AMLE Documentation

Release 0.1.0

Matthew John Hayes

Dec 04, 2017

Contents

AMLE (Automated Machine Learning Environment) is designed to help automate running machine learning (ML) tests to reduce effort and make it easier to innovate. Read More

Contents:

CHAPTER 1

Introduction

The AMLE *(rhymes with camel)* project makes it easier to design, run and tune machine learning for specific use cases.

TBD - more here...

CHAPTER 2

Configure

The dataset class provides methods available through *project_policy.yaml* to manipulate the ingested data so that it is suitable for processing.

2.1 Column Operations

Here are operations that can be performed on the dataset columns:

2.1.1 delete_columns

TBD

2.1.2 duplicate_column

TBD

2.1.3 one_hot_encode

Creates new column(s) with one hot encoded values. This is useful when you have more than two result types in a column.

You need to specify a column that is used as the source for creating one-hot-encoded columns. Note that this specified column is not updated.

Values in the column are listed that should be used to create new one-hot-encoded columns. Note that the value is used as the column name.

Be careful to avoid column name collisions.

Example:

```
- one_hot_encode:
    - column: class
    - values:
        - Iris-setosa
        - Iris-versicolor
        - Iris-virginica
```

2.1.4 rescale

2.1.5 set_output_columns

Sets what columns are used as output data from dataset (i.e. what columns contain the expected answer(s) Pass it a list of output column names

Example:

```
- set_output_columns:
    - Iris-setosa
    - Iris-versicolor
    - Iris-virginica
```

2.1.6 translate

TBD

2.1.7 trim_to_columns

TBD

2.2 Data Import

2.2.1 ingest

TBD

2.3 Data Export

2.3.1 get_data

TBD

2.3.2 inputs_array

TBD

2.3.3 outputs_array

TBD

2.4 General

2.4.1 set_name

TBD

2.4.2 transform

TBD

2.5 Partitioning

2.5.1 in_partition

TBD

2.5.2 partition

TBD

2.5.3 partition_sets

TBD

2.6 Row Operations

2.6.1 shuffle

TBD

2.6.2 trim_to_rows

TBD

2.7 Visibility

2.7.1 display

TBD

CHAPTER 3

Recipes

TBD

Recipes:

- *TBD Recipe*

3.1 TBD Recipe

CHAPTER 4

Modules

4.1 amle module

Automated Machine Learning Environment (AMLE)

This code is a simple shim to help automate running machine learning (ML) tests to reduce effort and make it easier to innovate.

Requires various packages including YAML: sudo apt-get install python-pip git git-flow python-pytest python-yaml

Requires PIP packages coloredlogs, voluptuous and numpy. Install with: pip install coloredlogs voluptuous numpy

Principles (aspirational):

- Generic. Just a shim, does not contain ML code, and tries to not be opinionated about how ML works or data types
- Reproducibility. Run the same test with same inputs and get the same output(s) - or at least statistically similar.
- Reduce experimentation work effort. Support comparative testing across different parameters and/or ML algorithms, retains historical parameters and results
- Add value to experimentation. Support evolutionary genetic approach to configuring algorithm parameters
- Visibility. Make it easy to understand how experiments are running / ran

class amle.**AMLE**(*CLI_arguments*)
Bases: *baseclass.BaseClass*

This class provides core methods for an Automated Machine Learning Environment (AMLE)

load_aggregator(*agg_name*)
Passed file location for an aggregator module and return it as an object

load_algorithm(*alg_name*)
Passed file location for an algorithm module and return it as an object

run()
Run AMLE

run_aggregator(*agg_name*, *agg_parameters*)
Run an aggregator, as per spec from policy

run_experiment(*experiment_name*)
Run an experiment, as per spec from policy

amle.**print_help**()
Print out the help instructions

4.2 baseclass module

The baseclass module is part of the AMLE suite and provides an inheritable class methods for logging

class baseclass.**BaseClass**
Bases: object

This class provides the common methods for inheritance by other classes

configure_logging(*name*, *s_name*, *c_name*)
Configure logging for the class that has inherited this method

4.3 evaluate module

Automated Machine Learning Environment (AMLE)

Evaluate library provides a class with methods to evaluate result data against desired results

class evaluate.**Evaluate**(*logger*)
Bases: object

Class with methods for evaluating result data

simple_accuracy(*results*, *threshold*)
Evaluation of simple results data that is in the form of a list of dictionaries, each of which contain two KVPs:

- actual
- computed

All result values are floats

A threshold is passed in, and if the actual result is +/- threshold of computed result then it is recorded as correct otherwise incorrect.

Returns accuracy percentage as an integer between 0 and 100.

4.4 config module

The config module is part of the AMLE suite.

