MyAutoML

Erik Jan de Vries

Oct 20, 2021

CONTENTS

Date: Oct 20, 2021 **Version**: 0.6.1

Useful links: Binary Installers | Source Repository | Download the docs

`myautoml` is an open source library providing tools to automate machine learning processes.

To the getting started guides

To the user guide

To the reference guide

To the development guide

CHAPTER

ONE

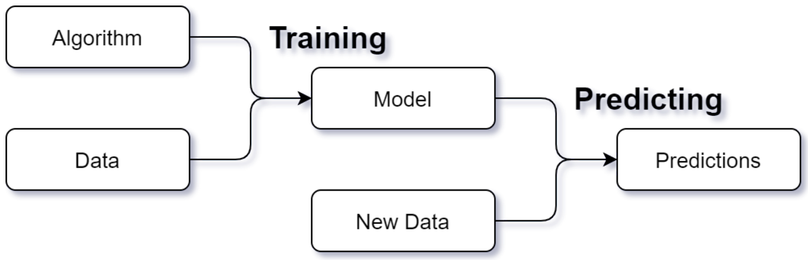
GETTING STARTED

1.1 Intro to MyAutoML

There are a lot of great open source tools for data scientists to use. Basically all data scientists know pandas, numpy and scikit-learn. Many will be familiar with tools such as MLflow and Hyperopt. However, while all these packages provide great functionalities, we still have to tie them all together when we want to build a functioning data science product. For this, many of us will be familiar with the feeling we're doing the same things over and over again.

MyAutoML aims to fill this gap: **allowing data scientists to focus on what makes their individual projects unique.**

MyAutoML focuses on scikit-learn type data science projects of classification and regression:



What makes your project unique? Indeed, the data. The overarching process and the algorithms tend to be mostly the same for every project. Scikit-learn and other open source packages provides the algorithms. MyAutoML aims to cover the process tying everything together, so you as a data scientist can focus on what's most important:

- translating your business problem into an analytics problem,
- preparing the target variable and features you need,
- generating business value.

1.2 What to expect

Perhaps it is easier to start off with what not to expect. MyAutoML is not a port of AutoML as offered by the Amazons, Googles and Microsofts of this world to your local environment. Perhaps one day in the future we may go in that direction, but at least for now we offer you tools (functions, classes and template scripts) to automate most of the repetitive work you do for your projects.

To get the most out of MyAutoML you will need a basic infrastructure setup, built upon open source software, such as MLflow and Hyperopt. Please have a look at the *Environment* page for more information.

1.3 Quick questions

Installation

The simplest way to install MyAutoML is to from PyPI via pip:

```
pip install myautoml
```

Glossary

In the User Guide we have included a *Glossary*.

1.3.1 Installation

The easiest way to install MyAutoML is to install it from PyPI.

Installing from PyPI

MyAutoML can be installed via pip from PyPI.

```
pip install myautoml
```

1.3.2 Machine Learning Process

The main two processes that we aim to cover with MyAutoML are the training and predicting processes. They are two separate processes, one for training a model and one for making predictions using a trained model. Each process is executed by running a Python script, e.g. `train.py` and `predict.py`. This can be as simple or as complex as you like: you can run the scripts manually (you can even run the code from a Jupyter notebook), or as an automated script in a Docker container on a Kubernetes platform scheduled by Airflow.

