Automated Machine Learning

OpenML Robot Assistant for algorithm selection and hyperparameter tuning

Student
J.M.A.P. van Hoof

Supervisor
Joaquin Vanschoren

Date
20-12-2018

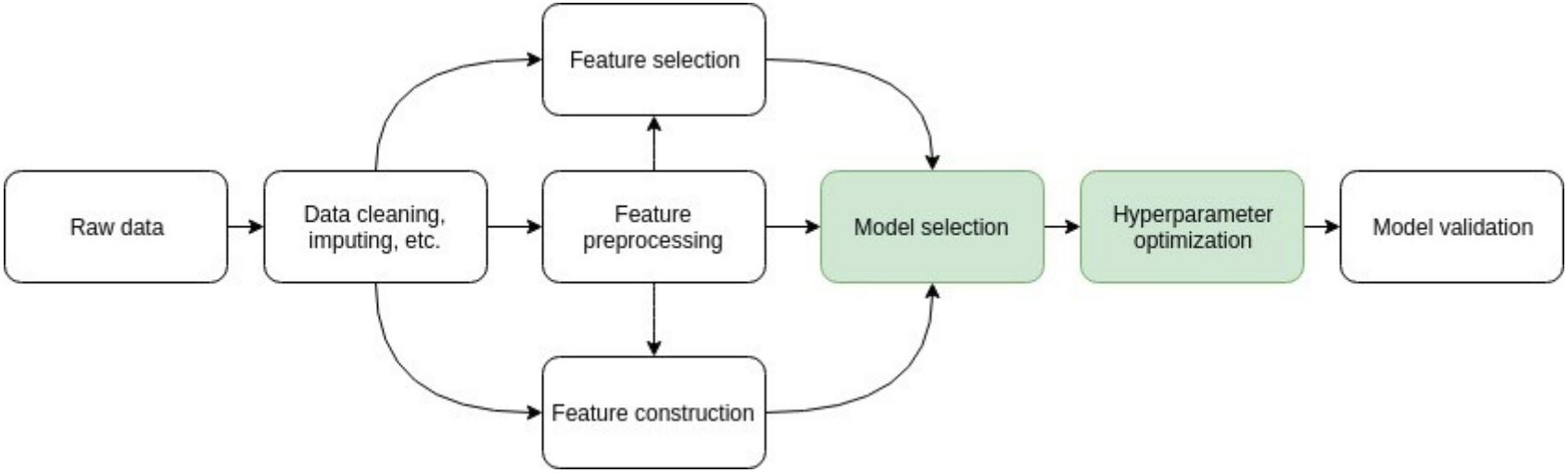
Committee
J. Vanschoren
V. Menkovski
M. Holenderski

Overview

- Introduction, AutoML techniques and current state of the art
- Improvements on hyperparameter optimization
- Improvements on meta-learning
- Exploring algorithm selection as a multi-armed bandit problem
- Conclusion

Automated machine learning

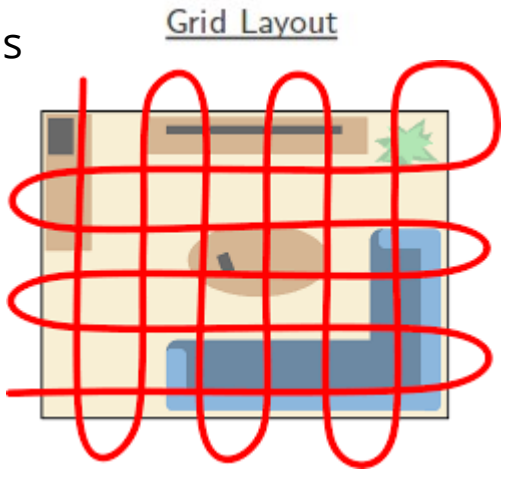
- Motivation
  - What **algorithm** for what **task**?
  - Algorithms rely on choosing the right parameters (**hyperparameters** or **configuration**)
  - Requires testing and manual work
  - Automating → faster creation & models that outperform hand-designed models
- Hyperparameter optimization, model selection and meta-learning

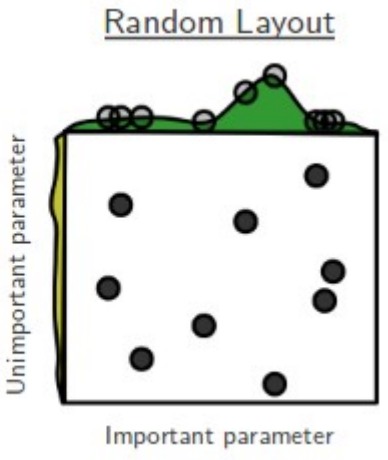


Grid search and randomized search

- **Grid search**
  - Try out all possible combinations → brute force
  - No trade-off between **computation time** and **accuracy** → only checked one corner
- **Randomized search**
  - Try out random combinations
  - Possible to draw values from continuous distributions
  - Can be terminated at any point in time → checked everything a little bit, just not as detailed

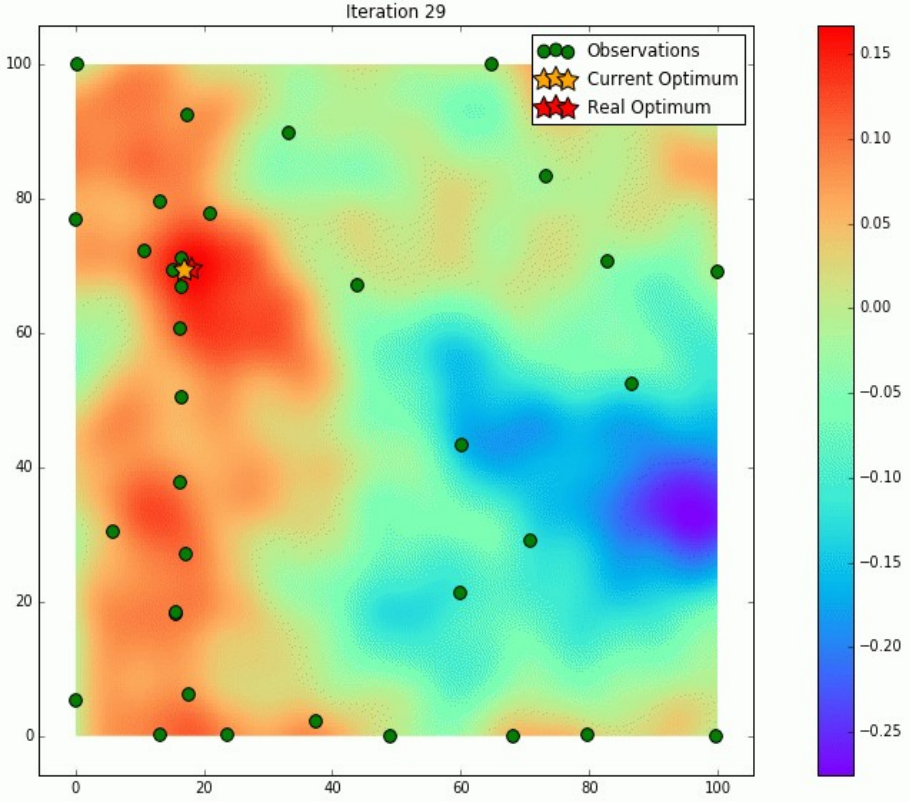
```
parameters = {
    'bootstrap': [True, False],
    'max_depth': [2, 4, 6, 8, 10, None],
    'max_features': ['log2', 'sqrt'],
    'min_samples_leaf': [1, 2, 4],
    'min_samples_split': [2, 5, 10],
    'n_estimators': [128, 256, 512, 1024]
}
```





