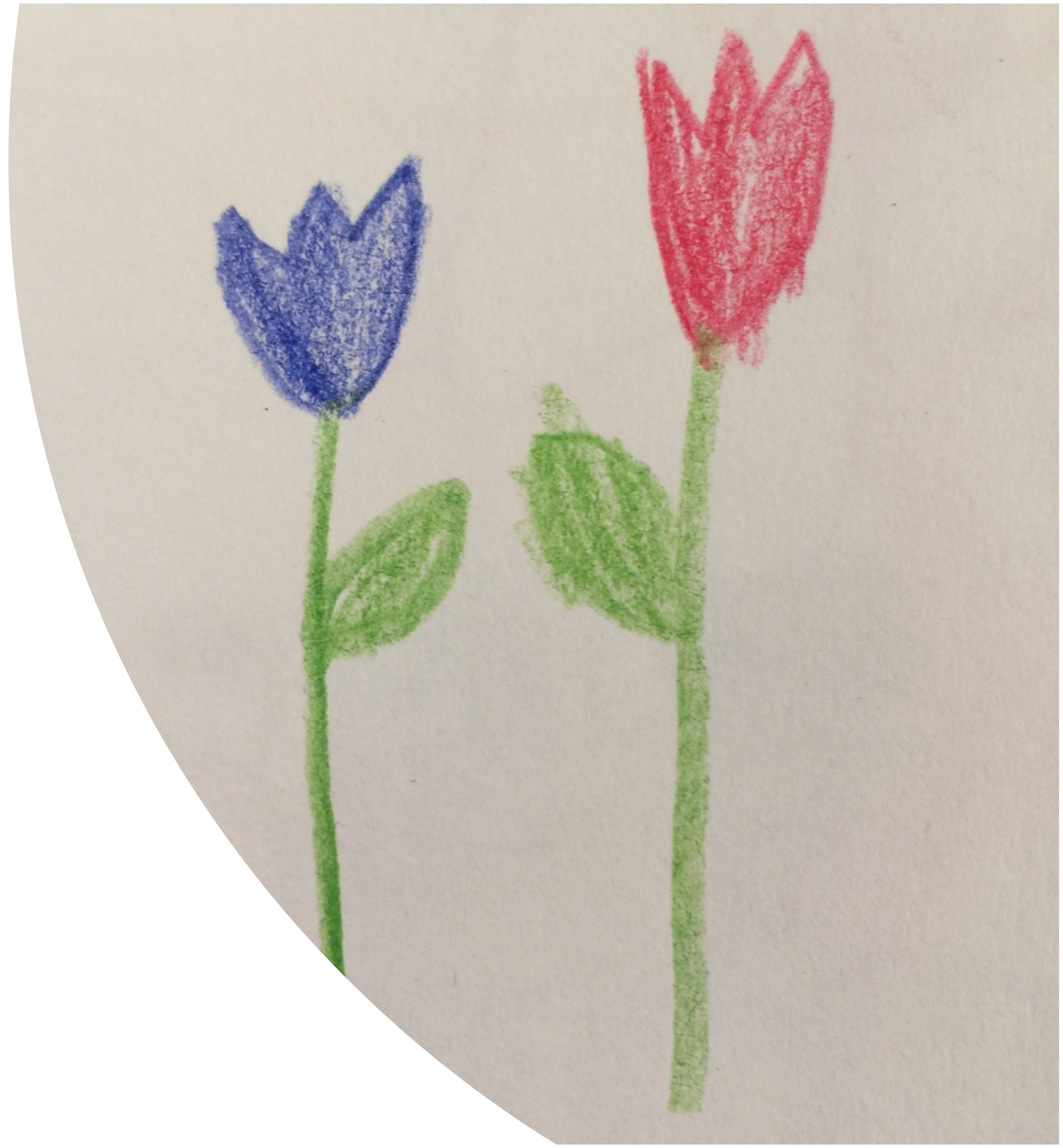
Type-Driven Automated Learning with LALE

