AutoML: Automatic Machine Learning

In recent years, the demand for machine learning experts has outpaced the supply, despite the surge of people entering the field. To address this gap, there have been big strides in the development of user-friendly machine learning software that can be used by non-experts. The first steps toward simplifying machine learning involved developing simple, unified interfaces to a variety of machine learning algorithms (e.g. H2O).

Although H2O has made it easy for non-experts to experiment with machine learning, there is still a fair bit of knowledge and background in data science that is required to produce high-performing machine learning models. Deep Neural Networks in particular are notoriously difficult for a non-expert to tune properly. In order for machine learning software to truly be accessible to non-experts, we have designed an easy-to-use interface which automates the process of training a large selection of candidate models. H2O's AutoML can also be a helpful tool for the advanced user, by providing a simple wrapper function that performs a large number of modeling-related tasks that would typically require many lines of code, and by freeing up their time to focus on other aspects of the data science pipeline tasks such as data-preprocessing, feature engineering and model deployment.

H2O's AutoML can be used for automating the machine learning workflow, which includes automatic training and tuning of many models within a user-specified time-limit. Stacked Ensembles – one based on all previously trained models, another one on the best model of each family – will be automatically trained on collections of individual models to produce highly predictive ensemble models which, in most cases, will be the top performing models in the AutoML Leaderboard.

H2O offers a number of model explainability methods that apply to AutoML objects (groups of models), as well as individual models (e.g. leader model). Explanations can be generated automatically with a single function call, providing a simple interface to exploring and explaining the AutoML models.

AutoML Interface

The H2O AutoML interface is designed to have as few parameters as possible so that all the user needs to do is point to their dataset, identify the response column and optionally specify a time constraint or limit on the number of total models trained.

In both the R and Python API, AutoML uses the same data-related arguments, `x`, `y`, `training_frame`, `validation_frame`, as the other H2O algorithms. Most of the time, all you'll need to do is specify the data arguments. You can then configure values for `max_runtime_secs` and/or `max_models` to set explicit time or number-of-model limits on your run.

Required Parameters

Required Data Parameters

- y: This argument is the name (or index) of the response column.
- training_frame: Specifies the training set.

Required Stopping Parameters

One of the following stopping strategies (time or number-of-model based) must be specified. When both options are set, then the AutoML run will stop as soon as it hits one of either of these limits.

- max_runtime_secs: This argument specifies the maximum time that the AutoML process will run for, prior to training the final Stacked Ensemble models. The default is 0 (no limit), but dynamically sets to 1 hour if none of `max_runtime_secs` and `max_models` are specified by the user.
- max_models: Specify the maximum number of models to build in an AutoML run, excluding the Stacked Ensemble models. Defaults to `NULL/None`.

Optional Parameters

Optional Data Parameters

- x: A list/vector of predictor column names or indexes. This argument only needs to be specified if the user wants to exclude columns from the set of predictors. If all columns (other than the response) should be used in prediction, then this does not need to be set.
- validation_frame: This argument is ignored unless `nfolds == 0`, in which a validation frame can be specified and used for early stopping of individual models and early stopping of the grid searches (unless `max_models` or `max_runtime_secs` overrides metric-based early stopping). By default and when `nfolds > 1`, cross-validation metrics will be used for early stopping and thus `validation_frame` will be ignored.
- **leaderboard_frame**: This argument allows the user to specify a particular data frame to use to score and rank models on the leaderboard. This frame will not be used for anything besides leaderboard scoring. If a leaderboard frame is not specified by the user, then the

leaderboard will use cross-validation metrics instead, or if cross-validation is turned off by setting `nfolds = 0`, then a leaderboard frame will be generated automatically from the training frame.