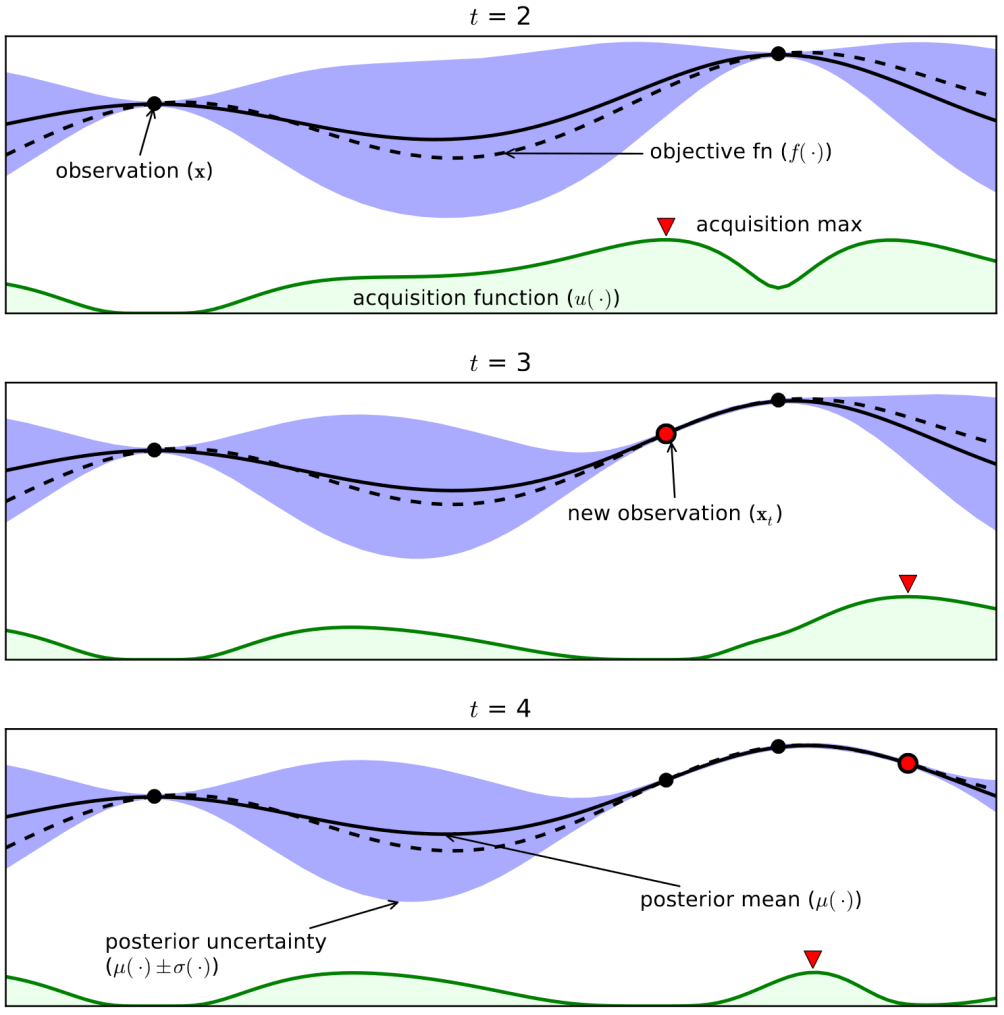
SMBO

- Better: look at previous observations and guess where to look next → actively search for the optimum
- Sequential model-based optimization (SMBO) a.k.a. Bayesian Optimization
- **Sequential:** runs trials one after another and alternates between
  - **evaluating configurations** and
  - deciding **which configuration to evaluate** next by reasoning on past observations
- **Model-based:** updating a probability model → **surrogate model**.



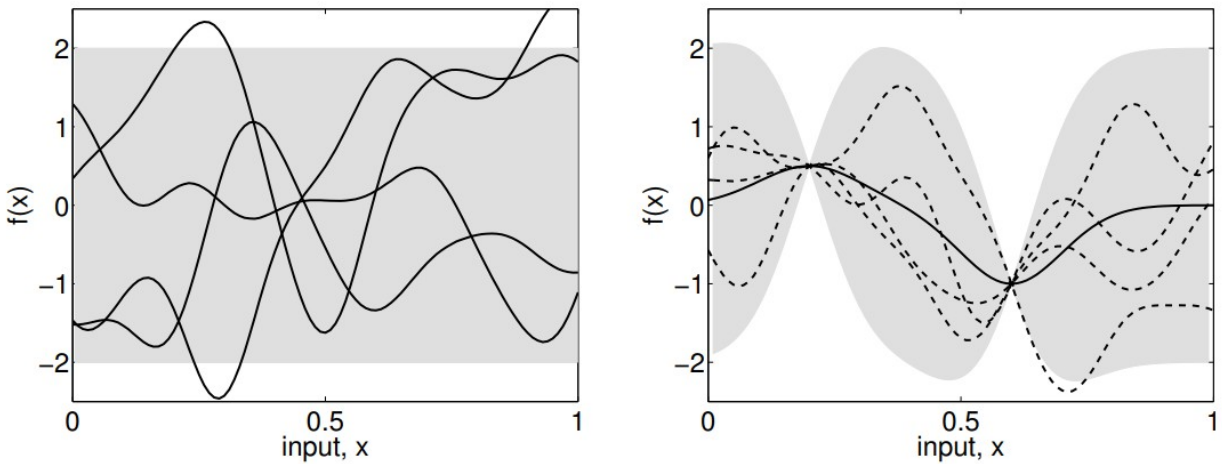
SMBO

- **Surrogate model**
  - A machine learning algorithm
  - Learn from observations so far
  - Gives a mean estimate → black line
  - Uncertainty (std) → purple area
  - Determine next point using an **acquisition function** → green line
- Expected improvement: **EI**
  - Estimate for the size of improvement over **incumbent** (current optimum)
- Expected improvement per (log) second: **EI/s**
  - Configurations that are believed to be **better,** but also **fast** to evaluate
  - Requires **performance estimation** (for EI) and **runtime estimation**



Surrogate models

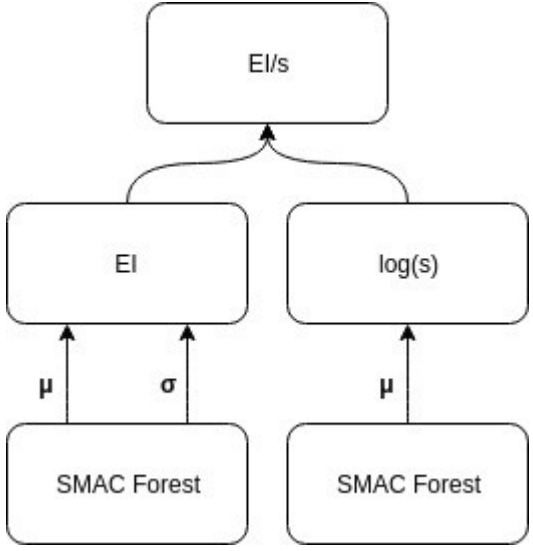
- Popular surrogate model for SMBO: **Gaussian Processes** (GP)
- Defines a **distribution** on a set of **functions**
  - Parameters for distribution optimized during learning process
- **Draw random functions** from this distribution
- **Throw away** functions that do not match **observed points**
- The functions that remain express the **uncertainty** of the GP.
- GP's training time grows **cubically** with the number of samples



(a), prior (b), posterior

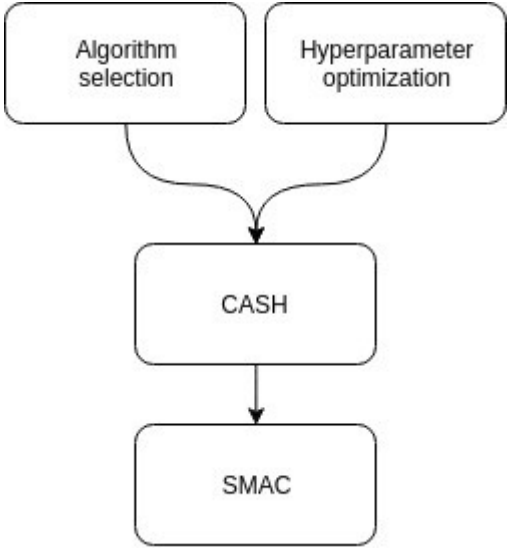
SMAC

- **SMAC** (Sequential Model-Based Algorithm Configuration)
- **Random Forests** instead of Gaussian Processes → scale better than GP
- SMBO requires model uncertainty **(σ)** → compute the **variance** of training data associated with **each leaf** → combine with **law of total variance**
- Uses Random Forests for estimating both **performance** and **runtime estimation**
- We will refer to this model as **SMAC Forest**



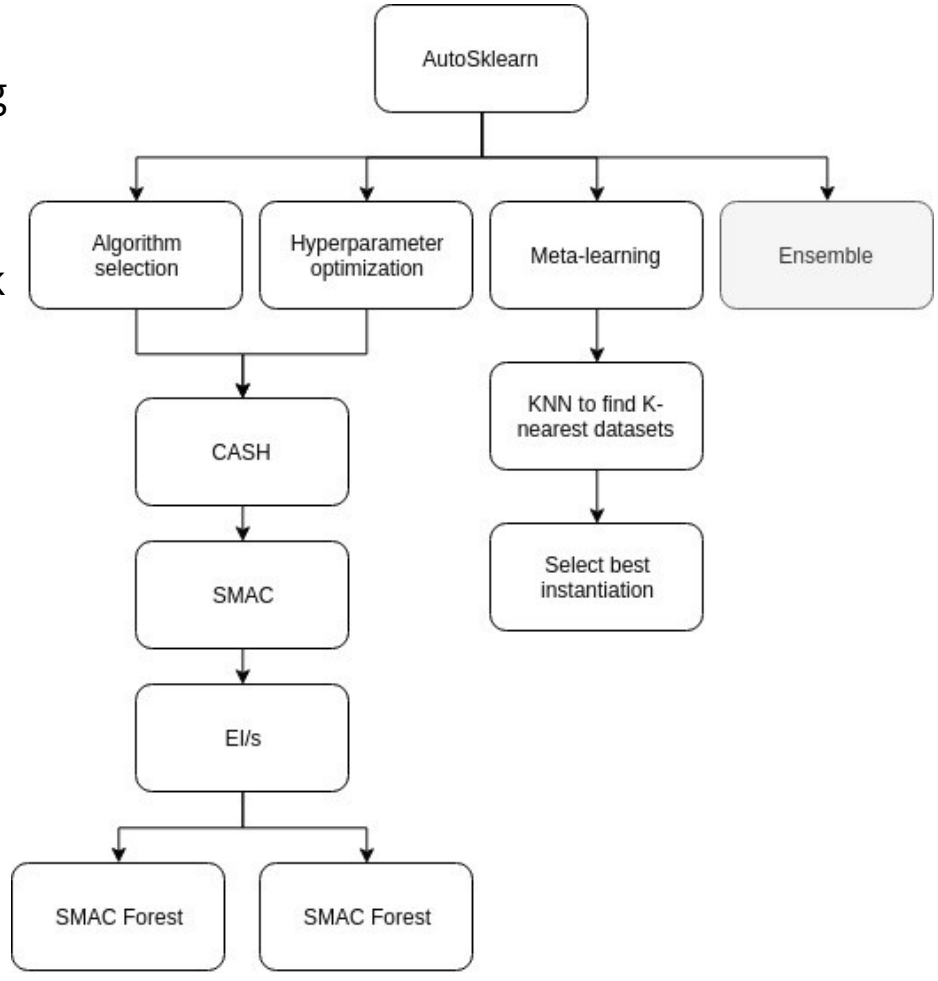
AutoSklearn

- **AutoSklearn:** state of the art machine learning tool
- **Based on SMAC**
- Combines hyperparameter optimization and algorithm selection into a single problem (CASH)
  - Concatenates all parameters into a long list
  - Flag determines whether algorithm is used