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

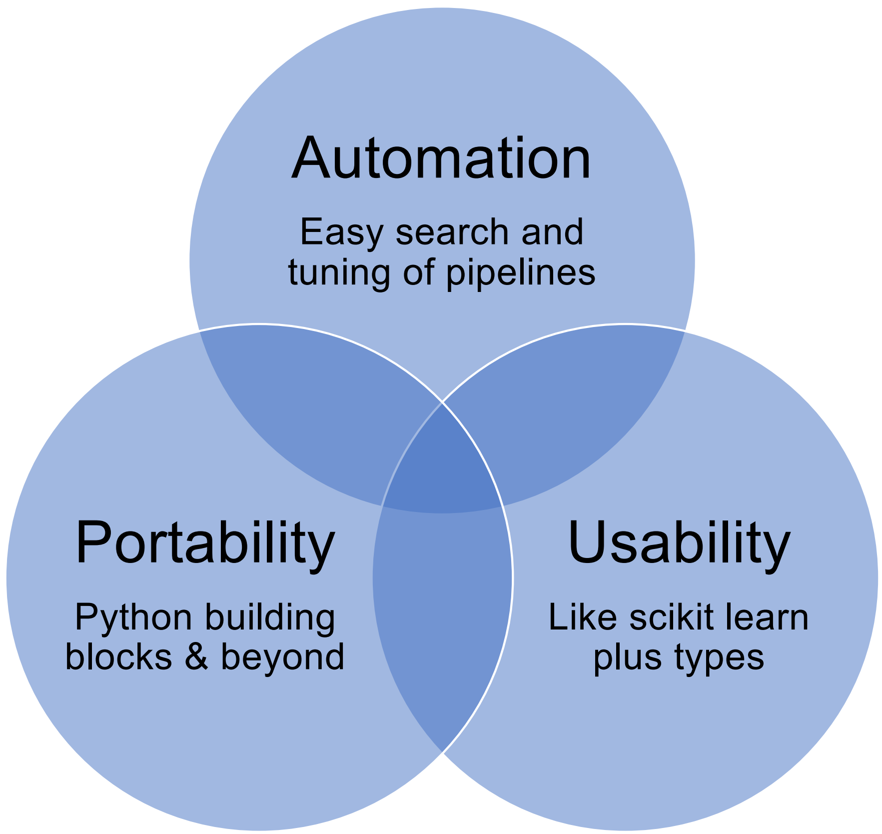
Monday, December 9th 2019

IBM PL Day 2019



Value Proposition

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



Manual ML with Sklearn

Prior work: scikit learn, popular machine learning package

```
pca_lr = make_pipeline(PCA(svd_solver='full', n_components=0.3),
                       LR(solver='liblinear', penalty='l1'))
```

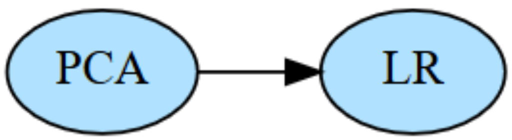
```
pca_lr.fit(train_X, train_y)
predicted = pca_lr.predict(test_X)
print(f'accuracy  {accuracy_score(test_y, predicted):.1%}')
```

```
accuracy  70.2%
```

Manual ML with LALE

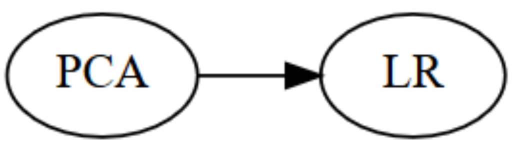
Our work: Language for Automated Learning Exploration

```
pca_lr = PCA(PCA.svd_solver.full, n_components=0.3) \
      >> LR(LR.solver.liblinear, LR.penalty.l1)
to_graphviz(pca_lr)
```




```
trained = pca_lr.fit(train_X, train_y)
predicted = trained.predict(test_X)
print(f'accuracy  {accuracy_score(test_y, predicted):.1%}')
to_graphviz(trained)
```

```
accuracy  70.2%
```



LALE Pipelines

- Pipeline Combinators:

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

- Example:

```
((PCA & MinMaxScaler)
 >> Concat
 >> (KNN | DecisionTree))
```

Constraints in Manual ML

Conditional hyperparameters

```
pca_lr = make_pipeline(PCA(svd_solver='full', n_components=0.3),
                       LR(solver='sag', penalty='l1'))
```

```
pca_lr.fit(train_X, train_y)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-7-de82d92d1962> in <module>
----> 1 pca_lr.fit(train_X, train_y)

~/python3.7venv/lib/python3.7/site-packages/sklearn/pipeline.py in fit(self, X, y, **fit_params)
    265         Xt, fit_params = self._fit(X, y, **fit_params)
    266         if self._final_estimator is not None:
--> 267             self._final_estimator.fit(Xt, y, **fit_params)
    268         return self
    269 

~/python3.7venv/lib/python3.7/site-packages/sklearn/linear_model/logistic.py in fit(self, X, y, sample_weight)
   1275                              "positive; got (tol=%r)" % self.tol)
   1276 
-> 1277         solver = _check_solver(self.solver, self.penalty, self.dual)
   1278 
   1279         if solver in ['newton-cg']:

~/python3.7venv/lib/python3.7/site-packages/sklearn/linear_model/logistic.py in _check_solver(solver, penalty, dual)
    445     if solver not in ['liblinear', 'saga'] and penalty != 'l2':
    446         raise ValueError("Solver %s supports only l2 penalties, "
--> 447                          "got %s penalty." % (solver, penalty))
    448     if solver != 'liblinear' and dual:
    449         raise ValueError("Solver %s supports only "

ValueError: Solver sag supports only l2 penalties, got l1 penalty.
```

