Accessible Machine Learning in Python

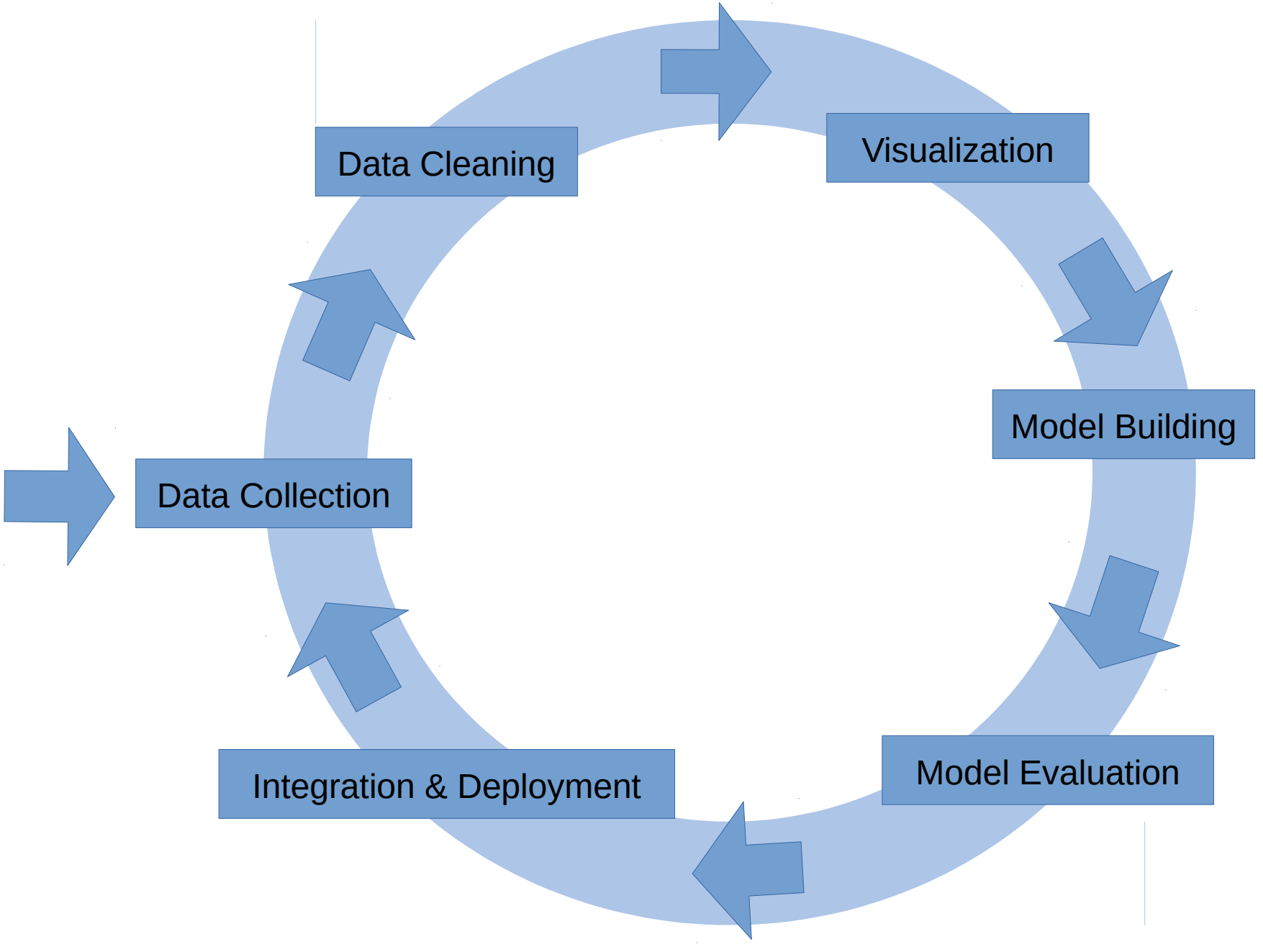
Andreas Müller

Associate Research Scientist
Columbia University
Scikit-learn Technical Committee

Alfred P. Sloan
FOUNDATION

COLUMBIA UNIVERSITY
IN THE CITY OF NEW YORK

A real world ML workflow

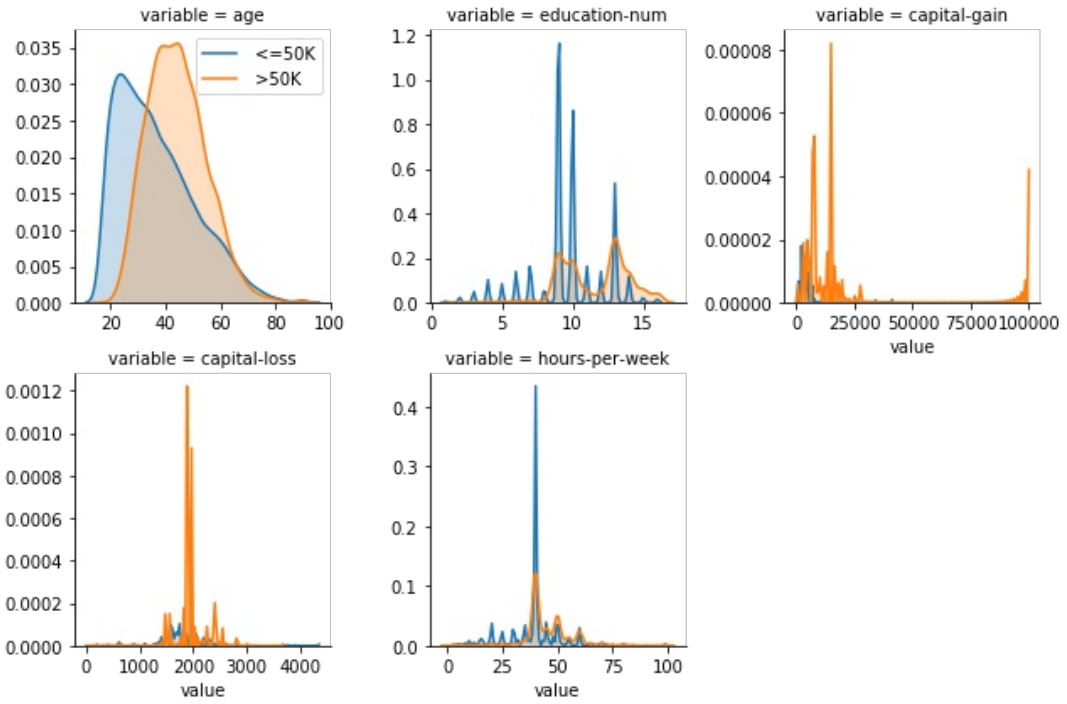


ML with sklearn & pandas

```
import pandas as pd
import seaborn as sns

data = pd.read_csv("adult.csv", index_col=0)

cols = data.columns[data.dtypes != object].tolist() + ['income']
df = data.loc[:, cols].melt("income")
g = sns.FacetGrid(df, col='variable', hue='income',
                  sharey=False, sharex=False, col_wrap=3)
g = g.map(sns.kdeplot, "value", shade=True)
g.axes[0].legend()
```



ML with sklearn & pandas

```
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression

categorical_columns = data_features.dtypes == object

cont_pipe = Pipeline([('scaler', StandardScaler()),
                      ('imputer', SimpleImputer(strategy='median', add_indicator=True))])
cat_pipe = Pipeline([('ohe', OneHotEncoder(handle_unknown='ignore')),
                     ('imputer', SimpleImputer(strategy='most_frequent', add_indicator=True))])

pre = ColumnTransformer([('categorical', cat_pipe, categorical_columns),
                         ('continuous', cont_pipe, ~categorical_columns),
                        ])

model = Pipeline([('preprocessing', pre), ('clf', LogisticRegression())])
param_grid = {'clf__C': np.logspace(-3, 3, 7)}
grid_search = GridSearchCV(model, param_grid=param_grid)
grid_search.fit(X_train, y_train)
```