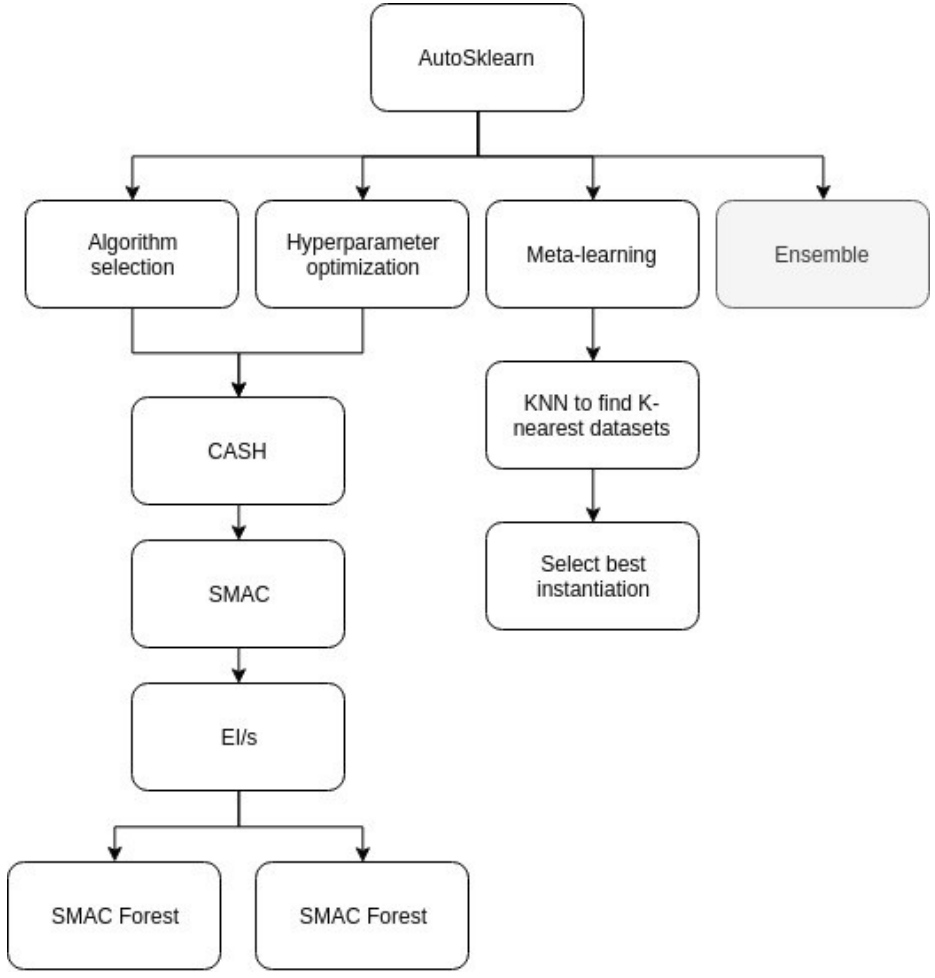
Meta-learning and AutoSklearn

- **AutoSklearn** adds 2 components: **meta-learning** and **ensemble-building** → we focus on meta-learning
- **Meta-learning**
  - Normally: Hyperparameter optimization needs to start from **zero knowledge** for every new task
    - Random initialization
  - Learn from performance on other tasks
- **Warm-starting:** Select configurations that worked well on similar datasets
- AutoSklearn selects **25** most similar datasets (based on **meta-features** of these datasets)
- Then uses the best **machine learning instantiation** (algorithm and configuration) of each dataset



Baseline

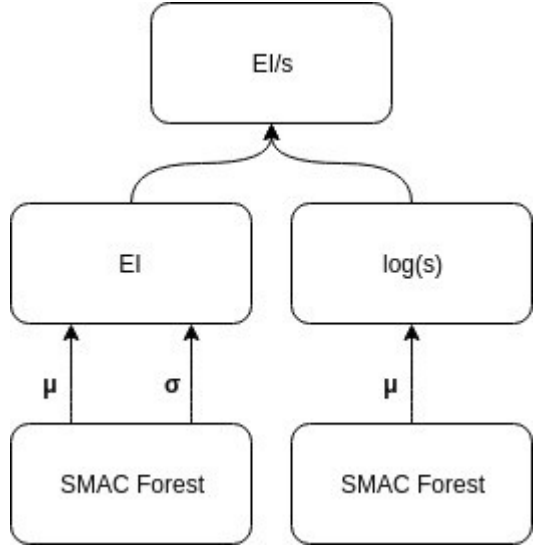
- We use AutoSklearn as baseline, and propose:
  - Improvements on **hyperparameter optimization** and **runtime estimation**
  - Training a **meta-learner** (on OpenML data) and improve warm-starting
  - Explore **algorithm selection** as a separate problem



Hyperparameter optimization

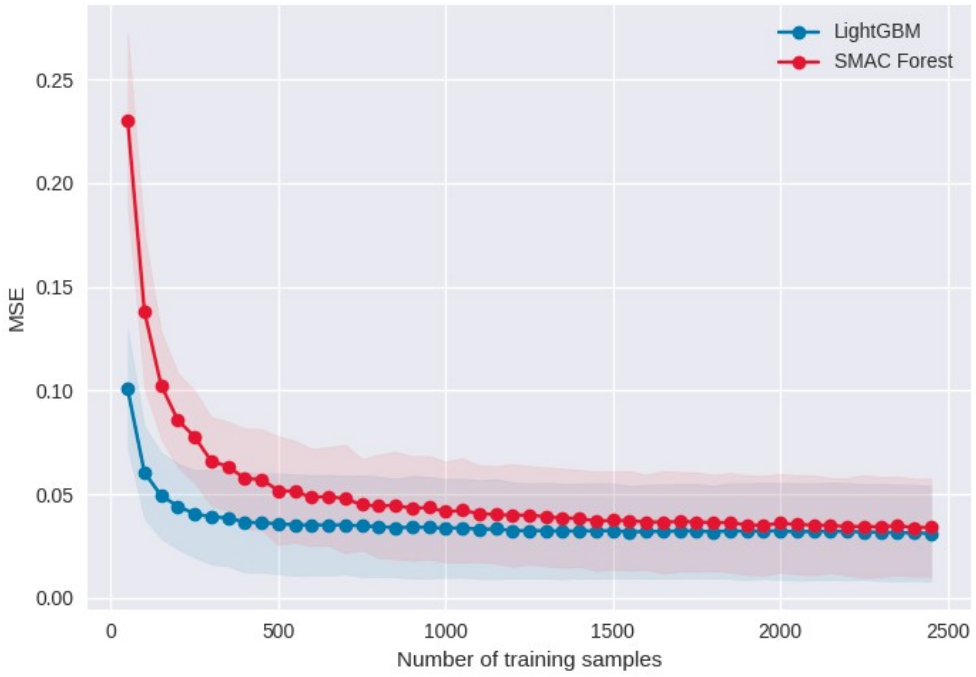
Hyperparameter optimization

- Better method for runtime estimation
- Better method for performance estimation
- Both methods based on LightGBM (gradient boosting) vs SMAC Forest (random forest)
- Combined into one method



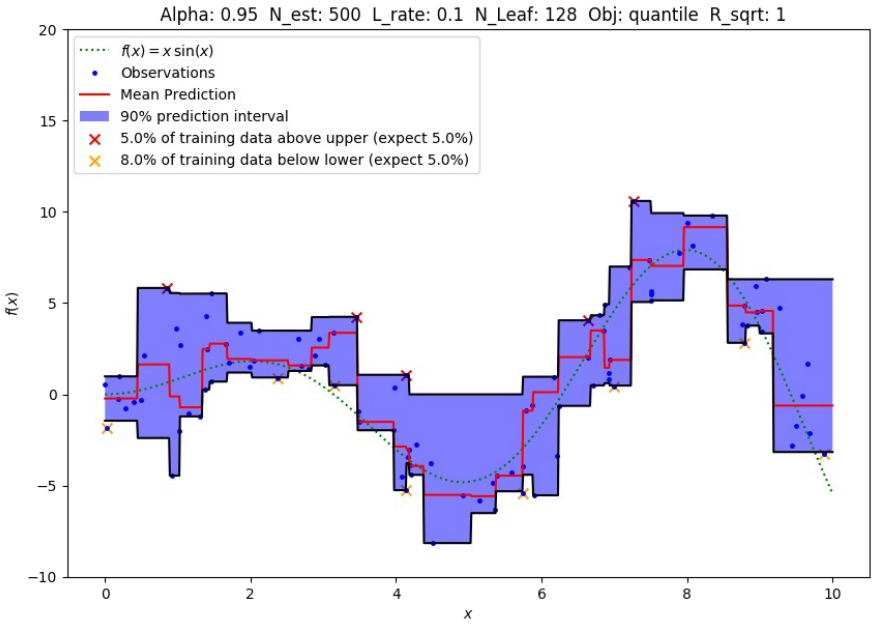
Runtime estimation

- Model for estimating the s in EI/s
- Baseline uses **SMAC Forest**
- Instead, we use **LightGBM**, a **gradient boosting** technique developed by Microsoft.
  - Ensemble of **100 trees** (vs 10)
  - **Shallow trees** of a maximum of 4 leaves
  - Optimized to run faster
- **Learning curve** plots MSE (y-axis) against number of samples (x-axis)
- Needs less data to perform better than SMAC's forest → headstart