Training

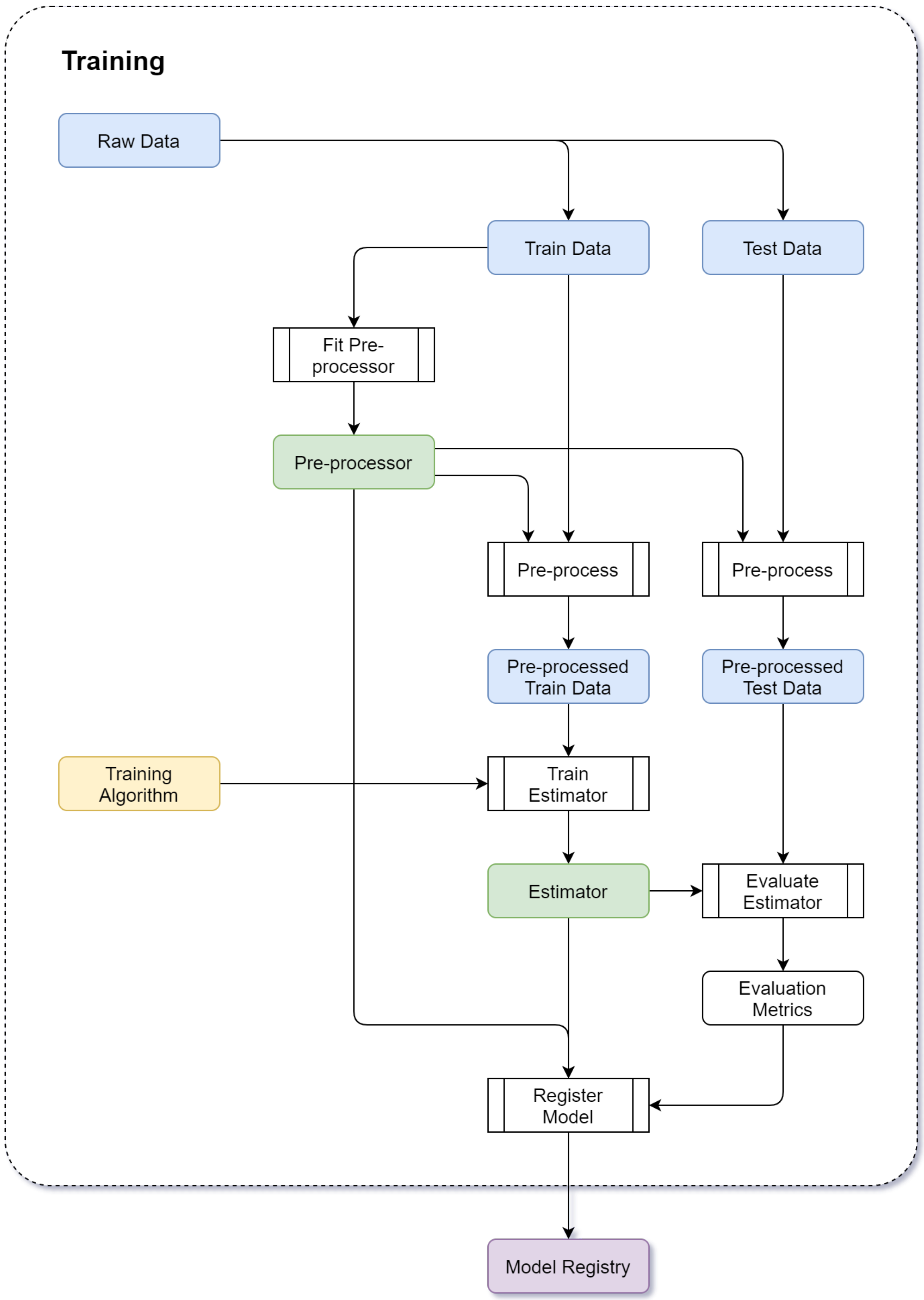
The purpose of the training process is to start with some data, process it with a certain algorithm and produce a model that captures the interesting patterns in the training data.

