35th European Modeling & Simulation Symposium
20th International Multidisciplinary Modeling & Simulation Multiconference

2724-0029 © 2023 The Authors.
doi: 10.46354/i3m.2023.emss.028

Performance Comparison of Microsoft's AutoML API

Philipp Neuhauser[1,*] and Stefan Wagner[1]

[1]Josef Ressel Center for Adaptive Optimization in Dynamic Environments, University of Applied Sciences Upper Austria, Softwarepark 11, Hagenberg, 4323, Austria

[*]Corresponding author. Email address: philipp.neuhauser@fh-hagenberg.at

Abstract

In recent years, many software libraries for automated machine learning (AutoML), such as H2O AutoML or Auto-Sklearn, have become increasingly popular as they propose to significantly simplify the ML workflow and to sustainably reduce the time required for manual feature engineering, hyperparameter tuning as well as model selection and evaluation. Among the younger and therefore less well-known representatives is Microsoft's ML.NET, about whose model builder API only little literature and performance comparisons exists. This paper summarizes the functionality of such frameworks and discusses general requirements for automated machine learning in more detail. Finally, several experiments compare ML.NET with already established AutoML libraries based on some datasets from the field of supervised learning with respect to model quality, the API's scope of functions and required computational resources.

Keywords: Automated Machine Learning; Microsoft AutoML; Performance Comparison; Hyperparameter Tuning

1. Introduction

Machine learning has become an integral part of many areas of daily life as it enables us to predict future events as well as to discover and extract hidden knowledge in a vast amount of data. However, changes in the underlying data often require the training of a new or updated model. Take for example the dynamic environment of Industrial Internet of Things (IIoT), where data sources (e.g., sensors, databases, real-time message brokers, etc.) are added or modified almost on a daily basis, it is of enormous importance that models are relearned as quickly as possible in order to continue delivering reliable results and predictions. There is often not much time for a laborious and manual training and hyperparameter optimization of precise models. AutoML provides a remedy for this problem and attempts to train models independently out of a set of available algorithms (also called trainers) and corresponding parameters within a given period. It then automatically selects that model, which achieves the best results on some provided validation data. Besides well-known AutoML frameworks such as H2O AutoML (LeDell and Poirier, 2020) and Auto-Sklearn (Feurer et al., 2020), ML.NET (Ahmed et al., 2019) is the way to go for .NET developers. Unfortunately, there is hardly any literature or benchmark comparisons available for the software library mentioned last. This paper aims to address this issue by evaluating the performance, features and functionality of Microsoft's AutoML API based on various datasets in order to assess its practical applicability.

In Section 2 the fundamentals and current state-of-the-art regarding automated machine learning are summarized and the model builder API of ML.NET (Microsoft AutoML) is introduced in more details. Section 3 focuses on a basic function comparison of the three selected software frameworks mentioned above. Some benchmark tests based on several datasets, representing traditional supervised machine learning tasks, as well as the further evaluation of the performance results are then gathered and discussed in Section 4. All key findings and a brief outlook for using Microsoft AutoML in a battle tested, real-world environment are finally outlined in Section 5.

2. AutoML: State of the Art

Data mining, speech processing, deep learning methods, information discovery and the creation of prediction and forecasting engines as well as recommendation systems are very time-consuming, cumbersome and costly challenges that often require experienced data scientists with fundamental background knowledge about the respective application domain (Hutter et al., 2019). This manual approach is like trying to find a needle in a haystack and mostly results in an highly iterative trial and error process until a satisfying model is found as different algorithms have to be tested and hyperparameters to be tuned. In order to speed up this process and to overcome expert knowledge, the field of automated machine learning tries to create suitable models with a minimum of human intervention (Tuggener et al., 2019).

2.1. Basic Workflow of AutoML Tools

Under the hood, AutoML libraries always follow a simple, iterative mental model that is presented in Figure 1. Depending on the quality and structure of the provided input dataset it must be decided if any preprocessing steps (e.g., data cleaning and transformation tasks or feature extraction and selection aspects) have to be applied and to what extent. Next, an appropriate machine learning algorithm must be chosen in order to solve the particular ML task. After that, a combination of hyperparameters must be explored and evaluated for the selected algorithm in order to receive an optimized, well-performing model within a fixed timespan and computational budget, such as CPU and memory usage (Feurer et al., 2015). Finally, the data scientist is responsible for selecting and deploying the best generated model to production or creating an ensemble out of a subset of best performing models. By creating multiple models out of a variety of different algorithms, AutoML can also be used at an early stage to determine if the provided dataset is suitable for solving a certain machine learning task at all or if none of the generated models performs better than the baseline (Tuggener et al., 2019).