It represents AMLE configuration data that is global, i.e. not project-specific

It loads configuration from file, validates keys and provides access to values

It expects a file called "config.yaml" to be in the config subdirectory, containing properly formed YAML

class config.**Config**(*dir_default='config'*, *dir_user='config/user'*, *config_filename='config.yaml'*)
Bases: *baseclass.BaseClass*

This class provides methods to ingest the configuration file and provides access to the config keys/values. Config file is YAML format in config subdirectory, and is called 'config.yaml'

get_value(*config_key*)
Passed a key and see if it exists in the config YAML. If it does then return the value, if not return 0

ingest_config_default(*config_filename*, *dir_default*)
Ingest default config file

ingest_config_file(*fullpath*)
Passed full path to a YAML-formatted config file and ingest into a dictionary

ingest_config_user(*config_filename*, *dir_user*)
Ingest user config file that overrides values set in the default config file.

inherit_logging(*config*)
Call base class method to set up logging properly for this class now that it is running

4.5 dataset module

The dataset module provides an abstraction for sets of data, primarily aimed at use in machine learning (ML).

class dataset.**DataSet**(*logger*)
Bases: object

Represents a set of ML data with methods to ingest, manipulate (i.e. preprocess) and extract

delete_columns(*column_names*)
Passed a list of columns and remove them from the dataset

display(*display_type*)
Display data

duplicate_column(*current_column_name*, *new_column_name*)
Passed name of a current column and copy that column to a new column with name passed for new column name

get_data()
Return data in native format

in_partition(*partition_name*, *row_number*)
Passed a partition name, row number and total number of rows in the dataset and after consulting internal partition settings, return a 1 if the given row belongs to the partition, otherwise 0

ingest(*filename*)
Load data CSV from file into class as a list of dictionaries of rows. Requires first row in file to be a header row and uses these values as keys in row dictionaries. Example row: {'dataset': 'ML', 'min_interpacket_interval': '0.001'}

inputs_array(*partition='A'*)
Return input data as a numpy array Filter out output column(s) and only include rows from specified partition, which defaults to 'A'

one_hot_encode(*column_name*, *keys*)
Take an existing column and use it to build new columns that are each one hot encoded for one of the specified keys.

Supplied with the column_name string and a list that has the specific key names to build new columns.

outputs_array(*partition='A'*)
Return output data as a numpy array Filter out input columns

partition(*partitions*)
Set partition parameters for split of dataset into arbitrary partitions, which are named by strings. Note that partitioning is applied when data is retrieved, not to internal dataset

Passed a list of partition names which are used to divide the dataset based on modulo division by the length of the list.

Setting partitions overwrites any previously set partition configuration

Default partition is partitions=['A'] (i.e. all data in partition 'A')

Standard convention for usage of partitions is: * Partition 'Training' is used as training data * Partition 'Validation' is used as validation (test) data

Example: Randomise row order, then allocate 75% of rows to partition 'Training' with the last 25% in partition 'Validation':

dataset.shuffle() dataset.partition(partitions=['Training', 'Training',

'Training', 'Validation'])

partition_sets()
Return the number of sets in the partition

rescale(*column_name*, *min_x*, *max_x*)
Rescale all values in a column so that they sit between 0 and 1. Uses rescaling formula: x' = (x - min(x)) / (max(x) - min(x))

set_name(*name*)
Set the name for the dataset

set_output_columns(*output_columns*)
Set what columns are used as output data from dataset (i.e. what columns contain the expected answer(s) Pass it a list of output column names

shuffle(*seed=0*)
Shuffle dataset rows. Set seed=1 if want predictable randomness for reproduceable shuffling

transform(*transform_policy*)
Passed policy transforms and run them against the dataset.

translate(*column_name*, *value_mapping*)
Go through all values in a column replacing any occurences of key in value_mapping dictionary with corresponding value

trim_to_columns(*fields*)
Passed a list of fields (columns) to retain and trim the internal representation of the training data to just those columns

trim_to_rows(*key*, *fields*)
Passed a key (column name) and list of fields (column values) match rows that should be retained and remove other rows

4.6 policy module

Automated Machine Learning Environment (AMLE)

Policy library that handles reading in policy, validating it and providing values to other parts of AMLE

class policy.**Policy**(*config*, *project_directory*)
Bases: *baseclass.BaseClass*

This policy class serves these purposes: - Ingest policy (policy.yaml) from file - Validate correctness of policy against schema - Methods and functions to check various parameters

against policy

Note: Class definitions are not nested as not considered Pythonic Main Methods and Variables: - ingest # Read in policy and check validity

get_aggregator(*name*)
Return policy for a named aggregator

get_aggregators()
Return a list of policy aggregators

get_algorithms()
Return a list of policy algorithms

get_datasets()
Return a list of policy datasets

get_experiment(*name*)
Return policy for a named experiment

get_experiments()
Return a list of policy experiments

get_run_items()
Return a list of run items

policy.**validate**(*logger*, *data*, *schema*, *where*)
Generic validation of a data structure against schema using Voluptuous data validation library Parameters:

- logger: valid logger reference
- data: structure to validate
- schema: a valid Voluptuous schema
- where: string for debugging purposes to identity the policy location

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index