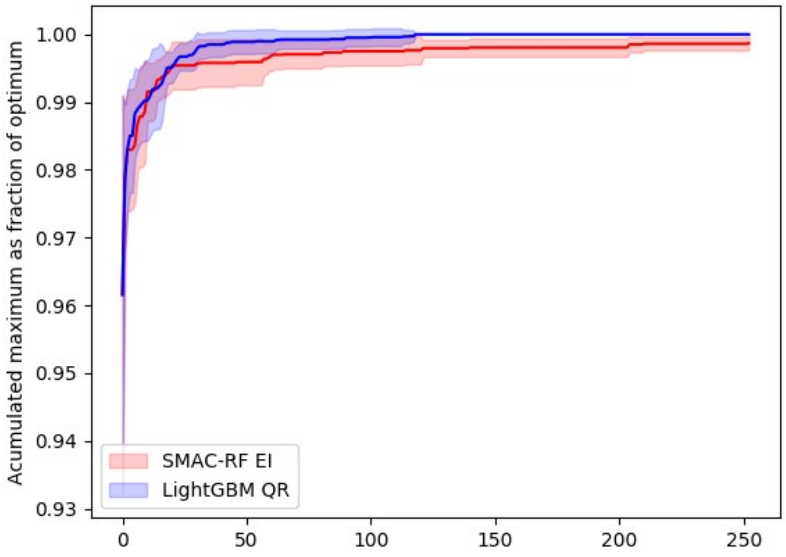
Performance estimation

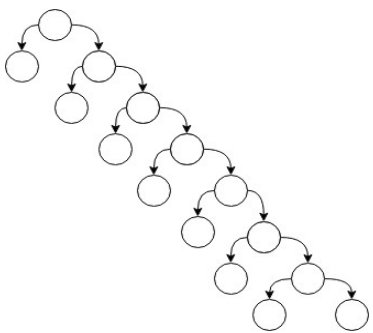
- Baseline: **SMAC Forest** with **expected improvement**
- We propose another method using **LightGBM** again.
- **Problem:** each tree is built on the residuals of the earlier trees → computing variance no longer works
- **Quantile regression:** estimate quantile (e.g. median) instead of mean
- **Normally:** need separate model for each quantile → upperbound, lowerbound and median → possible to estimate uncertainty for **EI**
- **Instead:** Only use the **90% upper bound**, and we no longer need **EI**
  - We select next configuration where upper bound is highest

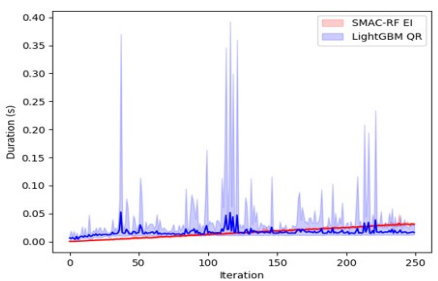


Performance estimation

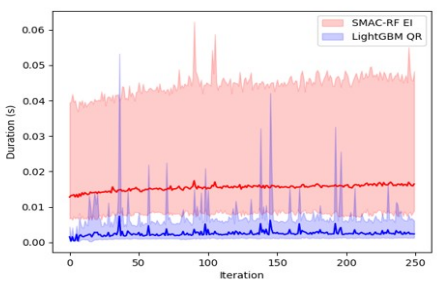
- We use **LightGBM** with 90th-quantile quantile regression, 100 trees and a maximum of 8 leaves
- Outperforms the basline SMAC Forest
- LightGBM **predicts faster**, but training time seems to be "**spiky**", but **scales better** with larger data.
- Spiky → gradient boosting is sequential, LightGBM parallelizes node-building, we limit number of leaves instead of maximum depth



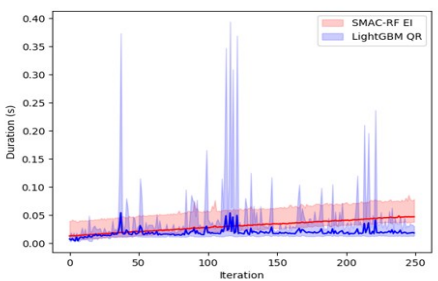




(a) Training time (over observed data)



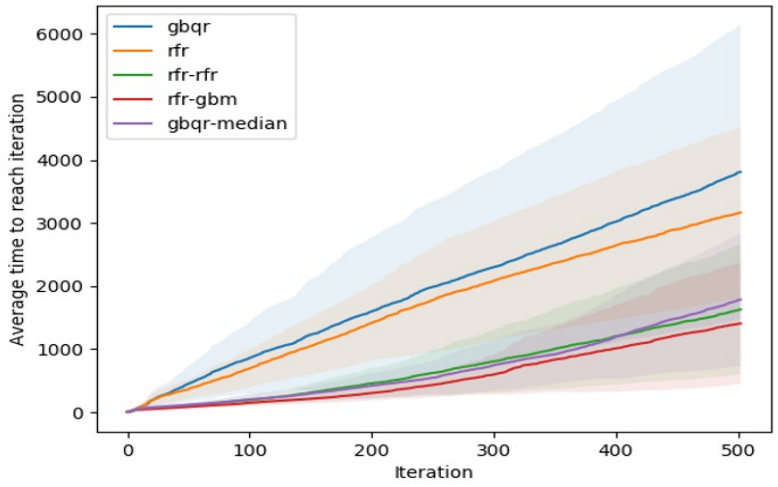
(b) Prediction time (over all data)



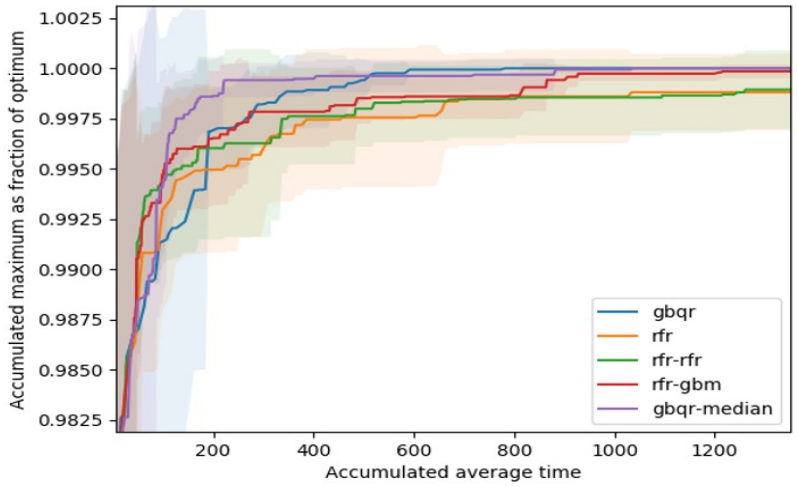
(c) Total time

Combined estimation

- Tested different combinations of: *<performance estimator> – <runtime estimator>* (and <performance estimator> without runtime estimator)
- **RFR:** SMAC's Forest, also called Random Forest Run
- **GBQR**: Our Gradient Boosting with Quantile Regression solution
- **GBQR-Median**: GBQR (predictions normalized by removing median from upper bound estimates) in combination with GBM runtime estimator



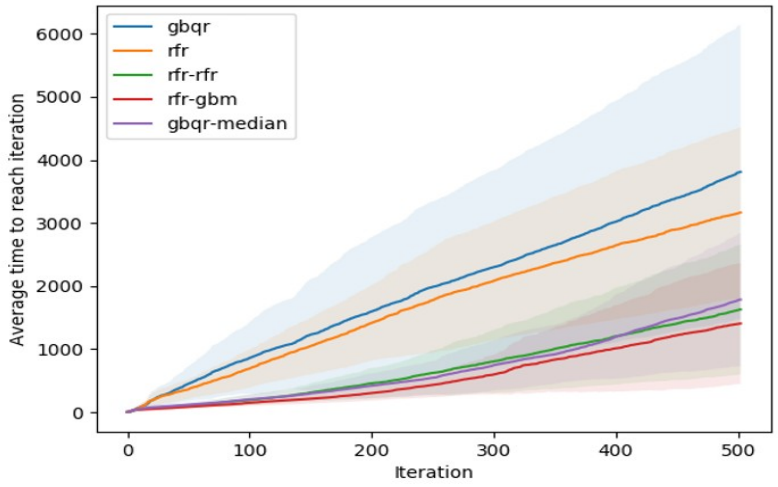
(a) Evaluation time



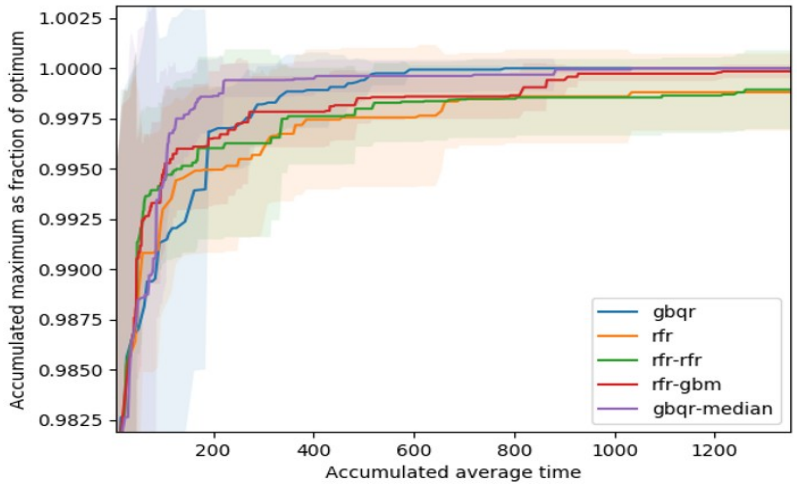
(b) Optimum over time

Combined estimation

- SMAC Forest (RFR) in combination with GBM works better than RFR with RFR
- GBQR outperforms all other methods in terms of performance, even GBQR with time estimation
- After more investigation → methods that include time estimation perform worse in situations where runtime and performance are too correlated



