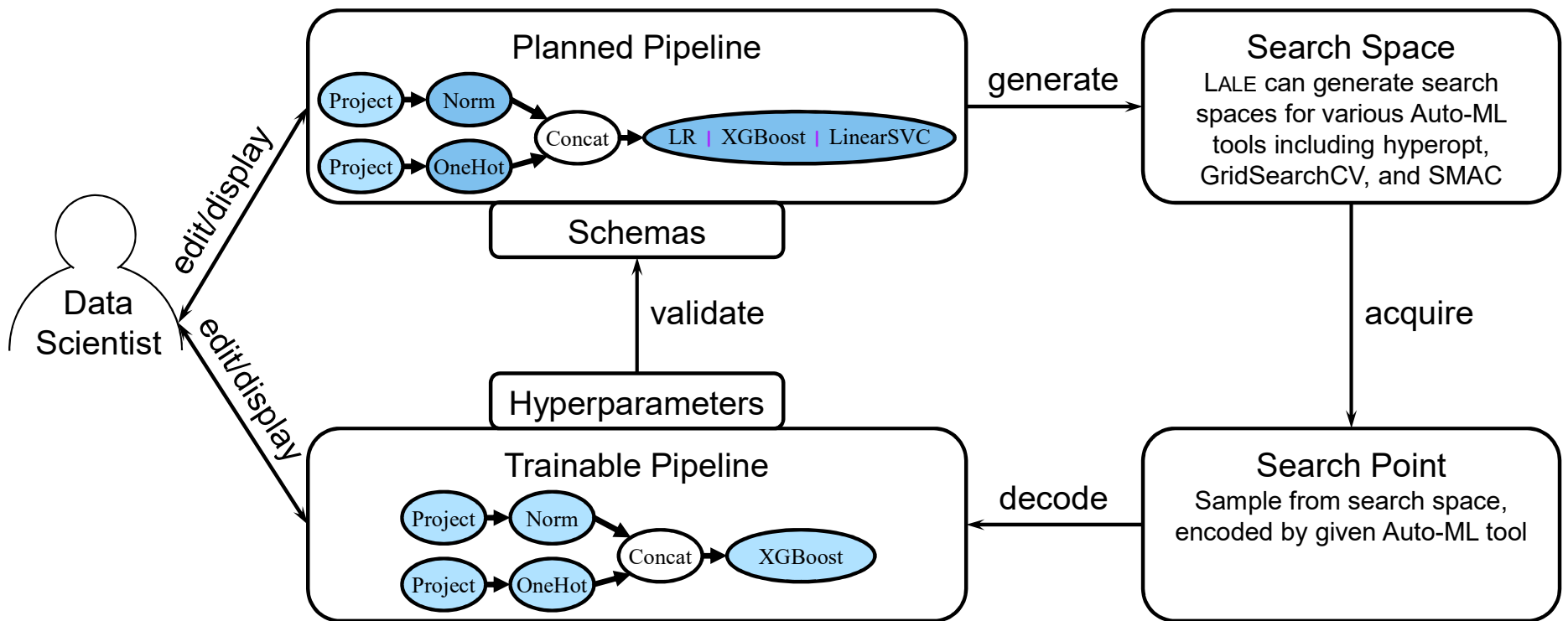
```
In [18]: 1 XGBoost.hyperparam_schema('booster')
```

```
Out[18]: {'description': 'Specify which booster to use.',  
          'enum': ['gbtree', 'gblinear', 'dart'],  
          'default': 'gbtree'}
```

# Schemas as Types



# Types as Search Spaces



"Type-Driven Automated Learning with Lale", <https://arxiv.org/pdf/1906.03957.pdf>



# Customizing Schemas

```
In [19]: 1 import lale.schemas as schemas
          2 Grove = XGBoost.customize_schema(
          3     n_estimators=schemas.Int(min=2, max=6),
          4     booster=schemas.Enum(['gbtree']))

In [20]: 1 grove_planned = ( Project(columns={'type': 'number'}) >> Norm
          2                     & Project(columns={'type': 'string'}) >> OneHot
          3                     ) >> Concat >> Grove

In [21]: 1 grove_optimizer = HyperoptClassifier(grove_planned, cv=3, max_evals=10)
          2 grove_trained = grove_optimizer.fit(train_X, train_y)
          3 grove_y = grove_trained.predict(test_X)
          4 print(f'accuracy {sklearn.metrics.accuracy_score(test_y, grove_y):.1%}')

100%|██████████| 10/10 [00:25<00:00, 2.45s/it, best loss: -0.7358263933376041]
accuracy 71.2%
```

# Scikit-learn Compatible Interopability

| Modality    | Dataset                                    | Pipeline ( <b>bold: best found choice</b> )                                |
|-------------|--------------------------------------------|----------------------------------------------------------------------------|
| Text        | Movie reviews (sentiment analysis)         | <pre>&gt;&gt; (BERT   TFIDF) &gt;&gt; (LR   MLP   KNN   SVC   PAC)</pre>   |
| Table       | Car (structured with categorical features) | <pre>J48   ArulesCBA   LR   KNN</pre>                                      |
| Images      | CIFAR-10 (image classification)            | <pre>ResNet50</pre>                                                        |
| Time-series | Epilepsy (seizure classification)          | <pre>WindowTransformer &gt;&gt; (KNN   XGBoost   LR) &gt;&gt; Voting</pre> |

# Ongoing Work

- General improvements
  - More operators
  - More Auto-ML tools
  - More robustness
- Resource usage
  - Memory
  - Compute
- Expressiveness
  - Grammars
  - Ensembles

*We welcome your suggestions and contributions!*

# <https://github.com/ibm/lale>

