Type-Driven Automated Learning with Lale

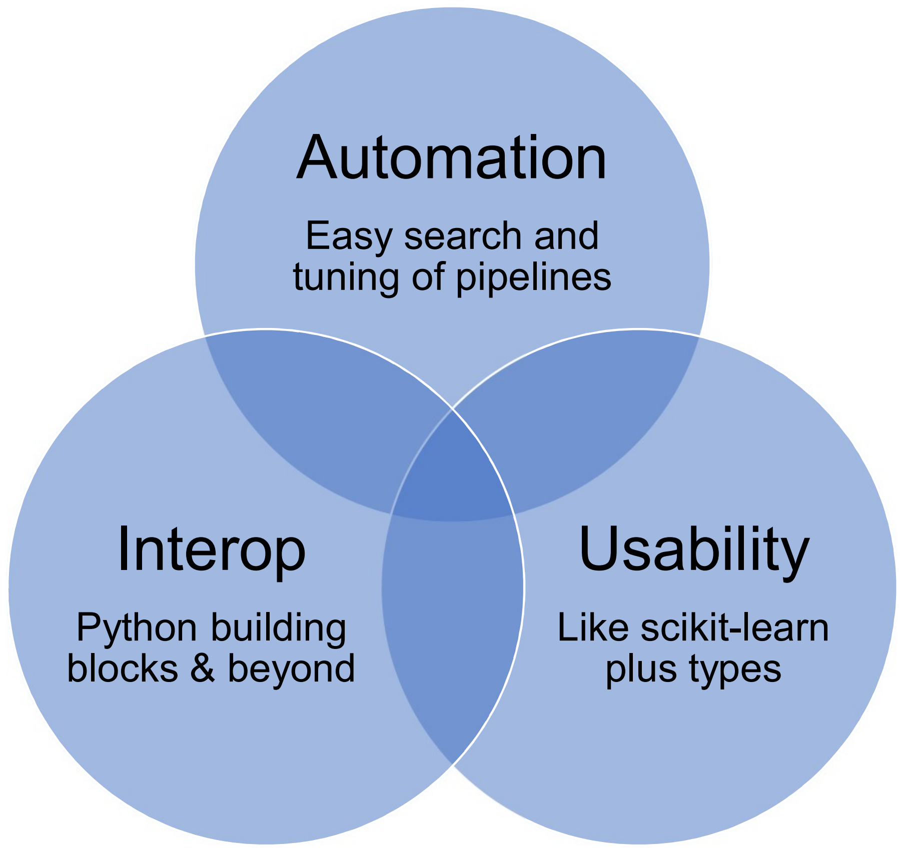
Martin Hirzel, Kiran Kate, Avi Shinnar, Pari Ram, and Guillaume Baudart

Tuesday 4 November 2019

https://github.com/ibm/lale

Value Proposition

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



Categorical + Continuous Dataset

In [1]:

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

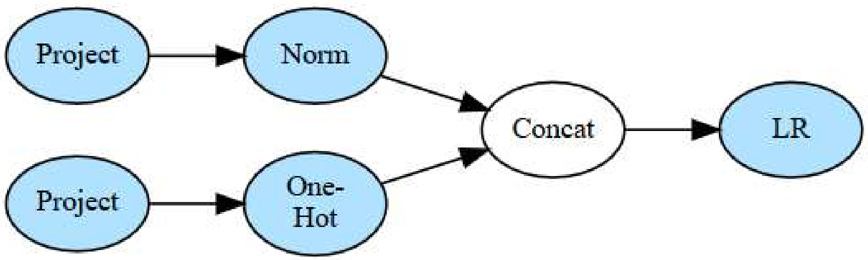
https://nbviewer.jupyter.org/github/IBM/lale/blob/master/examples/talk_2019-1105-lale.ipynb