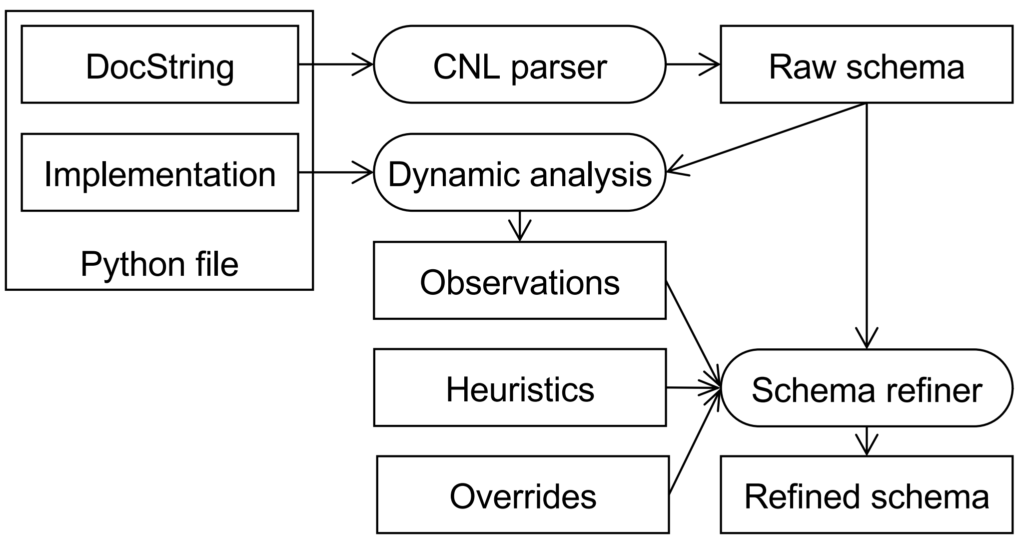
(JSON) Schemas for ML Algorithms

- Use of JSON schema for defining hyper-parameter types and search spaces, input and output types.
- Example hyper-parameter schema for LogisticRegression:

```
LR: { allOf: [
  { type: object,
    properties: {
      S: { description: "Optimization problem solver",
           enum: [linear, sag, lbfgs], default: linear},
      P: { description: "Penalization norm",
           enum: [l1, l2], default: l2}}},
  { description: "Solvers sag and lbfgs support only l2.",
    anyOf: [
      { not: { type: object, properties: {S: {enum: [sag, lbfgs]}}}},
      { type: object, properties: {P: {enum: [l2]}}}]}]}
```

Most users do not need to write these schemas. Usually, the operator writer adds these once and uses them multiple times for multiple purposes.

Automated Schema Extractor



https://github.com/IBM/lale/tree/master/lale/lib/autogen

Customizing Schemas by Hand

```
from lale.schemas import Null, Enum, Int, Float, Object, Array, Not, AnyOf

MyLR = MyLR.customize_schema(
    relevantToOptimizer=['solver', 'penalty', 'C'],
    solver=Enum(desc='Algorithm for optimization problem.',
                values=['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga'],
                default='liblinear'),
    penalty=Enum(desc='Norm used in the penalization.',
                 values=['l1', 'l2'],
                 default='l2'),
    C=Float(desc='Inverse regularization strength. '
                 'Smaller values specify stronger regularization.',
            min=0.0, exclusiveMin=True,
            minForOptimizer=0.03125, maxForOptimizer=32768,
            distribution='loguniform',
            default=1.0))
```

https://nbviewer.jupyter.org/github/IBM/lale/blob/master/examples/docs_new_operators_schemas_api.ipynb

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}
```

AutoML (GridSearchCV)

Create Hyperparameter Search Space

```
# Create regularization penalty space
penalty = ['l1', 'l2']