- blending_frame: Specifies a frame to be used for computing the predictions that serve as the training frame for the Stacked Ensemble models metalearner. If provided, all Stacked Ensembles produced by AutoML will be trained using Blending (a.k.a. Holdout Stacking) instead of the default Stacking method based on cross-validation.
- fold_column: Specifies a column with cross-validation fold index assignment per observation. This is used to override the default, randomized, 5-fold cross-validation scheme for individual models in the AutoML run.
- weights_column: Specifies a column with observation weights. Giving some observation a weight of zero is equivalent to excluding it from the dataset; giving an observation a relative weight of 2 is equivalent to repeating that row twice. Negative weights are not allowed.

Optional Miscellaneous Parameters

- nfolds: Specify a value >= 2 for the number of folds for k-fold cross-validation of the models in the AutoML run. This value defaults to 5. Use 0 to disable cross-validation; this will also disable Stacked Ensembles (thus decreasing the overall best model performance).
- balance_classes: Specify whether to oversample the minority classes to balance the class distribution. This option is not enabled by default and can increase the data frame size. This option is only applicable for classification. If the oversampled size of the dataset exceeds the maximum size calculated using the `max_after_balance_size parameter`, then the majority classes will be undersampled to satisfy the size limit.
- class_sampling_factors: Specify the per-class (in lexicographical order) over/under-sampling ratios. By default, these ratios are automatically computed during training to obtain the class balance. Note that this requires `balance_classes=true`.
- max_after_balance_size: Specify the maximum relative size of the training data after balancing class counts (**balance_classes** must be enabled). Defaults to 5.0. (The value can be less than 1.0).
- max_runtime_secs_per_model: Specify the max amount of time dedicated to the training of each individual model in the AutoML run. Defaults to 0 (disabled). Note that setting this parameter can affect AutoML reproducibility.
- stopping_metric: Specify the metric to use for early stopping. Defaults to `AUTO`. The available options are:

    - `AUTO`: This defaults to `logloss` for classification and `deviance` for regression.
    - `deviance` (mean residual deviance)
    - `logloss`
    - `MSE`
    - `RMSE`
    - `MAE`
    - `RMSLE`
    - `AUC` (area under the ROC curve)
    - `AUCPR` (area under the Precision-Recall curve)
    - `lift_top_group`
    - `misclassification`
    - `mean_per_class_error`

- stopping_tolerance: This option specifies the relative tolerance for the metric-based stopping criterion to stop a grid search and the training of individual models within the AutoML run. This value defaults to 0.001 if the dataset is at least 1 million rows; otherwise it defaults to a bigger value determined by the size of the dataset and the non-NA-rate. In that case, the value is computed as 1/sqrt(nrows * non-NA-rate).
- stopping_rounds: This argument is used to stop model training when the stopping metric (e.g. AUC) doesn't improve for this specified number of training rounds, based on a simple moving average. In the context of AutoML, this controls early stopping both within the random grid searches as well as the individual models. Defaults to 3 and must be an non-negative integer. To disable early stopping altogether, set this to 0.
- sort_metric: Specifies the metric used to sort the Leaderboard by at the end of an AutoML run. Available options include:

    - `AUTO`: This defaults to `AUC` for binary classification, `mean_per_class_error` for multinomial classification, and `deviance` for regression.
    - `deviance` (mean residual deviance)
    - `logloss`
    - `MSE`
    - `RMSE`
    - `MAE`
    - `RMSLE`
    - `AUC` (area under the ROC curve)
    - `AUCPR` (area under the Precision-Recall curve)
    - `mean_per_class_error`

- seed: Integer. Set a seed for reproducibility. AutoML can only guarantee reproducibility under certain conditions. H2O Deep Learning models are not reproducible by default for performance reasons, so if the user requires reproducibility, then `exclude_algos` must contain `"DeepLearning"`. In addition `max_models` must be used because `max_runtime_secs` is resource limited, meaning that if the available compute resources are not the same between runs, AutoML may be able to train more models on one run vs another. Defaults to `NULL/None`.