Manual Pipeline

In [4]:

```
manual_trainable = (
        (   Project(columns={'type': 'number'}) >> Norm()
         & Project(columns={'type': 'string'}) >> OneHot())
    >> Concat
    >> LR(LR.penalty.l1, C=0.001))
lale.helpers.to_graphviz(manual_trainable)
```

Out[4]:



In [5]:

```
import sklearn.metrics
manual_trained = manual_trainable.fit(train_X, train_y)
manual_y = manual_trained.predict(test_X)
print(f'accuracy {sklearn.metrics.accuracy_score(test_y, manual_y):.1%}')
```

```
accuracy 29.1%
```

Pipeline Combinators

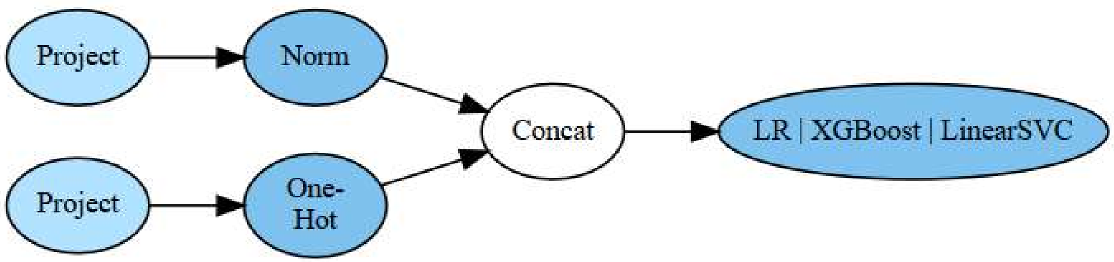
LALE features	Name	Description	Scikit-learn features
`>>` or `make_pipeline`	pipe	feed to next	`make_pipeline`
`&` or `make_union`	and	run both	`make_union` or `ColumnTransformer`
`|` or `make_choice`	or	choose one	N/A (specific to given Auto-ML tool)

Automated Pipeline

In [7]:

```
auto_planned = (
        (  Project(columns={'type': 'number'}) >> Norm
         & Project(columns={'type': 'string'}) >> OneHot)
    >> Concat
    >> (LR | XGBoost | LinearSVC))
lale.helpers.to_graphviz(auto_planned)
```

Out[7]:



In [8]:

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

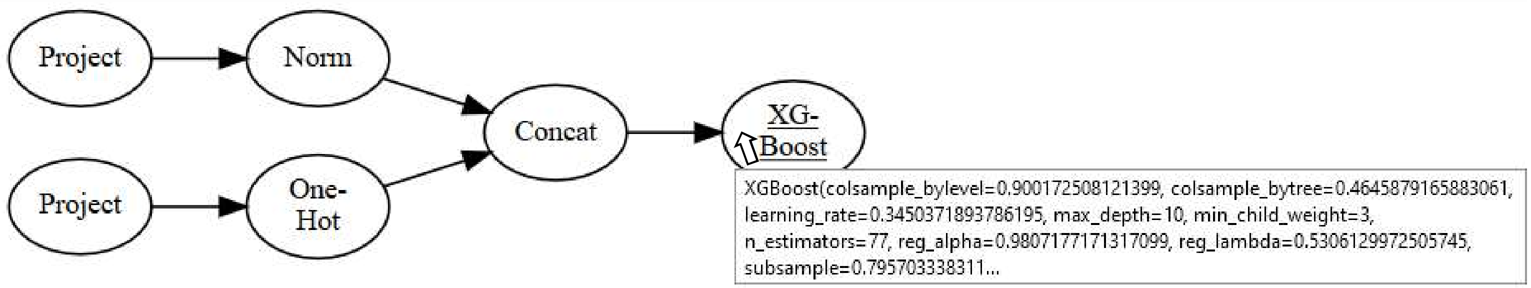
```
100%|██████████| 10/10 [00:30<00:00,  2.91s/it, best loss: -0.7373278347213325]
accuracy 75.2%
```