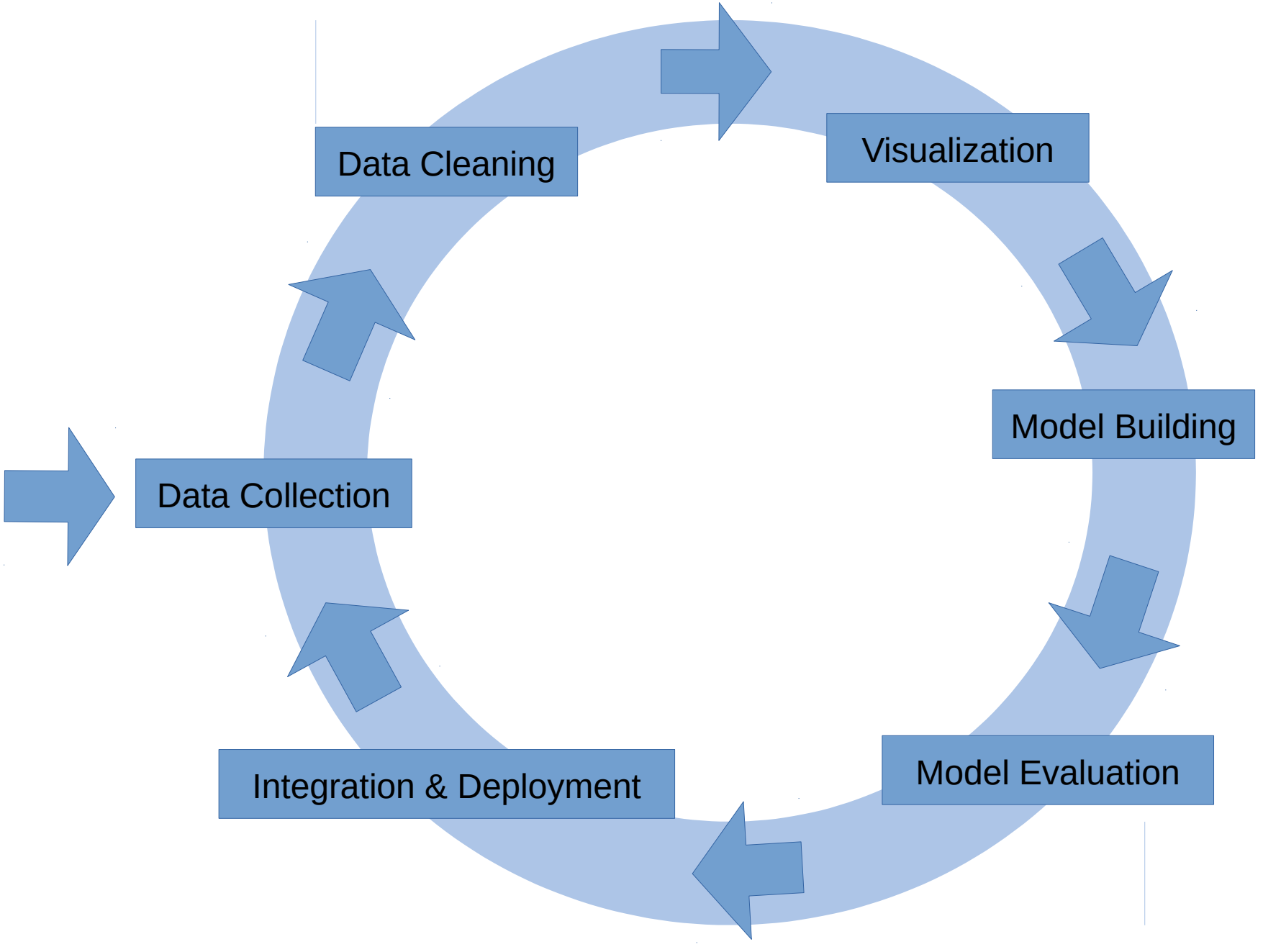
Current Automatic ML frameworks

Example

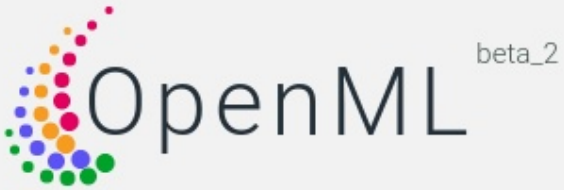
```
>>> import autosklearn.classification
>>> import sklearn.model_selection
>>> import sklearn.datasets
>>> import sklearn.metrics
>>> X, y = sklearn.datasets.load_digits(return_X_y=True)
>>> X_train, X_test, y_train, y_test = \
        sklearn.model_selection.train_test_split(X, y, random_state=1)
>>> automl = autosklearn.classification.AutoSklearnClassifier()
>>> automl.fit(X_train, y_train)
>>> y_hat = automl.predict(X_test)
>>> print("Accuracy score", sklearn.metrics.accuracy_score(y_test, y_hat))
```

This will run for one hour and should result in an accuracy above 0.98.

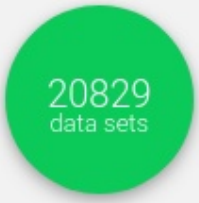
A real world ML workflow



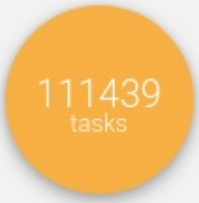
Enabling AutoML Research with OpenML



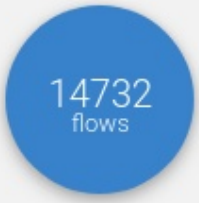
Machine learning, better, together



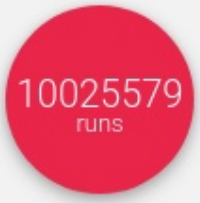
Find or add **data** to analyse



Download or create scientific **tasks**



Find or add data analysis **flows**



Upload and explore all **results** online.

Supervised Classification on credit-g

Task 31 | Supervised Classification | credit-g | 418762 runs submitted

4 likes | downloaded by 61 people , 116 total downloads | 0 issues

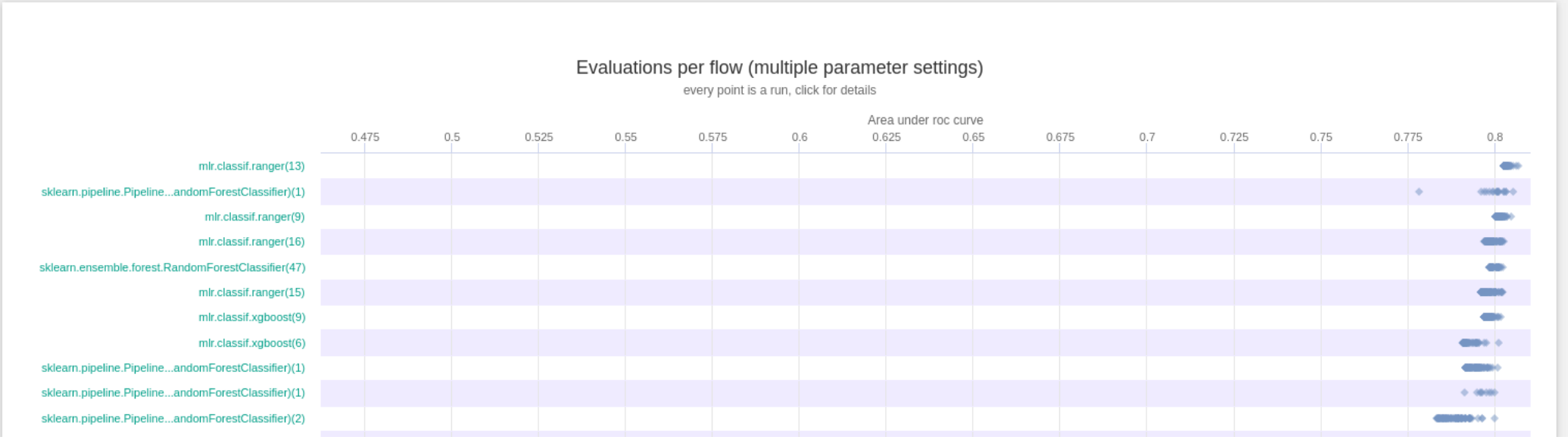
Visibility: Public

at2 basic mythbusting mythbusting_1 OpenML-CC18 OpenML100 python study_1 study_107 study_123 study_130 study_14 study_15 study_20 study_218 study_41 study_50 study_7 study_70 study_98 study_99 under100k under1m + Add tag

