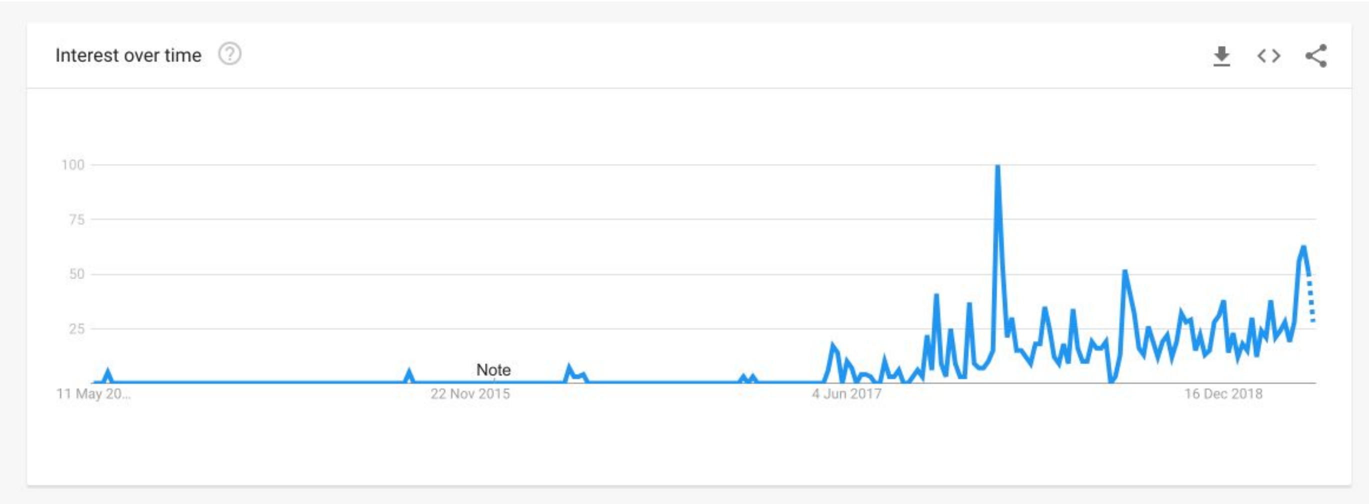
# AutoML with h2o

## Parameters and Model Optimization

6 minute read

The interest in AutoML is rising over time. This graph shows the trends in Google for the AutoML search term.




AutoML algorithms are reaching really good rankings in data science competitions (see [this article](https://towardsdatascience.com/achieving-a-top-5-position-in-an-ml-competition-with-automl-89a5a6fb8060) (https://towardsdatascience.com/achieving-a-top-5-position-in-an-ml-competition-with-automl-89a5a6fb8060))

But *what is* AutoML ? How does it work? And mainly, how can you implement an AutoML in Python?

# What is AutoML?

AutoML is a framework whose role is to optimize the machine learning workflow, which includes automatic training and tuning of many models within a user-specified time-limit.

The fasten the work of the Data Scientist when it comes to model selection and p . On the other hand, the user simply inputs the training data, eventually some v nd a time limit.

AutoML will automatically try several models, choose the best performing models, tune the parameters of the *leader* models, try to stack them…

AutoML outputs a leaderboard of algorithms, and you can select the best performing algorithm given several criteria that are measured (MSE, RMSE, log loss, Auc…).

# Why and when should you use AutoML?

Building models and tuning the hyperparameters is a long process for any data scientist. The search space for the optimal parameters is enormous, and this is only for 1 chosen model.

AutoML can be highly parallelized, so bear in mind that a couple of GPUs will help.