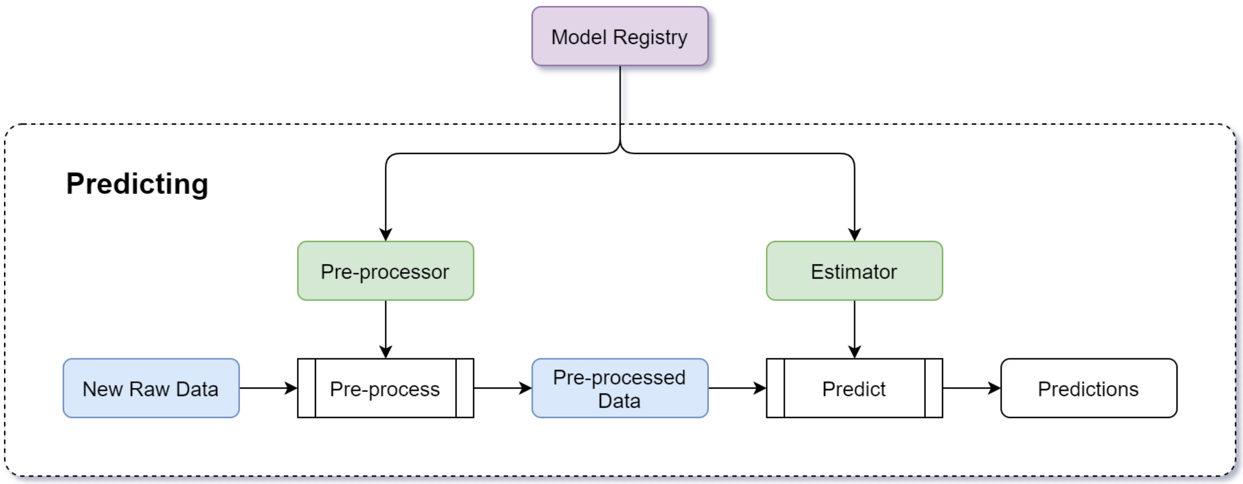
Predicting

The goal of the prediction process is to use a (trained) model and apply it to some new data to make predictions. A prediction script can make predictions for a batch of items, or it can spawn an API for real-time, on-demand predictions.

Calibrating

In some classification use cases we need to calibrate the output of our models to actual probabilities, rather than generic scores. While sometimes this can be done directly in the training process, in other cases it is more pragmatic to train a model first, and perform the calibration separately using the following process:


Training
Raw Data
Train Data
Test Data
Fit Pre-processor
Pre-processor
Pre-process
Pre-process
Pre-processed Train Data
Pre-processed Test Data
Training Algorithm
Train Estimator
Estimator
Evaluate Estimator
Evaluation Metrics
Register Model
Model Registry



Further reading

Wikipedia: Training, validation, and test sets

Machine Learning Mastery: How to Use ROC Curves and Precision-Recall Curves for Classification in Python

Machine Learning Mastery: How and When to Use a Calibrated Classification Model with scikit-learn

1.3.3 Environment

To get the most out of MyAutoML you will need to install and setup several components in your environment for MyAutoML to work with. Please have a look at the *Machine Learning Process* to see where these components fit in.