EVALUATIONS | PEOPLE | RUNS | + ADD RESULTS

Metric: AREA UNDER ROC CURVE

418762 Runs



```python
import openml
import sklearn.tree, sklearn.impute, sklearn.pipeline
# obtain a benchmark suite
benchmark_suite = openml.study.get_suite('OpenML-CC18')
clf = sklearn.pipeline.Pipeline(steps=[
    ('imputer', sklearn.impute.SimpleImputer()),
    ('estimator', sklearn.tree.DecisionTreeClassifier()),
])  # build a sklearn classifier
for task_id in benchmark_suite.tasks:  # iterate over all tasks
    task = openml.tasks.get_task(task_id)  # download the OpenML task
    run = openml.runs.run_model_on_task(clf, task)  # run classifier on splits
    # run.publish()  # upload the run to the server, optional
```

Figure 2: Training and evaluating a decision tree classifier from scikit-learn on each task of the OpenML-CC18 benchmark suite (Bischl et al., 2019).

Feurer, Rijn, Kadra, Gijsbers, Mallik, Ravi, Mueller, Vanschoren, Hutter: OpenML-Python: an extensible Python API for OpenML
https://arxiv.org/abs/1911.02490 (submitted to JMLR OSS)

Learning (Multiple) Defaults

Pfisterer, Rijn, Probst, Mueller, Bischl: Learning Multiple Defaults for Machine Learning Algorithms
https://arxiv.org/abs/1811.09409

Optimizing a Portfolio of Configurations

$$\max_{\hat{\theta} \subset \theta, |\hat{\theta}| = k} \sum_{j=1}^{N} \max_{c \in \hat{\theta}} S_j(c)$$

Given a space of parameters θ, datasets S_j, find an optimum portfolio $\hat{\theta}$ of size k.

	Dataset 1	Dataset 2	Dataset 3	Dataset 4	...
Configuration 1	0.9	0.1	0.5	0.4	
Configuration 2	0.8	0.5	0.4	0.6	
Configuration 3	0.4	0.9	0.3	0.1	
Configuration 4	0.4	0.2	0.4	0.3	
...					

$$\max_{\hat{\theta}\subset\theta,|\hat{\theta}|=k} \sum_{j=1}^{N} \max_{c\in\hat{\theta}} S_j(c)$$

	Dataset 1	Dataset 2	Dataset 3	Dataset 4	...
Configuration 1	0.9	0.1	0.5	0.4	
Configuration 2	0.8	0.5	0.4	0.6	
Configuration 3	0.4	0.9	0.3	0.1	
Configuration 4	0.4	0.2	0.4	0.3	
...					

$$\max_{\hat{\theta}\subset\theta,|\hat{\theta}|=k} \sum_{j=1}^{N} \max_{c\in\hat{\theta}} S_j(c)$$

	Dataset 1	Dataset 2	Dataset 3	Dataset 4	...
Configuration 1	0.9	0.1	0.5	0.4	
Configuration 2	0.8	0.5	0.4	0.6	
Configuration 3	0.4	0.9	0.3	0.1	
Configuration 4	0.4	0.2	0.4	0.3	
...					

$$\max_{\hat{\theta}\subset\theta,|\hat{\theta}|=k} \sum_{j=1}^{N} \max_{c\in\hat{\theta}} S_j(c)$$

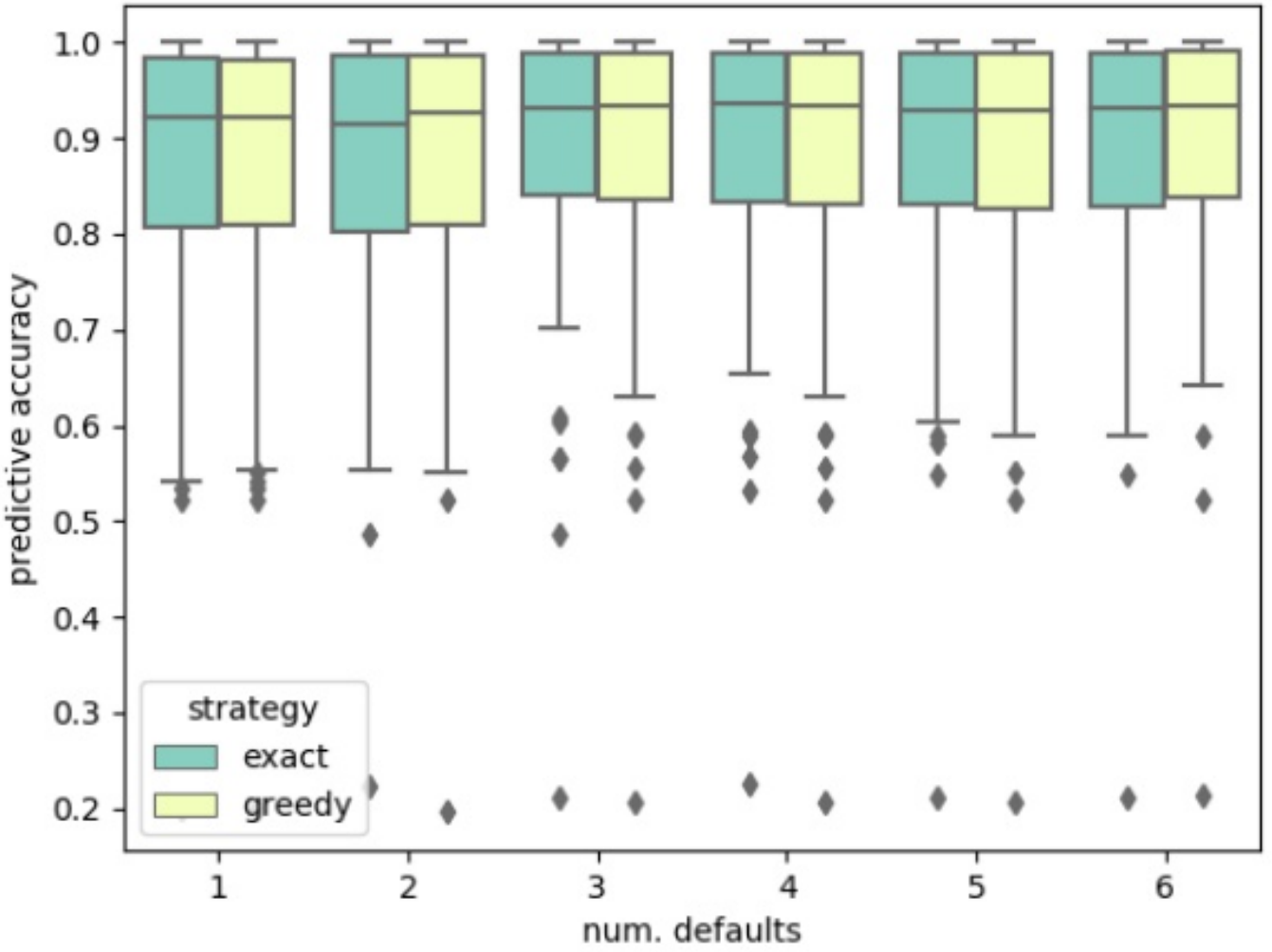
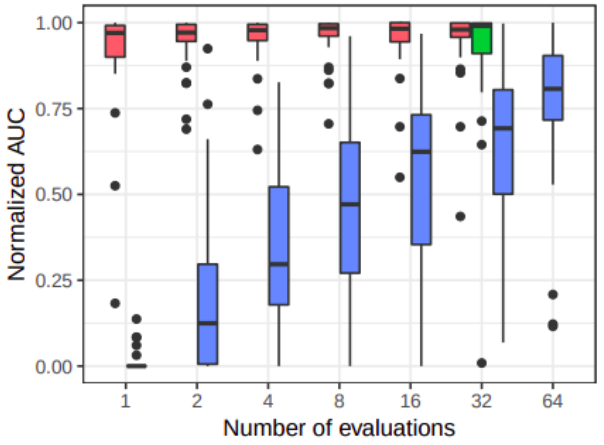
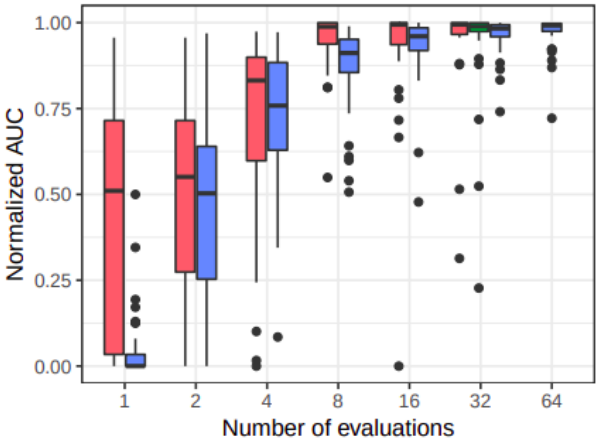