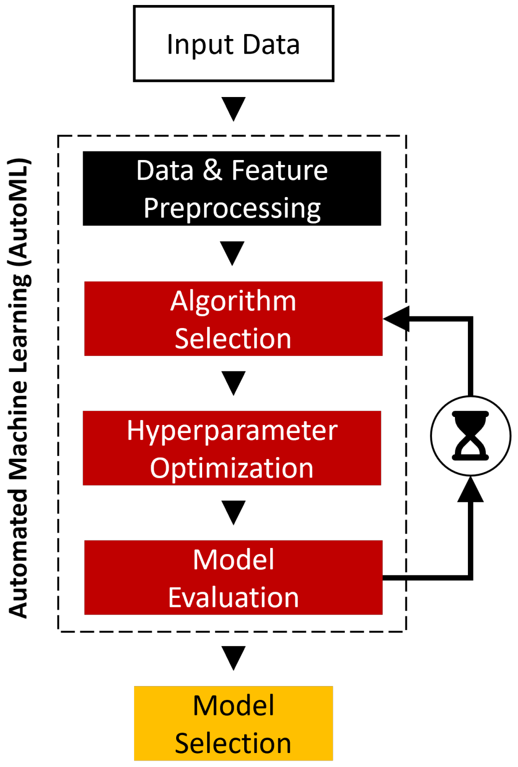

Input Data
Automated Machine Learning (AutoML)
Data & Feature Preprocessing
Algorithm Selection
Hyperparameter Optimization
Model Evaluation
Model Selection

Figure 1. The basic iterative AutoML workflow for automatically creating models with a minimum of human intervention, given a fixed timespan and computational budget (CPU and memory usage).

2.2. Common AutoML Optimization Techniques

Looking at the machine learning workflow in Figure 1, one can add automation for any pipeline step. Waring et al. (2020) provided an excellent overview over the possible AutoML disciplines in their article which are briefly summarized for the sake of completeness.

Automated feature engineering is the process of automatically identifying the most relevant variables or generating entirely new variables by combining existing ones, that are predictive of the outcome of interest with the overall goal to improve the model quality. As this is typically a very time-consuming task for a data scientist, many different approaches have emerged. Kanter and Veeramachaneni (2015) introduced the "expand-reduce" method, where a set of transformations is applied at once to a feature vector, followed by a feature selection and hyperparameter tuning step. Other approaches make use of genetic programming, reinforcement learning, hierarchical greedy search or meta learning.

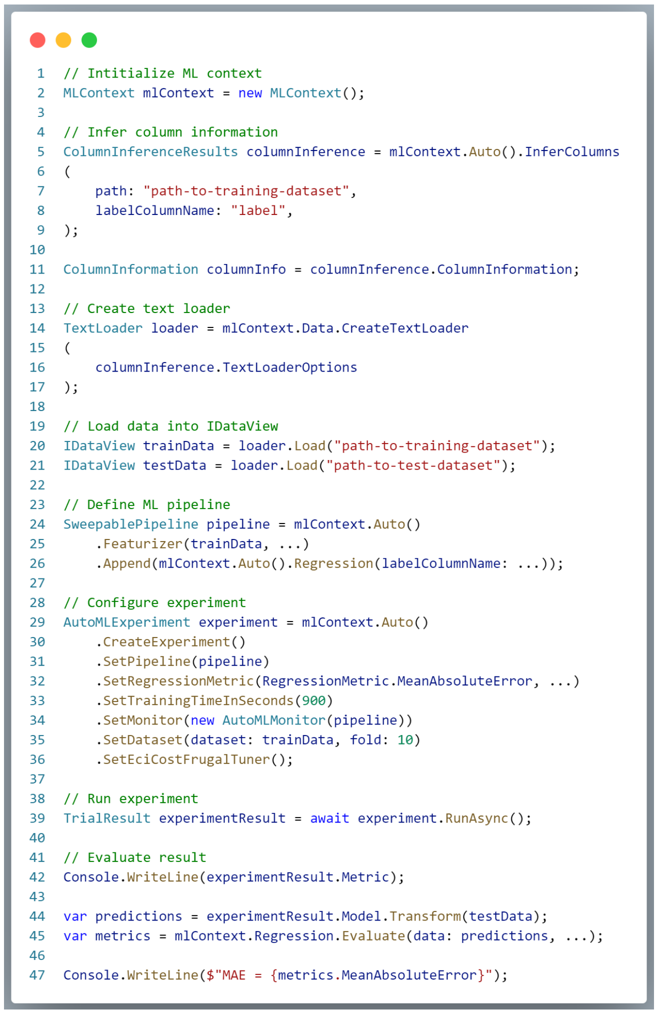
Automated hyperparameter optimization is a highly complex discipline where different selection mechanisms try to find an optimal combination for a training algorithm's "steering parameter" values that optimize (i.e., maximize/minimize) a certain performance metric for the machine learning task at hand. One possible approach is using unguided strategies that make no assumptions about the search space, e.g., Grid Search (brute-force) or Random Search (given a user-defined search space for hyperparameters). As these previous kinds of methods do not make use of past performance evaluations, better strategies can be found by using so-called "optimization from samples" methods. Evolutionary algorithms (Bäck, 1996) or the particle swarm optimization (Kennedy and Eberhart, 1995; Escalante et al., 2009) exchange information of good hyperparameter combinations without losing diversity. Nevertheless, Bayesian optimization has emerged as the state-of-the-art in this AutoML discipline by using a surrogate model and an acquisition function that try to optimize a mapping of various hyperparameter configurations to their achieved performance. The surrogate model is used to approximate the objective function, while the acquisition function is used to determine which hyperparameters to evaluate next.

