Evaluation of Automated Machine Learning

Hongyu Xin
Rutgers University
hx152@rutgers.edu

Yanping Wang
Rutgers University
yw882@rutgers.edu

Junyi Li
Rutgers University
jl2707@rutgers.edu

Abstract

Development of end-to-end automated machine learning workflows is important in order to deploy them in a robust way in real life applications. We present a benchmark of current open source AutoML solutions using 15 large diverse QSAR data sets that are taken from Merck's drug discovery effort. We use the framework to conduct a thorough comparison of 3 AutoML systems included autosklearn, TPOT, H2O's AutoML and 1 robust methods random forest and analyze the results. We find that H2O performs the best across 15 QSAR regression datasets.

1 Introduction

Designing and tuning machine learning systems is a labor and time intensive task which requires extensive expertise. In the pharmaceutical industry it is common to generate many QSAR models from training sets containing a large number of molecules and a large number of descriptors. The best QSAR methods are those that can generate the most accurate predictions but that are not overly expensive computationally.

In 2012, Merck sponsored a Kaggle competition (`https://www.kaggle.com/competitions/MerckActivity/data`) to examine how well the state of art of machine learning methods can perform in QSAR problems. 15 datasets of various sizes (2000 - 50 000 molecules) using a common descriptor type was provided by our project supervisors from Merck research group. Each data set was required divided into a 80% training set and 20%test set. Deep neural net (DNN)[1] and Extreme gradient boosting machine (GBM)[2] was shared by supervisors from Merck research contains background of 15 qsar datasets, sammary table of data dimentions and good method for quantitative structure- Acativity Relationship.

Automated machine learning (AutoML) is a way that automates the ML pipeline in real-world applications. AutoML covers the total pipeline from the crude dataset to the deployable AI model. It allows developers to build ML models with high scale, efficiency, and productivity all while sustaining model quality and apply machine learning without much expertise. In recent years, many approaches have been developed to examine how well the state of art of machine learning methods can perform in QSAR problems. This paper focuses on design of an AutoML workflow and its subsequent evaluation on QSAR data sets and leverage the experience to develop a guidance for an effective use of AutoML framework in real life.

Our code and result are avalibale on the github `https://github.com/arthurxin/workflow_of_automl_framework`

1.1 Baseline Model: Random Forest

RF(Random Forest) is an ensemble recursive partitioning method where each recursive partitioning "tree" is generated from a bootstrapped sample of compounds and a random subset of descriptors is used at each branching of each node. RF can handle regression problems or classification problems. RF naturally handles correlation between descriptors, and does not need a separate descriptor selection procedure to obtain good performance. Importantly, while there are a handful of adjustable parameters (e.g., number of trees, fraction of descriptors used at each branching, node size, etc.), the quality of predictions is generally insensitive to changes in these parameters. Therefore, the same set of parameters can be effectively used in various problems.

In particular, RF has been very popular since it was introduced as a QSAR method by Svetnik[3]. Due to its high prediction accuracy, ease of use, and robustness to adjustable parameters, RF has been something of a "gold standard" to which other QSAR methods are compared.

1.2 Selected AutoML Frameworks

1.2.1 Auto Sklearn

Auto-sklearn is one of the most popular AutoML packages in Python. Figure 1 shows the Auto-sklearns process for pipeline optimization. It uses the Sci-kit Learn machine learning library for transforming data and machine learning algorithms. It uses a Bayesian Optimization search procedure to discover a top-performing model pipeline for a given dataset in an effective way. One of the key advantages of using Auto-Sklearn is that it not only discovers the data preparation and the best model for the provided data but also, learns from models that performed well on similar datasets and can automatically create an ensemble of top-performing models discovered as part of the optimization process.