- **project_name**: Character string to identify an AutoML project. Defaults to `NULL/None`, which means a project name will be auto-generated based on the training frame ID. More models can be trained and added to an existing AutoML project by specifying the same project name in multiple calls to the AutoML function (as long as the same training frame is used in subsequent runs).
- exclude_algos: A list/vector of character strings naming the algorithms to skip during the model-building phase. An example use is `exclude_algos = ["GLM", "DeepLearning", "DRF"]` in Python or `exclude_algos = c("GLM", "DeepLearning", "DRF")` in R. Defaults to `None/NULL`, which means that all appropriate H2O algorithms will be used if the search stopping criteria allows and if the `include_algos` option is not specified. This option is mutually exclusive with `include_algos`. See `include_algos` below for the list of available options.
- include_algos: A list/vector of character strings naming the algorithms to include during the model-building phase. An example use is `include_algos = ["GLM", "DeepLearning", "DRF"]` in Python or `include_algos = c("GLM", "DeepLearning", "DRF")` in R. Defaults to `None/NULL`, which means that all appropriate H2O algorithms will be used if the search stopping criteria allows and if no algorithms are specified in `exclude_algos`. This option is mutually exclusive with `exclude_algos`. The available algorithms are:
    - `DRF` (This includes both the Random Forest and Extremely Randomized Trees (XRT) models. Refer to the Extremely Randomized Trees section in the DRF chapter and the histogram_type parameter description for more information.)
    - `GLM`
    - `XGBoost` (XGBoost GBM)
    - `GBM` (H2O GBM)
    - `DeepLearning` (Fully-connected multi-layer artificial neural network)
    - `StackedEnsemble`

- **modeling_plan**: The list of modeling steps to be used by the AutoML engine. (They may not all get executed, depending on other constraints.)
- **preprocessing**: The list of preprocessing steps to run. Only `["target_encoding"]` is currently supported. There is more information about how Target Encoding is automatically applied here. Experimental.
- **exploitation_ratio**: Specify the budget ratio (between 0 and 1) dedicated to the exploitation (vs exploration) phase. By default, the exploitation phase is disabled (exploitation_ratio=0) as this is still experimental; to activate it, it is recommended to try a ratio around 0.1. Note that the current exploitation phase only tries to fine-tune the best XGBoost and the best GBM found during exploration. Experimental.
- monotone_constraints: A mapping that represents monotonic constraints. Use +1 to enforce an increasing constraint and -1 to specify a decreasing constraint.

- keep_cross_validation_predictions: Specify whether to keep the predictions of the cross-validation predictions. This needs to be set to TRUE if running the same AutoML object for repeated runs because CV predictions are required to build additional Stacked Ensemble models in AutoML. This option defaults to FALSE.
- keep_cross_validation_models: Specify whether to keep the cross-validated models. Keeping cross-validation models may consume significantly more memory in the H2O cluster. This option defaults to FALSE.
- keep_cross_validation_fold_assignment: Enable this option to preserve the cross-validation fold assignment. Defaults to FALSE.
- **verbosity**: (Optional: Python and R only) The verbosity of the backend messages printed during training. Must be one of `"debug", "info", "warn"`. Defaults to `NULL/None` (client logging disabled).
- export_checkpoints_dir: Specify a directory to which generated models will automatically be exported.

Notes

If the user sets `nfolds == 0`, then cross-validation metrics will not be available to populate the leaderboard. In this case, we need to make sure there is a holdout frame (aka. the "leaderboard frame") to score the models on so that we can generate model performance metrics for the leaderboard. Without cross-validation, we will also require a validation frame to be used for early stopping on the models. Therefore, if either of these frames are not provided by the user, they will be automatically partitioned from the training data. If either frame is missing, 10% of the training data will be used to create a missing frame (if both are missing then a total of 20% of the training data will be used to create a 10% validation and 10% leaderboard frame).