Figure 1: Performance of defaults obtained by exact discretized vs. the greedy.

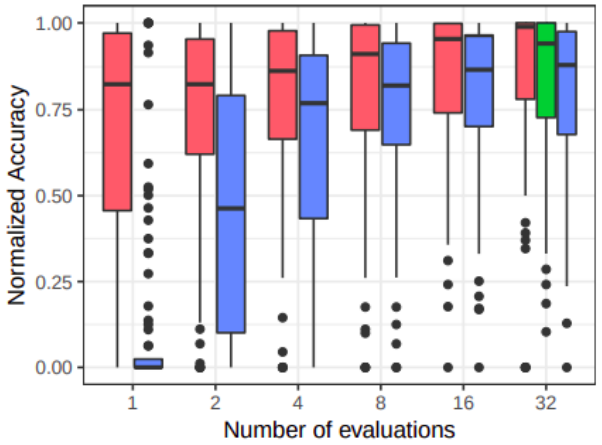
(a) Elastic net



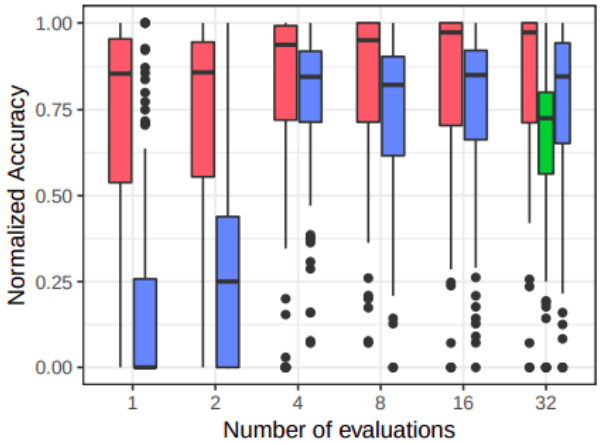
(b) Decision tree



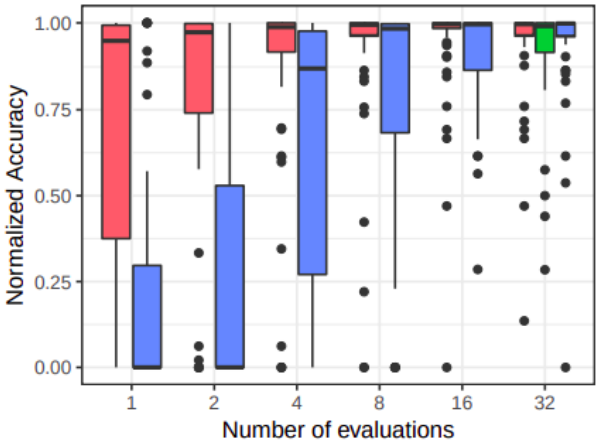
(c) Gradient boosting



(d) Adaboost



(e) Random forest

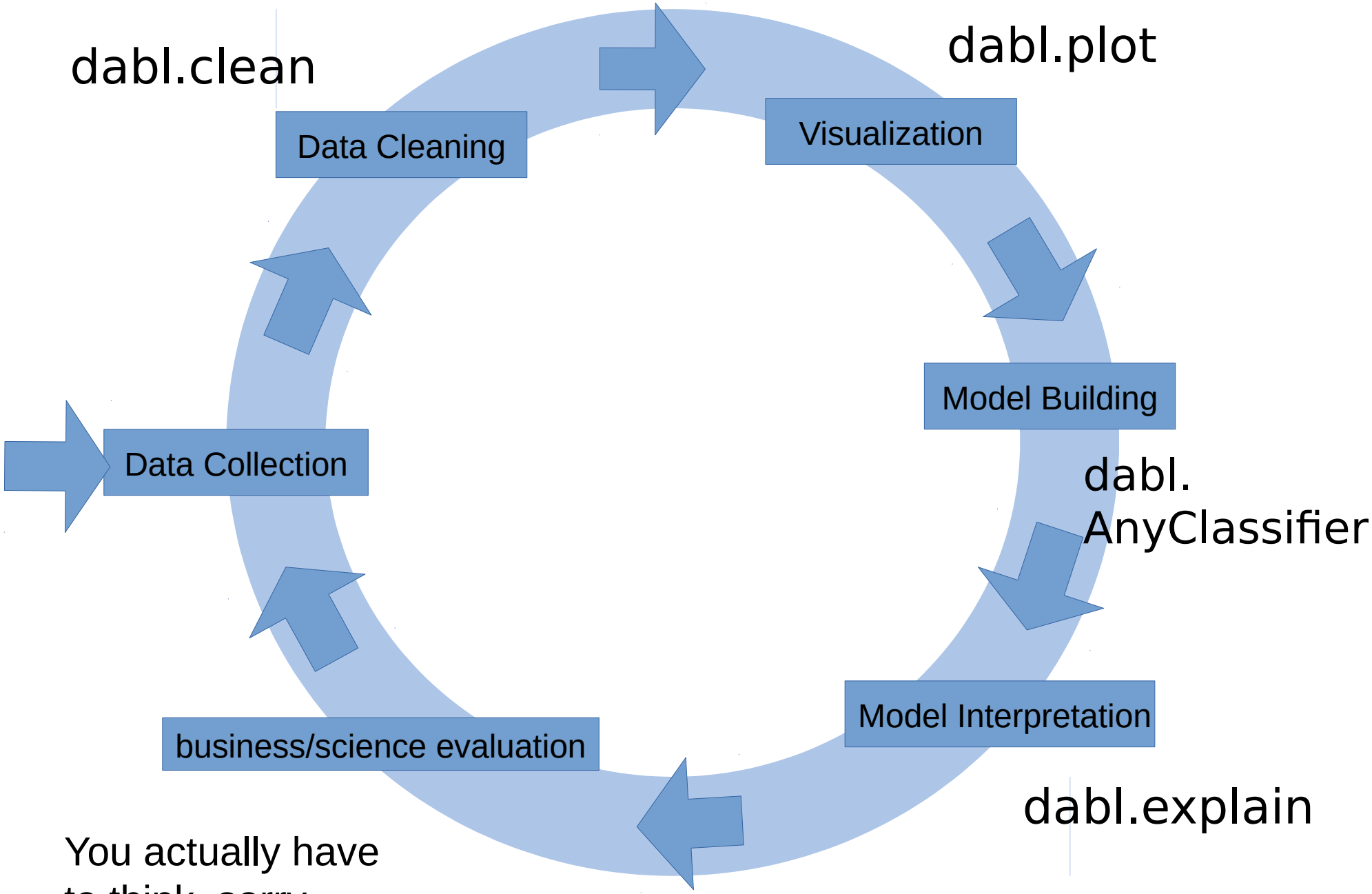


(f) SVM

Table 2: Comparison between vanilla defaults and symbolic defaults, on 98 datasets from the OpenML100 (Bischl et al., 2017). Full results are displayed in Table 3 in the appendix.