Displaying Automation Results

In [9]:

```
lale.helpers.to_graphviz(auto_trained)
```

Out[9]:

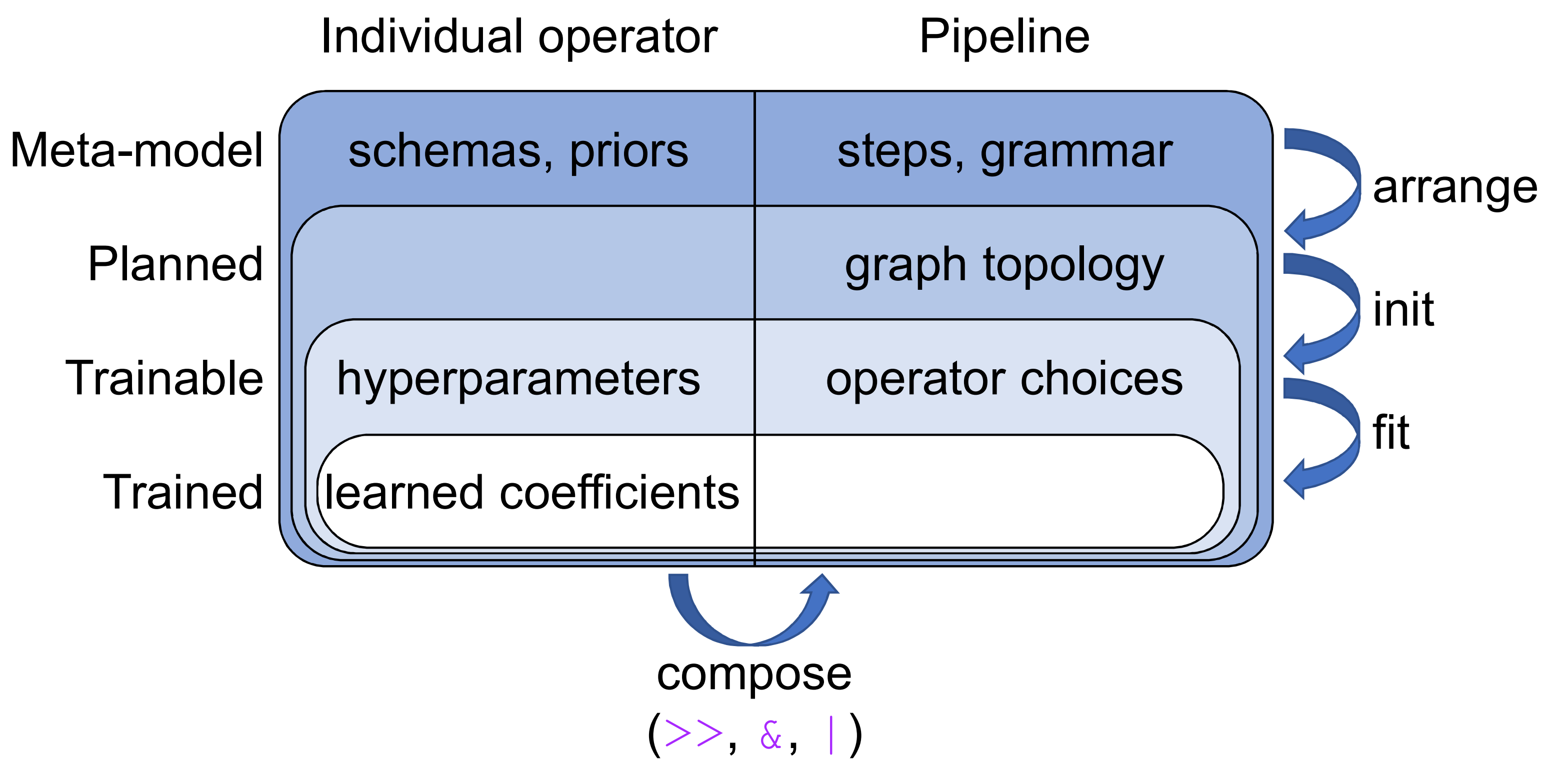


In [10]:

```
import lale.pretty_print
lale.pretty_print.ipython_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, le
pipeline = ((project >> Norm) & (project_1 >> OneHot)) >> Concat >> xgboost
```

