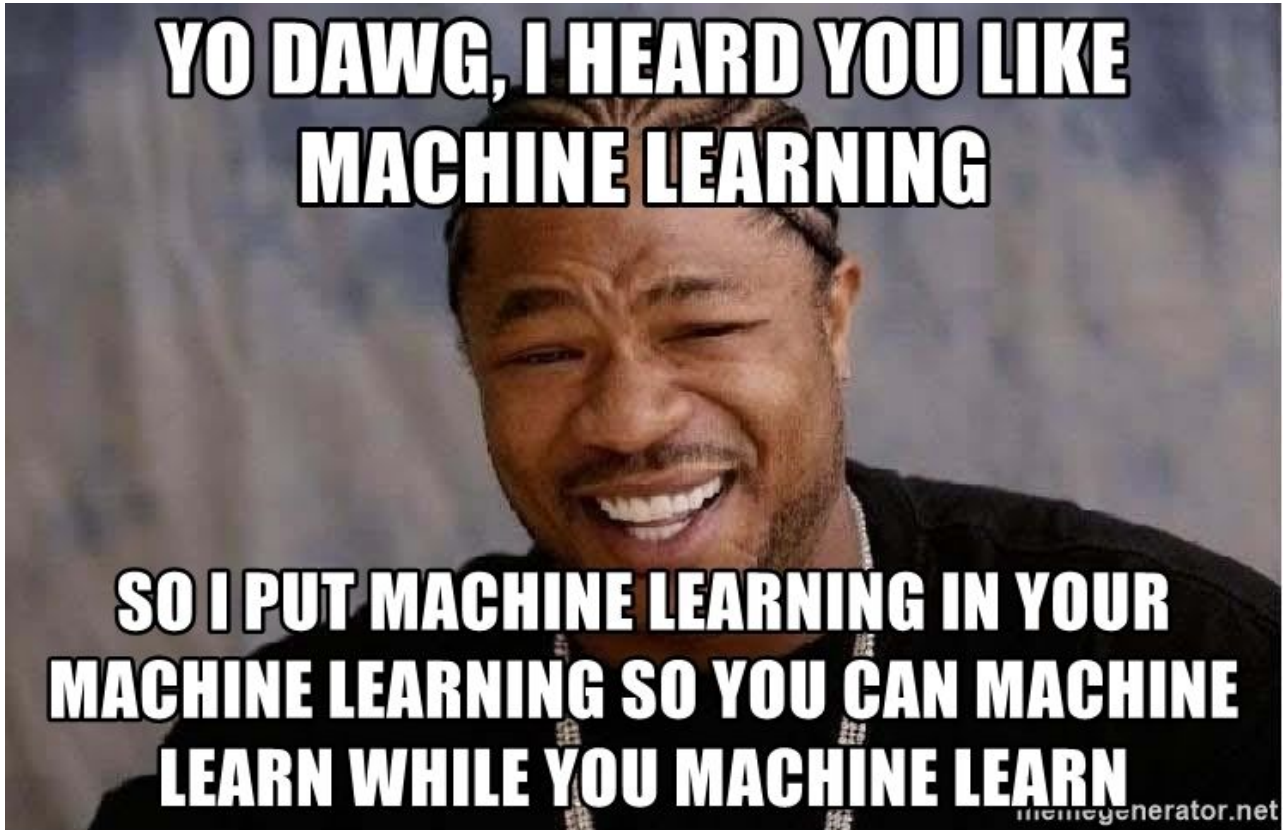
auto-sklearn

Automatic model selection and hyperparameter tuning

First, let's get this out of our system...



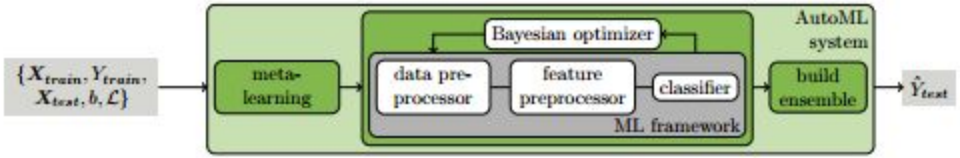
CASH problem

Combined Algorithm Selection and Hyperparameter optimization:

$$A^*, \boldsymbol{\lambda}_* \in \underset{A^{(j)} \in \mathcal{A}, \boldsymbol{\lambda} \in \Lambda^{(j)}}{\operatorname{argmin}} \frac{1}{K} \sum_{i=1}^{K} \mathcal{L}(A_{\boldsymbol{\lambda}}^{(j)}, D_{train}^{(i)}, D_{valid}^{(i)}).$$

