Exploring Auto-Sklearn Models with PipelineProfiler

How to open the black box of AutoML

Jorge Piazentin Ono Jun 3, 2020 · 4 min read ★

Image credit: Negative Space

Building machine learning pipelines is often a difficult, time-consuming and trial-and-error task. AutoML makes this process easier, by automatically selecting computational steps, tuning hyperparameters and training end-to-end models that solve a machine learning problem. Auto-Sklearn[1] is one of the most popular open source AutoML systems. Given a dataset, a problem type (classification or regression) and a metric score, Auto-Sklearn is able to produce ensemble pipelines that optimize the chosen metric and produce good results.

Take, for example, the *Digits* dataset, where we want to classify images of numbers. We can automatically create a pipeline that solves this problem with a few lines of code:

```
import sklearn.datasets
import autosklearn.classification
X, y = sklearn.datasets.load_digits(return_X_y=True)
automl = autosklearn.classification.AutoSklearnClassifier()
automl.fit(X, y, dataset_name='digits')
```

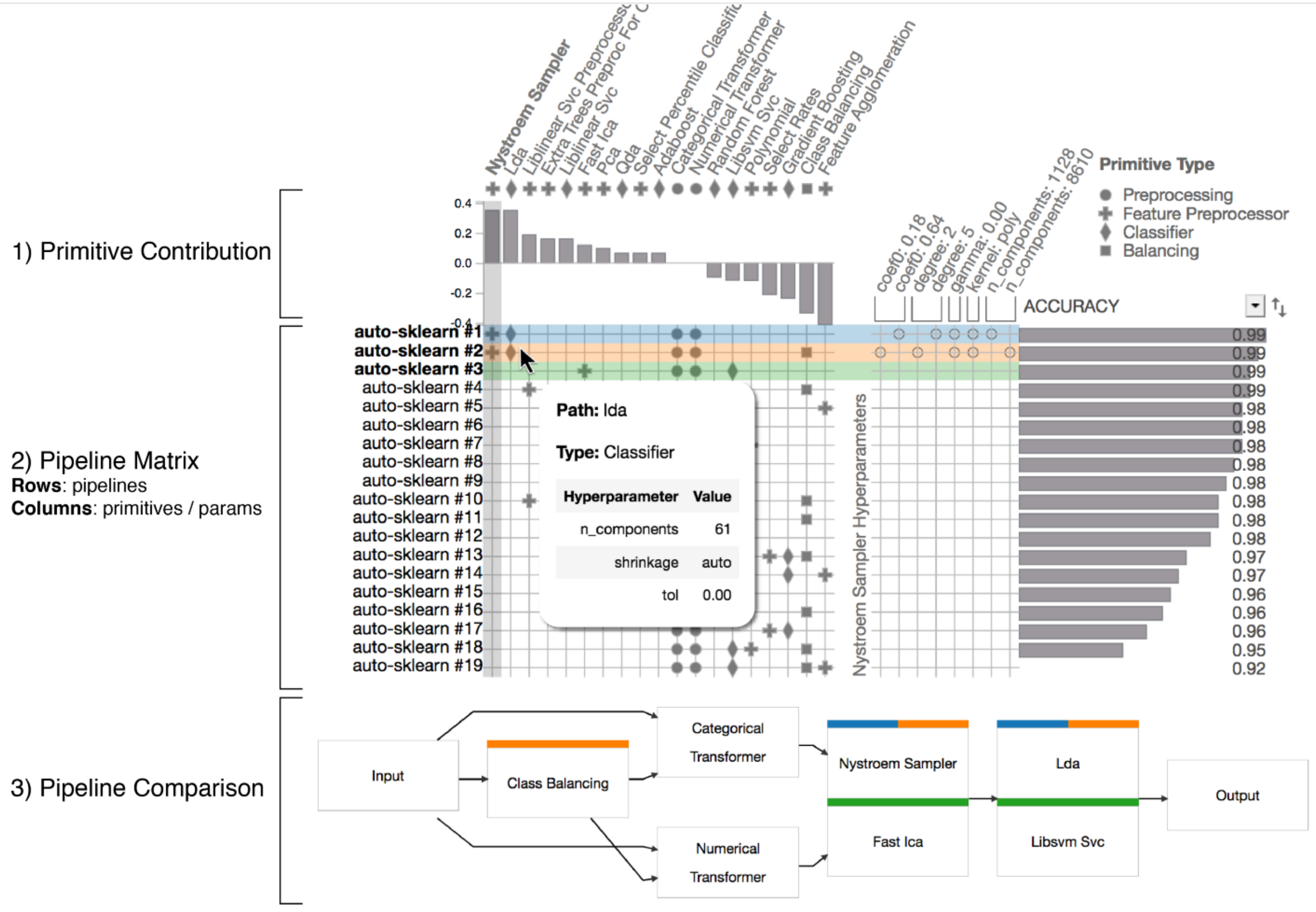
obscure: the data scientist does not know what steps were taken to produce the pipeline, what part of the search space was explored, and what hyperparameters were tuned.