# Create regularization hyperparameter space
C = np.logspace(0, 4, 10)

# Create hyperparameter options
hyperparameters = dict(C=C, penalty=penalty)
```

Create Grid Search

```
# Create grid search using 5-fold cross validation
clf = GridSearchCV(logistic, hyperparameters, cv=5, verbose=0)
```

Constraints in AutoML

Problem: Some automated iterations raise exceptions

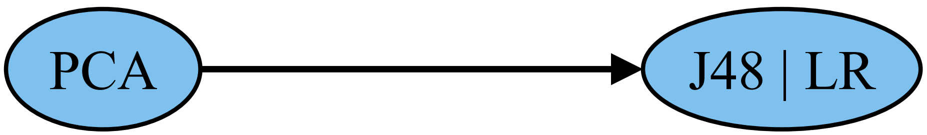
Solution 1: Unconstrained search space

- $\{S{:}[\mathit{linear},\mathit{sag},\mathit{lbfgs}],\ P{:}\ [\mathit{l1},\mathit{l2}]\}$
- Catch exception
- Return made-up loss `np.float.max`

Solution 2: Constrained search space

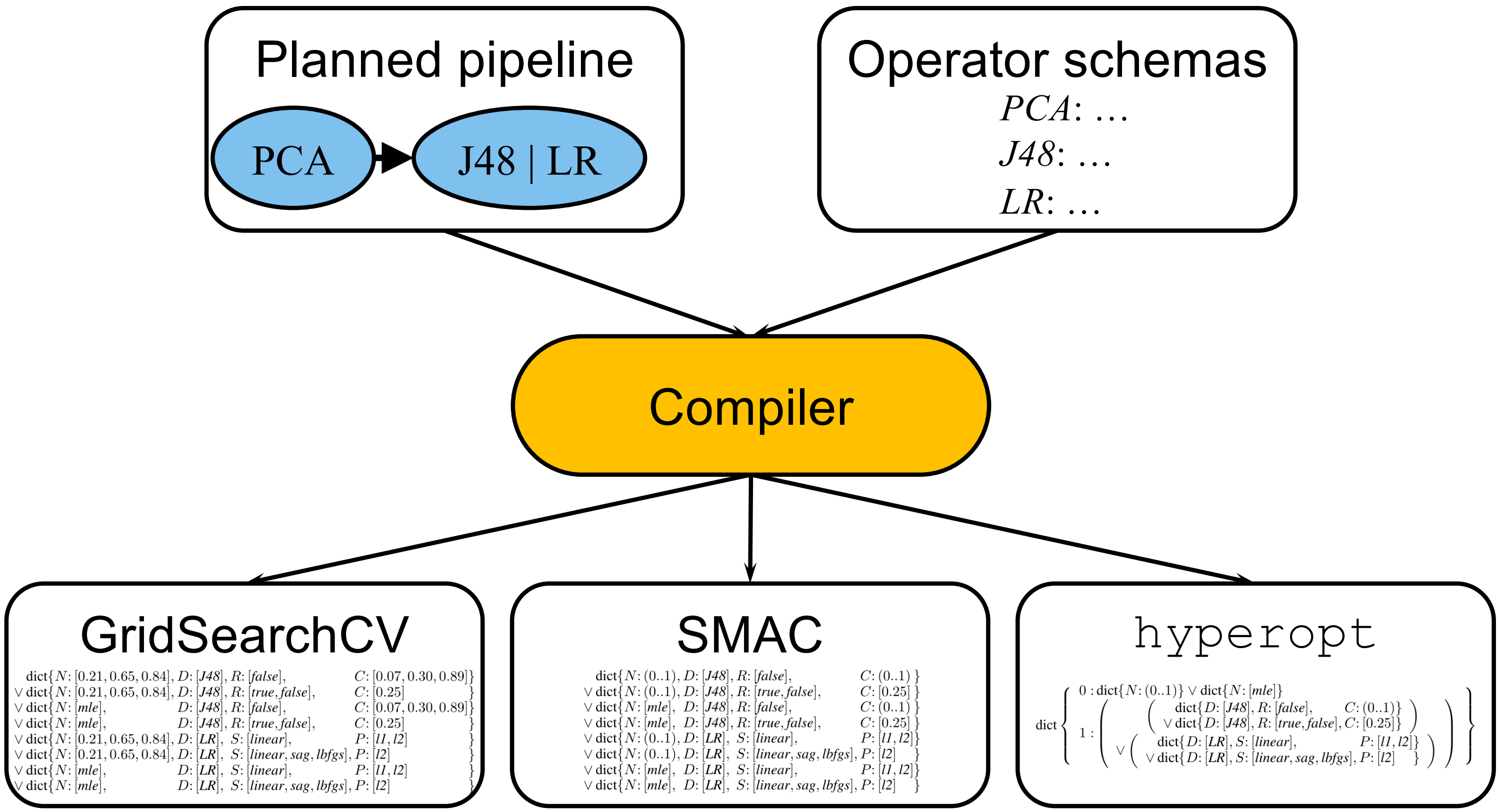
- $\{S{:}[\mathit{linear},\mathit{sag},\mathit{lbfgs}],\ P{:}\ [\mathit{l1},\mathit{l2}]\}$ **and** (**if** S: $[\mathit{sag},\mathit{lbfgs}]$ **then** P: $[\mathit{l2}]$)
- No exceptions
- No made-up loss

Algorithm Selection



Types as Search Spaces

LALE auto-generates search spaces for AutoML tools



GridSearchCV Search Space

AutoML included with Sklearn

$$
\begin{array}{lllll}
 & \text{dict}\{N\colon [0.21, 0.65, 0.84], & D\colon [\mathit{J48}], R\colon [\mathit{false}], & C\colon [0.07, 0.30, 0.89] & \} \\
\vee & \text{dict}\{N\colon [0.21, 0.65, 0.84], & D\colon [\mathit{J48}], R\colon [\mathit{true}, \mathit{false}], & C\colon [0.25] & \} \\
\vee & \text{dict}\{N\colon [\mathit{mle}], & D\colon [\mathit{J48}], R\colon [\mathit{false}], & C\colon [0.07, 0.30, 0.89] & \} \\
\vee & \text{dict}\{N\colon [\mathit{mle}], & D\colon [\mathit{J48}], R\colon [\mathit{true}, \mathit{false}], & C\colon [0.25] & \} \\
\vee & \text{dict}\{N\colon [0.21, 0.65, 0.84], & D\colon [\mathit{LR}], \ S\colon [\mathit{linear}], & P\colon [\mathit{l1}, \mathit{l2}] & \} \\
\vee & \text{dict}\{N\colon [0.21, 0.65, 0.84], & D\colon [\mathit{LR}], \ S\colon [\mathit{linear}, \mathit{sag}, \mathit{lbfgs}], & P\colon [\mathit{l2}] & \} \\
\vee & \text{dict}\{N\colon [\mathit{mle}], & D\colon [\mathit{LR}], \ S\colon [\mathit{linear}], & P\colon [\mathit{l1}, \mathit{l2}] & \} \\
\vee & \text{dict}\{N\colon [\mathit{mle}], & D\colon [\mathit{LR}], \ S\colon [\mathit{linear}, \mathit{sag}, \mathit{lbfgs}], & P\colon [\mathit{l2}] & \}