(a) Evaluation time



(b) Optimum over time

Meta-learning

Meta-learning

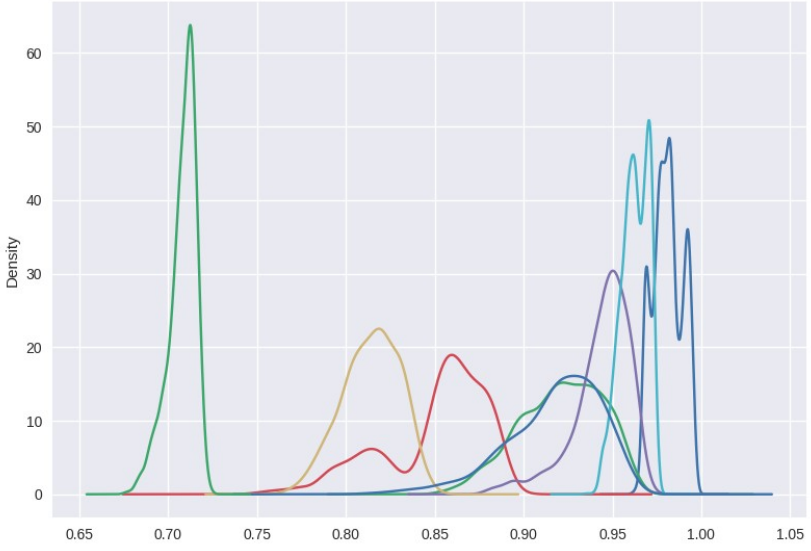
- Goal: using historical results on other datasets, predict results on new task
- Combine **hyperparameters** of each run and the **metafeatures** the run was executed on
  - Metafeatures
    - Simple features: #features, #samples, % missing, etc.
    - Landmarkers: simple machine learning algorithms
  - Hyperparameters
    - Translate configuration to numerical representation

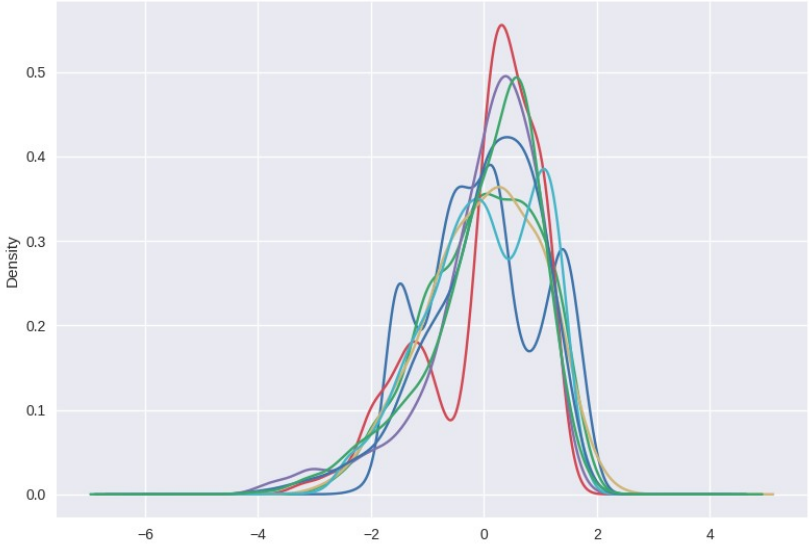
parameter	unique values
n_estimators	10, 100, 500
criterion	“gini”, “entropy”
max_features	0.05, 0.10, “log2”, “sqrt”, None
max_depth	9, 10, 11, 12, None
min_samples_split	2, 3, 5, 8
min_samples_leaf	1, 2, 7, 9
bootstrap	True, False, 0, 1
@preprocessor	OneHotEncoder(), PCA(), None

parameter	values
n_estimator	10, 100, 100, 500, 100
criterion_gini	1, 1, 1, 0, 1
criterion_entropy	0, 0, 0, 1, 0
max_features_num	0.05, 0.10, 0.0576, 0.0913, 1
max_features_nom_null	1, 1, 0, 0, 0
max_features_nom_None	0, 0, 0, 0, 1
max_features_nom_sqrt	0, 0, 0, 1, 0
max_features_nom_log2	0, 0, 1, 0, 0
max_detph_num	9, 10, 11, 12, 20
max_depth_nom_null	1, 1, 1, 1, 0
max_depth_nom_None	0, 0, 0, 0, 1
min_samples_split	2, 3, 8, 5, 8
min_samples_leaf	1, 7, 2, 7, 9
bootstrap	1, 0, 0, 1, 1
@preprocessor_OneHotEncoder	1, 0, 0, 0, 0
@preprocessor_PCA	0, 1, 1, 1, 0
@preprocessor_None	0, 0, 0, 0, 1

Meta-learner

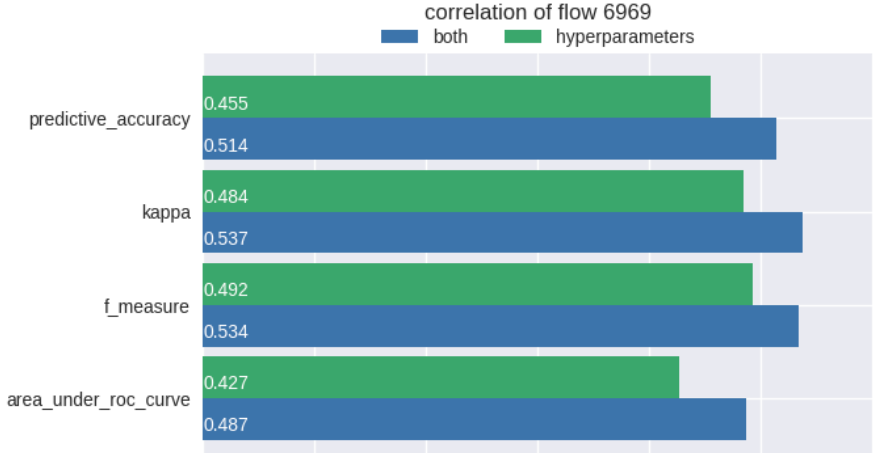
- Distinguish **high-performing** configurations from **low-performing** configurations
  - Rather than just giving a precise estimate (harder to predict)
- **Standardize** performance per dataset (see below) → relative performance of each configuration
- **Kendall rank correlation coefficient** to determine how well meta-learner “ranks” these configurations → Measures correlation of meta-estimator estimates with actual actual ranking

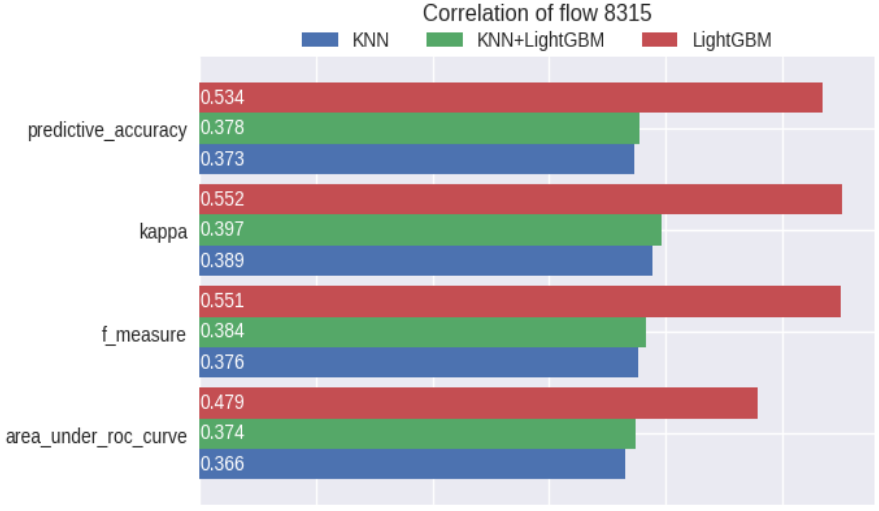




Meta-learner

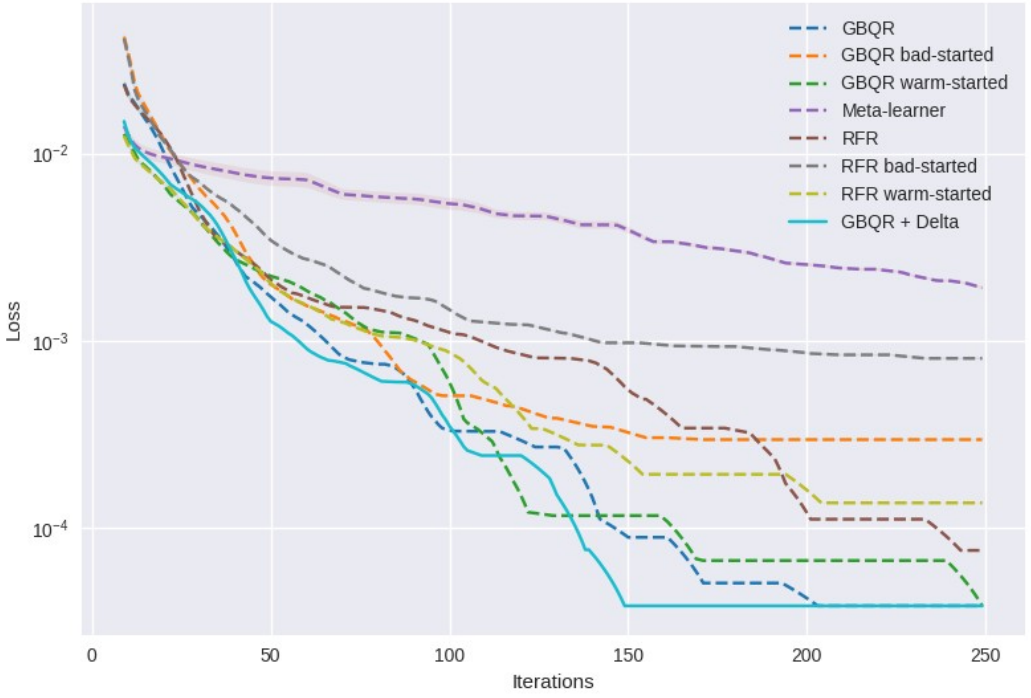
- We optimized LightGBM estimator to “rank” configurations
- Measured average **correlation** between **estimates** and **actual ranking**
- Shows **ability to learn interaction** between hyperparameters and metafeatures
- Better than selecting K similar datasets and training a model on hyperparameters from these datasets





