Automated Data Science

Technical Review

Author: Laurent Querella, PhD, MSc

Senior Data Scientist

Version : draft version 1.0 – Dec 2021

Contents

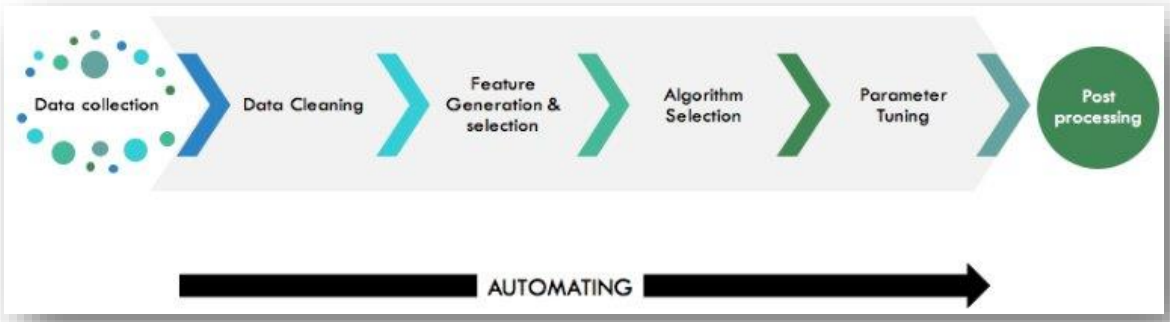
1 References and Links

- KDNuggets
  - http://www.kdnuggets.com/2016/03/automated-data-science.html
  - http://www.kdnuggets.com/2016/06/kdnuggets-blog-contest-automated-data-science.html
- Sotware/Vendors
  - http://www.kdnuggets.com/software/automated-data-science.html
- AutoML (*taking the human expert out of the loop*)
  - http://www.automl.org/

2 Overview

The growing trend towards automation has also touched Data Science, with more companies and research projects aiming to build the ultimate Data Science platform - the user just points it to the data and it does the rest.

2.1 Data Science Pipeline



2.1.1 Data Preparation

Lot of research has been undergoing in automating **Data Cleaning & Feature Generation & Selection** aspect of DS pipeline because they consume almost 70% of the data science project time.

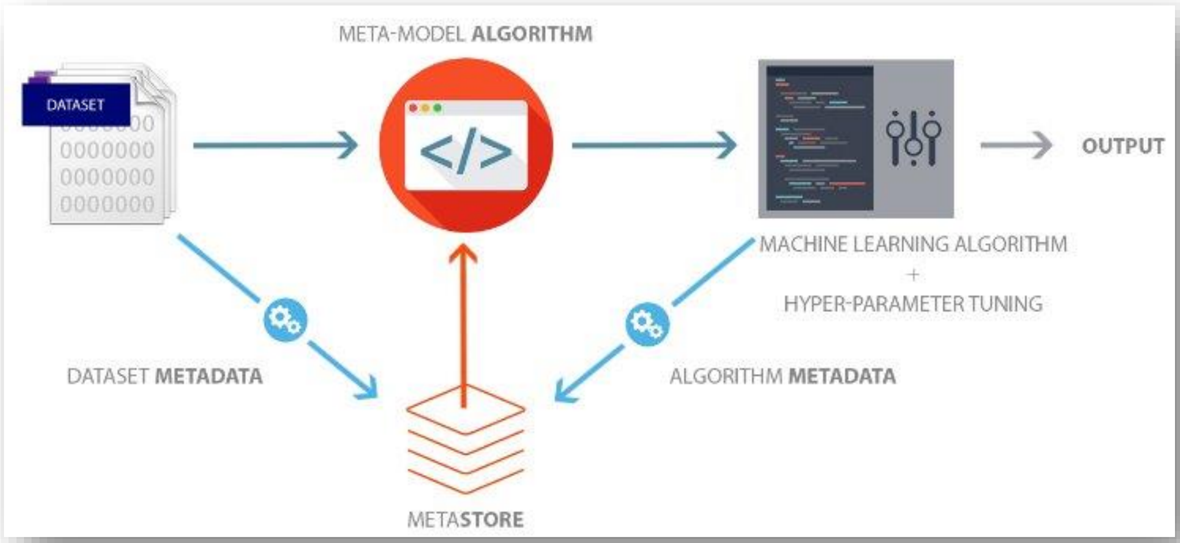
2.1.2 Algorithm Selection and Parameter Tuning

Both of these components are considered manual in nature. Data scientists spend hours applying multiple ML algorithms and then tuning the parameters. For every new problem they follow the same process and over period they develop the intuition of which algorithm will work better for which type of dataset and how best they can tune the parameters. If you look closely this is in itself a learning problem. Lately lot of research & development has been started to automate these components to get best accuracy in least amount of time and efforts.