Pipeline optimization techniques go one step further as they try to improve the entire AutoML process and not only specific aspects. One common approach is to tackle the so called "Combined Algorithm Selection and Hyperparameter optimization" (CASH) problem (Thornton et al., 2013). AutoML algorithms try to select an appropriate algorithm out of a list of available solvers that is then steadily optimized to score the highest validation performance amongst all algorithm-hyperparameter combinations (Tuggener et al., 2019). Again, this can be done by exploiting Baysian optimization methods, like in the Auto-WEKA platform that utilizes the SMAC (Sequential Model-based optimization for general Algorithm Configuration) tuner (Hutter et al., 2010). Auto-sklearn introduced two essential improvements to solve the CASH problem even more efficiently by using a meta-learning mechanism to "warmstart" the optimization with AutoML pipeline configurations that worked well for similar datasets. Another technical refinement is the automated ensemble construction of models evaluated during the tuning process that makes a pipeline configuration generally more robust and less prone to overfitting. Other optimization techniques make use of reinforcement learning or genetic programming (TPOT - Tree-based Pipeline Optimization Tool, Olson and Moore (2016)) for constructing sophisticated AutoML pipelines.

2.3. Introduction of the ML.NET AutoML API

As an integral part of the constantly evolving ML.NET (Microsoft.ML) library (Ahmed et al., 2019), the AutoML extension is relatively new and therefore less known in the automated machine learning community. Although there is an official CLI tool as well as an intuitive graphical user interface in Visual Studio for generating strong models without the need of writing any code, we focus solely on the AutoML model builder API which consists of some essential aspects. The following is a brief summary of the well prepared and detailed step-by-step tutorial, found on the official Microsoft website (Microsoft, 2023). The interplay of all required API classes is illustrated as a code snippet in Figure 2.

The first step for using ML.NET is initializing the `MLContext` which creates a new machine learning environment. Several operations and actions within the machine learning workflow can be generated with this instance in a catalog-based manner. In order to access the AutoML functionalities, one has to install an additional NuGet package. Next, the input data must be loaded from either a text file, a database, or even in-memory-collections. Microsoft AutoML is also capable



```
// Intitialize ML context
MLContext mlContext = new MLContext();

// Infer column information
ColumnInferenceResults columnInference = mlContext.Auto().InferColumns
(
    path: "path-to-training-dataset",
    labelColumnName: "label",
);

ColumnInformation columnInfo = columnInference.ColumnInformation;

// Create text loader
TextLoader loader = mlContext.Data.CreateTextLoader
(
    columnInference.TextLoaderOptions
);

// Load data into IDataView
IDataView trainData = loader.Load("path-to-training-dataset");
IDataView testData = loader.Load("path-to-test-dataset");

// Define ML pipeline
SweepablePipeline pipeline = mlContext.Auto()
    .Featurizer(trainData, ...)
    .Append(mlContext.Auto().Regression(labelColumnName: ...));

// Configure experiment
AutoMLExperiment experiment = mlContext.Auto()
    .CreateExperiment()
    .SetPipeline(pipeline)
    .SetRegressionMetric(RegressionMetric.MeanAbsoluteError, ...)
    .SetTrainingTimeInSeconds(900)
    .SetMonitor(new AutoMLMonitor(pipeline))
    .SetDataset(dataset: trainData, fold: 10)
    .SetEciCostFrugalTuner();

// Run experiment
TrialResult experimentResult = await experiment.RunAsync();

// Evaluate result
Console.WriteLine(experimentResult.Metric);

var predictions = experimentResult.Model.Transform(testData);
var metrics = mlContext.Regression.Evaluate(data: predictions, ...);

Console.WriteLine($"MAE = {metrics.MeanAbsoluteError}");
```