A, a set of algorithms
lambda, their hyperparameters
L, a loss function
D, data
K, number of folds in cross-validation

Auto-sklearn = metalearning + bayesopt + ensemble building

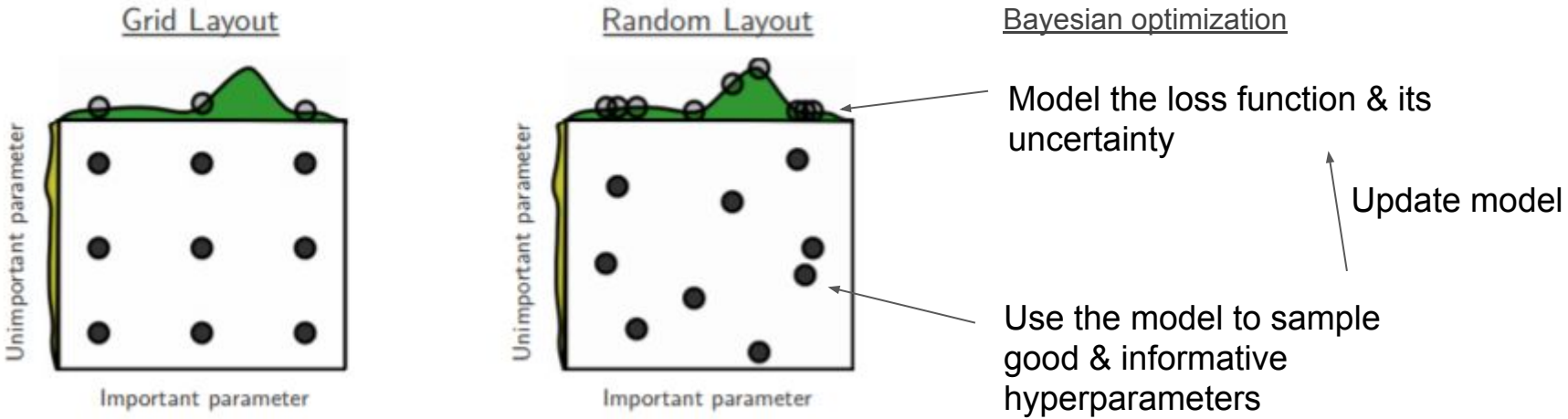


Scikit-learn provides:

15 classifiers
14 feature preprocessing methods
4 data preprocessing methods

110 hyperparameters

Bayesian Optimization



SMAC - not GP, but random forest.

Tree-based models have been shown to be more successful in high-dimensional, structured, and partly discrete problems

Meta-learning

Given a new dataset D, we compute its meta-features, rank all datasets by their L1 distance to D in meta-feature space and select the stored ML framework instantiations for the k = 25 nearest datasets for evaluation before starting Bayesian optimization with their results.

Some meta-features:

Number of samples, features and classes.
Number and percentage of missing features and samples.
Ratio of samples to features. Ratio of numeric to nominal features.
Statistics of features: mean, std, kurtosis, skewness.
Class entropy.

Automated ensemble construction of models evaluated during optimization

Automatic ensemble construction avoids committing itself to a single hyperparameter setting and is thus more robust (and less prone to overfitting) than using the point estimate that standard hyperparameter optimization yields.

Done by ensemble selection, a greedy procedure that starts from an empty ensemble and then iteratively adds the model that maximizes ensemble validation performance (with uniform weight, but allowing for repetitions)

Why would you use this?

- You don’t care what model you use, you just want decent predictions without trying every algorithm in the book
- You don’t enjoy solving CASH problems manually
- You don’t care about interpretability
- You have a cluster lying around doing nothing and you’re not in a hurry

The rest of today:

1. Make sure there's one linux laptop per table with auto-sklearn installed
2. Go through the API
3. Go through the manual
4. Read, understand and run the regression example (don't worry about the warnings)
5. Perform classification on Iris data (or your own dataset) and inspect the results
    a. sklearn.datasets.load_iris
    b. Remember to set a time limit! 3 minutes should do it.
    c. Describe the resulting model (e.g "The model is an ensemble of 4 pipelines. The first pipeline consists of preprocessing by PCA with 4 components, followed by (...)")
6. Perform classification on Iris data (or your own dataset) but with
    a. no preprocessing and only random forests
    b. no preprocessing , no ensembling and only random forests