Delta warm-starting

- How to use this meta-learner to improve optimization?
- **Meta-learner:** using meta-learner only (descending from best-predicted downwards)
- **RFR** and **GBQR:** randomly initialized
- **Warm-started:** initialized with top 3 predicted by meta-learner
  - Does not work well: **RFR** better than **RFR warm-started**, **GBQR** better than **GBQR warm-started**
  - Predictions too close → less information (top 3 prediction only varies unimportant hyperparameter)
- **Delta:** use warm-starting, but spread our selection (e.g. best, 10th place and 20th place)



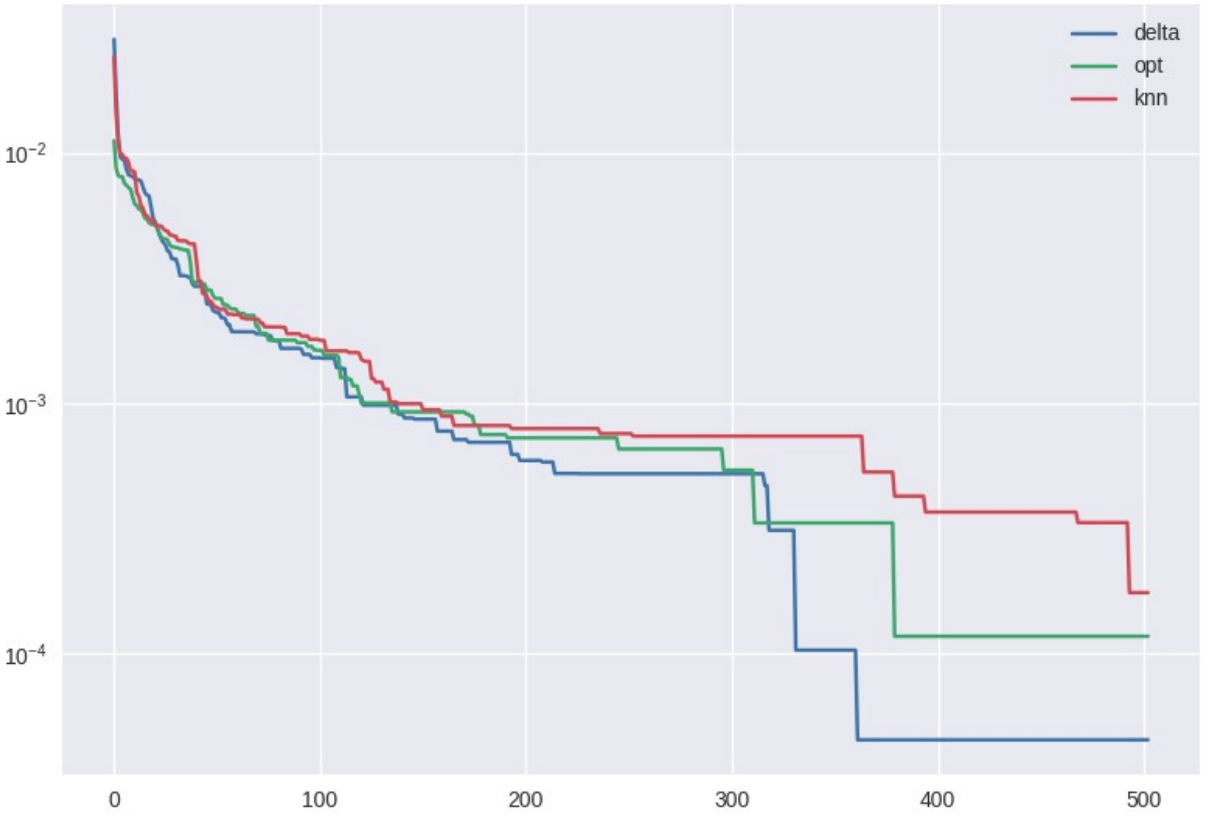
$$\texttt{score}(p_i, \Delta) = (1 - p_i)^{\Delta} + p_i$$

Delta warm-starting

- How does this relate AutoSklearn?
  - AutoSklearn: selects 25 most similar datasets using KNN on metafatures → best **instantiations** to warm-start **CASH**
  - We use 3 **configurations** selected by the delta-method to warm-start the optimization of a **single algorithm**
- Comparison
  - KNN to select 3 most similar datasets → use top configuration of each dataset
  - Delta-method to select 3 configurations

Ranking

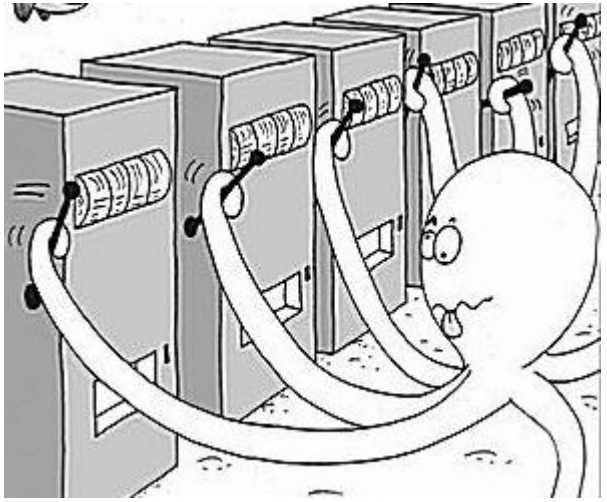
- **KNN:** select 3 most similar datasets with KNN based on metafeatures (very similar to AutoSklearn)
- **OPT:** theoretical best 3 datasets to select
  - Based on 4050 configurations executed across all datasets
  - Comparing the ranking of these configurations on every pair of datasets
  - The 3 datasets with the highest correlation with the current dataset are selected
- **Delta:** using my delta-method