Figure 2. Basic usage of Microsoft AutoML for solving a regression machine learning task.

of inferring the corresponding data types and columns by using the `InferColumns` method. The class `TrainTestSplit` can then be used to separate the integrated data into a training and test (validation) set.

The next step is the definition of the machine learning pipeline which is represented as an instance of `SweepablePipeline`. That is basically a collection of `SweepableEstimator` that is a combination of an `Estimator` (untrained transformer) instance with a corresponding `SearchSpace` (range of available hyperparameters). With the `Featurizer` API one can automatically define basic data preprocessing steps by using the inferred column information. The resulting numeric feature vector is then used for model training by appending a trainer (algorithm) to the `SweepablePipeline`. Currently AutoML supports default trainers and search space configurations for binary classification, multi-class classification and regression machine learning tasks.

One last task to do is to define an `AutoMLExperiment` where the following components are chained together. The previously created `SweepablePipeline` defines how the input (training) dataset has to be properly transformed. Over that, an evaluation metric (e.g., area under ROC curve for binary classification tasks) must be defined which the pipeline tries to optimize during the AutoML process. Finally, one must specify a maximum time limit in seconds or a maximum number of models for the experiment to run. A `Trial` is described as a single hyperparameter optimization run and a trial runner is a component that uses the AutoML pipeline and trial settings to generate a `TrialResult`.

Optionally, Microsoft AutoML also offers the opportunity to configure the algorithm that is used for hyperparameter tuning. By default, the ECI Cost Frugal Tuner (Wang et al., 2021) for hierarchical search spaces is used. This can be changed by setting the experiment's `Tuner` to Cost Frugal Tuner (for concerning training cost) (Wu et al., 2020), SMAC (Bayesian optimization) (Hutter et al., 2010), Grid Search (recommended for small search spaces) or Random Search. Next, the question arises as to how this software library now compares directly to other AutoML frameworks in order to assess its practicality.

3. Functional Comparison of AutoML Libraries

After having described the fundamental aspects for automated machine learning frameworks, this section will focus on a detailed comparison of Microsoft AutoML (.NET) based on some important requirements. Although there are many different automated machine learning libraries available that Microsoft AutoML could be compared with, the choice fell on H2O AutoML (Java) and Auto-Sklearn (Python) as they are very popular, similar in their concrete usage and based on different runtimes. First, the three frameworks are evaluated for meeting the AutoML requirements mentioned in Section 3.1. After that, they are tested on some datasets for solving different machine learning tasks in the next Section 4.

3.1. Basic Requirements for AutoML Libraries

With modern AutoML APIs come several requirements, requests and limitations. On the one hand, such libraries should be able to solve a variety of common machine learning tasks by applying a single mental programming model. Over this, it should be possible to load data from many different sources and to automatically infer the column and datatype information. Nevertheless, manual ETL operations and optional data preprocessing steps must also be applicable to a machine learning pipeline. For every single machine learning task, there should be offered numerous training algorithms as well as different hyperparameter tuning mechanisms, which at best also can be enabled and disabled or entirely exchanged. Furthermore, it is also desirable to bring in some expert knowledge by manually adjusting the search space. For debugging, evaluation and deeper analysis of the trained models, there must be integrated possibilities for logging and monitoring. In times of Big Data and IIoT, support for huge data sets is becoming increasingly important. Online processing of data windows is the desired way to go instead of keeping the entire dataset in memory. Sufficient performance, framework updates on a regular basis and mechanisms to evaluate the feature importance are also fundamental requirements for AutoML libraries. Finally, it should be technically possible to somehow limit the available memory, CPU and training time in order to make the AutoML framework applicable to less powerful machines or edge devices.