In this blog post, we show how *PipelineProfiler* can be used to debug and explore the models produced by Auto-Sklearn. *PipelineProfiler* is a new library aimed at producing detailed visualizations of end-to-end machine learning pipelines. It is integrated with Python Notebooks (Jupyter and Google Colab), and can be used within the data science workflow.

The PipelineProfiler Tool

Given the trained models produced by Auto-Sklearn, we can use PipelineProfiler with three lines of code:

```
import PipelineProfiler
profiler_data = PipelineProfiler.import_autosklearn(automl)
PipelineProfiler.plot_pipeline_matrix(profiler_data)
```



Path: lda

Type: Classifier

Hyperparameter	Value
n_components	61
shrinkage	auto
tol	0.00

Fig. 1 PipelineProfiler applied to the Digits dataset.

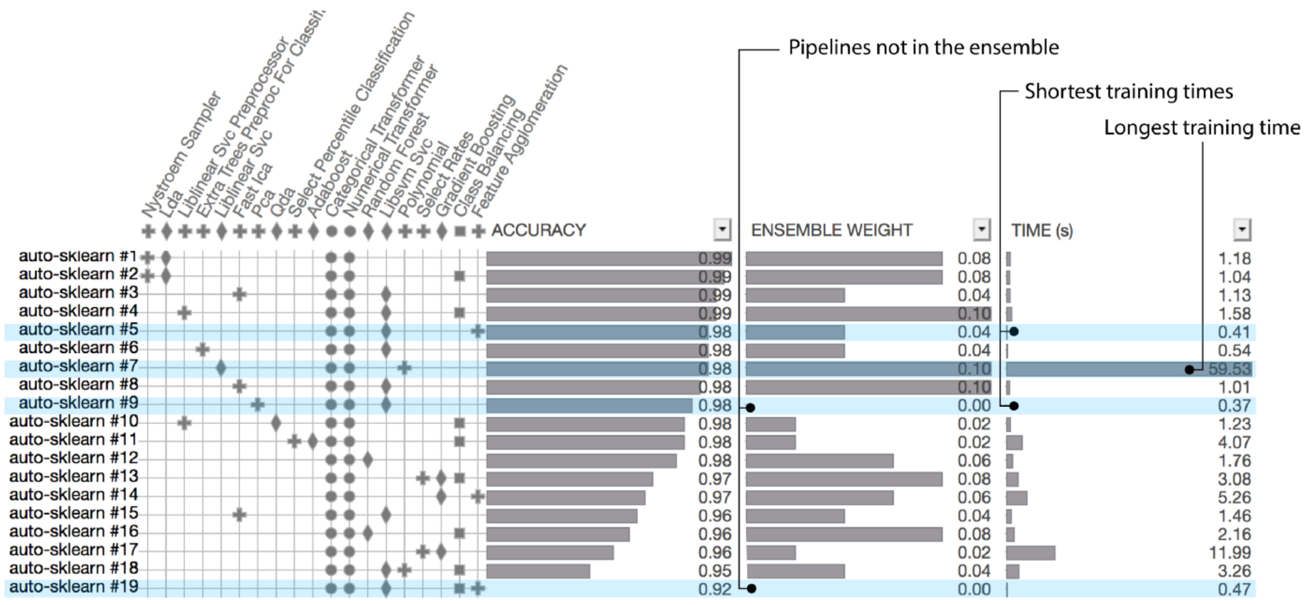
Fig. 1 shows *PipelineProfiler* applied to the Digits dataset. The system is divided into three parts. 1) The primitive contribution, showing the correlation of primitive usage with the pipeline scores. 2) The Pipeline Matrix, presenting a summary of the pipelines, the primitives used and the hyperparameters. 3) The Pipeline Comparison view, highlighting the differences among selected pipelines.

Pipeline Analysis

Fig 1. shows that the top two pipelines use *Nystroem Sampler*, *LDA* and *Transformers*, but differ on the use of *Class Balancing* and the *Sampler* hyperparameters. The Primitive Contribution shows us that *Nystroem Sampler* and LDA are correlated with high Accuracy Scores. Conversely, *Feature Agglomeration* is correlated with low scores.