AutoML can be used to :

- Assess the feature importance
- Try a lot of models and parameters as a first guess

Once a model and a set of parameters have been identified, you have 2 options :

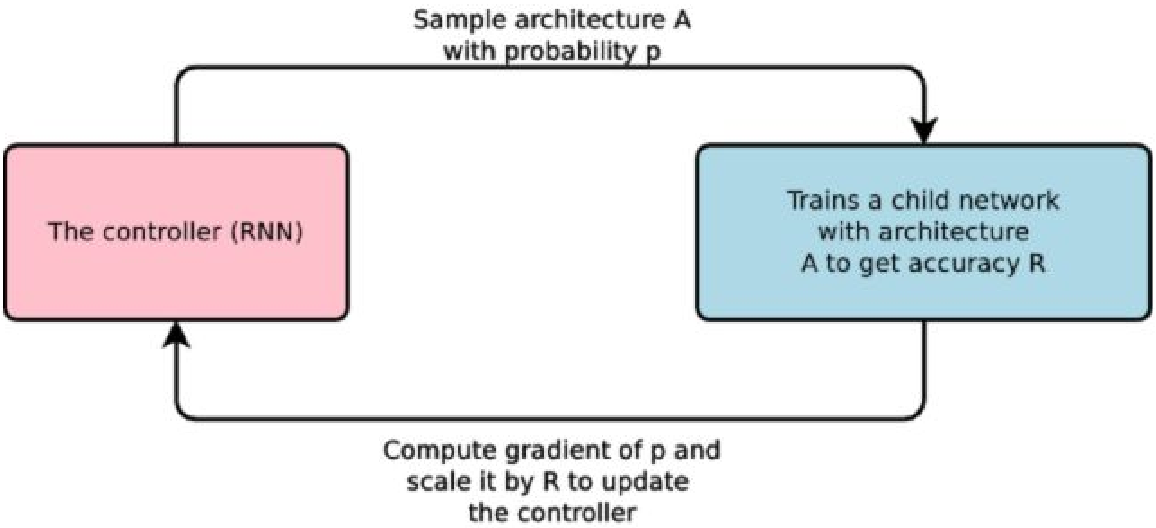
- either the model is good enough and satisfies your criteria
- or you can use the selected set of model + parameters as a starting point for a GridSearch or Bayesian HyperOpt

# How does AutoML work?

AutoML **does not** use a GIANT double for-loop to test every model and every parameter. It's much smarter than that. It uses Reinforcement Learning.

A controller neural net can propose a "child" model architecture, which can then be trained and evaluated for quality on a particular task. That feedback is then used to inform the controller how to improve its proposals for the next round.

Eventually, the controller learns to assign a high probability to areas of architecture space that achieve better accuracy on a held-out validation dataset, and low probability to areas of architecture space that score poorly.




To make the controller a little more complex, it uses anchor points, and set-selection attention to form skip connections.

At that point, you might think that AutoML frameworks are extremely long to run. In AutoML, each gradient update to the controller parameters θ corresponds to training one child network to convergence.

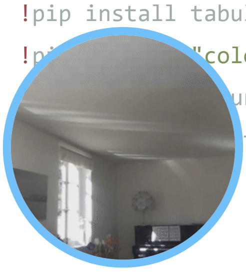
You're right, training a single child network can take hours. For this reason, according to Google's Blog, AutoML uses distributed training and asynchronous parameter updates to speed up the learning process of the controller. It uses a parameter-server scheme where we have a parameter server of S shards, that store the shared parameters for K controller replicas. Each controller replica samples m different child architectures that are trained in parallel. The controller then collects gradients according to the results of that minibatch of m architectures at convergence and sends them to the parameter server to update the weights across all controller replicas.

# Example in Python

Several companies are currently AutoML pipelines. Among them, Google and h2o. In this example, we'll use h2o's solution. I suggest you run this in Google Colab using GPU's, but you can also run it locally.

Start by importing the necessary packages :

```
!pip install requests
!pip install tabulate
!pi              "colorama>=0.3.8"
                ure
                 ttp://h2o-release.s3.amazonaws.com/h2o/latest_stable_Py.html h2o
```



# The Data

We'll use the Credit Card Fraud detection, a famous Kaggle dataset that can be found here (https://www.kaggle.com/mlg-ulb/creditcardfraud).