Automating Algorithm selection can be done using concept called **Meta Learning** while to automate Parameter tuning there are different techniques like Grid search, bayesian optimization, etc.

2.1.2.1 Automating Algorithm Selection using Meta Learning

There are two essential parts of Meta-learning - Meta Store & Meta-model algorithm.

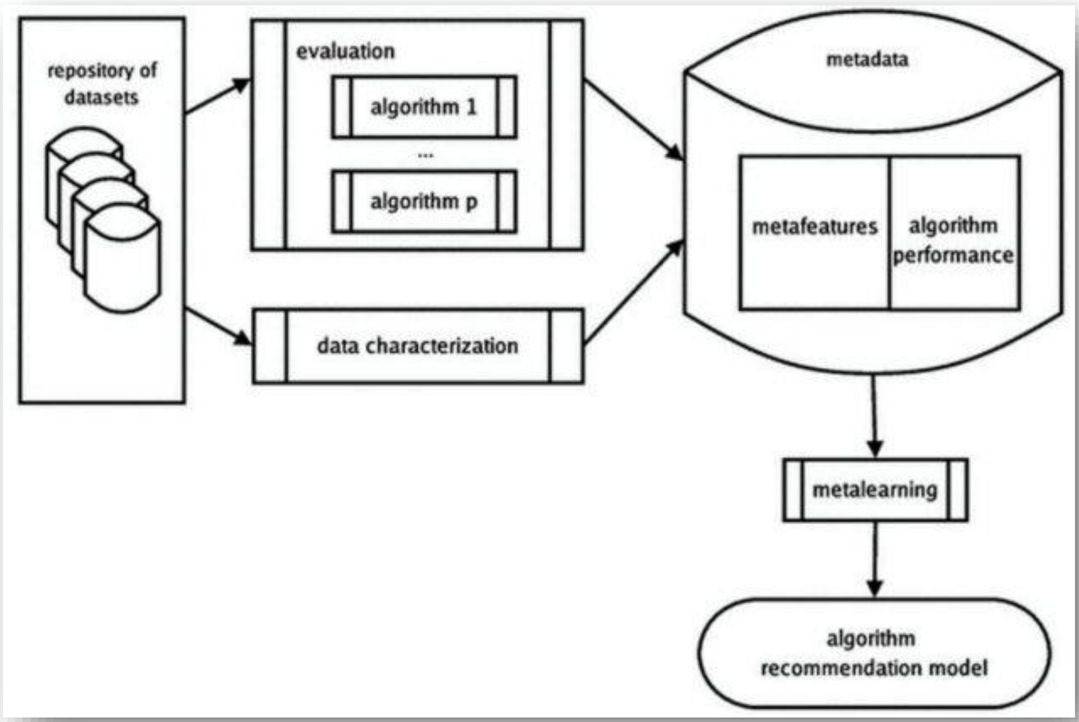


Meta store is used to capture metadata of datasets and metadata of algorithm which gave best results after hyper-parameter tuning. As soon as dataset is loaded into the environment, its metadata is generated and stored in the centralized MetaStore. Before running any machine learning algorithm (specially for classification and regression model) Meta-model algorithm will analyze the metadata of this dataset and run algorithm ranking logic which will predict the machine learning model out of 10-20 different models that which model might perform better on this dataset.

Once the algorithm is selected, next step is to run hyper-parameter tuning algorithms to tune this model. Meta model algorithm will essentially use MetaStore and machine learning algorithm. Meta-model algorithm will also help in reducing the hyper parameter space which boosts convergence rapidly to provide best result.