`H2OAutoML` can interact with the `h2o.sklearn` module. The `h2o.sklearn` module exposes 2 wrappers for `H2OAutoML` (`H2OAutoMLClassifier` and `H2OAutoMLRegressor`), which expose the standard API familiar to `sklearn` users: `fit`, `predict`, `fit_predict`, `score`, `get_params`, and `set_params`. It accepts various formats as input data (H2OFrame, `numpy` array, `pandas` Dataframe) which allows them to be combined with pure `sklearn` components in pipelines. For an example using `H2OAutoML` with the `h2o.sklearn` module, click here.

Explainability

AutoML objects are fully supported though the H2O Model Explainability interface. A large number of multi-model comparison and single model (AutoML leader) plots can be generated automatically with a single call to `h2o.explain()`. We invite you to learn more at page linked above.

Code Examples

Training

Here's an example showing basic usage of the `h2o.automl()` function in *R* and the `H2OAutoML` class in *Python*. For demonstration purposes only, we explicitly specify the the *x* argument, even though on this dataset, that's not required. With this dataset, the set of predictors is all columns other than the response. Like other H2O algorithms, the default value of `x` is "all columns, excluding `y`", so that will produce the same result.

R Python

```r
library(h2o)
h2o.init()

# Import a sample binary outcome train/test set into H2O
train <- h2o.importFile("https://s3.amazonaws.com/erin-data/higgs/higgs_train_10k.csv")
test <- h2o.importFile("https://s3.amazonaws.com/erin-data/higgs/higgs_test_5k.csv")

# Identify predictors and response
y <- "response"
x <- setdiff(names(train), y)

# For binary classification, response should be a factor
train[, y] <- as.factor(train[, y])
test[, y] <- as.factor(test[, y])

# Run AutoML for 20 base models (limited to 1 hour max runtime by default)
aml <- h2o.automl(x = x, y = y,
                  training_frame = train,
                  max_models = 20,
                  seed = 1)

# View the AutoML Leaderboard
lb <- aml@leaderboard
print(lb, n = nrow(lb))  # Print all rows instead of default (6 rows)

#                                               model_id       auc   logloss
mean_per_class_error      rmse       mse
# 1      StackedEnsemble_AllModels_AutoML_20181210_150447 0.7895453 0.5516022
0.3250365 0.4323464 0.1869234
# 2   StackedEnsemble_BestOfFamily_AutoML_20181210_150447 0.7882530 0.5526024
0.3239841 0.4328491 0.1873584
# 3                      XGBoost_1_AutoML_20181210_150447 0.7846510 0.5575305
0.3254707 0.4349489 0.1891806
# 4          XGBoost_grid_1_AutoML_20181210_150447_model_4 0.7835232 0.5578542
0.3188188 0.4352486 0.1894413
# 5          XGBoost_grid_1_AutoML_20181210_150447_model_3 0.7830043 0.5596125
0.3250808 0.4357077 0.1898412
# 6                      XGBoost_2_AutoML_20181210_150447 0.7813603 0.5588797
0.3470738 0.4359074 0.1900153
# 7                      XGBoost_3_AutoML_20181210_150447 0.7808475 0.5595886
0.3307386 0.4361295 0.1902090
# 8                          GBM_5_AutoML_20181210_150447 0.7808366 0.5599029
0.3408479 0.4361915 0.1902630
# 9                          GBM_2_AutoML_20181210_150447 0.7800361 0.5598060
0.3399258 0.4364149 0.1904580
# 10                         GBM_1_AutoML_20181210_150447 0.7798274 0.5608570
0.3350957 0.4366159 0.1906335
# 11                         GBM_3_AutoML_20181210_150447 0.7786685 0.5617903
0.3255378 0.4371886 0.1911339
# 12         XGBoost_grid_1_AutoML_20181210_150447_model_2 0.7744105 0.5750165
0.3228112 0.4427003 0.1959836
# 13                         GBM_4_AutoML_20181210_150447 0.7714260 0.5697120
0.3374203 0.4410703 0.1945430
# 14             GBM_grid_1_AutoML_20181210_150447_model_1 0.7697524 0.5725826
0.3443314 0.4424524 0.1957641
# 15             GBM_grid_1_AutoML_20181210_150447_model_2 0.7543664 0.9185673
0.3558550 0.4966377 0.2466490
# 16                         DRF_1_AutoML_20181210_150447 0.7428924 0.5958832
0.3554027 0.4527742 0.2050045
```

```
# 17                          XRT_1_AutoML_20181210_150447 0.7420910 0.5993457
0.3565826 0.4531168 0.2053148
# 18  DeepLearning_grid_1_AutoML_20181210_150447_model_2 0.7388505 0.6012286
0.3695292 0.4555318 0.2075092
# 19       XGBoost_grid_1_AutoML_20181210_150447_model_1 0.7257836 0.6013126
0.3820490 0.4565541 0.2084417
# 20                 DeepLearning_1_AutoML_20181210_150447 0.6979292 0.6339217
0.3979403 0.4692373 0.2201836
# 21  DeepLearning_grid_1_AutoML_20181210_150447_model_1 0.6847773 0.6694364
0.4081802 0.4799664 0.2303678
# 22           GLM_grid_1_AutoML_20181210_150447_model_1 0.6826481 0.6385205
0.3972341 0.4726827 0.2234290
#
# [22 rows x 6 columns]

# The leader model is stored here
aml@leader
```

