In a recent [blog post](https://www.statworx.com/blog/a-performance-benchmark-of-google-automl-vision-using-fashion-mnist/) our CEO Sebastian Heinz wrote about Google's newest stroke of genius – [AutoML Vision](https://cloud.google.com/vision/automl/docs/). A cloud service "*that is able to build deep learning models for image recognition completely fully automated and from scratch*". AutoML Vision is part of the [current trend](https://www.h2o.ai/blog/the-different-flavors-of-automl/) towards the automation of machine learning tasks. This trend started with automation of hyperparameter optimization for single models (Including services like [SigOpt](https://sigopt.com/), [Hyperopt](http://jaberg.github.io/hyperopt/), [SMAC](https://github.com/automl/SMAC3)), went along with automated feature engineering and selection (see my colleague [Lukas' blog post](https://www.statworx.com/blog/data-science/automated-feature-selection-using-bouncer) about our [bounceR](https://github.com/STATWORX/bounceR) package) towards full automation of complete data pipelines including automated model stacking (a common model ensembling technique).

One company at the frontier of this development is certainly [h2o.ai](https://www.statworx.com/de/blog/a-performance-benchmark-of-different-automl-frameworks/www.h2o.ai). They developed both a free Python/R library ([H2O AutoML](http://docs.h2o.ai/h2o/latest-stable/h2o-docs/automl.html?_ga=2.24204030.573118300.1535804795-2033992266.1523379628)) as well as an enterprise ready software solution called [Driverless AI](https://www.h2o.ai/products/h2o-driverless-ai/). But H2O is by far not the only player on the field. This blog post will provide you with a short comparison between two freely available Auto ML solutions and compare them by predictive performance as well as general usability.

**H2O AutoML**

[H2O AutoML](http://docs.h2o.ai/h2o/latest-stable/h2o-docs/automl.html) is an extension to H2O's popular java based open source machine learning framework with APIs for Python and R. It automatically trains, tunes and cross-validates models (including [Generalized Linear Models [GLM]](https://www.statworx.com/blog/pushing-ordinary-least-squares-to-the-limit-with-xy/), [Gradient Boosting Machines [GBM]](https://www.statworx.com/blog/xgboost-tree-vs-linear/), Random Forest [RF], [Extremely Randomized Forest [XRF]](https://www.statworx.com/blog/using-machine-learning-for-causal-inference/), and [Neural Networks](https://www.statworx.com/de/blog/wie-lernen-neuronale-netze/)). Hyperparameter optimization is done using a random search over a list of reasonable parameters (both RF and XRF are currently not tuned). In the end, H2O produces a leaderboard of models and builds two types of stacked ensembles from the base models. One including all base models, the other including only the best base model of each family.

Model training can be controlled by either the number of models to be trained, or the total training time. Especially the later makes model training quite transparent. One of the big advantages of H2O is that all models are parallelized out-of-the-box.

**auto-sklearn**

[auto-sklearn](https://automl.github.io/auto-sklearn/stable/) is an automated machine learning toolkit based on Python's Scikit-Learn Library. A detailed explanation of auto-sklearn can be found in Feurer et al. (2015). In H2O AutoML, each model was independently tuned and added to a leaderboard. In auto-sklearn, the authors combine model selection and hyperparameter optimization in what they call "*Combined Algorithm Selection and Hyperparameter optimization*" (CASH). This joint optimization problem is than solved using a tree-based Bayesian optimization methods called "*Sequential Model-based Algorithm Configuration*" (SMAC) (see Bergstra 2011).

So contrary to H2O AutoML, auto-sklearn optimizes a complete modeling pipeline including various data and feature preprocessing steps as well as the model selection and hyperparameter optimization. Data preprocessing includes one-hot-encoding, scaling, imputation, and balancing. Feature preprocessing includes, among others, feature agglomeration, ICA and PCA. Algorithms included in auto-sklearn are similar to those in H2O AutoML, but in addition also includes more traditional methods like k-Nearest-Neighbors (kNN), Naive Bayes, and Support Vector Machines (SVM).

Similar to H2O AutoML, auto-sklearn includes a final model ensemble step. Whereas H2O AutoML uses simple but efficient model stacking, auto-sklearn uses ensemble selection. A greedy method that adds individual models iteratively to the ensemble if and only if they increase the validation performance. Like H2O, auto-sklearn allows model training to be controlled by the total training time.