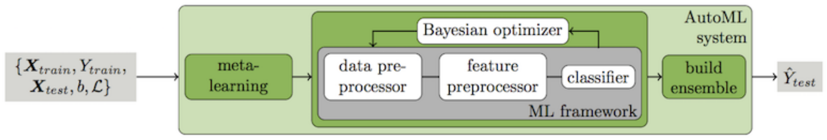


Figure 1: Auto-sklearns process for pipeline optimization

1.2.2 H2O AutoMl

AutoML is a function in H2O that automates the process of building a large number of models, with the goal of finding the ”best” model without any prior knowledge or effort by the Data Scientist. The current version of AutoML trains and cross-validates a default Random Forest, an Extremely-Randomized Forest, a random grid of Gradient Boosting Machines (GBMs), a random grid of Deep Neural Nets, a fixed grid of GLMs, and then trains two Stacked Ensemble models at the end. One ensemble contains all the models (optimized for model performance), and the second ensemble contains just the best performing model from each algorithm class/family (optimized for production use).

1.2.3 TPOT

TPOT stands for the Tree-based Pipeline Optimization Tool. It optimizes machine learning pipelines using genetic programming. It mechanizes the most monotonous piece of AI by cleverly investigating a large number of potential pipelines to locate the best one for your information. It automates portions of the machine learning process detailed in Figure 2.

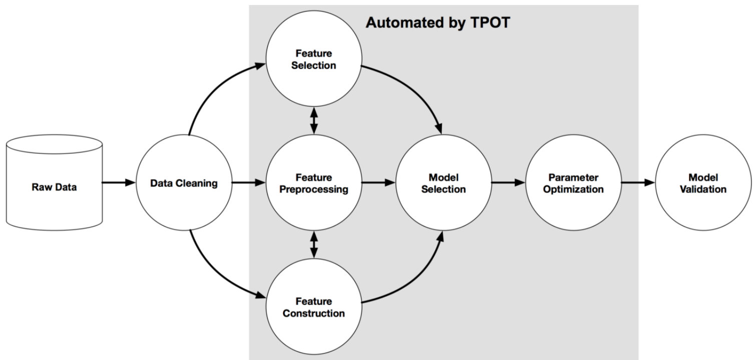


Figure 2: Pieces of the machine learning process automated by TPOT

When TPOT is done looking (or you become weary of pausing), it gives you the Python code for the best pipeline it found so you can fiddle with the pipeline from that point. TPOT is built on top of sci-kit learn, so all of the code it generates should look familiar. Figure 3 shows the process.

1.2.4 Comparison of the AutoML tools

AutoML methods differ in their optimization method, the pipelines they generate, the library of algorithms they select from, whether they use meta-learning to learn from runs on prior datasets, or whether they perform post-processing. Table 1 shows a simplified comparison of the AutoML tools compared in this paper.

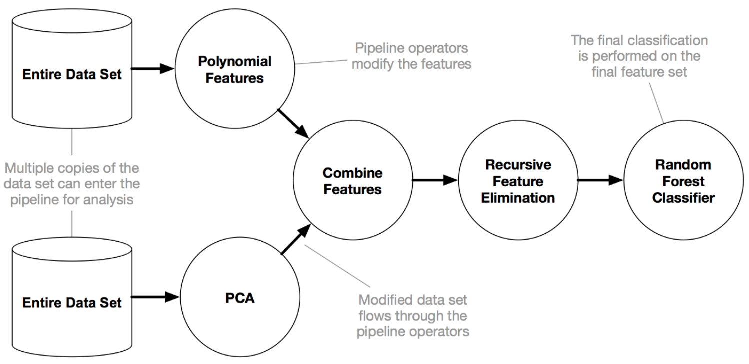

Entire Data Set
Polynomial Features
Pipeline operators modify the features
The final classification is performed on the final feature set
Combine Features
Recursive Feature Elimination
Random Forest Classifier
Multiple copies of the data set can enter the pipeline for analysis
Entire Data Set
PCA
Modified data set flows through the pipeline operators