2.1.2.2 Meta-model algorithm

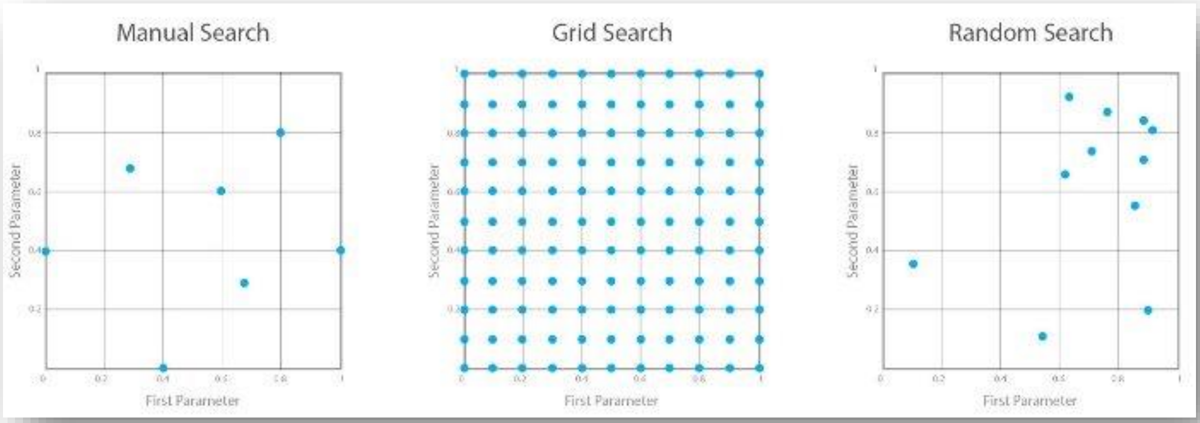
This is the brain of Meta-learning. This itself is a machine learning algorithm which basically learns from the various run of different algorithms on a lot of datasets. Algorithm metadata which we derived from best algorithm run on a particular dataset and stored it in MetaStore, and dataset metadata are combined together to form a new dataset which is fed to machine learning algorithm which ranks section of ML algorithms for a particular dataset.



2.1.2.3 Hyper parameter tuning

Almost all machine learning algorithm have hyper-parameters. SVM has Regularization parameter(C), k-means has Number of clusters(k), Neural network has Number of hidden layers, dropout, etc as hyperparameters.

Efficiently finding the best solution of these hyper-parameters is the key to any ML solution. There are basically three famous techniques to tune the hyper-parameters of any machine learning algorithms:



- **Manual Search**: Data Scientists try different values and by luck may end up getting the best solution. It is very common among DS community.

- **Grid Search**: Machine does all the work by evaluating each parameters combinations and checking for best result. It works only for less feature; as the feature set increases so does the number of combination and thereby computation time.
- **Random Search**: Works with large features

There has been some recent advancements in this field where data scientists are using **Bayesian approach to optimize hyper-parameters**. This approach works well with very large number of features but can be an overkill for less data and dataset with less features. The goal is to maximize some true unknown function f. Information about this function is gained by making observations, which are evaluations of the function at specific hyper-parameter values. These observations are used to infer a posterior distribution over the function values representing the distribution of possible functions. This approach takes less number of iterations to optimize the values of hyperparameters. It may not find the best solution always but will give relatively close to best solution in amazingly less number of iterations.

Opensource tools:

- Spearmint is a package to perform **Bayesian optimization** according to the algorithms outlined in the paper: Practical Bayesian Optimization of Machine Learning Algorithms. Jasper Snoek, Hugo Larochelle and Ryan P. Adams. Advances in Neural Information Processing Systems, 2012
  - https://github.com/JasperSnoek/spearmint/
- A Python tool that automatically creates and optimizes machine learning pipelines using **genetic programming**
  - https://github.com/rhiever/tpot
- Distributed Asynchronous Hyperparameter Optimization in Python
  - https://github.com/hyperopt/hyperopt

2.2 AutoML

Cf. Website above.

HPOlib is a hyperparameter optimization library.

3 Solutions

3.1 Auto-sklearn

Auto-sklearn is an open-source Python tool that automatically determines effective machine learning pipelines for classification and regression datasets. It is built around the successful scikit-learn library and won the recent AutoML challenge.

Despite the great success of machine learning (ML) in many fields (e.g., computer vision, speech recognition or machine translation), it is very hard for novice users to effectively apply ML: they need to decide between dozens of available ML algorithms and preprocessing methods, and tune the hyperparameters of the selected approaches for the dataset at hand. Doing this right often makes the difference between poor and state-of- the-art performance, but the need for these tedious manual tasks constitutes a substantial burden for real-world applications of machine learning

With ever-more industrial uses of machine learning, there is now a strong demand for robust machine learning systems that autonomously perform inference on a given dataset. Recent *automated machine learning* (AutoML) systems find the right algorithm and hyperparameters in a data-driven way without any human intervention. In this blog post we describe how this is done in our AutoML system Auto-sklearn that won the recent AutoML challenge.

