Evolutionary optimization of machine learning pipelines

Gabriela Suchopárová

Abstract

We have developed a system that, for a given supervised learning task, finds a suitable pipeline — combination of machine learning, ensembles and preprocessing methods — using the developmental genetic programming.

AutoML

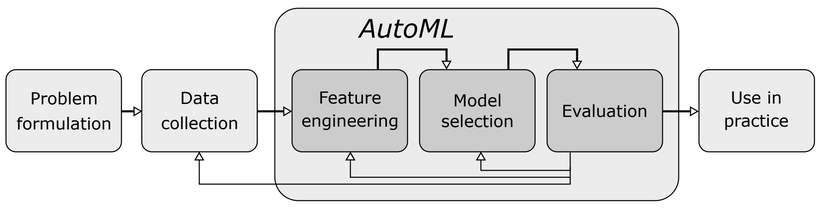

Figure 1: Schema of a general workflow

The AutoML systems aim to automatize a part of the machine learning workflow, which is the process of selection of suitable feature preprocessing methods and of a model with a particular hyperparameter setting.

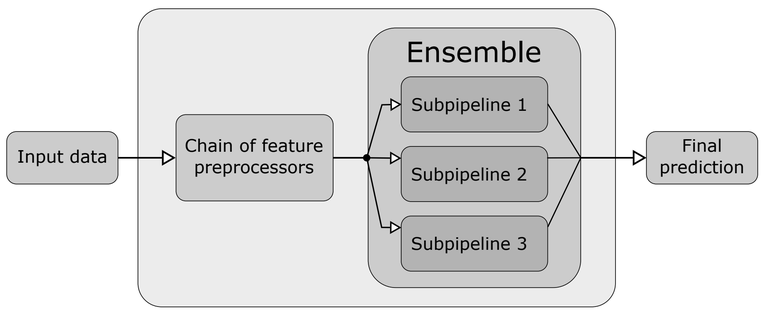

Figure 2: A more complex machine learning pipeline

One of the tasks targeted by AutoML systems is *pipeline optimization*. A machine learning pipeline is composed from a chain of feature preprocessing methods and one estimator. The estimator can be a simple method or an ensemble that contains one or more base-estimators. Because a base-estimator can be a pipeline as well, a general pipeline is a directed acyclic graph.

Our system

Our system aims to optimize scikit-learn pipelines. The search space comprises of all scikit-learn preprocessors, ensembles and simple methods. We also extended the Pipeline class to act as a base-estimator of an ensemble.

Algorithm 1: Pipeline optimization — main

Data: dataset d
$individuals \longleftarrow$ run developmental GP on a dataset
$pipelines \longleftarrow$ `compile(`$individuals$`)`

return $pipelines$

For pipeline optimization, we used the genetic programming (GP), which is a subfield of evolutionary algorithms. We evolve pipelines encoded as GP individuals. The result is a set of pipelines that perform the best on the input dataset.

Developmental GP

We created a specific encoding that enables to convert pipelines in the form of a DAG into a tree representation. Instead of directly encoding pipeline steps as nodes, we apply the developmental GP, where the nodes represent *operations* that create the pipeline. An example of the encoding is shown in Figure 3. The tree individual contains instructions which construct the actual pipeline:

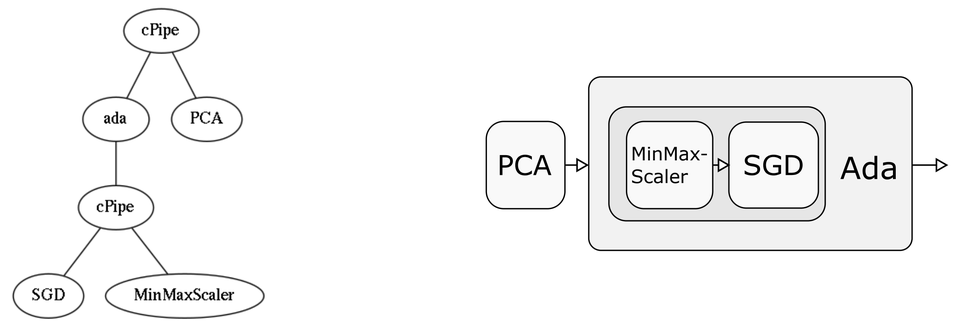

Figure 3: Example of an encoded pipeline

cPipe — create a pipeline with a preprocessor chain and an estimator
- **ada** — insert an AdaBoost ensemble with a base-estimator
  - **cPipe** — create a pipeline with a preprocessor chain
    - **SGD** — insert a SGD classifier
    - **MinMaxScaler** — insert a MinMaxScaler
- **PCA** — insert the PCA preprocessor

Evolutionary algorithm

In GP, the fittest tree individuals are combined to produce better individuals. The fitness is composed from the score of the encoded pipeline on the input dataset and from evaluation time.

Algorithm 2: Pipeline optimization — developmental GP

$P(0) \longleftarrow$ initial population of GP trees
while $n < max\_gen$**:**
  `evalutate(`$P(0)$`)`
  for i in `range(`$i/2$`):`
    $i_1, i_2 \longleftarrow$ tournament selection from $P(n)$
    `reproduction(`i_1, i_2`)`
    `evalutate(`i_1 *and* i_2`)`
    add i_1, i_2 to $P_o(n)$

$P(n+1) \longleftarrow$ NSGA-II selection from $P_o(n)$ and $P(n)$

return $pipelines$

OpenML experiment

Our system was tested on the OpenML-CC18 benchmark, which is a suite of 72 datasets available at OpenML. Each of the datasets is associated with a classification task, the evaluation method is 10-fold cross-validation. We present a comparison of our results with all other results uploaded to OpenML. On most of the datasets our system performed in the upper quartile of all results.

OpenML-CC18 benchmark