Figure 3: Pieces of the machine learning process automated by TPOT

Tool	Back-end	Optimization	Meta-learning	Post-processing
Auto-sklearn	scikit-learn	Bayesian	warm-start	ensemble selection
H2O AutoML	H2O	Random Search	-	stacked ensembles
TPOT	scikit-learn	Genetic Programming	-	-

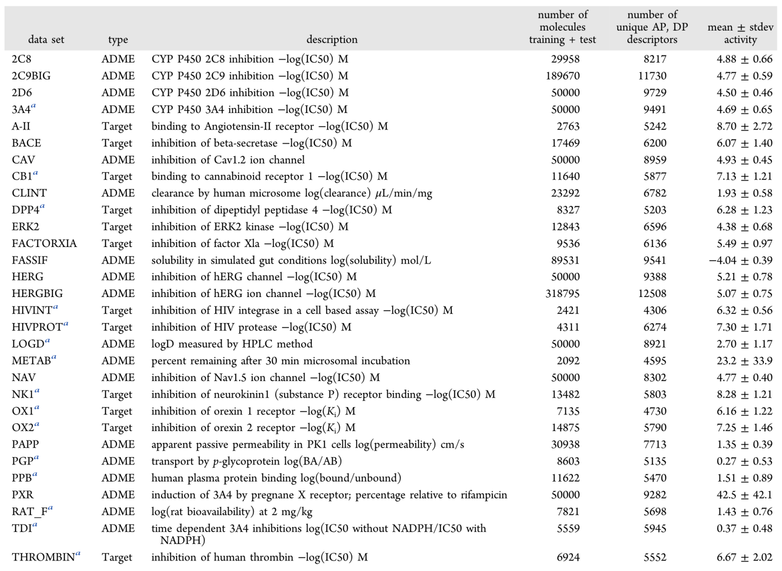
Table 1: Comparison of the AutoML tools

2 Benchmark Design

Each benchmark task consists of a dataset, a metric to optimize, and specific resources to use. We briefly explain our choice for each.

2.1 Datasets

Figure 4 download from the work of Robert P et al[2] shows the data sets used in this study. These are in-house Merck data sets including on-target and ADME (absorption, distribution, metabolism, and excretion) activities. The 15 labeled "Kaggle Data Sets" are the same data sets we used for Merck Challeng. In order to compare the predictive ability of QSAR methods, each of these data sets was random split into training set(80%) and test set(20%). These 15 QSAR datasets are avaliable in Kaggle competition (`https://www.kaggle.com/competitions/MerckActivity/data`) . These two paper([2])([1]) contain background of these 15 QSAR datasets and summary table of data dimensions.