The dataset contains transactions made by credit cards in September 2013 by European cardholders. This dataset presents transactions that occurred in two days, where we have 492 frauds out of 284,807 transactions. The dataset is highly unbalanced, the positive class (frauds) account for 0.172% of all transactions.

It contains only numerical input variables which are the result of a PCA transformation. Unfortunately, due to confidentiality issues, the original features are not provided. Features V1, V2, ... V28 are the principal components obtained with PCA, the only features which have not been transformed with PCA are 'Time' and 'Amount'. Feature 'Time' contains the seconds elapsed between each transaction and the first transaction in the dataset. The feature 'Amount' is the transaction Amount, this feature can be used for example-dependant cost-sensitive learning. Feature 'Class' is the response variable and it takes value 1 in case of fraud and 0 otherwise.

```
### General
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```
df = pd.read_csv('creditcard.csv')
df.head()
```

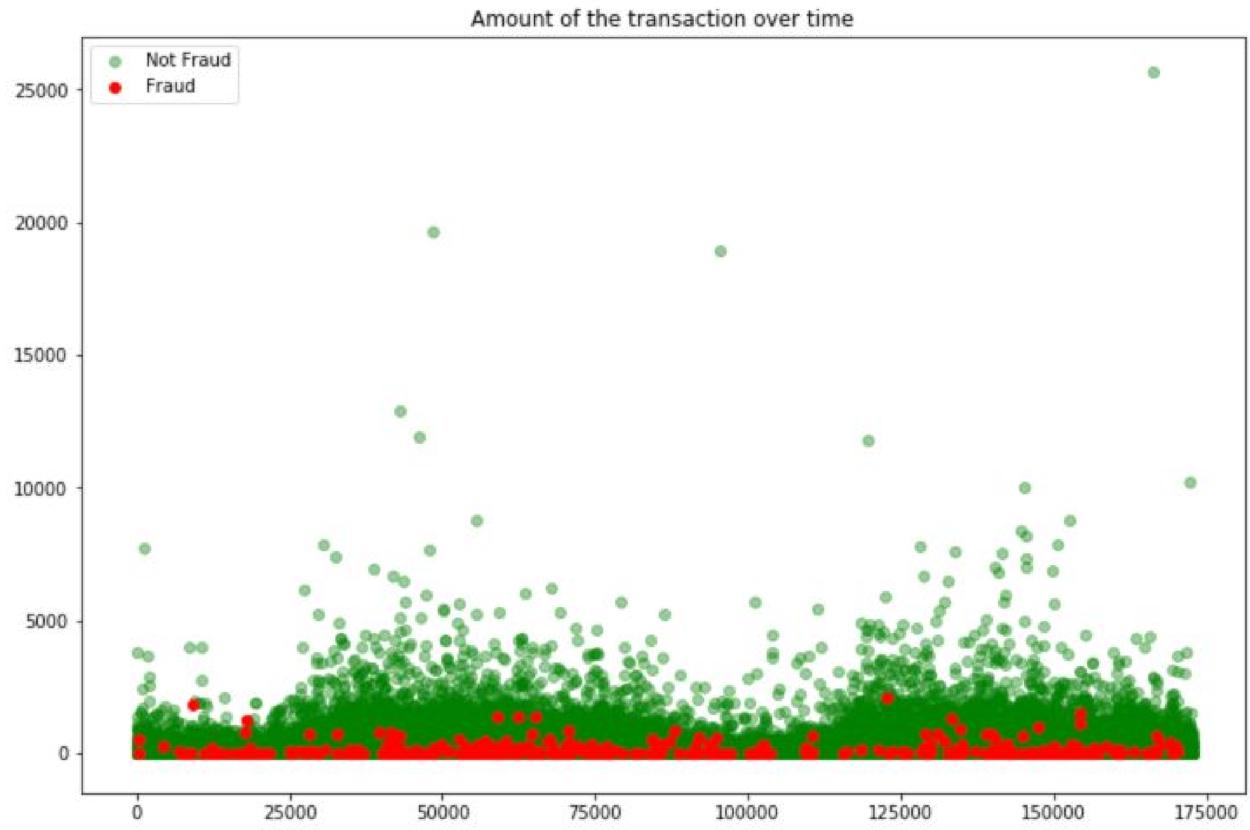
Out[96]:

| V5 | V6 | V7 | V8 | V9 | ... | V21 | V22 | V23 | V24 | V25 | V26 | V27 | V28 | Amount | Class |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| -0.338321 | 0.462388 | 0.239599 | 0.098698 | 0.363787 | ... | -0.018307 | 0.277838 | -0.110474 | 0.066928 | 0.128539 | -0.189115 | 0.133558 | -0.021053 | 149.62 | 0 |
| 0.060018 | -0.082361 | -0.078803 | 0.085102 | -0.255425 | ... | -0.225775 | -0.638672 | 0.101288 | -0.339846 | 0.167170 | 0.125895 | -0.008983 | 0.014724 | 2.69 | 0 |
| -0.503198 | 1.800499 | 0.791461 | 0.247676 | -1.514654 | ... | 0.247998 | 0.771679 | 0.909412 | -0.689281 | -0.327642 | -0.139097 | -0.055353 | -0.059752 | 378.66 | 0 |
| -0.010309 | 1.247203 | 0.237609 | 0.377436 | -1.387024 | ... | -0.108300 | 0.005274 | -0.190321 | -1.175575 | 0.647376 | -0.221929 | 0.062723 | 0.061458 | 123.50 | 0 |
| -0.407193 | 0.095921 | 0.592941 | -0.270533 | 0.817739 | ... | -0.009431 | 0.798278 | -0.137458 | 0.141267 | -0.206010 | 0.502292 | 0.219422 | 0.215153 | 69.99 | 0 |

If you explore the data, you'll notice that only 0.17% of the transactions are fraudulent. We'll use the F1-Score metric, a harmonic mean between the precision and the recall.

To understand the nature of the fraudulant transactions, simply plot the following graph :

```
plt.figure(figsize=(12,8))
plt.scatter(df[df.Class == 0].Time, df[df.Class == 0].Amount, c='green', alpha=0.4, label="Not Fraud")
plt.scatter(df[df.Class == 1].Time, df[df.Class == 1].Amount, c='red', label="Fraud")
plt.title("Amount of the transaction over time")
plt.legend()
plt.show()
```




Fraudulent transactions have a limited amount. We can guess that these transactions must remain "unseen" and not attracting too much attention.

# h2o AutoML

Now, let's import h2o AutoML :

```
### h2o AutoML
import h2o
from h2o.estimators.gbm import H2OGradientBoostingEstimator
from h2o.automl import H2OAutoML
```

T h2o session :