strategy	wins	symbolc default configuration
symbolic	59	$\gamma = 0.189824/\text{NumberOfFeatures}$, $C = 86.13$
vanilla	36	$\gamma = 0.001078$, $C = 4522.35$

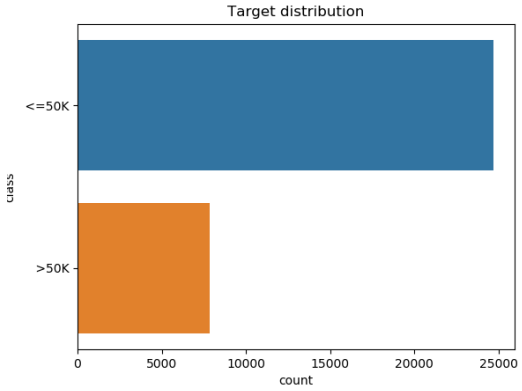
data
analysis
baseline
library

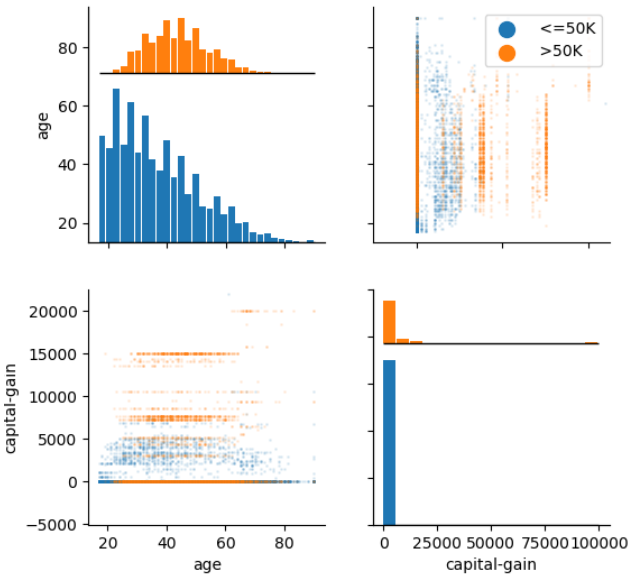

dabl.clean
dabl.plot
Data Cleaning
Visualization
Model Building
Data Collection
dabl.
AnyClassifier
Model Interpretation
business/science evaluation
dabl.explain
You actually have
to think, sorry.

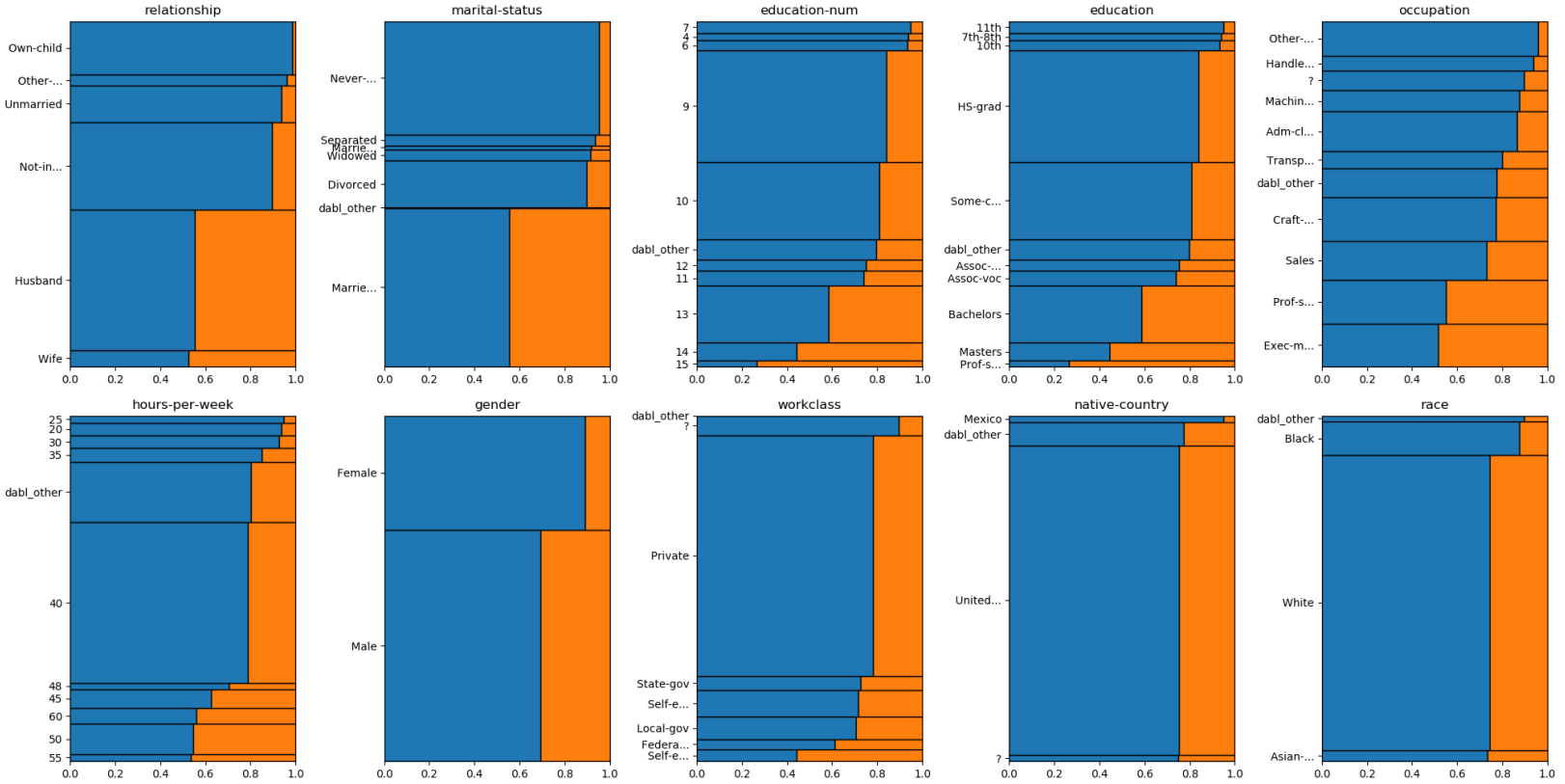
dabl.clean

- Detect types
- Detect Missing / rare values
- Detect ordinal vs categorical
- Detect near-constant
- Detect index

```
data = pd.read_csv("adult.csv")
plot(data, 'income')
```