The code above is the quickest way to get started, and the example will be referenced in the sections that follow. To learn more about H2O AutoML we recommend taking a look at our more in-depth AutoML tutorial (available in R and Python).

Prediction

Using the `predict()` function with AutoML generates predictions on the leader model from the run. The order of the rows in the results is the same as the order in which the data was loaded, even if some rows fail (for example, due to missing values or unseen factor levels).

Using the previous code example, you can generate test set predictions as follows:

R   Python

```
# To generate predictions on a test set, you can make predictions
# directly on the `"H2OAutoML"` object or on the leader model
# object directly
pred <- h2o.predict(aml, test)  # predict(aml, test) also works

# or:
pred <- h2o.predict(aml@leader, test)
```

AutoML Output

Leaderboard

The AutoML object includes a “leaderboard” of models that were trained in the process, including the 5-fold cross-validated model performance (by default). The number of folds used in the model evaluation process can be adjusted using the `nfolds` parameter. If you would like to score the models on a specific dataset, you can specify the `leaderboard_frame` argument in the AutoML run, and then the leaderboard will show scores on that dataset instead.

The models are ranked by a default metric based on the problem type (the second column of the leaderboard). In binary classification problems, that metric is AUC, and in multiclass classification problems, the metric is mean per-class error. In regression problems, the default sort metric is deviance. Some additional metrics are also provided, for convenience.

To help users assess the complexity of `AutoML` models, the `h2o.get_leaderboard` function has been been expanded by allowing an `extra_columns` parameter. This parameter allows you to specify which (if any) optional columns should be added to the leaderboard. This defaults to None. Allowed options include:

- `training_time_ms`: A column providing the training time of each model in milliseconds. (Note that this doesn’t include the training of cross validation models.)
- `predict_time_per_row_ms`: A column providing the average prediction time by the model for a single row.
- `ALL`: Adds columns for both training_time_ms and predict_time_per_row_ms.