data set	type	description	number of molecules training + test	number of unique AP, DP descriptors	mean ± stdev activity
2C8	ADME	CYP P450 2C8 inhibition −log(IC50) M	29958	8217	4.88 ± 0.66
2C9BIG	ADME	CYP P450 2C9 inhibition −log(IC50) M	189670	11730	4.77 ± 0.59
2D6	ADME	CYP P450 2D6 inhibition −log(IC50) M	50000	9729	4.50 ± 0.46
3A4[a]	ADME	CYP P450 3A4 inhibition −log(IC50) M	50000	9491	4.69 ± 0.65
A-II	Target	binding to Angiotensin-II receptor −log(IC50) M	2763	5242	8.70 ± 2.72
BACE	Target	inhibition of beta-secretase −log(IC50) M	17469	6200	6.07 ± 1.40
CAV	ADME	inhibition of Cav1.2 ion channel	50000	8959	4.93 ± 0.45
CB1[a]	Target	binding to cannabinoid receptor 1 −log(IC50) M	11640	5877	7.13 ± 1.21
CLINT	ADME	clearance by human microsome log(clearance) μL/min/mg	23292	6782	1.93 ± 0.58
DPP4[a]	Target	inhibition of dipeptidyl peptidase 4 −log(IC50) M	8327	5203	6.28 ± 1.23
ERK2	Target	inhibition of ERK2 kinase −log(IC50) M	12843	6596	4.38 ± 0.68
FACTORXIA	Target	inhibition of factor Xla −log(IC50) M	9536	6136	5.49 ± 0.97
FASSIF	ADME	solubility in simulated gut conditions log(solubility) mol/L	89531	9541	−4.04 ± 0.39
HERG	ADME	inhibition of hERG channel −log(IC50) M	50000	9388	5.21 ± 0.78
HERGBIG	ADME	inhibition of hERG ion channel −log(IC50) M	318795	12508	5.07 ± 0.75
HIVINT[a]	Target	inhibition of HIV integrase in a cell based assay −log(IC50) M	2421	4306	6.32 ± 0.56
HIVPROT[a]	Target	inhibition of HIV protease −log(IC50) M	4311	6274	7.30 ± 1.71
LOGD[a]	ADME	logD measured by HPLC method	50000	8921	2.70 ± 1.17
METAB[a]	ADME	percent remaining after 30 min microsomal incubation	2092	4595	23.2 ± 33.9
NAV	ADME	inhibition of Nav1.5 ion channel −log(IC50) M	50000	8302	4.77 ± 0.40
NK1[a]	Target	inhibition of neurokinin1 (substance P) receptor binding −log(IC50) M	13482	5803	8.28 ± 1.21
OX1[a]	Target	inhibition of orexin 1 receptor −log(K_i) M	7135	4730	6.16 ± 1.22
OX2[a]	Target	inhibition of orexin 2 receptor −log(K_i) M	14875	5790	7.25 ± 1.46
PAPP	ADME	apparent passive permeability in PK1 cells log(permeability) cm/s	30938	7713	1.35 ± 0.39
PGP[a]	ADME	transport by *p*-glycoprotein log(BA/AB)	8603	5135	0.27 ± 0.53
PPB[a]	ADME	human plasma protein binding log(bound/unbound)	11622	5470	1.51 ± 0.89
PXR	ADME	induction of 3A4 by pregnane X receptor; percentage relative to rifampicin	50000	9282	42.5 ± 42.1
RAT_F[a]	ADME	log(rat bioavailability) at 2 mg/kg	7821	5698	1.43 ± 0.76
TDI[a]	ADME	time dependent 3A4 inhibitions log(IC50 without NADPH/IC50 with NADPH)	5559	5945	0.37 ± 0.48
THROMBIN[a]	Target	inhibition of human thrombin −log(IC50) M	6924	5552	6.67 ± 2.02

Figure 4: summary table of data

2.2 Hardware choice and resource specifications

To accurately compare the selected AutoML frameworks, we need to control the hardware condition. We choose Rutgers Amarel cluster to deploy our program. We used 4 CPU cores, and 100 GB RAM memory. The CPUs are Intel(R) Xeon(R) CPU E5-2670 0 @ 2.60GHz, provide stable computing ability. The 100 GB memory is adequate in most situation for the QSAR datasets above. If not, the framework is not good at memory control.

2.3 AUTOML Framework Configuration

We selected 3 tools based on popularity, ease of use and variety of underlying techniques and Random Forest which is good for QSAR dataset.

- Baseline Method
  - Random Forest (The version of RF we are using is a default hyperparameter value. Since most users will use them in this way.)
- AutoML Framework
  - Auto-Sklearn (n_jobs:4, memory_limit=20480, metric:r^2, per_run_time_limit:600 , time_left_for_for_this_task = timelimit*)
  - H2O (exlude_algos: Deep learning, verbosity: Null, max_time_secs:)
  - TPOT (n_jobs:4, verbosity:3, generation:10, population_size:20, max_time_mins:)