3.1.1 Algorithm and hyperparameter selection

A prominent method for optimizing machine learning hyperparameters is Bayesian optimization, which iterates the following steps:

1. Build a probabilistic model to capture the relationship between hyperparameter settings and their performance
2. Use the model to select useful hyperparameter settings to try next by trading off exploration (searching in parts of the space where the model is uncertain) and exploitation (focussing on parts of the space predicted to perform well)
3. Run the machine learning algorithm with those hyperparameter settings.

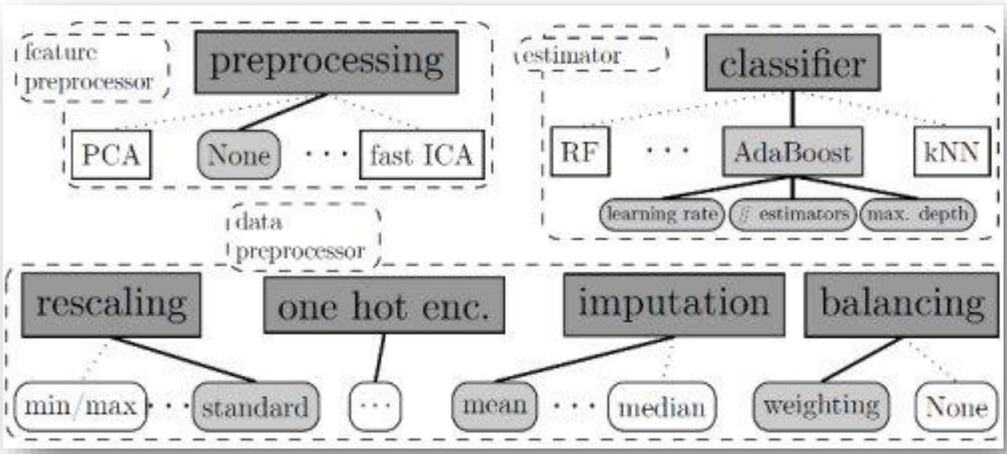
This process can be generalized to jointly select algorithms, preprocessing methods, and their hyperparameters as follows: the choices of classifier / regressor and preprocessing methods are top-level, categorical hyperparameters, and based on their settings the hyperparameters of the selected methods become active. The combined space can then be searched with Bayesian optimization methods that handle such high-dimensional, conditional spaces; we use the random-forest- based SMAC, which has been shown to work best for such cases.

This AutoML approach of using Bayesian optimization to automatically customize very general ML frameworks for given datasets was first introduced in Auto-WEKA. (As the name suggests, Auto-WEKA is based on WEKA; now in version 2.0, it is integrated with WEKA's package manager, has a GUI, an API, and a command line interface, and is very popular with the WEKA user base, with an average of 250 downloads per week). Auto-sklearn serves a very similar purpose in Python.

3.1.2 Auto-sklearn

Auto-sklearn is our implementation of the above idea. It is open source, implemented in python and built around the scikit-learn library. It contains a machine learning pipeline which takes care of missing values, categorical features, sparse and dense data, and rescaling the data. Next, the pipeline applies a preprocessing algorithm and an ML algorithm.