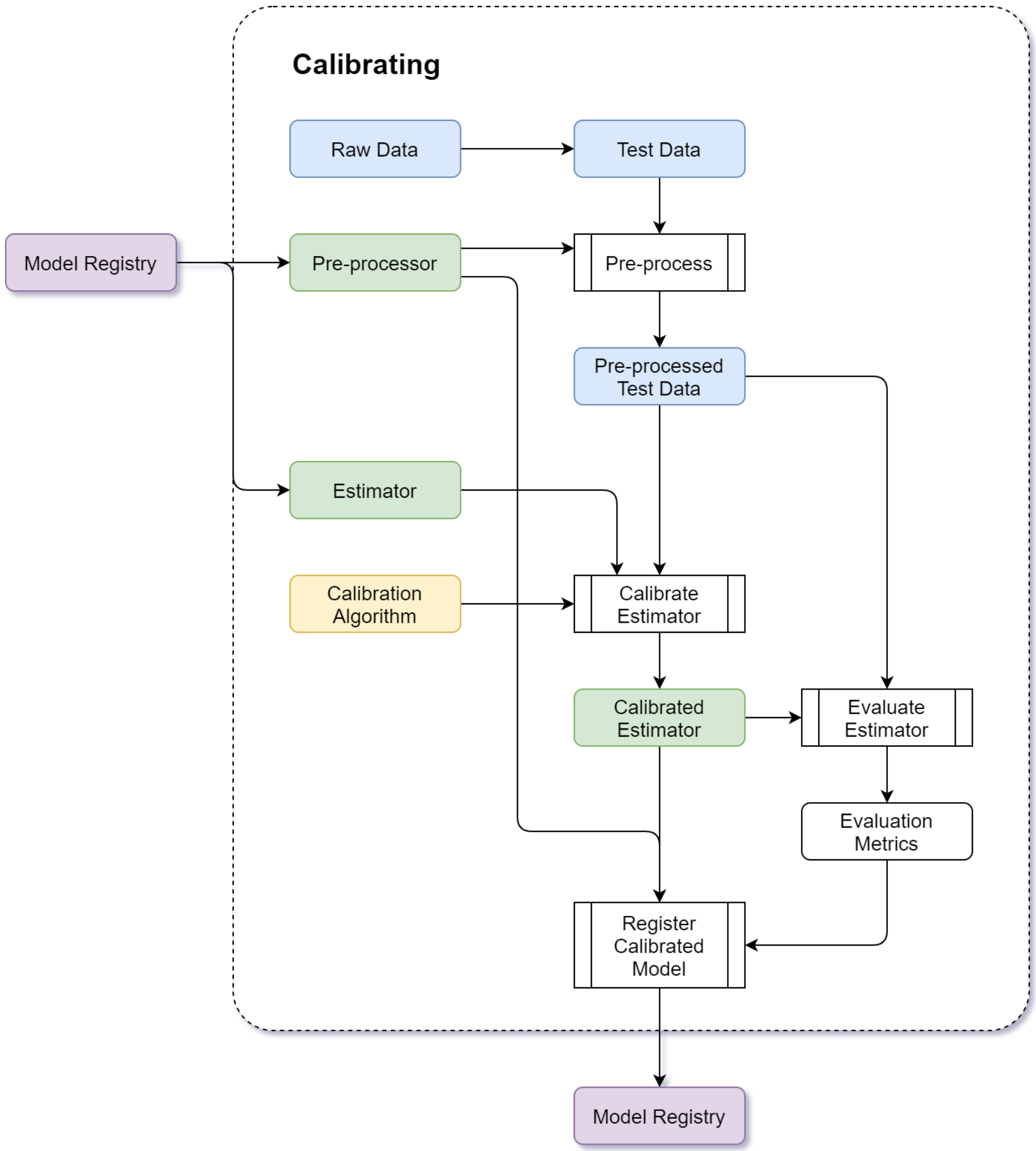
Model Registry: MLflow

As a model registry we work with MLflow. MLflow has two separate modules helping us to keep a good record of our models:

- MLflow Tracking
- MLflow Model Registry

In the *Machine Learning Process*, when we refer to a Model Registry, we mean both of these MLflow components above: every trained model is tracked in the MLflow Tracking Server. Additionally, some will be registered with a registered model name in the MLflow Model Registry. In the prediction process, a model is loaded from the MLflow Model Registry.

Please refer to the installation instructions and MLflow Tracking Servers to get you started. In order to use the MLflow Model Registry, you will need to setup an MLflow Tracking Server with a database Backend Store, such as SQLite or PostgreSQL.


Calibrating
Raw Data
Test Data
Model Registry
Pre-processor
Pre-process
Pre-processed Test Data
Estimator
Calibration Algorithm
Calibrate Estimator
Calibrated Estimator
Evaluate Estimator
Evaluation Metrics
Register Calibrated Model
Model Registry

1.3.4 Tutorial: train a model

In this tutorial we will show you step-by-step how to train a binary classification model with the help of MyAutoML.

- *Installation*
- ***Configuration***
  - *data*
  - *pre-processing*
  - *estimator & hyperparameters*
  - *config.yml*
- *Run & evaluate*
- *Wrapping up*

If you already have a local copy of MyAutoML including a working environment you may head over directly to *Configuration*. Otherwise, start with correctly installing MyAutoML.

Installation

Before we start you might wonder: what is Cookiecutter? Cookiecutter is a CLI tool that allows you to easily create a project structure including folders, scripts and documents. It is a way to save time if you find yourself creating and copy-pasting the same folders and scripts over time. A brief introduction can be found on medium. You can install Cookiecutter with pip.

```
pip install cookiecutter
```

If you already have Cookiecutter installed make sure it is up to date since we will use the –directory option. Note that we intend to move the cookiecutter to a separate repository, but for now it's still in a directory within this repository. More information on the installation of Cookiecutter can be found here. Once installed, move to the folder where you want to store MyAutoML and execute the following command.

```
cookiecutter https://github.com/myautoml/myautoml.git --directory="cookiecutter/binary_
↪classifier"
```

You will be prompted with a few questions regarding the project, this is a convenient feature of Cookiecutter to customize the new project to your wishes. When finished you are ready to create a new virtual environment.

A virtual environment allows you to install and use specific versions of Python and packages specifically for one project. This prevents cumbersome version conflicts when you run multiple projects from the same environment over time. More information on virtual environments can be found on real python.

Move to the `/scripts` folder and execute the following commands from your terminal.

```
conda env create -f environment.yml
conda activate <name_of_your_environment>
```

Configuration

MyAutoML comes with templated Python scripts and a config file, all meant to write as little custom code as possible, and to keep focused on what makes your project stand out: the data.

data

For this tutorial we are going to use the Bank Marketing Data Set from the UCI Machine learning repository. The official reference is:

> [Moro et al., 2014] S. Moro, P. Cortez and P. Rita. A Data-Driven Approach to Predict the Success of Bank Telemarketing. Decision Support Systems, Elsevier, 62:22-31, June 2014

This dataset holds a typical marketing classification task, where we are interested in predicting whether a customer will respond to a marketing campaign yes or no. The independent variables are a mix of demographics (age), customer specific data (balance), and behavioural data (response to previous campaigns). For demonstration purposes we will only use 6 independent variables plus the dependent variable of the original dataset.