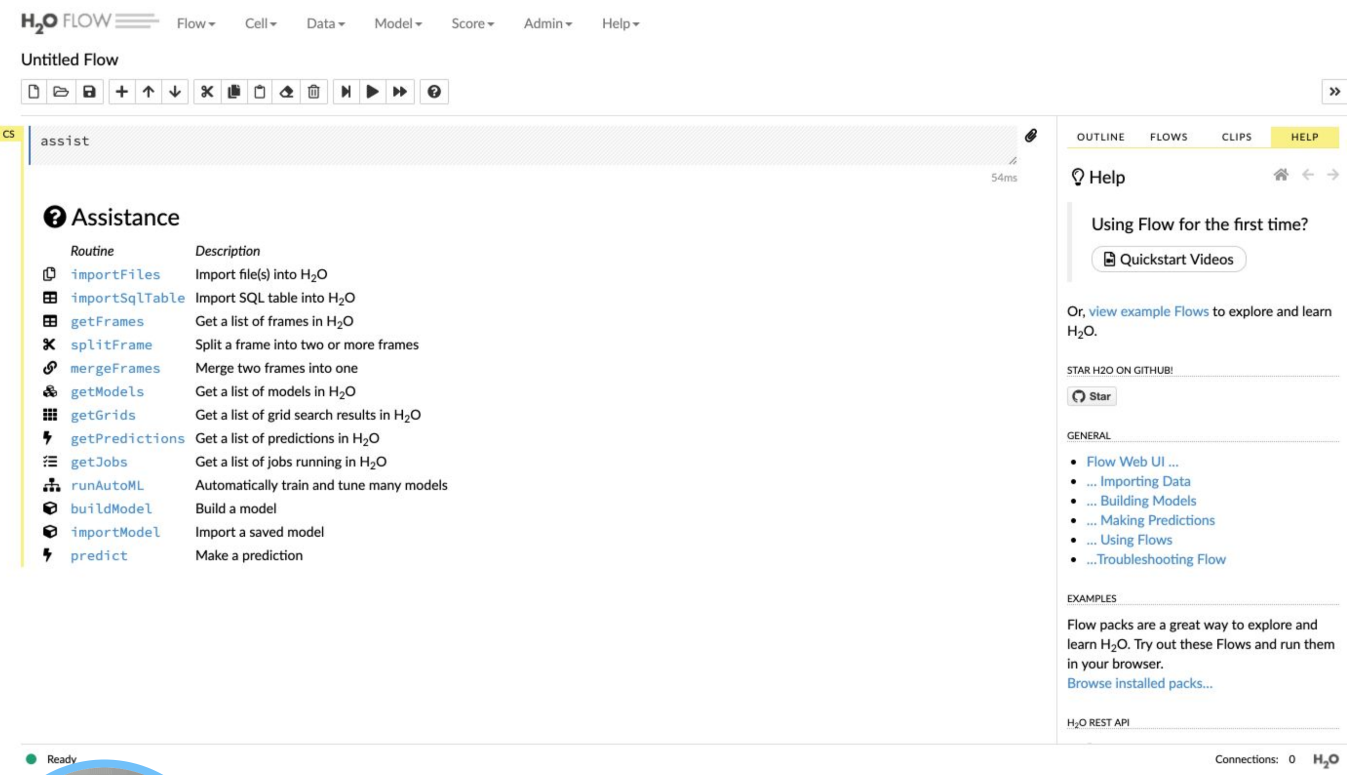
```
h2o.
```

If you're running this locally, you should see something like this :

```
Checking whether there is an H2O instance running at http://localhost:54321 . connected.
```

| | |
|---|---|
| H2O cluster uptime: | 05 secs |
| H2O cluster timezone: | Europe/Paris |
| H2O data parsing timezone: | UTC |
| H2O cluster version: | 3.24.0.2 |
| H2O cluster version age: | 18 days |
| H2O cluster name: | H2O_from_python_maelfabien_z2x0sv |
| H2O cluster total nodes: | 1 |
| H2O cluster free memory: | 3.546 Gb |
| H2O cluster total cores: | 4 |
| H2O cluster allowed cores: | 4 |
| H2O cluster status: | locked, healthy |
| H2O connection url: | http://localhost:54321 |
| H2O connection proxy: | None |
| H2O internal security: | False |
| H2O API Extensions: | Amazon S3, XGBoost, Algos, AutoML, Core V3, Core V4 |
| Python version: | 3.6.5 final |

If you follow the local link to the instance, you can access the h2o Flow :




I'[...]e Flow in another article, but Flow aims to do the same thing with a visual i[...] you need to import the dataset as an h2o object, and use built-in functions to sp[...]me :

```
# Load the data
df = h2o.import_file("/Users/maelfabien/Desktop/LocalDB/CreditCard/creditcard.csv")

d = df.split_frame(ratios = [0.8], seed = 1234)
df_train = d[0] # using 80% for training
df_test = d[1] #rest 20% for testing
```