The Pipeline Comparison view presents a node-link representation of the selected pipelines (multiple pipelines can be selected by *shift-clicking*). Fig 1. shows the

PipelineProfiler also displays performance metrics of the evaluated pipelines. In Fig 1., we see the accuracy score of all pipelines. However, we can switch this view to other pipeline metrics. For example, the ensemble produced by Auto-Sklearn consists of a weighted average of all the evaluated pipelines. We can switch the metric score to display the ensemble weights. We can also visualize the training time of each of the pipelines. Fig 2. shows the multiple metrics that can be displayed. We removed the primitive contribution view from this image, as the contribution would be different for each of the selected metrics.



	ACCURACY	ENSEMBLE WEIGHT	TIME (s)
auto-sklearn #1	0.99	0.08	1.18
auto-sklearn #2	0.99	0.08	1.04
auto-sklearn #3	0.99	0.04	1.13
auto-sklearn #4	0.99	0.10	1.58
auto-sklearn #5	0.98	0.04	0.41
auto-sklearn #6	0.98	0.04	0.54
auto-sklearn #7	0.98	0.10	59.53
auto-sklearn #8	0.98	0.10	1.01
auto-sklearn #9	0.98	0.00	0.37
auto-sklearn #10	0.98	0.02	1.23
auto-sklearn #11	0.98	0.02	4.07
auto-sklearn #12	0.98	0.06	1.76
auto-sklearn #13	0.97	0.08	3.08
auto-sklearn #14	0.97	0.06	5.26
auto-sklearn #15	0.96	0.04	1.46
auto-sklearn #16	0.96	0.08	2.16
auto-sklearn #17	0.96	0.02	11.99
auto-sklearn #18	0.95	0.04	3.26
auto-sklearn #19	0.92	0.00	0.47

Fig. 2 PipelineProfiler can be used to inspect metrics provided by Auto-Sklearn, e.g. Accuracy, Ensemble Weight and Time.

We notice that pipelines #9 and #19 do not belong to the ensemble, as their weight is set to zero. Furthermore, we see that pipeline #7 took the longest to train, and pipelines #9 and #5 had the fastest training times. We also notice that pipeline #1 has the best accuracy, but does not have the highest ensemble weight. This shows that Auto-Sklearn uses other criteria to assign weights to pipelines in the ensemble.

If we are interested in what primitives contributed to the long training time, we can investigate the primitive contribution view:

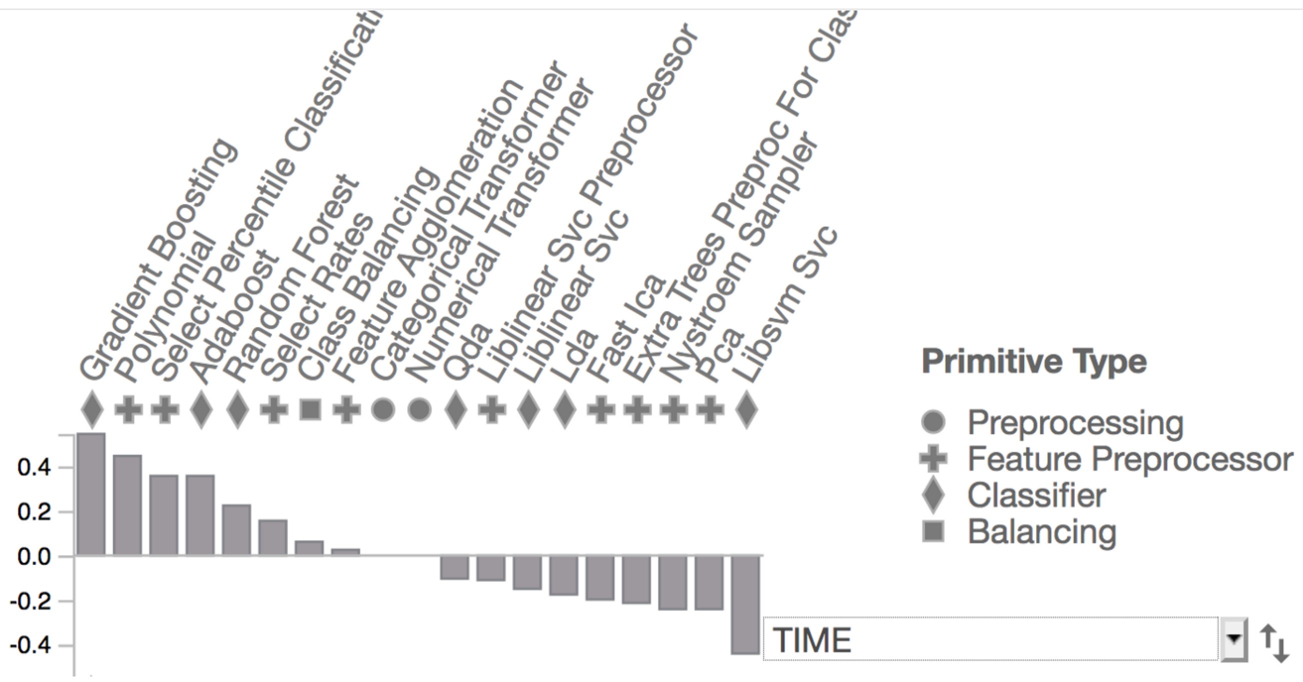


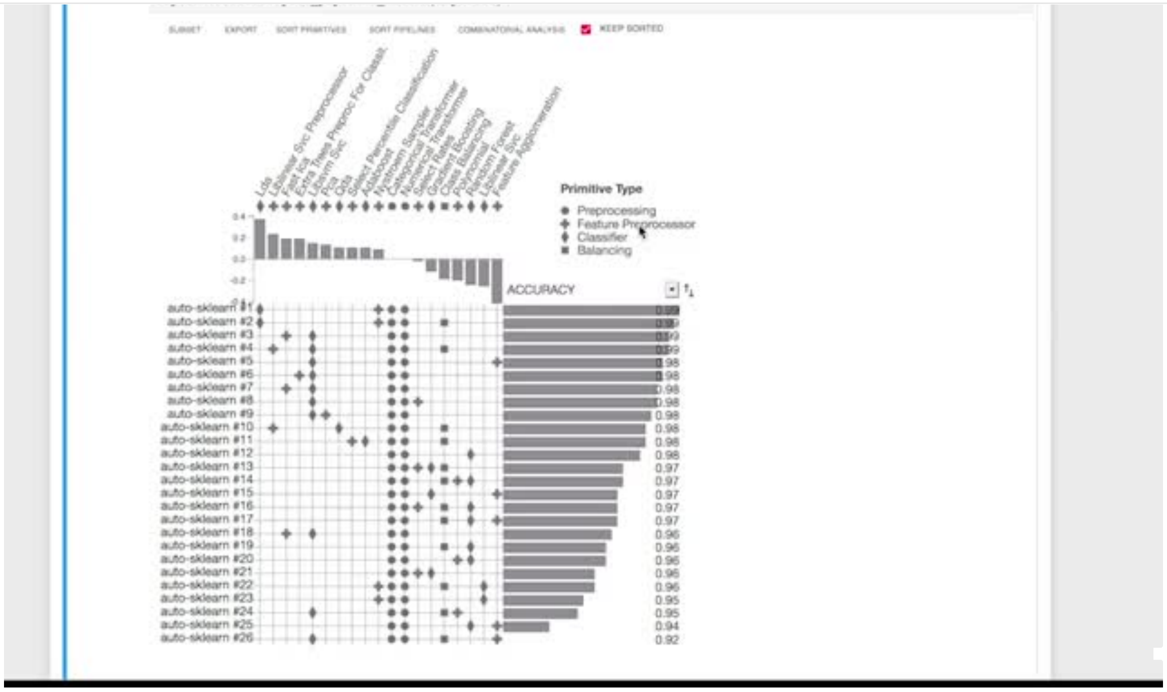
Fig 3. Primitive contribution computed based on the Time metric. Gradient Boosting is the primitive most correlated with the pipeline training time.

Here, we see that *Gradient Boosting* and *Polynomial* were two primitives correlated with the high training times, therefore they are probably to blame. Intuitively, the primitive *Polynomial* would result in longer training times, since it generates a large number of polynomial features from the original data.

Final Thoughts

This post presented a brief introduction to *PipelineProfiler*, a Python library that can be used to explore AutoML models. We have seen how we can import and visualize pipelines from Auto-Sklearn in Jupyter Notebook.

PipelineProfiler is a new tool and we are actively seeking feedback on what features we should support. Let us know if you have any questions or would like to collaborate!



4.2K views

gfycat

Installation and Demo

Machine Learning Automl Visualization Jupyter Notebook

The code is open sourced under the BSD 3 license. The github repository is: https://github.cc /VIDA-NYU/PipelineVis

PipelineProfiler is available in the PyPI repository. It can be installed via the command:

```
pip install pipelineprofiler
```

at r paper[2] and watch our video demo:

PipelineProfiler: A Visual Analytics Tool for the Exploration of AutoML Pipelines