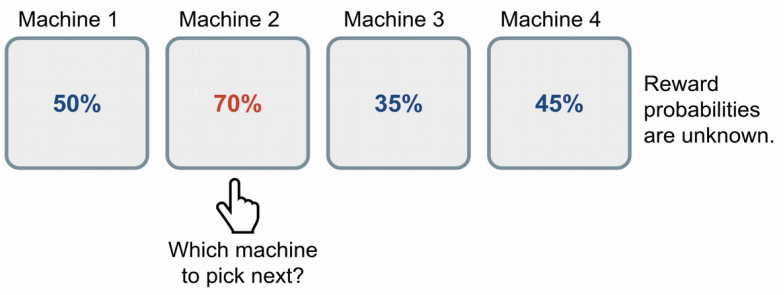


Algorithm selection

Multi-armed bandits

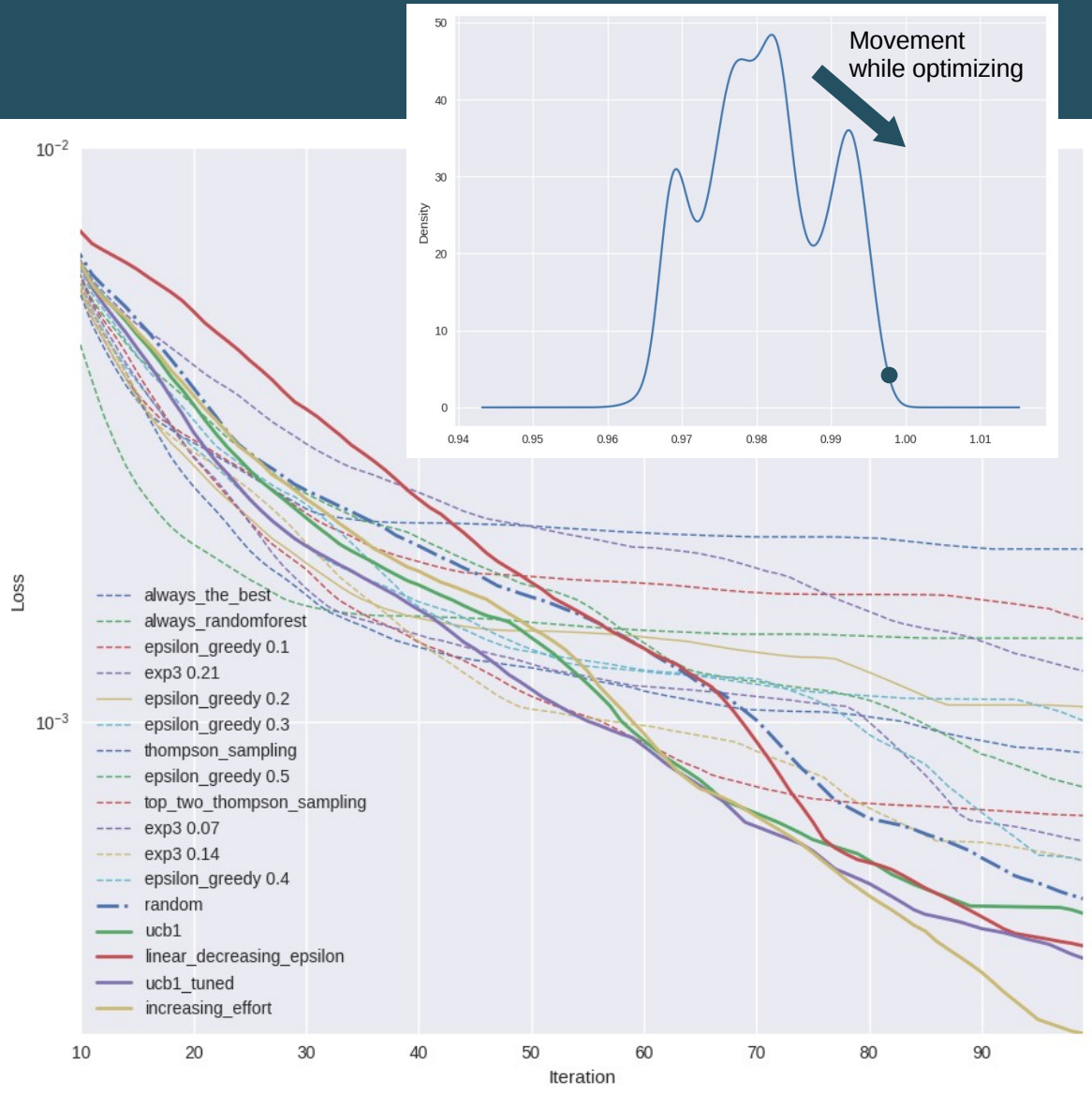
- Instead of CASH, explore algorithm selection as separate problem: **multi-armed bandit problem**.
- **One-armed bandit**: a slot machine
- Each slot has a **specific probability** with which it returns a **reward**
- Probabilities unknown → learn while maximize reward
- Gambler has to choose between
  - **Exploration:** trying out levers to estimate their probabilities more precisely
  - **Exploitation:** pulling the lever that gave him the highest rate of rewards
- In our case:
  - Lever = algorithm to optimize
  - Reward = observed performance





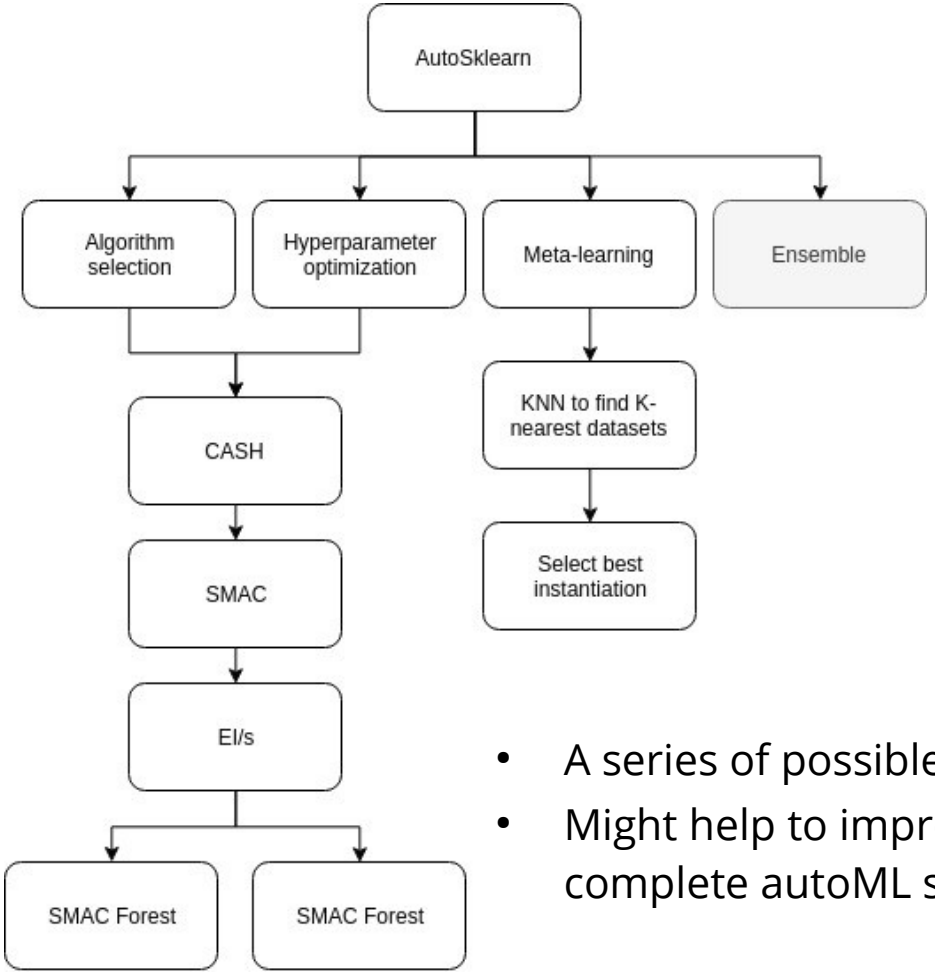
Algorithm selection

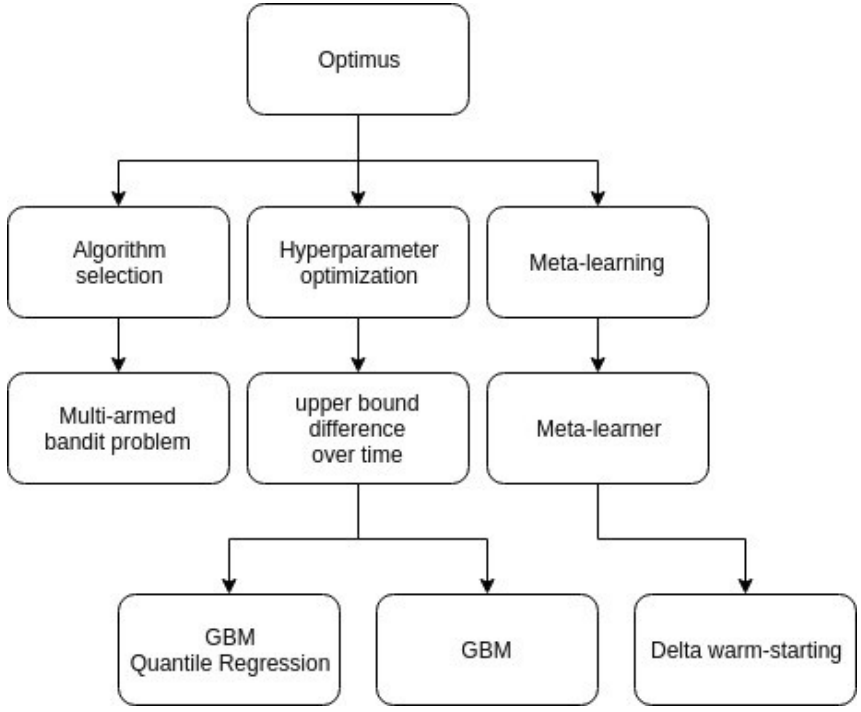
- Most algorithms not suitable
  - Find algorithm with highest top performance (not average performance)
  - Optimum located in sparse area of probability density function
  - Rewards do not reflect directly that one algorithm might become the best
- Upper confidence bound methods seem to work (e.g. UCB1-tuned)
- **Optimism in the face of uncertainty**: we believe an action is as good as possible given the available evidence



Conclusion

Conclusion





- A series of possible improvements on the current state of the art
- Might help to improve upon the state of the art or be bundled into a complete autoML solution

Questions

Defence

Overview

- Proof-of-concept demo: https://github.com/Yatoom/Lightning/blob/master/demo.ipynb
- Presentation video: https://www.youtube.com/watch?v=qFTHFYuRd8c
- Quantile regression 1
- Quantile regression 2
- Ranking 1
- Ranking 2
- CASH vs MAB
- AutoSklearn
- Meta-learning
- Paralellizing Bayesian optimization
- Miscellaneous

Quantile regression

- Why does quantile regression work better?
  - Uncertainty downwards is not included
  - Quantile more robust to outliers than mean estimate
  - Gradient boosting works better than Random Forests or Gaussian Processes
  - Training directly on the quantile
- Using three models to calculate median and standard deviation
  - 16^{th}, 50^{th} and 84^{th} percentile → uncertainty approximated by (high – low) / 2
  - Can not calculate percentile if no values above percentile → empty tree
  - Happens when observed performances above the percentile are all the same
  - Usually lower estimate can be calculated, while upper estimate needs more information
  - EI calculation still works, but now uses only 16^{th} percentile
  - Leads to more values at the top that are the same

Quantiles

- np.quantile([0.8, 0.9, 1.0], 0.9) →0.98
- np.quantile([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 9], 0.9) → 9
- np.quantile([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 9], 1) → 9