2.4 Experiment design and Performance metric

- Experiment on different datasets (In this experiment, program run the selected framework on the 15 datasets. For every dataset, the dataset is splited into training set(80%) and test set(20%). The framework fit the model on the train set with time limit 3600 seconds. Then calculate the R^2 score on the train and test set as output. The higher the test R^2, the better performance the framework. The train R^2 score is used to compare the over-fitting level of the framework.)
- Experiment on different time limit (In this experiment, program run the selected framework on the 4,13,14 datasets. The three datasets are the hardest three datasets in the 15. On hard datasets, we can observe the change of R^2 easier. Also, the dataset will be splited. Then, we set time limit list [600, 1200, 1800, 2400, 3000, 3600, 7200, 10800] seconds. The framework fit the model with different time limit and calculate the R^2 score on the train/test set as output. Plot the relationship between train/test R^2 and the time limit. We can compare how fast and stable the model is fitted.)
- Experiment on different memory usage (We recorded the peak value of memory usage in the two experiments above. A good framework should run with stable and limited memory usage.)

3 Results

Our 15 datasets are all regression problems, and we evaluated the frameworks from the following perspectives: R^2, time consumption, memory consumption, stability, and flexibility.

3.1 Analysis on different datasets

On different datasets, we all take the random forest model with defualt parameters as a base model. We can see from the Figure 5 and Figure 4 on most cases the R^2 scores of three frameworks are higher than our base model except for datasets 4, 13 and 14. For datasets 13 and 14, considering that the underlying model also has poor performance, so this may be due to the dataset itself. It is worth mentioning that on many datasets some frameworks also provide a random forest model, but it also provides better parameters compared to default parameters. And within the three frameworks, from the R^2 perspective alone, H2O wins in most cases

3.2 Analysis on different time limit

We do the experiment on time limit [600, 1200, 1800, 2400, 3000, 3600, 7200, 10800] seconds on [dataset4,dataset13,dataset14] on the three frameworks. The output are showed on the Figure7. The R^2 increases unstably on the Tpot framework. The H2O gets the fastest R^2 increasing rate and best final R^2. All the three framework would get the best final R^2 with the time limit 120 minutes. We would recommend using 3600 seconds as time limit.

3.3 Analysis on different memory consumption

We have some empirical formulas for memory usage. Let's assume storage of data in memory is X GB, and we have Y CPU cores for multi-thread. The Autosklearn would take X * Y + 40 GB RAM at most. And H2O would take 2 * X + 40 GB RAM at most. As for Tpot, due to the properties of Genetic Programming,