\end{array}
$$

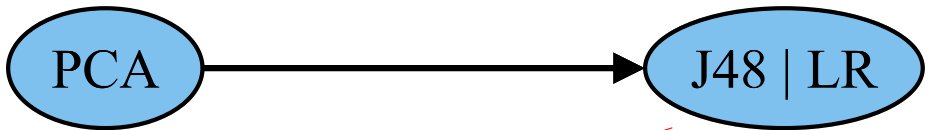
SMAC Search Space

Sequential Model-based Algorithm Configuration

$$
\begin{aligned}
& \text{dict}\{N\colon (0..1), D\colon [\mathit{J48}], R\colon [\mathit{false}], && C\colon (0..1)\ \} \\
\vee\ & \text{dict}\{N\colon (0..1), D\colon [\mathit{J48}], R\colon [\mathit{true}, \mathit{false}], && C\colon [0.25]\ \} \\
\vee\ & \text{dict}\{N\colon [\mathit{mle}],\ D\colon [\mathit{J48}], R\colon [\mathit{false}], && C\colon (0..1)\ \} \\
\vee\ & \text{dict}\{N\colon [\mathit{mle}],\ D\colon [\mathit{J48}], R\colon [\mathit{true}, \mathit{false}], && C\colon [0.25]\ \} \\
\vee\ & \text{dict}\{N\colon (0..1), D\colon [\mathit{LR}],\ S\colon [\mathit{linear}], && P\colon [\mathit{l1}, \mathit{l2}]\} \\
\vee\ & \text{dict}\{N\colon (0..1), D\colon [\mathit{LR}],\ S\colon [\mathit{linear}, \mathit{sag}, \mathit{lbfgs}], && P\colon [\mathit{l2}]\ \} \\
\vee\ & \text{dict}\{N\colon [\mathit{mle}],\ D\colon [\mathit{LR}],\ S\colon [\mathit{linear}], && P\colon [\mathit{l1}, \mathit{l2}]\} \\
\vee\ & \text{dict}\{N\colon [\mathit{mle}],\ D\colon [\mathit{LR}],\ S\colon [\mathit{linear}, \mathit{sag}, \mathit{lbfgs}], && P\colon [\mathit{l2}]\ \}
\end{aligned}
$$

Hyperopt Search Space

Supports parallel search

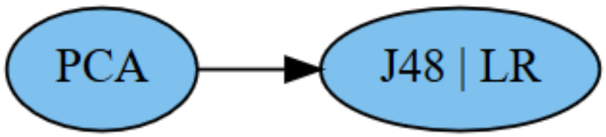


$$\mathrm{dict}\left\{ \begin{array}{l} 0 : \mathrm{dict}\{N\colon (0..1)\} \vee \mathrm{dict}\{N\colon [\mathit{mle}]\} \\ 1 : \left(\begin{array}{l} \left(\begin{array}{l} \quad \mathrm{dict}\{D\colon [\mathit{J48}], R\colon [\mathit{false}], \quad C\colon (0..1)\} \\ \vee \mathrm{dict}\{D\colon [\mathit{J48}], R\colon [\mathit{true}, \mathit{false}], C\colon [0.25]\} \end{array} \right) \\ \vee \left(\begin{array}{l} \quad \mathrm{dict}\{D\colon [\mathit{LR}], S\colon [\mathit{linear}], \quad P\colon [\mathit{l1}, \mathit{l2}]\} \\ \vee \mathrm{dict}\{D\colon [\mathit{LR}], S\colon [\mathit{linear}, \mathit{sag}, \mathit{lbfgs}], P\colon [\mathit{l2}]\} \end{array} \right) \end{array} \right) \end{array} \right\}$$

Automated ML with LALE

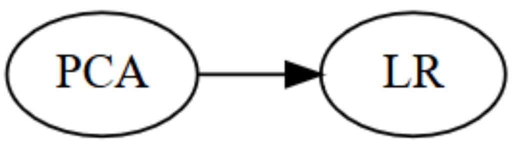
Combined algorithm selection and hyperparameter tuning