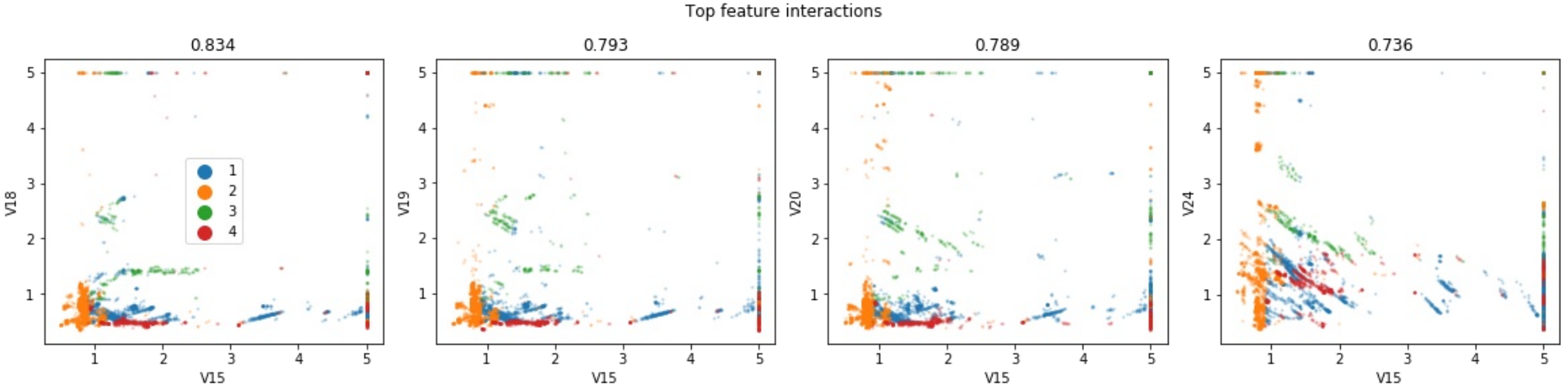






Automatic Scatterplot Selection

2d Decision Trees as Proxy for Perception



Work in progress: diverse sets of plots

Preprocessing

```
X, y = ames_df.drop('SalePrice', axis=1), ames_df.SalePrice
ep =  EasyPreprocessor().fit(X, y)
```

```
/home/andy/checkout/dabl/dabl/preprocessing.py:258: UserWarning: Discarding near-constant
'Land Slope', 'Condition 2', 'Roof Matl', 'Heating', 'Low Qual Fin SF', 'Kitchen AbvGr',
rea', 'Misc Val']
  near_constant.index[near_constant].tolist()))
```

```
ep.ct_
```

```
ColumnTransformer(n_jobs=None, remainder='drop', sparse_threshold=0.1,
                  transformer_weights=None,
                  transformers=[('continuous',
                                 Pipeline(memory=None,
                                          steps=[('simpleimputer',
                                                  SimpleImputer(add_indicator=False,
                                                                copy=True,
                                                                fill_value=None,
                                                                missing_values=nan,
                                                                strategy='median',
                                                                verbose=0)),
                                                 ('standardscaler',
                                                  StandardScaler(copy=True,
                                                                 with_mean=True,
                                                                 with_std=True))],...
```

```python
from dabl import SimpleClassifier
data = pd.read_csv("adult.csv", index_col=0)
SimpleClassifier().fit(data, target_col='income')
```

```
/home/andy/checkout/dabl/dabl/preprocessing.py:258: UserWarning: Discarding near-constant features:
  near_constant.index[near_constant].tolist()))
Running DummyClassifier(strategy='prior')
accuracy: 0.759 average_precision: 0.241 f1_macro: 0.432 recall_macro: 0.500 roc_auc: 0.500
=== new best DummyClassifier(strategy='prior') (using recall_macro):
accuracy: 0.759 average_precision: 0.241 f1_macro: 0.432 recall_macro: 0.500 roc_auc: 0.500

Running GaussianNB()
accuracy: 0.407 average_precision: 0.288 f1_macro: 0.405 recall_macro: 0.605 roc_auc: 0.607
=== new best GaussianNB() (using recall_macro):
accuracy: 0.407 average_precision: 0.288 f1_macro: 0.405 recall_macro: 0.605 roc_auc: 0.607

Running MultinomialNB()
accuracy: 0.831 average_precision: 0.773 f1_macro: 0.787 recall_macro: 0.815 roc_auc: 0.908
=== new best MultinomialNB() (using recall_macro):
accuracy: 0.831 average_precision: 0.773 f1_macro: 0.787 recall_macro: 0.815 roc_auc: 0.908

Running DecisionTreeClassifier(class_weight='balanced', max_depth=1)
accuracy: 0.710 average_precision: 0.417 f1_macro: 0.682 recall_macro: 0.759 roc_auc: 0.759
Running DecisionTreeClassifier(class_weight='balanced', max_depth=5)
accuracy: 0.784 average_precision: 0.711 f1_macro: 0.750 recall_macro: 0.811 roc_auc: 0.894
Running DecisionTreeClassifier(class_weight='balanced', min_impurity_decrease=0.01)
accuracy: 0.718 average_precision: 0.561 f1_macro: 0.693 recall_macro: 0.779 roc_auc: 0.848
Running LogisticRegression(C=0.1, class_weight='balanced')
accuracy: 0.819 average_precision: 0.789 f1_macro: 0.783 recall_macro: 0.832 roc_auc: 0.915
=== new best LogisticRegression(C=0.1, class_weight='balanced') (using recall_macro):
accuracy: 0.819 average_precision: 0.789 f1_macro: 0.783 recall_macro: 0.832 roc_auc: 0.915


Best model:
LogisticRegression(C=0.1, class_weight='balanced')
Best Scores:
accuracy: 0.819 average_precision: 0.789 f1_macro: 0.783 recall_macro: 0.832 roc_auc: 0.915
```

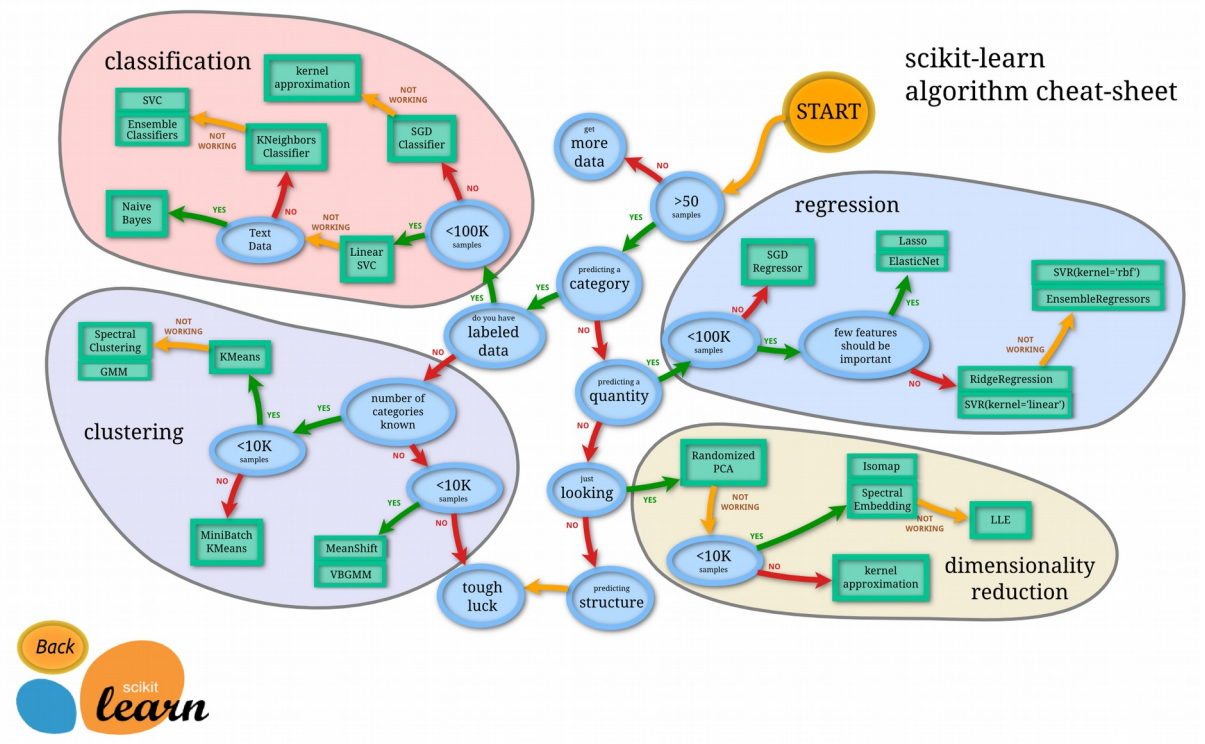
Automatic Model Search

Successive Halving on a fixed, diverse portfolio:

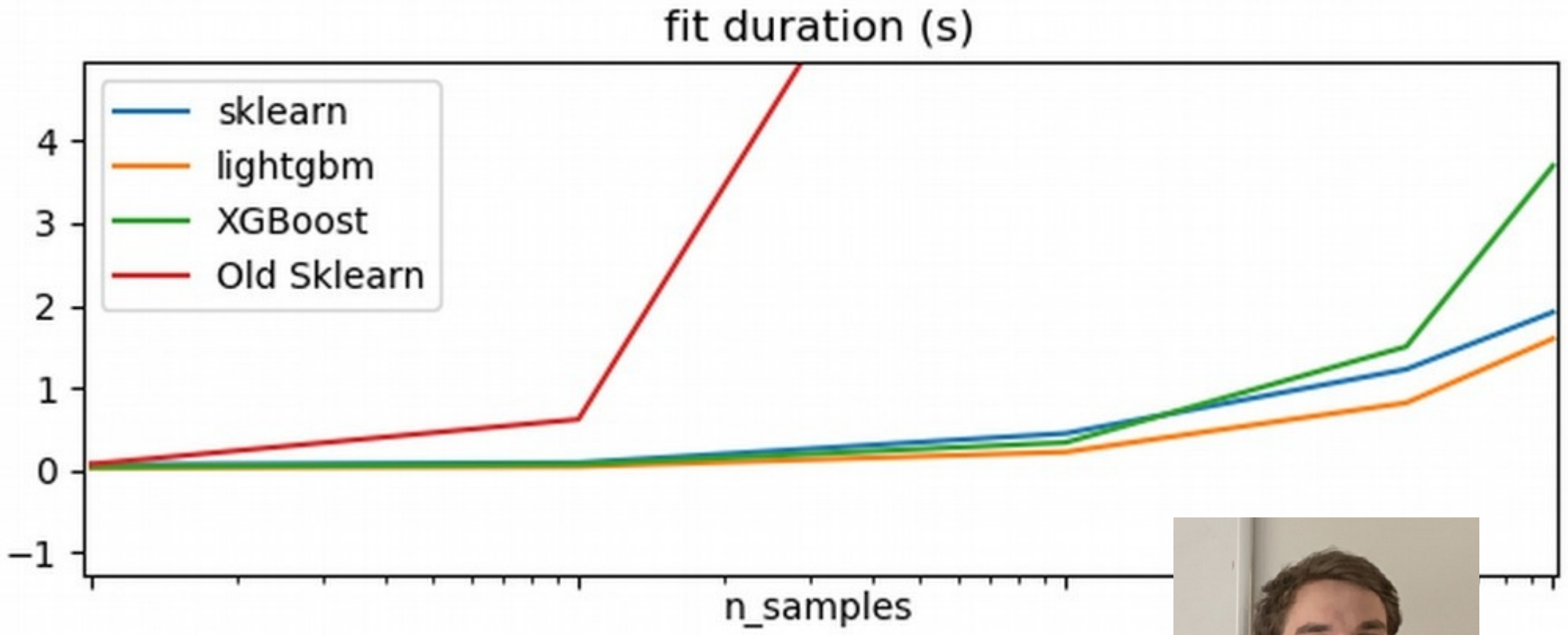
```
from sklearn.model_selection import train_test_split
df_train, df_test = train_test_split(data)
ac = AnyClassifier().fit(df_train, target_col='target')
```

Complex Models

- More Linear Models
- Random forest
- Gradient boosting
- Kernel methods



Side note: HistGradientBoosting



Successive Halving

- Given n configuration and budget B
- pick η=2 or η=3 (wording follows 2)
- Each iteration, keep best halve of configurations

SUCCESSIVEHALVING (Finite horizon)

input: Budget B, and n arms where $\ell_{i,k}$ denotes the kth loss from the ith arm, maximum size R, $\eta \geq 2$ ($\eta = 3$ by default).

Initialize: $S_0 = [n]$, $s = \min\{t \in \mathbb{N} : nR(t+1)\eta^{-t} \leq B, t \leq \log_\eta(\min\{R, n\})\}$.

For $k = 0, 1, \ldots, s$

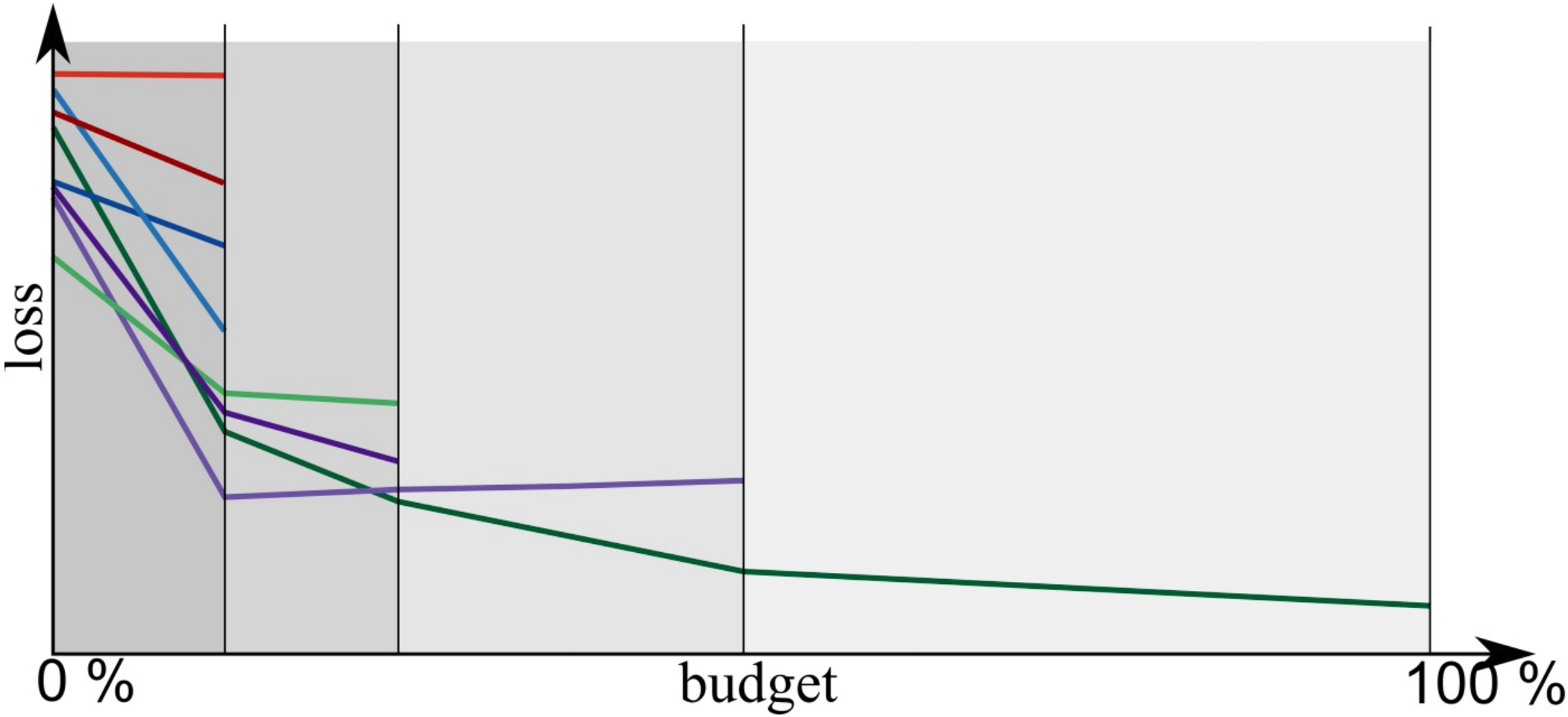
Set $n_k = \lfloor n\eta^{-k} \rfloor$, $r_k = \lfloor R\eta^{k-s} \rfloor$

Pull each arm in S_k for r_k times.