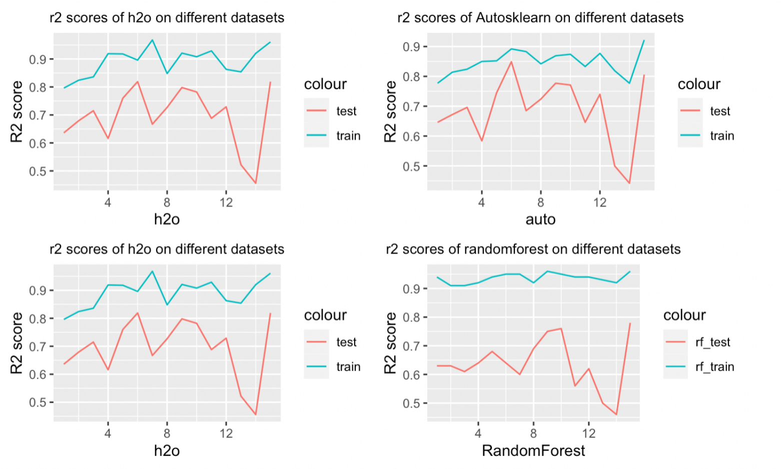


Figure 5: R^2 scores of three frameworks on different datasets

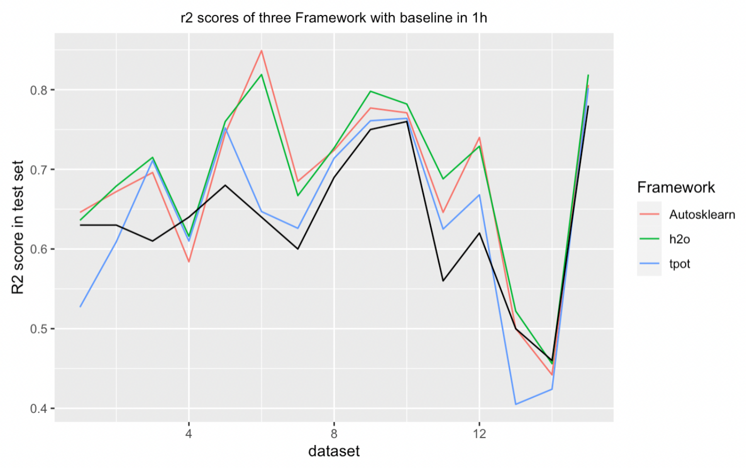


Figure 6: R^2 scores in test set with time limit of 1h

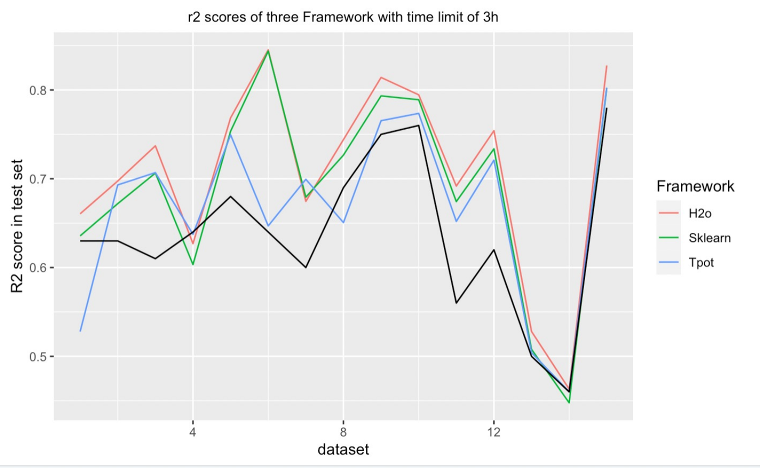


Figure 7: R^2 scores in test set with time limit of 3h

the number of models in a generation is a random number. Thus we can hardly estimate the memory usage of Tpot. What's worse is that the memory usage of Tpot can grow exponentially depending on the number of generations, the population and offspring sizes, also data size and multi-thread, may take more than 200 GB RAM.

3.4 Analysis on stability

As we can see from the graphs. Autosklearn and H2O's performance improved over time, slowly but steadily. However, This is not true for Tpot, Sometimes it even drops over time. This instability comes from the randomness in the Tpot algorithm, More specifically, the instability comes from random mutation and randomly crossover. This randomness might be helpful when a model gets stuck at some performance level and can't improve through hyperparameters searching. For example in the Figure 8 we can see Tpot made a breakthrough with time limit of 2h randomly. However the randomness also brings some disadvantages, which means you may need more time to keep its performance stable. And this instability is also manifested in memory usage. Tpot often has a sharp rise in memory demand at a certain point, causing memory overflow, which caused a lot of trouble to our experiments. Last the time instability, for example, we do the test within time limit of 1 hour. But Tpot always overshoot the time limit, while H2O and Autosklearn will complete the task and only have a short timeout at most. This may be because the algorithm genetic algorithm inside TPOT: it always prioritizes doing complete generations, while H2O and Autosklearn stops training when there is not enough time left to train next model. But even though Tpot took more time than the other two, it's not significantly better.