**Benchmark**

In order to compare the predictive performance of H2O's AutoML with auto-sklearn, one can conduct a small simulation study. My colleague André's R package [Xy](https://www.statworx.com/blog/benchmarking-feature-selection-algorithms-with-xy/) offers a straightforward way to simulate regression datasets with linear, non-linear, and noisy relationships. Using multiple (ten in total) simulation runs makes the whole simulation a bit more robust. The following R code was used to simulate the data:

library(Xy)

library(caret)

library(dplyr)

library(data.table)

# Number of datasets

n\_data\_set <- 10

for (i in seq(n\_data\_set)) {

# Sim settings

n <- floor(runif(1, 1000, 5000))

n\_num\_vars <- c(sample(2:10, 1), sample(2:10, 1))

n\_cat\_vars <- c(0, 0)

n\_noise\_vars <- sample(1:5, 1)

inter\_degree <- sample(2:3, 1)

# Simulate data

sim <- Xy(n = n,

numvars = n\_num\_vars,

catvars = n\_cat\_vars,

noisevars = n\_noise\_vars,

task = Xy\_task(),

nlfun = function(x) {x^2},

interactions = 1,

sig = c(1,4),

cor = c(0),

weights = c(-10,10),

intercept = TRUE,

stn = 4)

# Get data and DGP

df <- sim$data

dgp <- sim$dgp

# Remove Intercept

df[, "(Intercept)"] <- NULL

# Rename columns

names(df) <- gsub("(?

Since auto-sklearn is only available in Python, switching languages is necessary. Therefore, loading the raw data in Python is the next step:

import pandas as pd

# Load data

df\_train = pd.read\_csv("../data/Xy/1\_train.csv")

df\_test = pd.read\_csv("../data/Xy/1\_test.csv")

# Columns

cols\_train = df\_train.columns.tolist()

cols\_test = df\_test.columns.tolist()

# Target and features

y\_train = df\_train.loc[:, "label"]

X\_train = df\_train.drop("label", axis=1)

y\_test = df\_test.loc[:, "label"]

X\_test = df\_test.drop("label", axis=1)

Having the data in Python, the training procedure can start. In order to make the results comparable, both frameworks used, where possible, similar settings. This included 60 minutes of training for each dataset, 5-fold crossvalidation for model evaluation and ensemble building, no preprocessing (not available in H2O AutoML and therefore deactivated in auto-sklearn), and a limitation to similar algorithms (namely GLM, RF, XRF, and GBM).

As previously noted, H2O supports out-of-the-box parallelization. By default, auto-sklearn only uses two cores, while also supporting more cores, at least in theory. While there is a [manual](https://automl.github.io/auto-sklearn/stable/manual.html#parallel-computation) on how to do that, I was not able to get it working on my system (OSX 10.13, Python 3.6.2 Anaconda). Therefore H2O was also limited to only two cores.

from autosklearn.regression import AutoSklearnRegressor

from autosklearn.metrics import mean\_squared\_error

# Settings

estimators\_to\_use = ["random\_forest", "extra\_trees", "gradient\_boosting", "ridge\_regression"]

preprocessing\_to\_use = ["no\_preprocessing"]

# Init auto-sklearn

auto\_sklearn = AutoSklearnRegressor(time\_left\_for\_this\_task=60\*60,

per\_run\_time\_limit=360,

include\_estimators=estimators\_to\_use,

exclude\_estimators=None,

include\_preprocessors=preprocessing\_to\_use,

exclude\_preprocessors=None,

ml\_memory\_limit=6156,

resampling\_strategy="cv",

resampling\_strategy\_arguments={"folds": 5})

# Train models

auto\_sklearn.fit(X=X\_train.copy(), y=y\_train.copy(), metric=mean\_squared\_error)

it\_fits = auto\_sklearn.refit(X=X\_train.copy(), y=y\_train.copy())

# Predict

y\_hat = auto\_sklearn.predict(X\_test)

# Show results

auto\_sklearn.cv\_results\_

auto\_sklearn.sprint\_statistics()

auto\_sklearn.show\_models()

auto\_sklearn.get\_models\_with\_weights()

import h2o

from h2o.automl import H2OAutoML

# Shart h2o cluster

h2o.init(max\_mem\_size="8G", nthreads=2)