3.2. Comparison Based on AutoML Requirements

Table 1 compares Microsoft AutoML, H2O AutoML and Auto-Sklearn based on the previously defined requirements in Section 3.1. All frameworks are able to automatically solve classification and regression tasks by using cross-validation to check model performance, execute experiments in parallel and can import data from files or in-memory collections. Moreover, Microsoft.ML is also capable of loading files directly from an SQL server database. Every AutoML framework can cope with datatype inference as well as manually defined ETL operations. Nevertheless, the libraries differ in the number of available training algorithms and hyperparameter tuning mechanisms, where Microsoft AutoML is clearly in the lead, which may turn out to be a big advantage for certain ML problem tasks. Other striking differences are the required software runtimes and supported operating systems. In times of virtualization, this should not be a big deal as all libraries can be deployed or hosted in e.g., Docker containers. Moreover, all frameworks use a maximum time limit as default stopping criteria. In addition to that, Microsoft AutoML and H2O AutoML also support a maximum number of models that should be trained before the process stops. Furthermore, the development of these two libraries is driven by renowned companies (Microsoft and H2O.ai), while Auto-Sklearn is more regarded as an academic software solution.

4. Benchmarking Microsoft AutoML

Ferreira et al. (2021) already bench-marked various popular AutoML frameworks and tools. For every traditional supervised learning task (binary-/multi-class classification and regression), they have used the four most downloaded (and sanitized) datasets from OpenML (Vanschoren et al., 2013), that are also shown in Table 2. They are characterized by a different number of entries (rows), features (columns) as well as possible values of the target variable to be predicted (labels). For the purpose of direct comparison, the same datasets are used in this paper.

Table 1. AutoML framework comparison

Requirements	Microsoft AutoML	H2O AutoML	Auto-Sklearn
Supported ML Tasks	Binary-/Multi-class Classification, Regression	Binary-/Multi-class Classification, Regression	Binary-/Multi-class Classification, Regression
Supported Data Sources	Text Files, SQL Server, In-Memory-Collections	Text Files, In-Memory-Collections	Text Files, In-Memory-Collections
Auto Data Type Inference	Yes	Yes	Yes
Manual ETL Support	Yes	Yes	Yes
Training Algorithms	Averaged Perceptron, SDCA, Symbolic SGD Logistic Regression, LBFGS Logistic Regression, Light GBM, Fast Tree, Fast Forest, GAM, Field Aware Factorization Machine, Prior Trainer, Linear SVM, Light GBM Multiclass, SDCA Maximum Entropy Multiclass, LBFGS Maximum Entropy, Naive Bayes, One Versus All Trainer, Pairwise Coupling, LBFGS Poisson Regression, Light GBM Regression, OLS, Online Gradient Descent, Fast Tree Tweedie	Distributed Random Forest (DRF), Extremely Randomized Trees (XRT), Generalized Linear Model with regularization (GLM), GLM, XGBoost GBM, Deep Learning, Stacked Ensemble	AdaBoost, Bernoulli NB, Descision Tree, Extra Trees, Gaussian NB, Gradient Boosting, K Nearest Neighbors, LDA, Liblinear SVC, Libsvm SVC, MLP, Multinominal NB, Passive Aggressive, QDA, Random Forest, SGD, ARD Regression, Gaussian Process, Gradient Boosting, Liblinear SVR, Libsvm SVR
Exchangeable Hyperparameter Tuners	ECI Cost Frugal, Cost Frugal, SMAC, Grid Search, Random Search	Random Grid Search	SMAC
Adjustable Search Space	Yes	Yes	Yes
Stopping Criterias	Max. training time, Max. # models to train	Max. training time, Max. # models to train	Max. training time
Validation Mechanism	Cross Validation	Cross Validation	Cross Validation
Support for Parallelism	Yes	Yes	Yes
Operating System	Windows and Linux	Windows and Linux	Linux
Framework/Runtime	.NET	Java (Core SDK) and Python (API)	Python
Integrated Logging Mechanisms	Yes	Yes	Yes

Table 2. Description of chosen OpenML datasets (Ferreira et al., 2021)