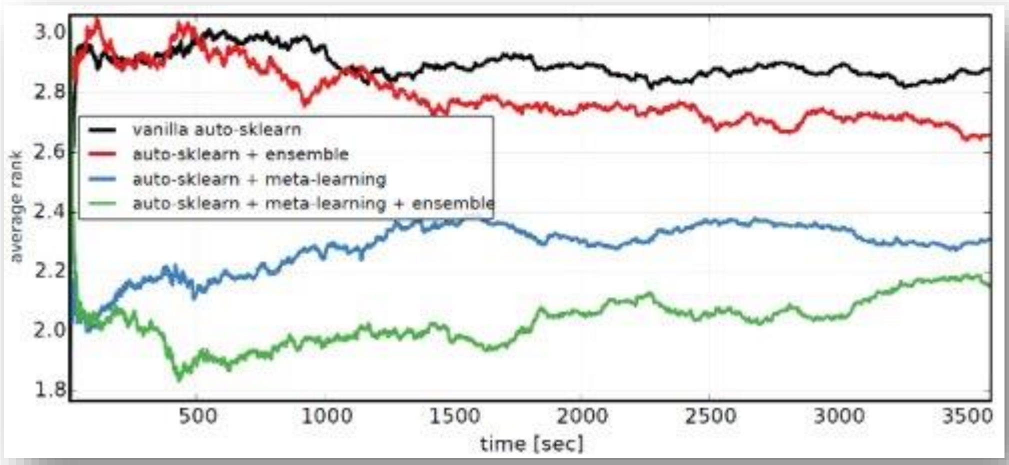
Auto-sklearn includes 15 ML algorithms, 14 preprocessing methods, and all their respective hyperparameters, yielding a total of 110 hyperparameters; the resulting space is illustrated in Figure 1, where only the hyperparameters in grey boxes are active.



Optimizing performance in Auto-sklearn's space of 110 hyperparameters can of course be slow, and to jumpstart this process we use meta-learning to start from good hyperparameter settings for previous similar datasets. Specifically, Auto-sklearn comes with a database of previous optimization runs on 140 diverse datasets from OpenML. For a new dataset, it first identifies the most similar datasets and starts from the saved best settings for those.

A second improvement was to automatically construct ensembles: instead of returning a single hyperparameter setting (as standard Bayesian optimization would), we automatically construct ensembles from the models trained during the Bayesian optimization. Specifically, we used *Ensemble Selection* to create small, powerful ensembles with increased predictive power and robustness.

In our NIPS 2015 paper we benchmarked Auto-sklearn with and without these new components on a total of 140 datasets (using leave-one- dataset-out validation) and found both of them to help substantially: Figure 2 shows that meta-learning improves performance directly from the start and ensembling helps more for longer optimization runs.



3.1.3 Auto-Net

Deep neural networks gained a lot of attraction in recent years due to their great performance for many large datasets. However, due to their sensitivity to many hyperparameters they are even harder to use than other algorithms, and there is an obvious need for automatically-tuned neural networks. Neither Auto-WEKA nor Auto-sklearn support deep neural networks, and our recent Auto-Net extension to Auto-sklearn closes this gap. So far it is restricted to simple feed forward neural networks, but in the future we aim to extend it to convolutional and recurrent neural networks.

Auto-Net contains layer-independent network hyperparameters (such as the learning rate or the number of layers) as well as hyperparameters which are set for each layer individually (such as dropout rate or weight initialization). We use a maximum of 6 layers, with a total of 63 hyperparameters. To make use of Auto-sklearn's preprocessing methods and ensembling, Auto-Net is simply implemented as a 16th ML algorithm for Auto-sklearn.

3.1.4 Applying Auto-sklearn to your own problem

Auto-sklearn is a drop-in replacement for a scikit-learn estimator. You can use the following four lines of python code to obtain a machine learning pipeline:

```
import autosklearn.classification

cls = autosklearn.classification.AutoSklearnClassifier()
cls.fit(X_train, y_train)
y_hat = cls.predict(X_test)
```

3.2 Advertising campaigns

3.2.1 Hashing of Binary Indicators

Hashing all URL's into a space of fixed dimensionality, we allow our automated model-builders to operate in a static, consistent feature space even though the semantic meaning of a feature may change over time.

3.3

4 Methods

4.1 Bayesian Optimization for Hyperparameter Tuning

https://arimo.com/data-science/2016/bayesian-optimization-hyperparameter-tuning/

With example and references therein!

Hyperparameter tuning may be one of the most tricky, yet interesting, topics in Machine Learning. For most Machine Learning practitioners, mastering the art of tuning hyperparameters requires not only a solid background in Machine Learning algorithms, but also extensive experience working with real-world datasets.

The straightforward way to tune hyperparameters is based on human expertise. Experienced Machine Learning practitioners know approximately how to chose good hyperparameters. For a new dataset, they will follow a trial and error process with various configurations of hyperparameters. This process of experimenting with hyperparameters is heuristic and different people with different experience might come up with different settings and the process is not easily reproducible.