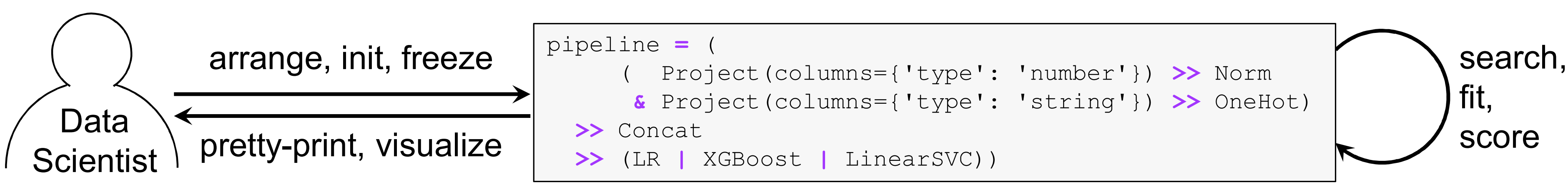
Bindings as Lifecycle: Venn Diagram



“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	• Interpretable • Based on licenses • Based on GPU requirements
Tweak graph topology	• Custom preprocessing • Multi-modal data • Fairness mitigation
Tweak hyperparameter schemas	• Adjust range for continuous • Restrict choices for categorical
Expand available operator choices	• Wrap existing library • Write your own operators




```
pipeline = (
     (   Project(columns={'type': 'number'}) >> Norm
       & Project(columns={'type': 'string'}) >> OneHot)
   >> Concat
   >> (LR | XGBoost | LinearSVC))
```

Constraints in Scikit-learn

In [13]:

```
sklearn_misconfigured = sklearn.pipeline.make_pipeline(
    sklearn.feature_extraction.text.TfidfVectorizer(),
    sklearn.linear_model.LogisticRegression(solver='sag', penalty='l1'))
print('no error detected yet')
```

```
no error detected yet
```

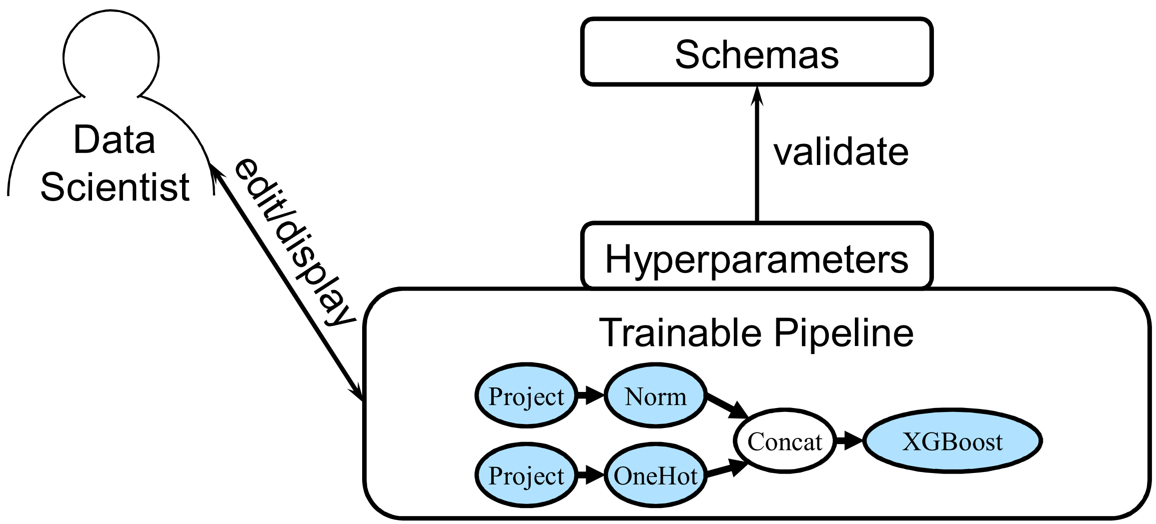
In [14]:

```
%%time
import sys
try:
    sklearn_misconfigured.fit(news_X, news_y)
except ValueError as e:
    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.
```