```
planned = PCA >> (J48 | LR)
to_graphviz(planned)
```




```
hyperopt_classifier = HyperoptClassifier(planned, max_evals=5)
best_found = hyperopt_classifier.fit(train_X, train_y)
predicted = best_found.predict(test_X)
print(f'accuracy  {accuracy_score(test_y, predicted):.1%}')
to_graphviz(best_found)
```

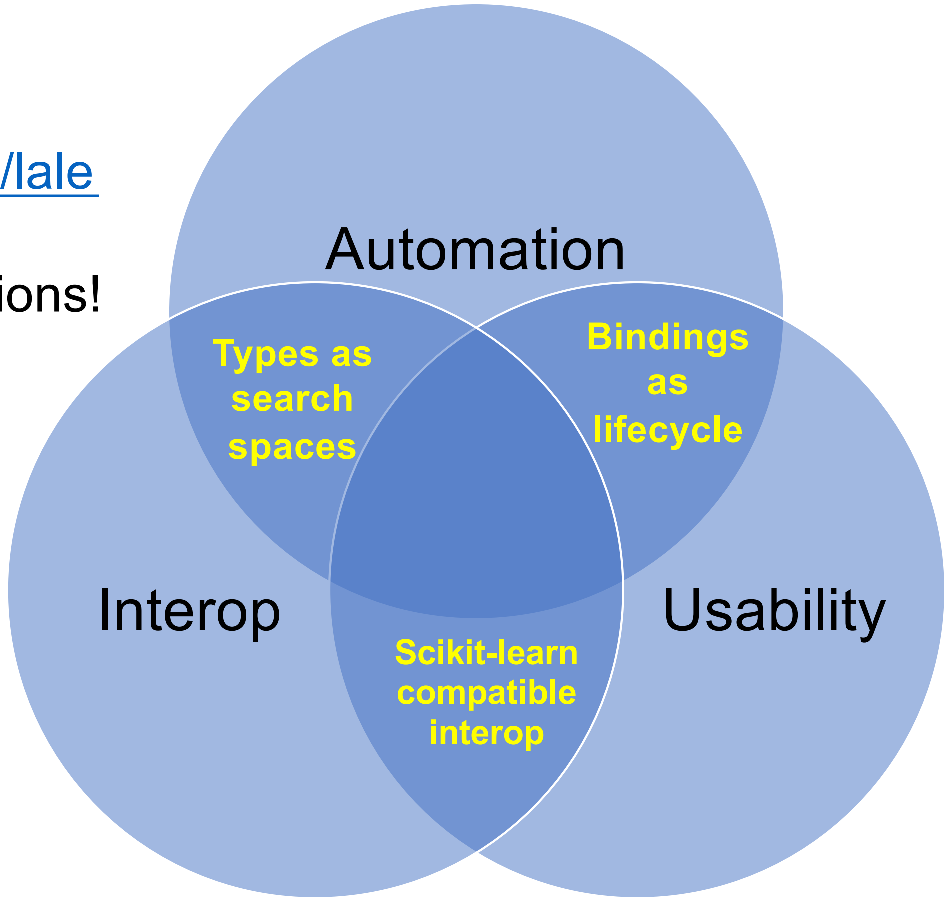
```
accuracy  96.4%
```



Summary

Github URL:
https://github.com/IBM/lale

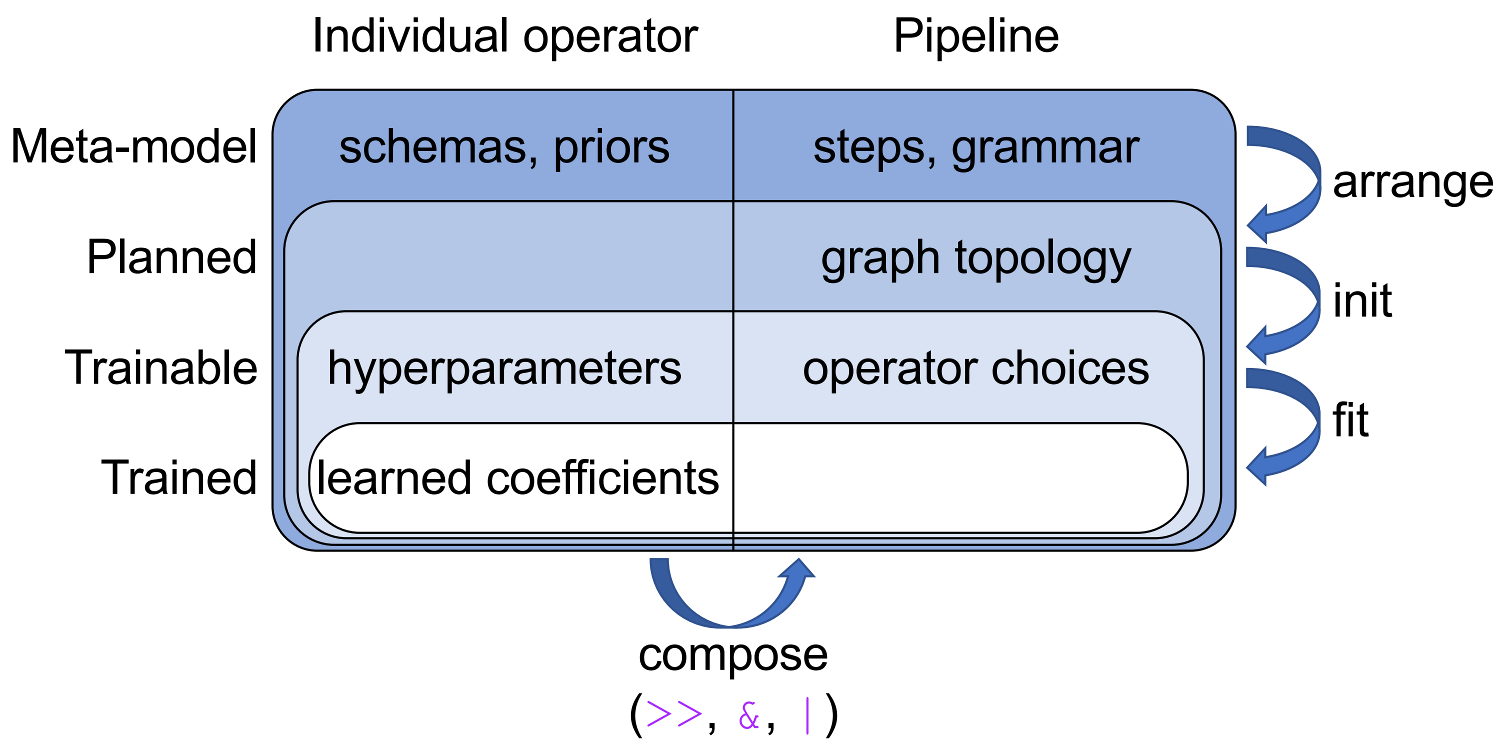
We welcome contributions!



Portability

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`**

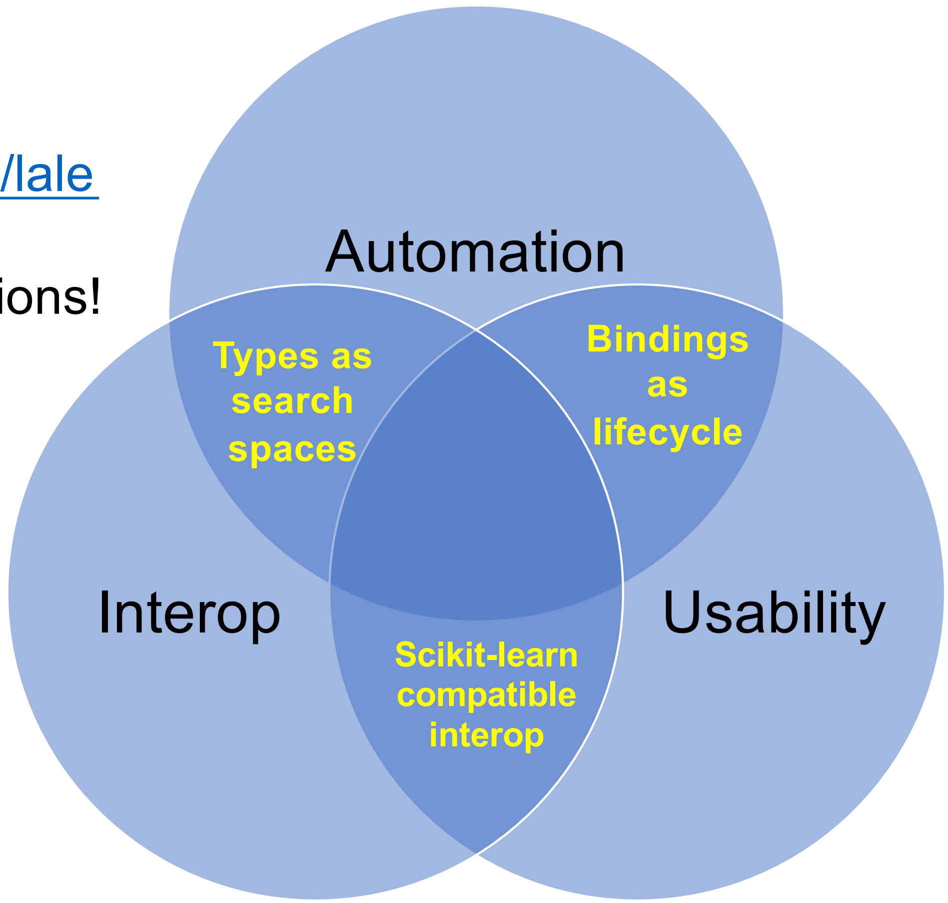
Bindings as Lifecycle



