In last few years, AutoML or automated machine learning as become widely popular among data science community. Big tech giants like Google, Amazon and Microsoft have started offering AutoML tools. There is still a split among data scientists when it comes to AutoML. Some fear that it is going to be a threat to their jobs and others believe that there is a bigger risk than a job; might cost the company itself. Others see it as a tool that they could use for non-critical tasks or for presenting proof-of-concepts. In-arguably, it has definitely made its mark among the data science community.

If you don’t know what AutoML is, a quick google search will give you a good intro to AutoML. According to wikipedia “**Automated machine learning** (**AutoML**) is the process of automating the process of applying machine learning to real-world problems. AutoML covers the complete pipeline from the raw dataset to the deployable machine learning model”

In this blog post, I will give my take on AutoML and introduce to few frameworks in R and Python.

**Pro’s**

* Time saving: It’s a quick and dirty prototyping tool. If you are not working on critical task, you could use AutoML to do the job for you while you focus on more critical tasks.
* Benchmarking: Building an ML/DL model is fun. But, how do you know the model you have is the best? You either have to spend a lot of time in building iterative models or ask your colleague to build one and compare it. The other option is to use AutoML to benchmark yours.

**Con’s**

* Most AI models that we come across are black box. Similar is the case with these AutoML frameworks. If you don’t understand what you are doing, it could be catastrophic.
* Based on my previous point, AutoML is being marketed as a tool for non-data scientists. This is a bad move. Without understanding how a model works and blindly using it for making decisions could be disastrous.

Personally, I do use AutoML frameworks for day-to-day tasks. It helps me save time and understand the techniques and tuning parameters behind these frameworks.

Now, let me introduce you to some of the top open source AutoML frame works I have come across.

**H2O**

H2O definitely goes on the top of the list. They offer ML, deep learning and stacked ensemble models in their frame work. Although it is written in java, they offer connectors for R and Python through API’s. The best feature that I have almost never seen is the “stopping time”, where I can set how long I want to train my model. Below is the code for running in R and Python for Iris data set.

**R**

# Load library

library(h2o)

# start h2o cluster

invisible(h2o.init())

# convert data as h2o type

train\_h = as.h2o(train)

test\_h = as.h2o(test)

# set label type

y = 'Species'

pred = setdiff(names(train), y)

#convert variables to factors

train[,y] = as.factor(train[,y])

test[,y] = as.factor(test[,y])

# Run AutoML for 20 base models

aml = h2o.automl(x = pred, y = y,

training\_frame = train\_h,

max\_models = 20,

seed = 1,

max\_runtime\_secs = 20

)

# AutoML Leaderboard

lb = aml@leaderboard

lb

# prediction result on test data

prediction = h2o.predict(aml@leader, test\_h[,-5]) %>%

as.data.frame()

# create a confusion matrix

caret::confusionMatrix(test$Species, prediction$predict)

# close h2o connection

h2o.shutdown(prompt = F)

**Python**

# load python libraries

import h2o

from h2o.automl import H2OAutoML

import pandas as pd

# start cluster

h2o.init()

# convert to h2o frame

traindf = h2o.H2OFrame(r.train)

testdf = h2o.H2OFrame(r.test)

y = "Species"

x = list(traindf.columns)

x.remove(y)

# create df to factors

traindf[y] = traindf[y].asfactor()

testdf[y] = testdf[y].asfactor()

#run automl

aml = H2OAutoML(max\_runtime\_secs = 60)

aml.train(x = x, y = y, training\_frame = traindf)

# view leader board

aml.leaderboard

# do pridiction and convert it to a data frame

predict = aml.predict(testdf)

p = predict.as\_data\_frame()

# convert to pandas dataframe

data = {'actual': r.test.Species, 'Ypredict': p['predict'].tolist()}

df = pd.DataFrame(data, columns = ['actual','Ypredict'])

# create a confusion matrix and print results

confusion\_matrix = pd.crosstab(df['actual'], df['Ypredict'], rownames=['Actual'], colnames=['Predicted'])

print (confusion\_matrix)

# close h2o connection

h2o.shutdown(prompt = False)

**automl Package in R**

The automl package is availabe on CRAN. The automl package fits from simple regression to highly customizable deep neural networks either with gradient descent or metaheuristic, using automatic hyper parameters tuning and custom cost function. A mix inspired by the common tricks on Deep Learning and Particle Swarm Optimization. Below is a sample code for how to use in R.

library(automl)

amlmodel = automl\_train\_manual(Xref = subset(train, select = -c(Species)),

Yref = subset(train, select = c(Species))$Species

%>% as.numeric(),

hpar = list(learningrate = 0.01,

minibatchsize = 2^2,

numiterations = 60))

prediction = automl\_predict(model = amlmodel, X = test[,1:4])

prediction = ifelse(prediction > 2.5, 3, ifelse(prediction > 1.5, 2, 1)) %>% as.factor()

caret::confusionMatrix(test$Species, prediction)