Table 1: dataset preview

y	age	default	balance	housing	loan	poutcome
False	58	no	2143	yes	no	success
False	44	no	29	yes	no	unknown
False	33	no	2	yes	yes	failure

To transform this dataset to actual training data we need to modify `scripts/data.py`, specifically the `load_training_data` function. Make sure to refer to the correct path of the dataset.

```python
import pandas as pd
from pathlib import Path
from sklearn.model_selection import train_test_split


def load_training_data():
    df_path = Path('..') / 'data' / 'bank' / 'bank-full.csv'
    df = pd.read_csv(df_path, sep=';', usecols=['age', 'default', 'balance', 'housing',
                                                 'loan', 'poutcome', 'y'])
    x = df.drop(labels='y', axis=1)
    y = df['y'].astype('category').cat.codes.astype('bool')

    x_train, x_test, y_train, y_test = train_test_split(x, y,
                                                        stratify=y,
                                                        test_size=0.2,
                                                        random_state=123)

    return x_train, y_train
```

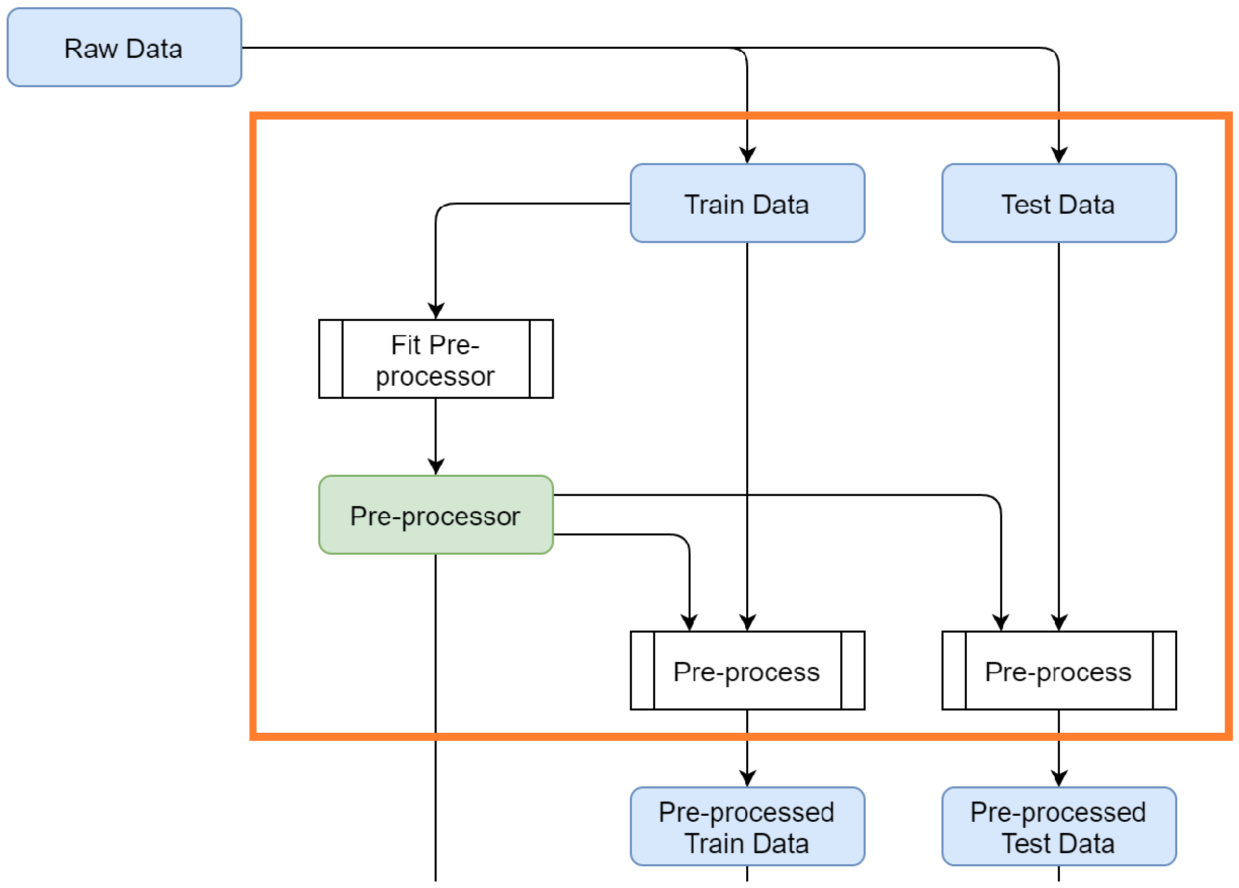
Now that we have the training data, we need to shape it so it can be used for modeling.