Ranking

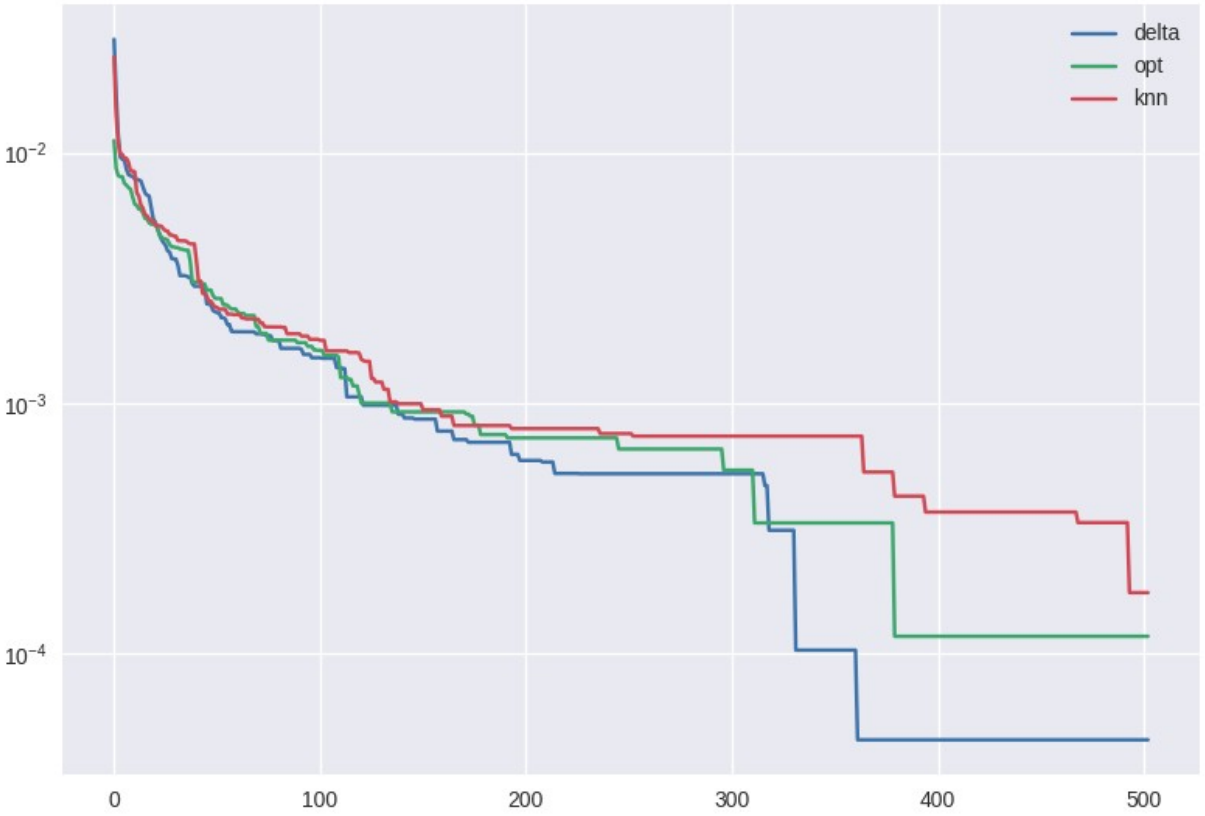
- Experiment on full grid with 31 datasets using LightGBM target model
- **Complete grid**: every configuration is executed across all datasets
  - Able to compare configurations one-on-one across datasets
  - $d_{i,OPT}$ → actual most similar dataset for *i* based on ranking of configurations
  - $d_{i,KNN}$ → dataset selected for *i* by KNN based on metafeatures
  - $d_{i,WORST}$ → most dissimilar dataset for *i*
  - Average correlation: -0.5759 for $d_{i,WORST}$, 0.7253 for $d_{i,OPT}$
- **OpenML:** different amount of configurations tested per dataset, might be partially overlapping
- **Incomplete grid:** randomly remove 50% of configurations $d_{i,OPT,\ 50\%}$

Input	Method	Tau
D	LightGBM	**0.6003**
$d_{i,OPT}$	LightGBM KNN (any K)	0.6939 **0.7253**
$d_{i,KNN}$	LightGBM KNN (any K)	0.4255 **0.4400**
$d_{i,OPT,\ 50\%}$	1NN 5NN LightGBM	0.4493 0.4575 **0.6937**

- KNN's performance drops rapidly for incomplete grid
- LightGBM much more robust, learns better
- Better than selecting most similar dataset with KNN
- Works with incomplete grid, missing values, missing features, etc.

Ranking

- **KNN:** select 3 most similar datasets with KNN based on metafeatures (very similar to AutoSklearn)
- **OPT:** theoretical best 3 datasets to select
  - Based on 4050 configurations executed across all datasets
  - Comparing the ranking of these configurations on every pair of datasets
  - The 3 datasets with the highest correlation with the current dataset are selected
- **Delta:** using my delta-method
- Experiment executed with a LightGBM target model on 31 datasets



CASH vs MAB

- Concerns with CASH
  - Hyperspace of one estimator larger → space is favoured because expected improvement does not decrease as fast
- Advantages of MAB
  - One specialized meta-learner and surrogate model per machine learning pipeline
  - Treating AS and HPO as two separate problems allows to evaluate performance separately
  - Observations divided over multiple surrogate models → faster
- Reward improvements over optimum
  - Per algorithm: initially high-performing algorithm gets rewarded less
  - Globally: algorithm that is just a little bit later doesn't get any reward

AutoSklearn

- Won first international challenge in 2015-2016 and in 2017-2018 (https://www.automl.org/blog-2nd-automl-challenge/)
- Best classification results in comparison with TPOT, H2O and auto_ml (https://arxiv.org/pdf/1808.06492.pdf) → 57 classification problems and 30 regression problems
- Hyperband has been shown to work better with small number of iterations (https://people.eecs.berkeley.edu/~kjamieson/hyperband.html) but slows down after more time (https://www.automl.org/blog_bohb/)
- PoSH-AutoSklearn: portfolio replaces warm-starting and includes SuccessiveHalving
  - Start with a small budget, evaluate all models
  - Double budget, halve amount of models, using only the best half
  - Budget = subset size of all data
- BOHB: random selection of configurations replaced by model-based search trained on the configurations that were evaluated so far

Meta-learning

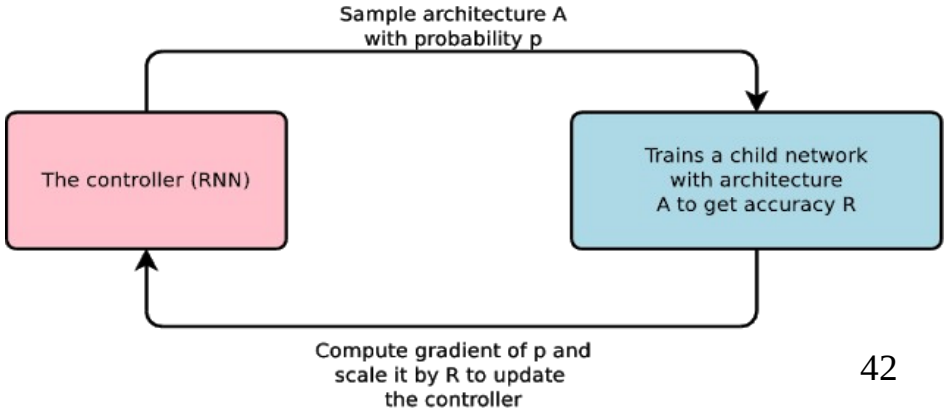
- How much features do we actually need?
  - Random Forest (flow 6969) predictive accuracy over 50+ datasets → correlation
  - Using hyperparameters (8/16 leaves) → 0.455
  - Using statistical features (and 8 leaves) → 0.509 ("ClassEntropy", "NumberOfFeatures", "NumberOfInstances", "MajorityClassPercentage", "MinorityClassPercentage")
  - Using landmarkers and statistical features (and 16 leaves) → 0.514
- What about AutoSklearn's portfolio building?
  - Might be able to gather metafeatures faster with more specific warm-starting triple with delta warm-starting
- Why ranking?
  - We want to distinguish between high-performing and low-performing configurations
- Why Kendall's tau?
  - Generally preferred over Spearman's rho (more robust, more reliable and interpretable confidence intervals, etc.)
  - Pearson: assumes normally distributed

Parallelizing Bayesian Optimization

- Machine learning algorithms itself can be parallelized
- Cross-fold evaluation can be parallelized
- Combined with randomized search for information gathering
- Fantasize outcomes → constant lier
  - CL-MAX: The lie in this Constant Liar strategy is fixed to the maximum of the current observations.
  - CL-MIN: The dummy response is fixed to the minimum of the current observations

Deep learning

- Possible to parametrize deep-learning models
- Warm-starting with well-known networks
- Controller that proposes architectures
- DARTS



Meta-learner

- Distinguish **high-performing** configurations from **low-performing** configurations
  - Rather than just giving a precise estimate (harder to predict)
- **Standardize** performance per dataset (by removing the mean and scaling to unit variance) → relative performance of each configuration
- **Kendall rank correlation coefficient** to determine how well meta-learner "ranks" these configurations
  - Measures correlation of meta-estimator estimates with actual actual ranking
  - Value between -1 and 1

$$\tau = \frac{(\text{number of concordant pairs}) - (\text{number of discordant pairs})}{n(n-1)/2}.$$

Candidate	Interviewer 1	Interviewer 2	Concordant	Discordant
A	1	1	11	
B	2	2		
C	3	4		
D	4	3		
E	5	6		
F	6	5		
G	7	8		
H	8	7		
I	9	10		
J	10	9		
K	11	12		
L	12	11		

Candidate	Interviewer 1	Interviewer 2	Concordant
A	1	1	11
B	2	2	10
C	3	4	8
D	4	[illegible]	
E	5	6	
F	6	5	
G	7	8	
H	8	7	
I	9	10	
J	10	9	
K	11	12	
L	12	11	