Using the previous example, you can retrieve the leaderboard as follows:

R   Python

```
# Get leaderboard with 'extra_columns = 'ALL'
lb <- h2o.get_leaderboard(object = aml, extra_columns = 'ALL')
lb
```

Here is an example of a basic leaderboard (no extra columns) for a binary classification task:

model_id	auc	logloss
StackedEnsemble_AllModels_AutoML_20191213_174603	0.789844	0.551067
StackedEnsemble_BestOfFamily_AutoML_20191213_174603	0.789768	0.550906
XGBoost_grid__1_AutoML_20191213_174603_model_4	0.784698	0.55681
XGBoost_3_AutoML_20191213_174603	0.784232	0.557749

model_id	auc	logloss
XGBoost_2_AutoML_20191213_174603	0.783533	0.555997
XGBoost_grid__1_AutoML_20191213_174603_model_3	0.782582	0.560218
GBM_5_AutoML_20191213_174603	0.78219	0.558353
XGBoost_1_AutoML_20191213_174603	0.781901	0.557944
XGBoost_grid__1_AutoML_20191213_174603_model_1	0.781648	0.561112
GBM_2_AutoML_20191213_174603	0.777673	0.562514
GBM_1_AutoML_20191213_174603	0.777294	0.562744
GBM_3_AutoML_20191213_174603	0.775488	0.564794
XGBoost_grid__1_AutoML_20191213_174603_model_2	0.773621	0.578141
GBM_grid__1_AutoML_20191213_174603_model_1	0.772656	0.568314
GBM_4_AutoML_20191213_174603	0.77248	0.569483
DRF_1_AutoML_20191213_174603	0.764975	0.5801
XRT_1_AutoML_20191213_174603	0.759957	0.585158
GBM_grid__1_AutoML_20191213_174603_model_2	0.748007	0.632981
DeepLearning_grid__2_AutoML_20191213_174603_model_1	0.739884	0.600688
DeepLearning_1_AutoML_20191213_174603	0.700406	0.63169
DeepLearning_grid__1_AutoML_20191213_174603_model_1	0.692235	0.671512
GLM_1_AutoML_20191213_174603	0.682648	0.63852

AutoML Log

When using Python or R clients, you can also access meta information with the following AutoML object properties:

- **event_log**: an `H2OFrame` with selected AutoML backend events generated during training.
- **training_info**: a dictionary exposing data that could be useful for post-analysis; for example various timings.

Experimental Features

Preprocessing

As of H2O 3.32.0.1, AutoML now has a `preprocessing` option with minimal support for automated Target Encoding of high cardinality categorical variables. The only currently supported option is `preprocessing = ["target_encoding"]`: we automatically tune a Target Encoder model and apply it to columns that meet certain cardinality requirements for the tree-based algorithms (XGBoost, H2O GBM and Random Forest). Work to improve the automated preprocessing support (improved model performance as well as customization) is documented in this ticket.

FAQ

- **Which models are trained in the AutoML process?**

  The current version of AutoML trains and cross-validates the following algorithms (in the following order): three pre-specified XGBoost GBM (Gradient Boosting Machine) models, a fixed grid of GLMs, a default Random Forest (DRF), five pre-specified H2O GBMs, a near-default Deep Neural Net, an Extremely Randomized Forest (XRT), a random grid of XGBoost GBMs, a random grid of H2O GBMs, and a random grid of Deep Neural Nets. In some cases, there will not be enough time to complete all the algorithms, so some may be missing from the leaderboard. AutoML then trains two Stacked Ensemble models (more info about the ensembles below). Particular algorithms (or groups of algorithms) can be switched off using the `exclude_algos` argument. This is useful if you already have some idea of the algorithms that will do well on your dataset, though sometimes this can lead to a loss of performance because having more diversity among the set of models generally increases the performance of the Stacked Ensembles. As a recommendation, if you have really wide (10k+ columns) and/or sparse data, you may consider skipping the tree-based algorithms (GBM, DRF, XGBoost).

  A list of the hyperparameters searched over for each algorithm in the AutoML process is included in the appendix below. More details about the hyperparameter ranges for the models in addition to the hard-coded models will be added to the appendix at a later date.