We then define a list of the columns we'll use as predictors :

```
# Predictor columns
predictors = list(df.columns)
predictors.remove('Time')
predictors.remove('Class')
```

As you might have guessed, we're facing a binary classification problem here. The default case is regression in AutoML. To "cast" a column type to integer, use this :

```
# Cast binary
df_train['Class'] = df_train['Class'].asfactor()
```

We are now ready to define the model and train it. We specify the maximal number of models to test, and the overall maximal runtime in seconds.

```
aml = H2OAutoML(max_models = 50, seed = 1, max_runtime_secs=21000)
aml.train(x = predictors, y = 'Class', training_frame = df_train, validation_frame = df_test)
```

By default, the maximal runtime is 1 hour. Your model will be training for 21'000 seconds now (I left it to train overnight). Now, let's display all the models that have been tested and their performance :

```
print(aml.leaderboard)
```

| model_id | auc | logloss | mean_per_class_error | rmse | mse |
|---|---|---|---|---|---|
| GBM_grid_1_AutoML_20190506_000950_model_8 | 0.981646 | 0.00279929 | 0.105491 | 0.0217763 | 0.000474206 |
| GBM_grid_1_AutoML_20190506_000950_model_13 | 0.97855 | 0.00278203 | 0.106747 | 0.0212029 | 0.000449564 |
| GLM_grid_1_AutoML_20190506_000950_model_1 | 0.977505 | 0.00403073 | 0.111985 | 0.0260326 | 0.000677699 |
| GBM_grid_1_AutoML_20190506_000950_model_10 | 0.97727 | 0.0027134 | 0.0977369 | 0.0202304 | 0.000409271 |
| GBM_grid_1_AutoML_20190506_000950_model_16 | 0.976829 | 0.00273863 | 0.0977566 | 0.0208157 | 0.000433294 |
| [illegible]id_1_AutoML_20190506_000950_model_5 | 0.969094 | 0.00313117 | 0.100327 | 0.0217719 | 0.000474017 |
| [illegible]AutoML_20190506_000950_model_17 | 0.963728 | 0.00304207 | 0.101602 | 0.0220067 | 0.000484294 |
| [illegible]AutoML_20190506_000950_model_6 | 0.95929 | 0.00727195 | 0.105513 | 0.0240526 | 0.000578527 |
| [illegible]_AutoML_20190506_000950_model_4 | 0.954595 | 0.00835674 | 0.114497 | 0.0240522 | 0.00057851 |
| D[illegible]_1_AutoML_20190506_000950_model_3 | 0.952148 | 0.0103404 | 0.106811 | 0.0246377 | 0.000607014 |

The leaderboard is established using Cross Validation, which more or less guarantees that the top performing models are indeed consistently performing well.

To display only the best model, use `print(aml.leader)`.

We can now make a prediction using the leader model, simply using:

```
aml.leader.predict(new_data)
```

We can then save the best model :

```
h2o.save_model(aml.leader, path = "./model_credit_card")
```

Once your work is over, shut down the session :

```
h2o.shutdown()
```

In this simple example, h2o outperformed the tuning I manually did.

> ***Conclusion*** *: I hope this article on AutoML was interesting. It's a really hot topic, and I do expect large improvements to be made over the next years in this field.*

Sources :

- How AutoML works (https://medium.com/@gangele397/how-does-automl-works-b0f9e45fbb24)
- Google AI Blog (https://ai.googleblog.com/2017/05/using-machine-learning-to-explore.html?m=1)
- H2o package exercises (http://eric.univ-lyon2.fr/~ricco/tanagra/fichiers/fr_Tanagra_Package_H2O_Python.pdf)



chinelearning

2019

#### LEAVE A COMMENT

cross-validation.

Best,
Márcio

Reply • Share ›

**Maël Fabien** Mod → Márcio Basgalupp • 2 years ago

Hi Márcio,
You are right ! I'll update the article accordingly, I never noticed this mistake. Thanks :)

Maël

Reply • Share ›