Keep the best $\lfloor n\eta^{-(k+1)} \rfloor$ arms in terms of the r_kth observed loss as S_{k+1}.

Output : $\hat{\imath}, \ell_{\hat{\imath},R}$ where $\hat{\imath} = \arg\min_{i \in S_{s+1}} \ell_{i,R}$

Successive Halving Illustrated



Portfolio creation

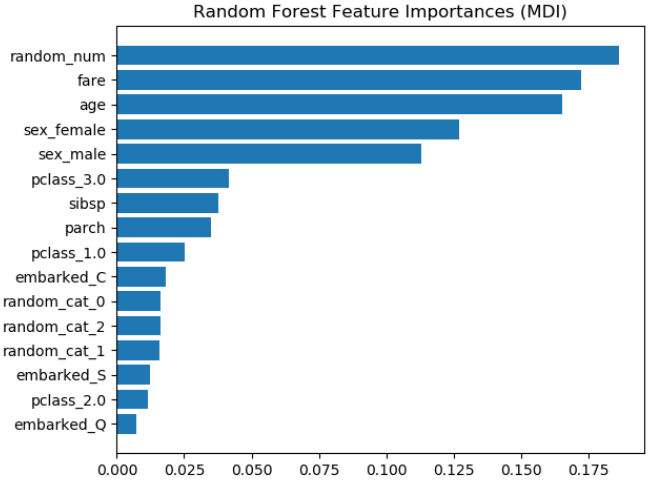
- Run Hyper-parameter optimization across models on large benchmark suite (OpenML-CC18)
- Evaluate all final models across all datasets
- Greedily create portfolio of best-performing, diverse models

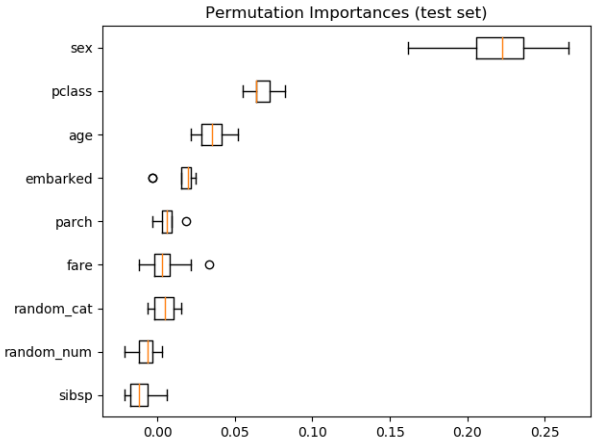
c.f. “Practical Automated Machine Learning for the AutoML Challenge 2018” Feurer et. al.

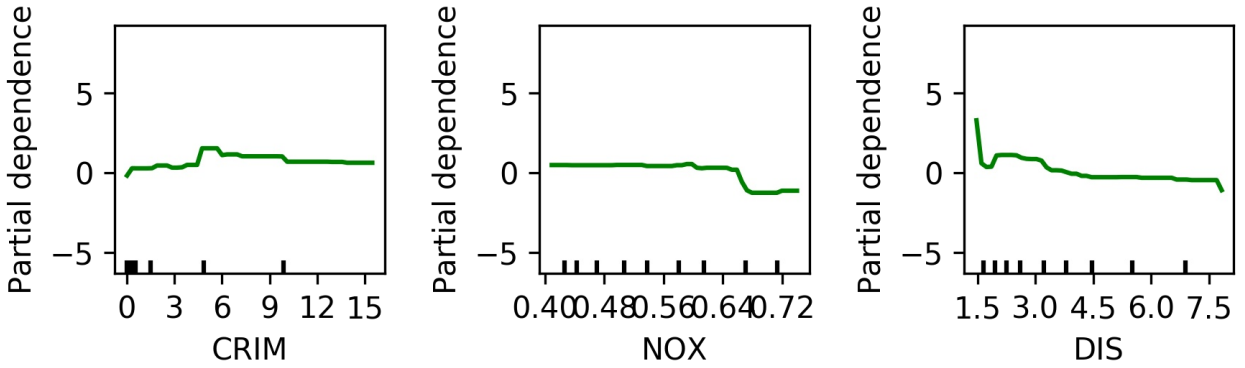
```
from sklearn.model_selection import train_test_split
df_train, df_test = train_test_split(data)
ac = AnyClassifier().fit(df_train, target_col='target')
```

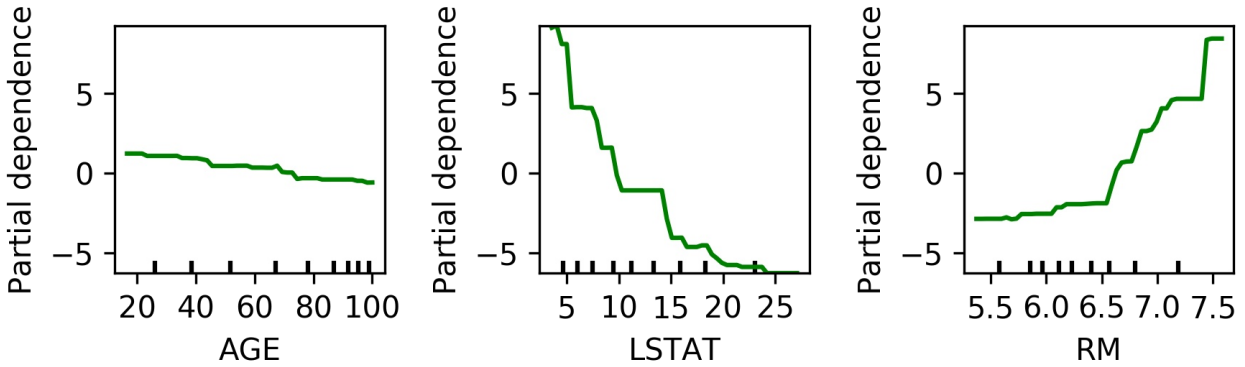
```
import dabl
dabl.explain(ac, X_val=df_test, target_col='target')
```

Model Explanation



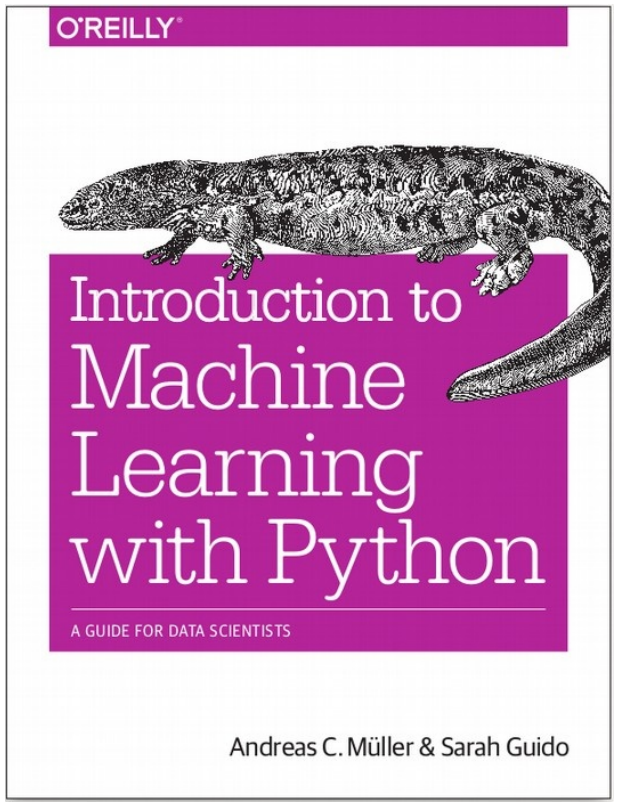






$ pip install dabl

https://amueller.github.io/dabl



amueller.github.io

@amuellerml

@amueller

andreas.mueller
@columbia.com