  Both of the ensembles should produce better models than any individual model from the AutoML run with the exception of some rare cases. One ensemble contains all the models, and the second ensemble contains just the best performing model from each algorithm class/family. The “Best of Family” ensemble is optimized for production use since it only contains six (or fewer) base models. It should be relatively fast to use (to generate predictions on new data) without much degradation in model performance when compared to the “All Models” ensemble. The metalearner in both ensembles is a variant of the default Stacked Ensemble metalearner: a non-negative GLM with regularization (Lasso or Elastic net, chosen by CV) to encourage more sparse ensembles. The metalearner also uses a logit transform (on the base learner CV preds) for classification tasks before training.

- **How do I save AutoML runs?**

Rather than saving an AutoML object itself, currently, the best thing to do is to save the models you want to keep, individually. A utility for saving all of the models at once, along with a way to save the AutoML object (with leaderboard), will be added in a future release.

- **Can we make use of GPUs with AutoML?**

  XGBoost models in AutoML can make use of GPUs. Keep in mind that the following requirements must be met:

  - NVIDIA GPUs (GPU Cloud, DGX Station, DGX-1, or DGX-2)
  - CUDA 8

  You can monitor your GPU utilization via the `nvidia-smi` command. Refer to https://developer.nvidia.com/nvidia-system-management-interface for more information.

- **Why don't I see XGBoost models?**

  AutoML includes XGBoost GBMs (Gradient Boosting Machines) among its set of algorithms. This feature is currently provided with the following restrictions:

  - XGBoost is not available on Windows machines.
  - XGBoost is used only if it is available globally and if it hasn't been explicitly disabled. You can check if XGBoost is available by using the `h2o.xgboost.available()` in R or `h2o.estimators.xgboost.H2OXGBoostEstimator.available()` in Python.

- **Why doesn't AutoML use all the time that it's given?**

  AutoML has a `max_runtime_secs` parameter, which is a limit on the total runtime. However, early stopping is also enabled by default: AutoML will stop once there's no longer "enough" incremental improvement. The user can tweak the early stopping paramters to be more or less sensitive. Set `stopping_rounds` higher if you want to slow down early stopping and let AutoML train more models before it stops. In a future release, we are planning to allow the user to specify an exact runtime, rather than just a maximum runtime.

Resources

- AutoML Tutorial (R and Python notebooks)
- Intro to AutoML + Hands-on Lab (1 hour video) (slides)
- Scalable Automatic Machine Learning in H2O (1 hour video) (slides)
- AutoML Roadmap

Citation

If you're citing the H2O AutoML algorithm in a paper, please cite our paper from the 7th ICML Workshop on Automated Machine Learning (AutoML). A formatted version of the citation would look like this:

Erin LeDell and Sebastien Poirier. *H2O AutoML: Scalable Automatic Machine Learning*. 7th ICML Workshop on Automated Machine Learning (AutoML), July 2020. URL https://www.automl.org/wp-content/uploads/2020/07/AutoML_2020_paper_61.pdf.

If you are using Bibtex:

```
@article{H2OAutoML20,
    title = {{H2O} {A}uto{ML}: Scalable Automatic Machine Learning},
    author = {Erin LeDell and Sebastien Poirier},
    year = {2020},
    month = {July},
    journal = {7th ICML Workshop on Automated Machine Learning (AutoML)},
    url = {https://www.automl.org/wp-content/uploads/2020/07/AutoML_2020_paper_61.pdf},
}
```

The H2O AutoML algorithm was first released in H2O 3.12.0.1 on June 6, 2017. If you need to cite a particular version of the H2O AutoML algorithm, you can use an additional citation (using the appropriate version replaced below) as follows:

```
@Manual{H2OAutoML_33212,
    title = {{H2O} {A}uto{ML}},
    author = {H2O.ai},
    year = {2021},
    note = {H2O version 3.32.1.2},
    url = {http://docs.h2o.ai/h2o/latest-stable/h2o-docs/automl.html},
}
```

Information about how to cite the H2O software in general is covered in the H2O FAQ.