There is also a real risk that a human won't achieve a near optimum setting of hyperparameters. Humans are not good at handling high dimensional data and can easily misinterpret or miss trends and relationships when trying to tune multiple hyperparameters. For instance, while tuning just two parameters, practitioners often fall back to tuning one parameter then tuning the second parameter. This may lead to concluding improvement in performance has plateaued while adjusting the second hyperparameter, while more improvement might be available by going back to changing the first hyperparameter. This difficulty requires an automatic and reproducible approach for hyperparameter tuning. The most popular approaches are outlined in this section.

4.1.1 Grid Search

In grid search, we try a set of configurations of hyperparameters and train the algorithm accordingly, choosing the hyperparameter configuration that gives the best performance. In practice, practitioners specify the bounds and steps between values of the hyperparameters, so that it forms a *grid* of configurations. Practitioners typically start with a limited grid with relatively large steps between parameter values, then extend or make the grid finer at the best configuration and continue searching on the new grid. This process is called manual grid search.

Grid search is a costly approach. Assuming we have *n*hyperparameters and each hyperparameter has two values, then the total number of configurations is 2^n. Therefore it is only feasible to do grid search on a small number of configurations. Fortunately, grid search is embarrassingly parallel, meaning it can be parallelized easily where each worker works on different parameter settings. This makes grid search a bit more feasible given enough computational power.

4.1.2 Random Search

In http://www.jmlr.org/papers/v13/bergstra12a.html *Random Search for Hyperparameter Optimization*, the authors show a surprising result: by navigating the grid of hyperparameters randomly, one can obtain similar performance to a full grid search. The authors show that if the close-to-optimal region of hyperparameters occupies at least 5% of the search space, then a random search with a certain number of trials (typically 40-60 trials) will be able to find that region with high probability.

Random search is surprisingly simple and effective, therefore it is considered by many practitioners as the *de facto* method for tuning hyperparameters. Like grid search, it is embarassingly parallel, yet the number of trials is much less, while the performance is comparable.

4.1.3 Automatic Hyperparameter Tuning

In Grid Search and Random Search, we try the configurations randomly and blindly. The next trial is independent to all the trials done before. In contrast, automatic hyperparameter tuning forms knowledge about the relation between the hyperparameter settings and model performance in order to make a smarter choice for the next parameter settings. Typically it will first collect the performance at several configurations, then make some inference and decide what configuration to try next. The purpose is to minimize the number of trials while finding a good optimum. This process, by nature, is sequential and not easily parallelizable.

It can be seen that hyperparameter tuning is an optimization problem, where we find the hyperparameter setting that maximizes the performance of the model on a validation set. However, the mapping from the hyperparameters to the performance on the validation set cannot be written into a formula, hence we don't know the derivatives of this function – it's known as a *blackbox function*. Because it's a blackbox function, optimization techniques like Newton method or Gradient Descent cannot be applied.

Most of the approaches for tuning hyperparameters fall into *Sequential Model-based Global Optimization* (SMBO). These approaches use a *surrogate* function to approximate the true blackbox function. Typically the inner loop of SMBO is the optimization of this surrogate, or some kind of transformation done on the surrogate. The configuration that maximizes this surrogate will be the one should be tried next. SMBO algorithms differ in the criteria by which it optimizes the surrogate, and in the way they model the surrogate given the observation history. Several SMBO approaches for hyperparameters have been recently proposed in literature:

4.1.3.1 Bayesian Optimization

Bayesian Optimization: uses a Gaussian Process to model the surrogate, and typically optimizes the Expected Improvement, which is the expected probability that new trials will improve upon the current best observation. Gaussian Process is a distribution over functions. A sample from a Gaussian process is an entire function. Training a Gaussian Process involves fitting this distribution to the given data, so that it generates functions that are close to the observed data. Using Gaussian process, one can compute the Expected Improvement of any point in the search space. The one gives the highest expected improvement will be tried next. Bayesian Optimization typically gives non-trivial, off-the-grid values for continuous hyperparameters (like the learning rate, regularization coefficient,…) and was shown to beat human performance on some good benchmark datasets. A well-known implementation of Bayesian Optimization is Spearmint.

4.1.3.2 Sequential Model-based Algorithm Configuration

Sequential Model-based Algorithm Configuration (SMAC): uses a random forest of regression trees to model the objective function, new points are sampled from the region considered optimal (high Expected Improvement) by the random forest. The implementation can be found here.

- Tree-structured Parzen Estimator (TPE): is an improved version of SMAC, where two separated models are used to model the posterior. A well-known implementation of TPE is hyperopt.