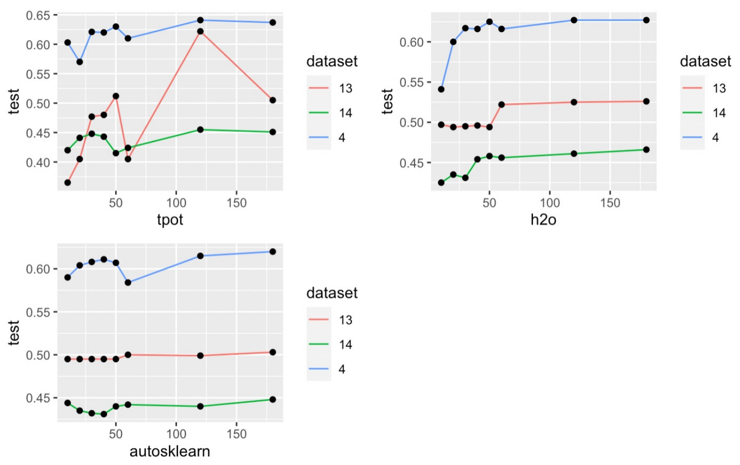


Figure 8: The Performance of three frameworks with different time limit

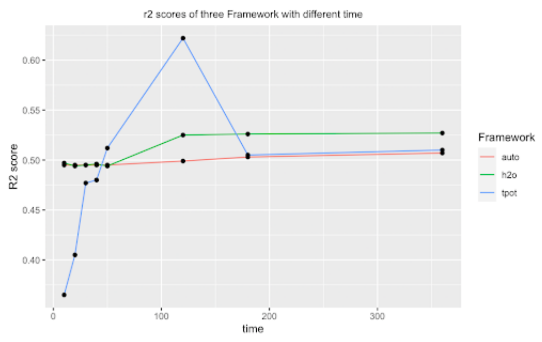


Figure 9: Comparison of the three frameworks on dataset 13

3.5 Analysis on flexibility

Tpot is totally based on python and scikit-learn package while H2O is written by JAVA in its core which means you need to configure a certain JAVA environment. Furthermore, Tpot allows you to export the best pipeline code in python easily. However, H2O do better on scalability and distributed computing. Because it builts on Map and Reduce model for large-scale data and it also compatibles with Hadoop and Spark.

4 Discussion, Conclusion and Guidance

We find that H2O performs the best on the QSAR regression datasets. The quantitative results from this experiment have extremely high variances, as such, it is important to think carefully about the quality of the code base, activity, feature set, and goals of these individual frameworks. Potential future work includes more granular comparison of the specific feature engineering, model selection, and hyperparameter optimization techniques of these projects and to perhaps expand the scope to more AutoML libraries and frameworks as they are developed.

For early data exploration, we would recommend Autosklearn. A personal computer is quite enough for Autosklearn. Running on one CPU core and limited memory, Autosklearn could provide a baseline stably. For companies which is trying to build a automatic analysis workflow, H2O would be the best choice. Benefit from the advantage of high multi-thread performance, the H2O is the most efficient and economic framework. As for the Tpot, make good use of the framework still need much experience and understanding of the data. It's far away from an automatic ML framework. However, with enough adjustment the Tpot would give the best model performance. Thus, we recommend regarding the Tpot as a tool to build the final model or to solve the most difficult problem.

References

[1] Farooq, Muhammad Shahid and Chaudhry, Azizul Haque and Shafiq, Mohammad and Berhanu, Girma. Deep neural nets as a method for quantitative structure–activity relationships. *Journal of chemical information and modeling*, 55(2):263–274, 2015.

[2] Sheridan, Robert P and Wang, Wei Min and Liaw, Andy and Ma, Junshui and Gifford, Eric M. Extreme gradient boosting as a method for quantitative structure–activity relationships. *Journal of chemical information and modeling*, 56(12):2353–2360, 2016.

[3] Svetnik, Vladimir and Liaw, Andy and Tong, Christopher and Culberson, J Christopher and Sheridan, Robert P and Feuston, Bradley P. Random forest: a classification and regression tool for compound classification and QSAR modeling. *Journal of chemical information and computer sciences*, 43(6):1947–1958, 2003.