pre-processing

There are 3 pre-processing steps we need to take:

- Scale the numerical variables
- Create numeric dummy variables for the categorical variables
- Select the correct columns for each pre-processing step

It is possible to perform these pre-processing steps with custom Python functions, but we opt to choose for a scikit-learn pipeline. There are a number of advantages of using a pipeline, such as being able to `fit` the transformations on the training data, and to apply these on the test data. This is an important step in building models but easily missed. The official documentation of MyAutoML illustrates this nicely.



The pre-processor can be set in `scripts/model.py`, where an example pipeline is already shown in the `get_preprocessor` function. We will overwrite the example with the following code.

```python
def get_preprocessor():
    numeric_transformer = Pipeline(steps=[
        ('scaler', StandardScaler())])

    categorical_transformer = Pipeline(steps=[
        ('onehot', OneHotEncoder(handle_unknown='ignore'))])

    preprocessor = ColumnTransformer(transformers=[
```

(continues on next page)

(continued from previous page)

```
        ('num', numeric_transformer, selector(pattern="age|balance")),
        ('cat', categorical_transformer, selector(pattern="default|housing|loan|poutcome
↪"))]
    )

    return preprocessor
```

If any of this code is unfamiliar to you we can highly recommend watching these short videos on calmcode or read the official pipeline documentation.

estimator & hyperparameters

To be able to build a full model pipeline, MyAutoML also uses scikit-learn for its estimators. For this tutorial we will use logistic regression, but you can use any estimator from scikit-learn that is suited for binary classification.

To setup the estimator in `scripts/model.py` we need to retrieve a few things, which are all available in the official LogisticRegression documentation.

- module name: sklearn.linear_model
- class name: LogisticRegression
- hyperparameters: C, class_weight

This information is used in the `get_estimator` and `get_params` functions.

```
from sklearn.linear_model import LogisticRegression


def get_estimator(**params):
    estimator = LogisticRegression(**params)
    estimator_tags = {'module': 'sklearn.linear_model',
                      'class': 'LogisticRegression'}

    return estimator, estimator_tags


def get_params():
    estimator_params = {}
    search_space = {
        'C': hp.quniform('C', 0, 1, 0.0001),
        'class_weight': hp.choice('class_weight', [None, 'balanced'])
    }

    return estimator_params, search_space
```

Make sure that the keys from the `search_space` dictionary exactly match the names of the hyperparameters. The `hp.` methods help to create a hyperparameter space which can be efficiently searched with `hyperopt` when training the model.

config.yml

The last part of the configuration is to setup the config.yml file. For now we increase the max_evals to 10 and set the shap_analysis to False. The rest of the settings will be discussed shortly, as they make more sense once we see the first results.

```yaml
experiment:
  name: tutorial

model:
  name: tutorial

training:
  max_evals: 10

evaluation:
  primary_metric: roc_auc_cv
  metrics:
    - roc_auc
    - accuracy
  shap_analysis: False

prediction:
  stage: Production
```

Run & evaluate

You are now almost ready to run the train script. Make sure to make a copy of `.env.general.template` with the name `.env.general`. The template shows the settings that are (may be) expected by the program, without containing any secrets such as passwords, so that it can be committed to a git repository. You can update your personal and private settings in the .env.general file. **Never commit your .env.general (with passwords, etc.) to a git repository!**

Change directory to the `/scripts` folder with the MyAutoML environment activated and enter the following code to start the train script:

```
python train.py
```

If everything is setup correctly the script will start and you will see lots of logging statements in the terminal. Once the training is finished we are ready to evaluate, and this is the part where MyAutoML really shines. Besides training the model in an efficient manner with `hyperopt`, a lot of other things were taken care of by the train script:

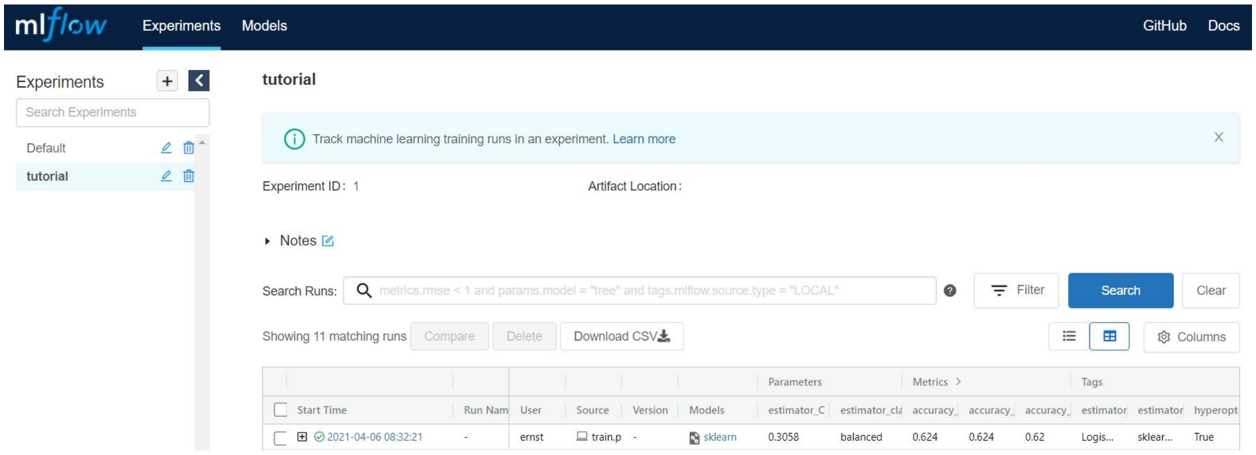
- Logging of the metadata of the estimator
- Logging of the metrics
- Creation of 5 typical binary classifier evaluation graphics
- Creation of the model as .pkl file, including a config file to easily distribute the model

All these things are integrated in MLflow, so you can use easily use them via the UI. If you are not familiar with MLflow, it is an open source platform for managing the end-to-end machine learning lifecycle. Amongst other things, it keeps track of experiments to track and compare results. More information can be found on their website.

To open the UI you first need to start it via a new terminal. Please move to the `scripts` folder, activate the MyAutoML environment, and execute the following command.

```
mlflow ui
```

Once you see the response in the terminal, head over to http://localhost:5000 and have a look. Note that we assume you are running this tutorial locally.



tutorial

Track machine learning training runs in an experiment. Learn more

Experiment ID: 1 Artifact Location:

Search Runs: metrics.rmse < 1 and params.model = "tree" and tags.mlflow.source.type = "LOCAL"