Type-Driven Manual Learning in LALE



Constraints in LALE

In [16]:

```
%%time
import jsonschema
try:
    lale_misconfigured = Tfidf >> LR(LR.solver.sag, LR.penalty.l1)
except jsonschema.ValidationError as e:
    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, s
ag, 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_inte
rcept': True, 'intercept_scaling': 1.0, 'class_weight': None, 'random_state': None, 'max_it
er': 100, 'multi_class': 'ovr', 'verbose': 0, 'warm_start': False, 'n_jobs': None}
```

Types 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'}
```

Constraints in Auto-ML

Problem: Some automated trials raise exceptions

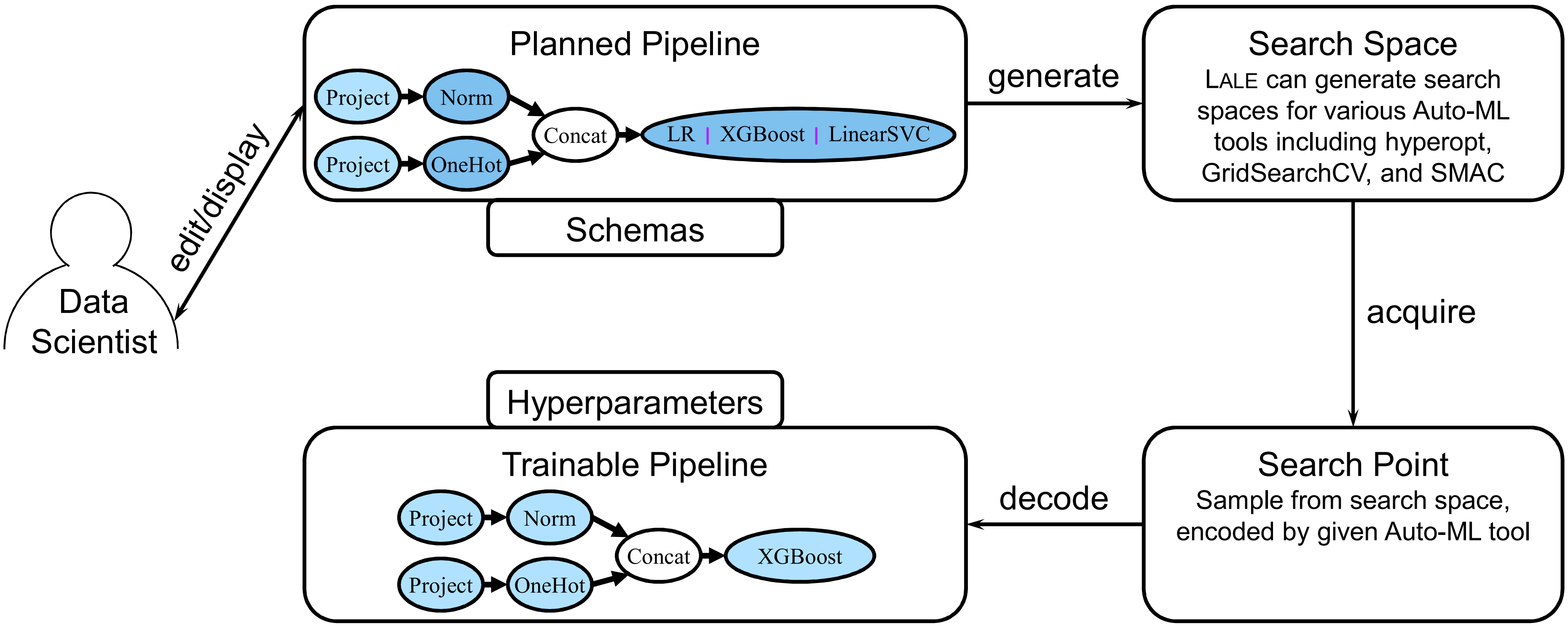
Solution 1: Unconstrained search space

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