Dataset	ML Task	Rows	Features	Classes	Values
churn	binary	5000	21	2	{0, 1}
credit	binary	1000	21	2	{0, 1}
diabetes	binary	768	9	2	{0, 1}
qsar	binary	1055	42	2	{0, 1}
cmc	multi-class	1473	10	10	{0 ... 9}
dmft	multi-class	797	5	6	{0 ... 5}
mfeat	multi-class	2000	7	10	{0 ... 9}
vehicle	multi-class	846	19	4	{0 ... 3}
cholesterol	regression	303	14	152	[126, 564]
cloud	regression	108	7	94	[0, 6]
liver	regression	345	6	16	[0, 20]
plasma	regression	315	14	257	[179, 1727]

4.1. Experiment Setup

All previously introduced datasets are stored in the CSV format where the first line contains the column headers. The remaining rows were then shuffled a billion times, before they were separated into two different files. 80% of the data was used for training and the remaining 20% for the later evaluation of the best found model. Moreover, it was ensured that there were no missing values or other impurities within the datasets. To ensure a fair comparison, the three AutoML libraries must try to solve the tasks as good as possible using their default pipeline settings. This means that no expert knowledge is involved. It was solely up to the respective libraries which algorithms are used and how their hyperparameters must be configured subsequently in order to obtain accurate models. For every AutoML experiment, the prepared training and test data files were initially loaded from the file system. The configured machine learning pipeline only consists of a featurization step and the definition for the ML task to be solved. The creation of the feature column that is used for prediction was done by utilizing the inferred data types and column names. Since the focus of this paper is on the comparison of general machine learning methods, Deep Learning algorithms were therefore disabled in H2O AutoML. The expected behavior is a broad horizontal search through traditional ML approaches. Internally a 10-fold cross validation was applied to determine the best model found during the experiment for each AutoML tool and task whose performance was then evaluated on the unseen 20% test data. For the binary classification experiments, the Area Under the Receiver Operating Characteristic Curve (ROC AUC) was chosen, where 1.0 denotes a perfectly trained model. The Macro Accuracy score was the evaluation metric of choice for all multi-class classification problems. The closer a specific metric is to 1.0, the better the learned model can be considered. Finally, the Mean Absolute Error (MAE) was used for regression tasks where 0.0 is the best quality result a model can achieve. All experiments were executed on an 11th Gen Intel Core i7-1185G7 with 4 cores and 32 GB of RAM.

Table 3. Achieved AutoML results for the selected OpenML datasets

Dataset	Tool	Metric	Quality	# Models
churn	MS AutoML	ROC AUC	0,914	107
	H2O AutoML		**0.931**	161
	Auto-Sklearn		0.884	32
credit	MS AutoML	ROC AUC	0,829	144
	H2O AutoML		**0.837**	315
	Auto-Sklearn		0.673	81
diabetes	MS AutoML	ROC AUC	0,782	159
	H2O AutoML		**0.786**	797
	Auto-Sklearn		0.674	176
qsar	MS AutoML	ROC AUC	0,953	183
	H2O AutoML		**0.963**	161
	Auto-Sklearn		0.878	74
cmc	MS AutoML	Macro Accuracy	0,536	135
	H2O AutoML		0.543	524
	Auto-Sklearn		**0.577**	326
dmft	MS AutoML	Macro Accuracy	**0,261**	172
	H2O AutoML		0.201	916
	Auto-Sklearn		0.213	315
mfeat	MS AutoML	Macro Accuracy	0,722	29
	H2O AutoML		**0.795**	53
	Auto-Sklearn		0.718	332
vehicle	MS AutoML	Macro Accuracy	0,780	38
	H2O AutoML		0.785	237
	Auto-Sklearn		**0.848**	285
cholesterol	MS AutoML	MAE	0,695	241
	H2O AutoML		**0.658**	1071
	Auto-Sklearn		N/A	0
cloud	MS AutoML	MAE	0,301	187
	H2O AutoML		0.247	2207
	Auto-Sklearn		**0.141**	155
liver	MS AutoML	MAE	**2,104**	192
	H2O AutoML		2.399	1473
	Auto-Sklearn		2.142	266
plasma	MS AutoML	MAE	**0.00**	40
	H2O AutoML		174.894	1499
	Auto-Sklearn		179.426	233

The training time for each experiment was specified with 15 minutes. Each AutoML framework was configured to use a 10-fold cross validation. All remaining parameters were specified with their default values. No limitations regarding CPU and memory were made, so every framework was allowed to use all available hardware resources.