[4] Olson, Randal S and Moore, Jason H TPOT: A tree-based pipeline optimization tool for automating machine learning *Workshop on automatic machine learning*,66–74,2016

[5] LeDell, Erin and Poirier, Sebastien H2o automl: Scalable automatic machine learning *Proceedings of the AutoML Workshop at ICML*,2020

[6] Feurer, Matthias and Klein, Aaron and Eggensperger, Katharina and Springenberg, Jost and Blum, Manuel and Hutter, Frank fficient and robust automated machine learning *Advances in neural information processing systems*,28,2015

[7] Balaji, Adithya and Allen, Alexander Benchmarking automatic machine learning frameworks *arXiv preprint arXiv:1808.06492*,2018

[toc,page]appendix

Dataset	R^2 in train set	trainning time	R^2 in test set	predicting time
1	0.83	3720.00	0.60	167.98
2	0.81	3624.58	0.67	24.58
3	0.82	3603.83	0.70	15.51
4	0.92	3603.06	0.62	5.24
5	0.85	3605.64	0.75	11.15
6	0.90	3601.71	0.74	182.77
7	0.97	3612.05	0.67	6.10
8	0.69	3701.71	0.65	34.72
9	0.84	3608.81	0.77	16.15
10	0.77	3705.95	0.71	31.68
11	0.75	3608.43	0.62	17.89
12	0.78	3701.29	0.68	24.64
13	0.61	3701.91	0.47	19.90
14	0.61	3607.43	0.40	14.74
15	0.89	3606.36	0.79	16.28

Table 2: Result of H2O

Dataset	R^2 in train set	trainning time	R^2 in test set	winning model
1	0.600	6327	0.527	SelectFwe
2	0.897	4218	0.609	GradientBoostingRegressor
3	0.871	4494	0.711	ExtraTreeRegressor
4	0.869	2144	0.610	RandomForestRegressor
5	0.939	3971	0.752	RandomForestRegressor
6	0.719	5890	0.647	LinearSVR
7	0.805	688	0.626	StackingEstimator
8	0.872	4525	0.714	ExtraTreesRegressor
9	0.872	3916	0.761	StackingEstimator
10	0.944	3983	0.764	RandomForestRegressor
11	0.863	4195	0.625	StackingEstimator
12	0.877	3991	0.668	RandomForestRegressor
13	0.750	4090	0.405	LinearSVR
14	0.714	3939	0.424	StackingEstimator
15	0.979	4215	0.802	GradientBoostingRegressor

Table 3: Result of Tpot

Dataset	R^2 in train set	trainning time	R^2 in test set	predicting time
1	0.77	3606.26	0.64	38.33
2	0.81	3612.49	0.67	8.69
3	0.82	3605.80	0.70	130.79
4	0.85	3621.75	0.58	9.13
5	0.85	3607.04	0.74	14.39
6	0.89	3604.21	0.85	37.89
7	0.88	3613.80	0.68	5.64
8	0.84	3605.47	0.72	21.76
9	0.89	3608.81	0.78	97.62
10	0.87	3607.17	0.77	30.03
11	0.83	3597.45	0.64	51.29
12	0.88	3607.79	0.74	27.30
13	0.82	3607.30	0.49	30.21
14	0.78	3610.67	0.44	128.90
15	0.92	3610.01	0.81	115.54

Table 4: Result of Autosklearn

Dataset	R^2 in train set	trainning time	R^2 in test set	predicting time
1	0.94	61.59	0.63	3.47
2	0.91	30.00	0.63	0.59
3	0.91	17.97	0.61	0.35
4	0.92	2.38	0.64	0.12
5	0.94	9.10	0.68	0.24
6	0.95	56.96	0.64	3.32
7	0.95	1.17	0.60	0.12
8	0.92	30.37	0.69	0.65
9	0.96	5.93	0.75	0.31
10	0.95	14.95	0.76	0.69
11	0.94	7.49	0.56	0.37
12	0.94	14.97	0.62	0.61
13	0.93	7.80	0.50	0.39
14	0.92	4.89	0.46	0.31
15	0.96	7.58	0.78	0.31

Table 5: Result of randomforest