Random Grid Search Parameters

AutoML performs a hyperparameter search over a variety of H2O algorithms in order to deliver the best model. In the table below, we list the hyperparameters, along with all potential values that can be randomly chosen in the search. If these models also have a non-default value set for a hyperparameter, we identify it in the list as well. Random Forest and Extremely Randomized Trees are not grid searched (in the current version of AutoML), so they are not included in the list below.

Note: AutoML does not run a standard grid search for GLM (returning all the possible models). Instead AutoML builds a single model with `lambda_search` enabled and passes a list of `alpha` values. It returns only the model with the best alpha-lambda combination rather than one model for each alpha-lambda combination.

GLM Hyperparameters

This table shows the GLM values that are searched over when performing AutoML grid search. Additional information is available here.

Note: GLM uses its own internal grid search rather than the H2O Grid interface. For GLM, AutoML builds a single model with `lambda_search` enabled and passes a list of `alpha` values. It returns a single model with the best alpha-lambda combination rather than one model for each alpha.

Parameter	Searchable Values
`alpha`	`{0.0, 0.2, 0.4, 0.6, 0.8, 1.0}`

XGBoost Hyperparameters

This table shows the XGBoost values that are searched over when performing AutoML grid search. Additional information is available here.

Parameter	Searchable Values
`booster`	`gbtree`, `dart`
`col_sample_rate`	`{0.6, 0.8, 1.0}`
`col_sample_rate_per_tree`	`{0.7, 0.8, 0.9, 1.0}`
`max_depth`	`{5, 10, 15, 20}`
`min_rows`	`{0.01, 0.1, 1.0, 3.0, 5.0, 10.0, 15.0, 20.0}`
`ntrees`	Hard coded: `10000` (true value found by early stopping)
`reg_alpha`	`{0.001, 0.01, 0.1, 1, 10, 100}`
`reg_lambda`	`{0.001, 0.01, 0.1, 0.5, 1}`
`sample_rate`	`{0.6, 0.8, 1.0}`

GBM Hyperparameters

This table shows the GLM values that are searched over when performing AutoML grid search. Additional information is available here.

Parameter	Searchable Values
`col_sample_rate`	`{0.4, 0.7, 1.0}`
`col_sample_rate_per_tree`	`{0.4, 0.7, 1.0}`
`learn_rate`	Hard coded: `0.1`
`max_depth`	`{3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17}`
`min_rows`	`{1, 5, 10, 15, 30, 100}`
`min_split_improvement`	`{1e-4, 1e-5}`
`ntrees`	Hard coded: `10000` (true value found by early stopping)
`sample_rate`	`{0.50, 0.60, 0.70, 0.80, 0.90, 1.00}`

Deep Learning Hyperparameters

This table shows the Deep Learning values that are searched over when performing AutoML grid search. Additional information is available here.

Parameter	Searchable Values
`activation`	Hard coded: `RectifierWithDropout`
`epochs`	Hard coded: `10000` (true value found by early stopping)
`epsilon`	`{1e-6, 1e-7, 1e-8, 1e-9}`
`hidden`	• Grid search 1: `{20}, {50}, {100}` • Grid search 2: `{20, 20}, {50, 50}, {100, 100}` • Grid search 3: `{20, 20, 20}, {50, 50, 50}, {100, 100, 100}`
	• Grid search 1: `{0.1}, {0.2}, {0.3}, {0.4}, {0.5}` • Grid search 2: `{0.1, 0.1}, {0.2, 0.2}, {0.3, 0.3}, {0.4, 0.4}, {`

Parameter	Searchable Values
`hidden_dropout_ratios`	• Grid search 3: `{0.1, 0.1, 0.1}, {0.2, 0.2, 0.2} {0.3, 0.3, 0.3},`
`input_dropout_ratio`	`{0.0, 0.05, 0.1, 0.15, 0.2}`
`rho`	`{0.9, 0.95, 0.99}`

Additional Information

AutoML development is tracked here. This page lists all open or in-progress AutoML JIRA tickets.