4.2. Results and Discussion

Multiple tests of all selected AutoML libraries with different training times led to almost always the same quality results with only a few percentage points between them, proving that Microsoft AutoML can compete with already well-established software libraries for automated machine learning (see Table 3). However, a serious difference could be observed with Auto-Sklearn because this framework was sometimes not able to learn models or model ensembles when the training time was set below 5 minutes. It is also worth mentioning, that the provided datasets are part of Auto-Sklearn's metadata but they were automatically removed before executing the AutoML experiments.

Generally speaking, all AutoML frameworks achieved similar model qualities as shown in Table 3 but some of them exhibited greater proficiency in solving specific ML tasks than others. For all binary classification machine learning tasks, H2O AutoML always performed slightly better than Microsoft AutoML (*churn*: +0.017, *credit*: +0.008, *diabetes*: +0.004, *qsar*: +0.010) but in most of the cases significantly better than Auto-Sklearn (*churn*: +0.047, *credit*: +0.164, *diabetes*: +0.112, *qsar*: +0.085). With regard to the four selected multi-class classification datasets, no clear winner could be identified as the scored metrics are very close to each other. For *cmc* and *vehicle* Auto-Sklearn trained the best model with a Macro Accuracy of 0.577 and 0.848. Microsoft AutoML reached a score of 0.261 for the *dmft* dataset and H2O AutoML a score of 0.795 for the *mfeat* dataset. Over that, non of the evaluated AutoML framework was always clearly better than the other ones for regression problem tasks. Although an achieved MAE of 0.0 on the *plasma* test dataset is an indicator for overfitting, Microsoft AutoML also obtained the lowest MAE value (2.104) for the *liver* dataset. H2O trained the best model for *cholesterol* (0.658) whereas Auto-Sklearn had serious difficulties with data preprocessing and was therefore not able to generate any model. Nevertheless, one can easily see, that Auto-Sklearn achieved the best MAE quality result of 0.141 for the *cloud* dataset. A direct comparison with the results of Ferreira et al. (2021) also shows that H2O AutoML and Auto-Sklearn slightly gained in performance with respect to model quality. In 2021, H2O AutoML achieved an ROC AUC score of 0.919 (-0,012) for *churn* and 0.803 (-0.034) for *credit*. Moreover two years ago, the best multi-class classification model that was trained with Auto-Sklearn reached a Macro Accuracy of 0.545 (- 0.032) for the *cmc* dataset and the best regression model scored an MAE of 2.329 (+ 0,187) for the *liver* dataset. Furthermore, the number of models created or trial runs also varies greatly. It can be clearly seen that H2O AutoML has often trained a remarkably larger number of prediction models, especially for the regression examples. Finally, regarding the required hardware resources it must be said that H2O AutoML has constantly used nearly the entire CPU power (around 90%) and 1000-1700 MB memory for running the experiments. Microsoft AutoML (between 40-70% and 50-200 MB) and Auto-Sklearn (30-60% and 500-600 MB), on the other hand, were somewhat more sparing with the provided resources. High resource requirements without any means of limiting them can prove to be a disadvantage in practice. Despite of limiting the maximum training time, only Microsoft AutoML and Auto-Sklearn made full use of the entire 15 minutes. H2O always stopped a bit earlier as no further improvements could be made, or additional models be trained.

5. Conclusion and Outlook

In this work, a brief overview of the fundamentals and current state-of-the art techniques regarding automated machine learning was given - a discipline that is increasingly getting more important for all research areas as it aims to generate reliable models with only a minimum amount of human intervention. The functionality and usage of Microsoft AutoML, a model builder API based on ML.NET, was then described in more detail. Being relatively new, there is limited literature and performance comparisons with established AutoML libraries, such as Auto-Sklearn or H2O AutoML. Therefore, this paper aimed to provide valuable insights into Microsoft AutoML's capabilities and potential compared to its counterparts. Several experiments on different datasets clearly proofed, that this library can easily compete with other popular and well-established automated machine learning frameworks. An easy-to-use and intuitive API as well as a variety of applicable and exchangeable training algorithms also speak in favor of using Microsoft AutoML. Furthermore, the possibility to leverage a variety of different hyperparameter tuning algorithms allows scoring better results for diverse datasets based on ubiquitous supervised machine learning tasks. Domain-specific expert knowledge can also be integrated by manually adjusting the search space of a training algorithm's available hyperparameters. ML.NET also handles unbounded data within windows, which can be very beneficial for Big Data applications. For example, by combining or extending Microsoft's powerful and robust AutoML API with modern message broker systems, such as Apache Kafka, one can consume real-time data streams and train multiple machine learning models with different training algorithms in parallel and increase the performance by exchanging the hyperparameter tuners on the fly. However, it is worth mentioning that currently none of the three AutoML libraries compared support tasks in the realm of unsupervised learning. Furthermore, the findings are based on a few different-sized datasets, which is why more experiments should be conducted in a future work. A detailed comparison of the different hyperparameter tuners in Microsoft AutoML could also be the content for another paper. Nevertheless, the first results and evaluations of this paper are very promising and clearly advocate for the usage of Microsoft's AutoML API, making it highly recommended not even for developers of .NET applications.