Summary

Github URL:
https://github.com/IBM/lale

We welcome contributions!

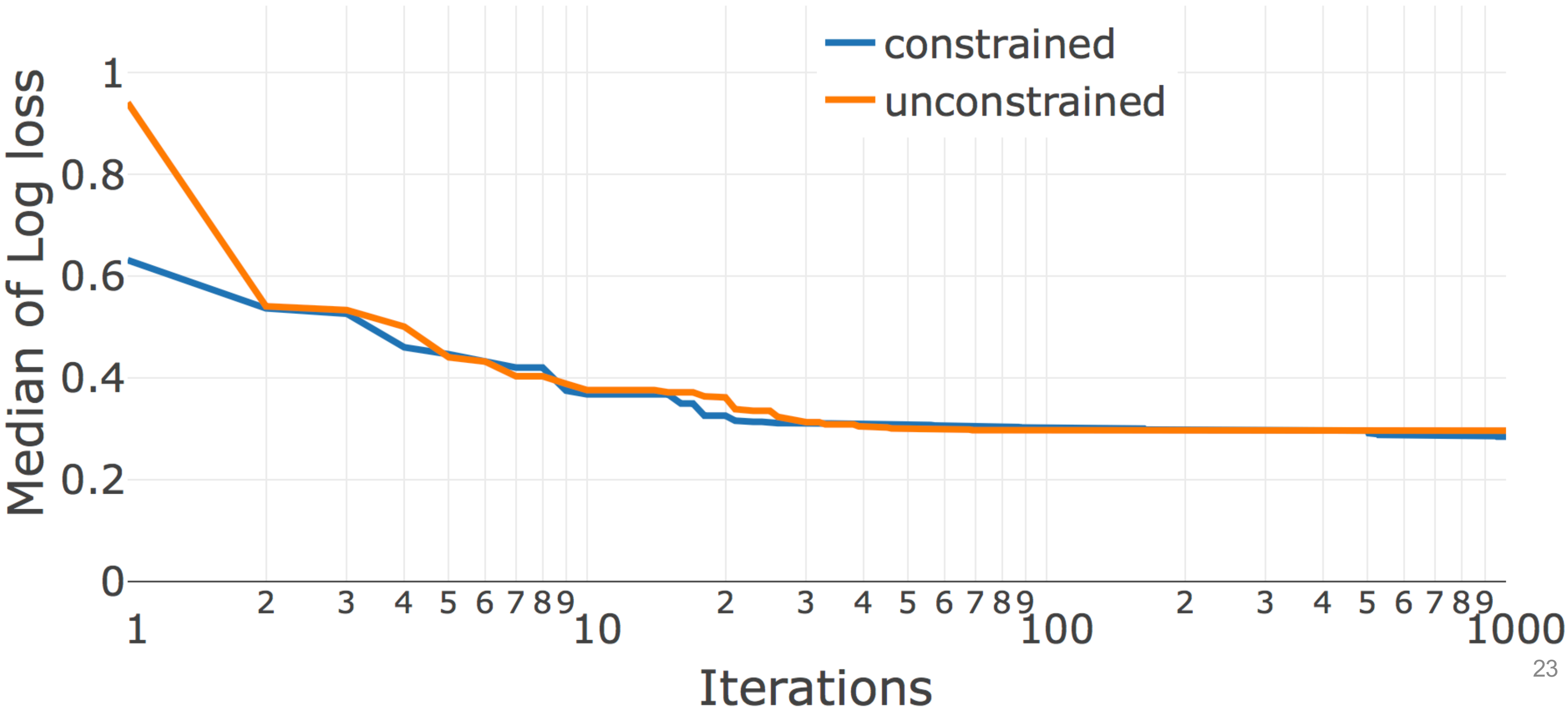


Search Convergence (1/3)

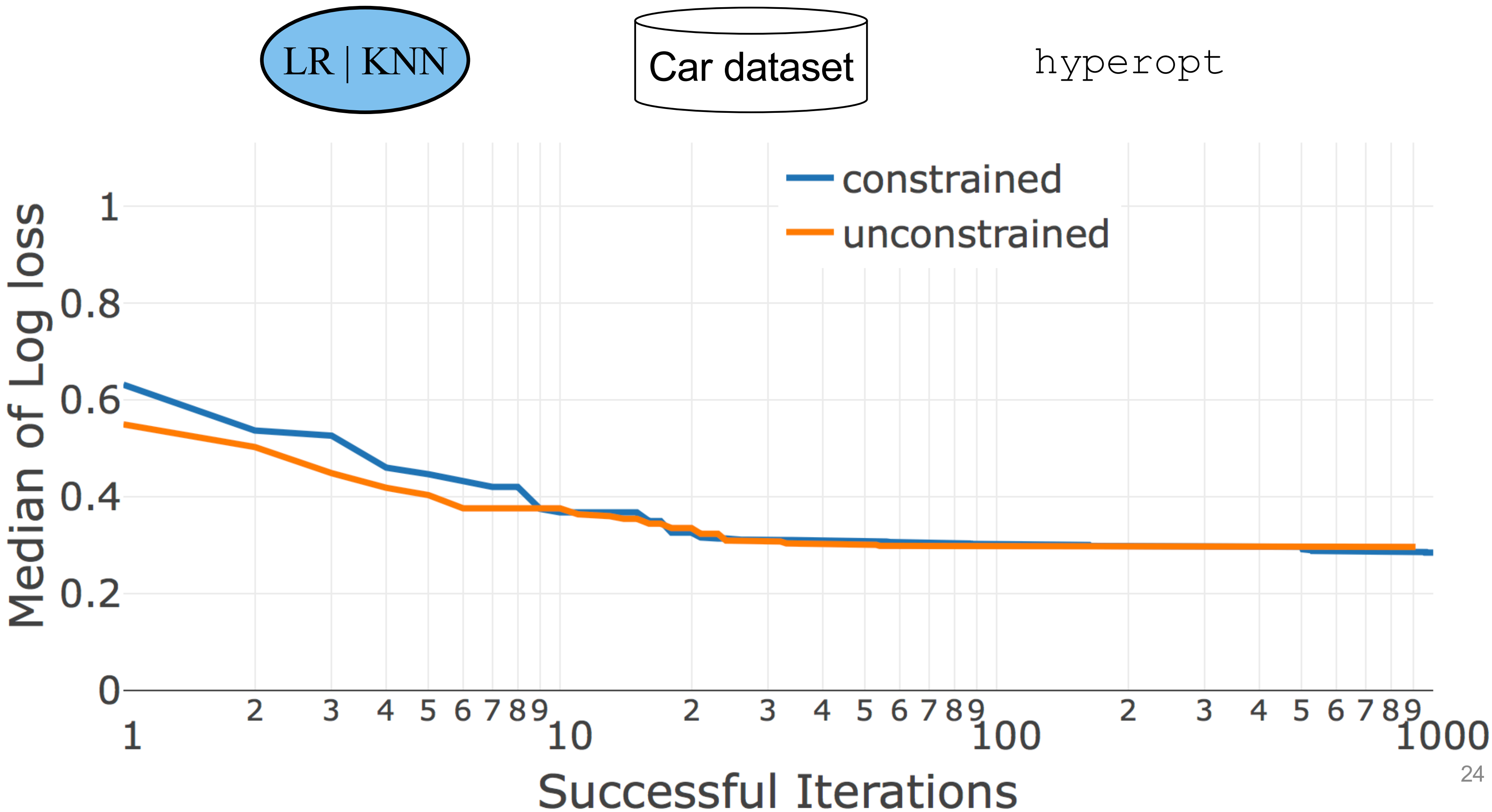
LR | KNN

Car dataset

hyperopt



Search Convergence (2/3)



Search Convergence (3/3)

J48 | LR | KNN

Car dataset

hyperopt