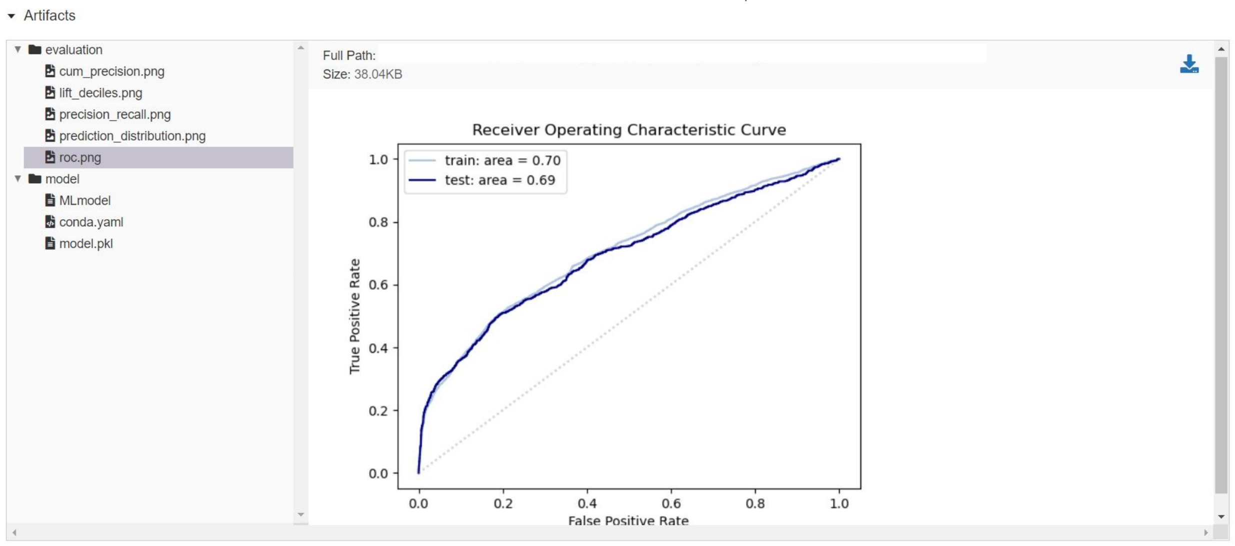
Showing 11 matching runs

						Parameters		Metrics			Tags		
Start Time	Run Nam	User	Source	Version	Models	estimator_C	estimator_cla	accuracy_	accuracy_	accuracy_	estimator	estimator	hyperopt
2021-04-06 08:32:21	-	ernst	train.p	-	sklearn	0.3058	balanced	0.624	0.624	0.62	Logis...	sklear...	True

By pressing the + button you gain access to every training evaluation (config.yml -> max_evals), which contains valuable information:

- hyperparameter settings (complexity, balanced y/n)
- evaluation metrics (accuracy in this case, specified for cv, train, and test)
- tags (estimator class & model)

Although informative, it gets even better when you click on one of the runs. Besides ~20 evaluation metrics there is a special section at the bottom which is called Artifacts. This section contains the graphical outputs of the specific run, as well as the actual trained model.



There is more information available than we can describe here, so we highly recommend to take your time exploring the experiment runs. Once finished, try to a different estimator and hyperparameter settings by adjusting the `get_estimator` and `get_params` functions. When you run the train script again the results of these new evaluations will also become visible in the UI, so you can easily compare which estimator and hyperparameter settings work best.

Wrapping up

Hopefully by now you have a better idea how MyAutoML works, and how it can help you to easily and efficiently train and compare binary classification models. In the following tutorial we will explain the next step in the modelling phase: prediction.

CHAPTER

TWO

USER GUIDE

The User Guide covers all of MyAutoML by topic area.

2.1 Glossary

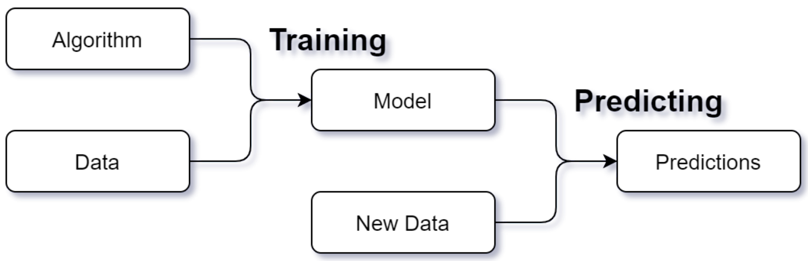
2.1.1 Estimator

An estimator is any object that learns from data. Most typically these are classification, regression and clustering algorithms.

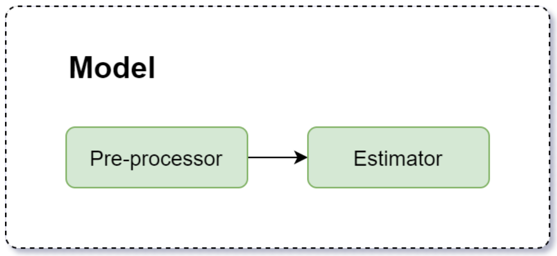
See the Scikit-Learn estimator.

2.1.2 Model

What exactly is a model? In MyAutoML we view a model as something that we can apply to data to make predictions.



Normally the raw data we start off with requires some preprocessing, before it can be used as input for the prediction algorithm (see the training and predicting processes above). Therefore we view a model as a *Pipeline* containing a preprocessor and an estimator:



2.1.3 Model Registry

A model registry is a tool to keep track of your models. Ideally it serves both as a log for your data science experiments (keeping track of all the variations of models you've trained, along with key metrics indicating model performance), and as a store for serving models to prediction processes.

MLflow has modules supporting these functions, namely the MLflow Tracking Server and the MLflow Model Registry respectively.

2.1.4 Pipeline

A pipeline is a sequence of operations to be carried out on a dataset.

See the Scikit-Learn Pipeline.

2.1.5 Pre-processor

A pre-processor is any object that transforms a raw data set into a form that can be used with an *Estimator*. A pre-processor often takes the form of a *Pipeline* containing multiple transformation steps.

See for example the Scikit-Learn ColumnTransformer and the Scikit-Learn preprocessing module.

CHAPTER

THREE

API REFERENCE

CHAPTER

FOUR

DEVELOPMENT

PYTHON MODULE INDEX

INDEX