6. Acknowledgements

The financial support by the Austrian Federal Ministry for Digital and Economic Affairs and the National Foundation for Research, Technology and Development and the Christian Doppler Research Association is gratefully acknowledged.

References

Ahmed, Z., Amizadeh, S., Bilenko, M., Carr, R., Chin, W.-S., Dekel, Y., Dupre, X., Eksarevskiy, V., Filipi, S., Finley, T., et al. (2019). Machine learning at Microsoft with ML. NET. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 2448–2458.

Bäck, T. (1996). *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*. Oxford University Press.

Escalante, H. J., Montes, M., and Sucar, L. E. (2009). Particle swarm model selection. *Journal of Machine Learning Research*, 10(2).

Ferreira, L., Pilastri, A., Martins, C. M., Pires, P. M., and Cortez, P. (2021). A Comparison of AutoML Tools for Machine Learning, Deep Learning and XGBoost. In *2021 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8.

Feurer, M., Eggensperger, K., Falkner, S., Lindauer, M., and Hutter, F. (2020). Auto-Sklearn 2.0: Hands-free AutoML via Meta-Learning. *arXiv:2007.04074 [cs.LG]*.

Feurer, M., Klein, A., Eggensperger, K., Springenberg, J., Blum, M., and Hutter, F. (2015). Efficient and Robust Automated Machine Learning. In Cortes, C., Lawrence, N., Lee, D., Sugiyama, M., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc.

Hutter, F., Hoos, H. H., and Leyton-Brown, K. (2010). Sequential model-based optimization for general algorithm configuration (extended version). *Technical Report TR-2010–10, University of British Columbia, Computer Science, Tech. Rep.*

Hutter, F., Kotthoff, L., and Vanschoren, J. (2019). *Automated Machine Learning*. Springer Nature.

Kanter, J. M. and Veeramachaneni, K. (2015). Deep feature synthesis: Towards automating data science endeavors. In *2015 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, pages 1–10.

Kennedy, J. and Eberhart, R. (1995). Particle swarm optimization. In *Proceedings of ICNN'95 - International Conference on Neural Networks*, volume 4, pages 1942–1948 vol.4.

LeDell, E. and Poirier, S. (2020). H2O AutoML: Scalable Automatic Machine Learning. *7th ICML Workshop on Automated Machine Learning (AutoML)*.

Microsoft (2023). How to use the ML.NET Automated Machine Learning (AutoML) API.

Olson, R. S. and Moore, J. H. (2016). TPOT: A tree-based pipeline optimization tool for automating machine learning. In *Workshop on automatic machine learning*, pages 66–74. PMLR.

Thornton, C., Hutter, F., Hoos, H. H., and Leyton-Brown, K. (2013). Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 847–855.

Tuggener, L., Amirian, M., Rombach, K., Lörwald, S., Varlet, A., Westermann, C., and Stadelmann, T. (2019). Automated Machine Learning in Practice: State of the Art and Recent Results. In *2019 6th Swiss Conference on Data Science (SDS)*, pages 31–36.

Vanschoren, J., van Rijn, J. N., Bischl, B., and Torgo, L. (2013). OpenML: Networked Science in Machine Learning. *SIGKDD Explorations*, 15(2):49–60.

Wang, C., Wu, Q., Weimer, M., and Zhu, E. (2021). Flaml: A fast and lightweight automl library. *Proceedings of Machine Learning and Systems*, 3:434–447.

Waring, J., Lindvall, C., and Umeton, R. (2020). Automated machine learning: Review of the state-of-the-art and opportunities for healthcare. *Artificial Intelligence in Medicine*, 104:101822.

Wu, Q., Wang, C., and Huang, S. (2020). Cost Effective Optimization for Cost-related Hyperparameters. *CoRR*, abs/2005.01571.