**Remix AutoML**

Remix AutoML was developed by remyx institute. “*This is a collection of functions that I have made to speed up machine learning and to ensure high quality modeling results and output are generated. They are great at establishing solid baselines that are extremely challenging to beat using alternative methods (if at all). They are intended to make the development cycle fast and robust, along with making operationalizing quick and easy, with low latency model scoring.*” Below is a sample code for how to use in R.

library(RemixAutoML)

train$Species = train$Species %>% as.integer()

remixml = AutoCatBoostRegression(data = train %>% data.matrix()

, TargetColumnName = "Species"

, FeatureColNames = c(1:4)

, MaxModelsInGrid = 1

, ModelID = "ModelTest"

, ReturnModelObjects = F

, Trees = 150

, task\_type = "CPU"

, GridTune = FALSE

)

predictions = AutoCatBoostScoring(TargetType = 'regression'

, ScoringData = test %>% data.table::data.table()

, FeatureColumnNames = c(1:4)

, ModelObject = remixml$Model

)

prediction = ifelse(predictions$Predictions > 2.5, 3, ifelse(predictions$Predictions > 1.5, 2, 1)) %>% as.factor()

caret::confusionMatrix(test$Species, prediction)

**AutoXGboost**

Below is a sample code for how to use in R.

# load library

library(autoxgboost)

# create a classification task

trainTask = makeClassifTask(data = train, target = "Species")

# create a control object for optimizer

ctrl = makeMBOControl()

ctrl = setMBOControlTermination(ctrl, iters = 5L)

# fit the model

res = autoxgboost(trainTask, control = ctrl, tune.threshold = FALSE)

# do prediction and print confusion matrix

prediction = predict(res, test[,1:4])

caret::confusionMatrix(test$Species, prediction$data$response)

**Auto-sklearn**

Auto-sklearn is an automated machine learning toolkit and a drop-in replacement for a scikit-learn estimator. According to Auto-sklearn team, “*auto-sklearn frees a machine learning user from algorithm selection and hyperparameter tuning. It leverages recent advantages in Bayesian optimization, meta-learning and ensemble construction. Learn more about the technology behind auto-sklearn by reading our paper published at NIPS 2015 .*” Also to note that, this framework is possibly the slowest among all the frameworks presented in this post. Below is a sample code for how to use in Python.

import autosklearn.classification

import sklearn.model\_selection

import sklearn.metrics

import pandas as pd

train = pd.DataFrame(r.train)

test = pd.DataFrame(r.test)

x\_train = train.iloc[:,1:4]

y\_train = train[['Species']]

print(y\_train.head())

x\_test = test.iloc[:,1:4]

y\_test = test[['Species']]

print(y\_test.head())

automl = autosklearn.classification.AutoSklearnClassifier()

print("classifier")

print("fittiong" )

automl.fit(x\_train, y\_train)

y\_hat = automl.predict(x\_test)

# convert to pandas dataframe

data = {'actual': r.test.Species, 'Ypredict': y\_hat.tolist()}

df = pd.DataFrame(data, columns = ['actual','Ypredict'])

# create a confusion matrix and print results

confusion\_matrix = pd.crosstab(df['actual'], df['Ypredict'], rownames=['Actual'], colnames=['Predicted'])

print (confusion\_matrix)

**Autogluon**

Autogluon is the latest offering by aws labs. According to the developers, “*AutoGluon enables easy-to-use and easy-to-extend AutoML with a focus on deep learning and real-world applications spanning image, text, or tabular data. Intended for both ML beginners and experts, AutoGluon enables you to:*

* *Quickly prototype deep learning solutions for your data with few lines of code.*
* *Leverage automatic hyperparameter tuning, model selection / architecture search, and data processing.*
* *Automatically utilize state-of-the-art deep learning techniques without expert knowledge.*
* *Easily improve existing bespoke models and data pipelines, or customize AutoGluon for your use-case.*“

Below is a sample code for how to use in Python.

#import autogluon as ag

from autogluon import TabularPrediction as task

import pandas as pd

train\_data = task.Dataset(file\_path = "TRAIN\_DATA.csv")

test\_data = task.Dataset(file\_path = "TEST\_DATA.csv")

label\_column = 'Species'

print("Summary of class variable: \n", train\_data[label\_column].describe())

predictor = task.fit(train\_data = train\_data, label = label\_column)

y\_test = test\_data[label\_column] # values to predict

y\_pred = predictor.predict(test\_data)

print("Predictions: ", y\_pred)

perf = predictor.evaluate\_predictions(y\_true=y\_test, y\_pred=y\_pred, auxiliary\_metrics=True)

print(perf)

Hope you enjoyed this post.