# Upload to h2o

df\_train\_h2o = h2o.H2OFrame(pd.concat([X\_train, pd.DataFrame({"target": y\_train})], axis=1))

df\_test\_h2o = h2o.H2OFrame(X\_test)

features = X\_train.columns.values.tolist()

target = "target"

# Training

auto\_h2o = H2OAutoML(max\_runtime\_secs=60\*60)

auto\_h2o.train(x=features,

y=target,

training\_frame=df\_train\_h2o)

# Leaderboard

auto\_h2o.leaderboard

auto\_h2o = auto\_h2o.leader

# Testing

df\_test\_hat = auto\_h2o.predict(df\_test\_h2o)

y\_hat = h2o.as\_list(df\_test\_hat["predict"])

# Close cluster

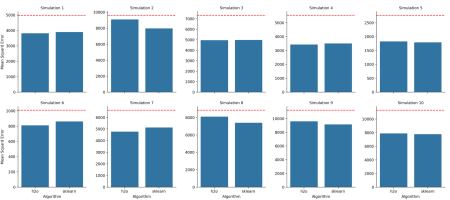
h2o.cluster().shutdown()

The complete code, including all simulation runs and visualization of results can be find on my [GitHub repo](https://github.com/fabianmax/ML-Automation).

**Results**

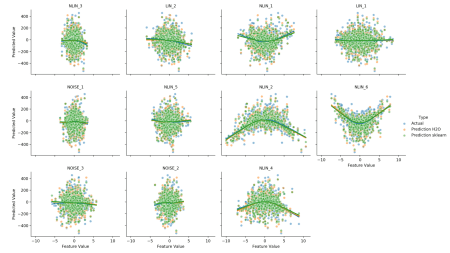
First, some words of caution: The results presented in the next sections are by no mean representative. Both H2O and the authors of auto-sklearn recommend to run their frameworks for hours, if not even days. Given ten different datasets, this was beyond the scope of a blog post. For the same reason of feasibility, the datasets are restricted to a rather small size. For a more elaborated performance comparison see for example Balaji and Allen (2018).

Figure 1 shows the Mean Squared Error of both frameworks produced on the test sample. The horizontal line, indicating the result from a vanilla Random Forest (from scikit-learn), serves as a benchmark. As one can see, the results are pretty similar for both frameworks and all data sets. Actually, it is a tie, with five wins for H2O and five wins for auto-sklearn.



The percentage difference between the average errors is 1.04\%in favor of auto-sklearn. Thus, auto-sklearn is on average about 1\%better than H2O. Compared with the vanilla RF, H2O's AutoML is on average 23.4\%better than the benchmark, while auto-sklearn is 24.6\%better.

The sheer closeness of the results can be further illustrated when taking a look at the predicted values. Figure 2 shows exemplary the predicted values for one particular dataset against all feature values (linear, non-linear and noise features). As one can see, the estimated effects for both frameworks are almost identical and pretty close to the actual relationship.



**Summary**

Automatic Machine Learning frameworks can provide promising results for standard machine learning task while keeping the manual efforts down to a minimum. This blog post compared two popular frameworks, namely H2O's AutoML and auto-sklearn. Both reached comparable results on ten simulated datasets, while outperforming vanilla models significantly. Beside predictive performance, H2O's AutoML offers some additional features like native parallelization, API for R, support for XGBoost and GPU training making it even more attractive.

**References**

* Feurer, Matthias, Aaron Klein, Katharina Eggensperger, Jost Tobias Springenberg, Manuel Blum, and Frank Hutter 2015. "Efficient and Robust Automated Machine Learning." NIPS 2015. <https://ml.informatik.uni-freiburg.de/papers/15-NIPS-auto-sklearn-preprint.pdf>
* Balaji, Adithya and Alexander Allen. 2018. "Benchmarking Automatic Machine Learning Frameworks." <https://arxiv.org/pdf/1808.06492.pdf>.
* Bergstra, James, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. 2011. "Algorithms for Hyper-Parameter Optimization." NIPS 2011. <https://papers.nips.cc/paper/4443-algorithms-for-hyper-parameter-optimization.pdf>