Solution 2: Constrained search space

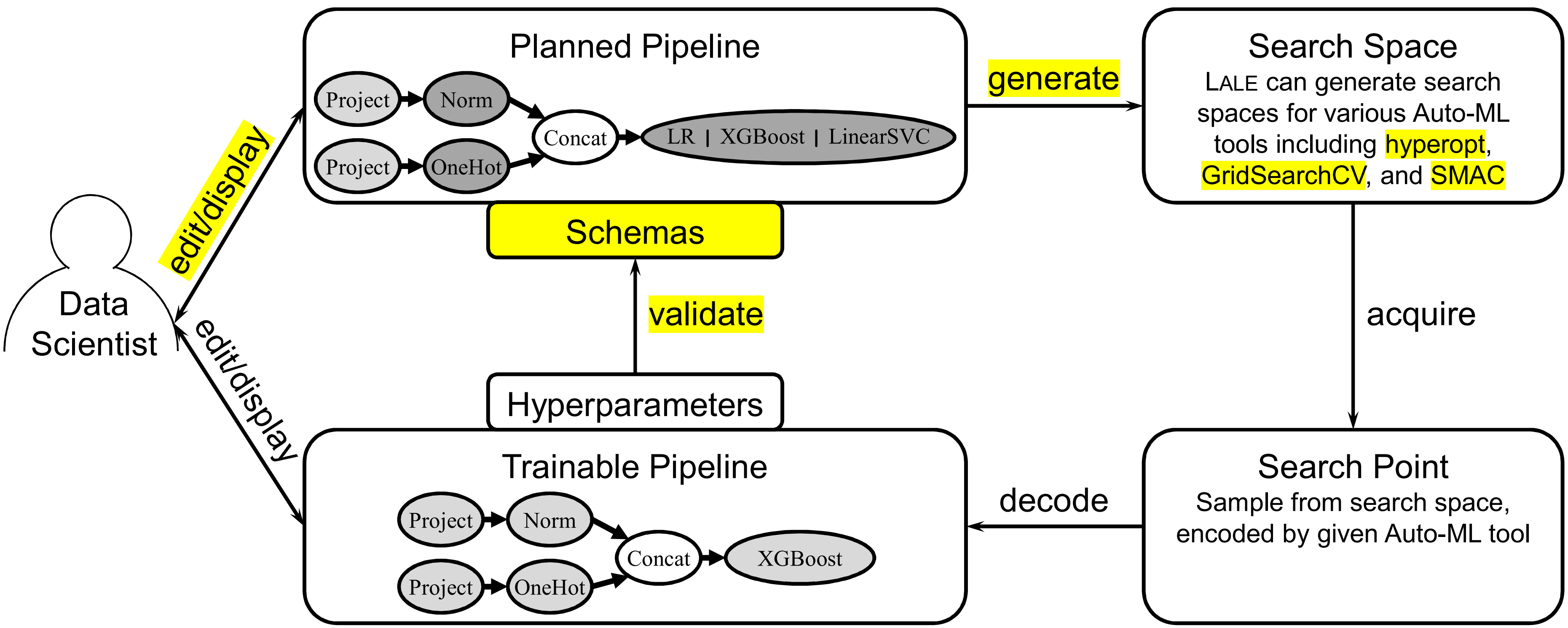
- {*solver*: [*linear*,*sag*,*lbfgs*], *penalty*: [*l1*,*l2*]} **and** (**if** *solver*: [*sag*,*lbfgs*] **then** *penalty*: [*l2*])
- No exceptions (no time wasted)
- No made-up loss

Types as Search Spaces



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

Types as Single Source of Truth



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

Customizing Types

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

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

In [21]:
```
grove_optimizer = HyperoptClassifier(grove_planned, cv=3, max_evals=10)
grove_trained = grove_optimizer.fit(train_X, train_y)
grove_y = grove_trained.predict(test_X)
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

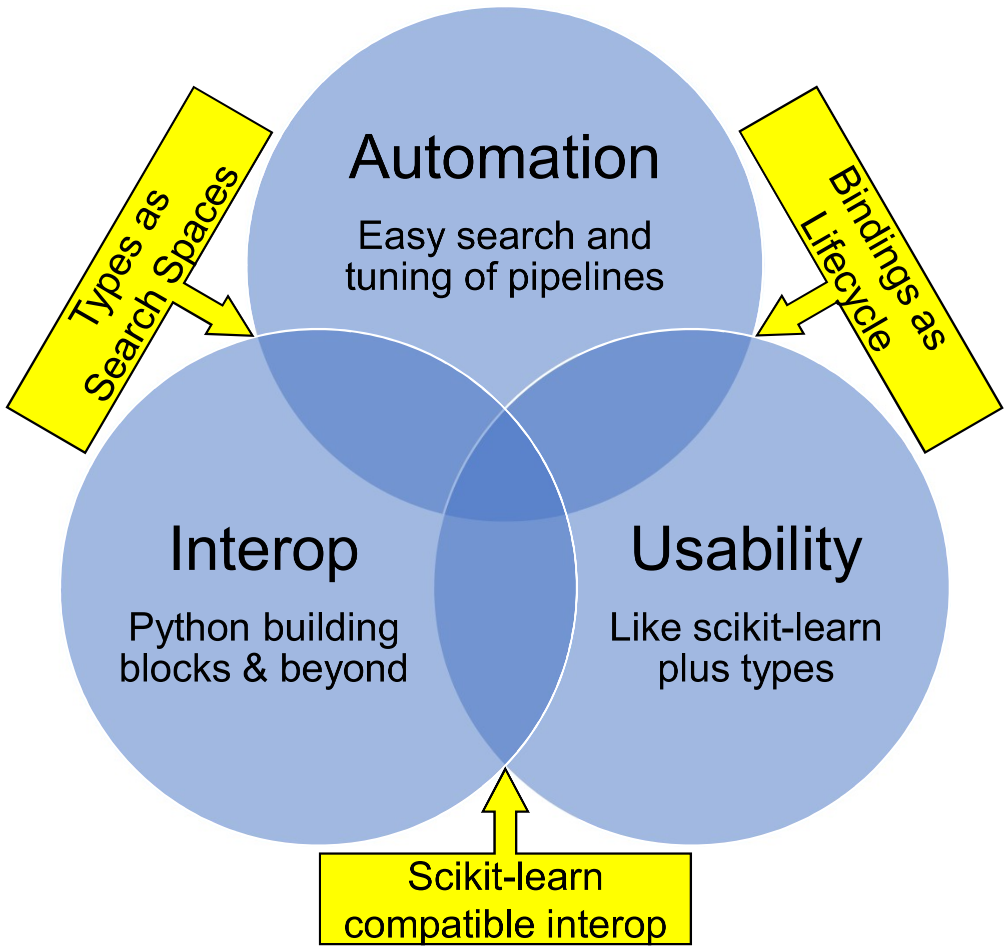
Modality	Dataset	Pipeline (**bold**: best found choice)
Text	Movie reviews (sentiment analysis)	`(`**`BERT`**` | TFIDF)` `>> (`**`LR`**` | MLP | KNN | SVC | PAC)`
Table	Car (structured with categorical features)	**`J48`**` | ArulesCBA | LR | KNN`
Images	CIFAR-10 (image classification)	**`ResNet50`**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

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!

Conclusion



github.com/ibm/lale