l3wrapper

Release 0.6.3-docs

Jul 08, 2020

Contents

Documentation https://l3wrapper.readthedocs.io/

Source Code https://github.com/g8a9/l3wrapper

Issue Tracker https://github.com/g8a9/l3wrapper/issues

PyPI https://pypi.org/project/l3wrapper/

Zenodo https://zenodo.org/record/3758480

CHAPTER 1

l3wrapper

A Python 3 wrapper around Live-and-Let-Live (L^3) classifier binaries implementing the `scikit-learn` estimator interface. The associative classifier was originally published in[1].

When imported, the package looks for L^3 compiled binaries in the user's `$HOME` directory. If they are not found, it downloads them. If you mind letting the wrapper do this for you, you can download the binaries for macOS Catalina or Ubuntu 18.04.

1.1 Installation

Install using pip with:

```
pip install l3wrapper
```

Or, download a wheel or source archive from PyPI.

1.1.1 Requirements

The package is dependent on `numpy`, `scikit-learn`, `tqdm`, and `requests`.

1.2 Usage

By design, the classifier **is intended for categorical/discrete attributes**. Therefore, using subtypes of `numpy.number` to fit the model is not allowed.

[1] Elena Baralis, Silvia Chiusano, and Paolo Garza. 2008. A Lazy Approach to Associative Classification. IEEE Trans. Knowl. Data Eng. 20, 2 (2008), 156–171. https://doi.org/10.1109/TKDE.2007.190677

1.2.1 Simple classification

A sample usage with the Car Evaluation dataset:

```
>>> from l3wrapper.l3wrapper import L3Classifier
>>> import numpy as np
>>> from sklearn.model_selection import train_test_split
>>> from sklearn.metrics import accuracy_score
>>> X = np.loadtxt('car.data', dtype=object, delimiter=',')
>>> y = X[:, -1]
>>> X = X[:, :-1]
>>> X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_
↪state=42)
>>> clf = L3Classifier().fit(X_train, y_train)
>>> accuracy_score(y_test, clf.predict(X_test))
0.9071803852889667
```

1.2.2 Column names and interpretable rules

Use the `column_names` and `save_human_readable` parameters to obtain an interpretable representation of the model:

```
>>> column_names = ['buying', 'maint', 'doors', 'persons', 'lug_boot', 'safety']
>>> clf = L3Classifier().fit(X_train, y_train, column_names=column_names, save_human_
↪readable=True)
```

The snippet will generate the *level1* and *level2* rule sets. An excerpt is:

```
0 persons:4,safety:high,maint:low,buying:high acc 12 100.0 4
1 doors:2,buying:vhigh,safety:med,lug_boot:med unacc 11 100.0 4
```

in the form:

```
<rule_id>\t<antecedent>\t<class label>\t<support count>\t<confidence(%)>\t<rule␣
↪length>
```

1.3 Known limitations

- **fixed** *The parallel training of multiple models cause failures (e.g. using ‘‘GridSearchCV‘‘, ‘‘joblib‘‘ or custom parallelism through ‘‘multiprocessing‘‘ with ‘‘njobs>1‘‘).*
- The scikit-learn's utility `check_estimator` still doesn't work, as L3Classifier doesn't support numerical input.

1.4 Compatibility

The underlying L^3 binaries are currently available for **macOS and Ubuntu**.

The package is currently tested with Python 3.6+.

1.5 License

The MIT License.

1.6 Authors

l3wrapper was written by g8a9.

1.6.1 l3wrapper.l3wrapper

The main module for the L3 estimator.

class l3wrapper.l3wrapper.**L3Classifier**(*min_sup=0.01*, *min_conf=0.5*, *l3_root='/home/docs/l3wrapper_data'*, *assign_unlabeled='majority_class'*, *match_strategy='majority_voting'*, *max_matching=1*, *specialistic_rules=True*, *max_length=0*, *rule_sets_modifier='standard'*)

The L3-based estimator implementing the scikit-learn estimator interface.

The model training relies on the L3 binaries. At this point they should be already iavailable. Instead, inference is enabled by the estimator itself. Hence, no L3 binaries are used at classification time (see `predict()`).

Parameters

- **min_sup** (*float, default='0.01'*) – The minimum support threshold to be used while training.
- **min_conf** (*float, default='0.5'*) – The minimum confidence threshold to be used while training.
- **l3_root** (*str, default='$HOME/l3wrapper_data'*) – The root folder where L3 binaries are located.
- **assign_unlabeled** (*str, default='majority_class'*) – The strategy used to assign the classification label whenever there is no rule that matches the data point to classified.
- **match_strategy** (*{'majority_voting'}, default='majority_voting'*) – The strategy used to pick which rule or set of rules should be used to classify a data point. Supported values: - 'majority_voting' (default): choose the label using majority voting among the class labels predicted by the top *max_matching* rules. In case of a tie, the label is chosen arbitrarily.
- **max_matching** (*int, default=1*) – The number of rules to be used for choosing the final label. It is used only when match_strategy='majority_voting'.
- **specialistic_rules** (*bool, default=True*) – Choose whether to prefer specialistic or general rules first at training time.
- **max_length** (*int, default=0*) – The maximum length of the mined rules. Note that as per the L3 training procedure, this applies to the *macro-itemset* mining step, i.e. it is possible that their relative traductions to normal itemset contain rule antecedents longer than max_length. (default=0, i.e. no limit)

- **rule_sets_modifier** (*{'standard', 'level1'}, default='standard'*) – Use this parameter to modify the extracted rule sets. Option 'level1' retains only the level 1 rule set, discarding level 2. If 'standard', the original behavior of L3 is unchanged.

X_
The input passed during *fit()*.

Type ndarray, shape (n_samples, n_features)

y_
The labels passed during *fit()*.

Type ndarray, shape (n_samples,)

classes_
The classes seen at *fit()*.

Type ndarray, shape (n_classes,)

n_items_used_
The number of different items seen.

Type int

lvl1_rules_
The level 1 rules (`Rule`) mined at *fit()*.

Type list

lvl2_rules_
The level 2 rules (`Rule`) mined at *fit()*.

Type list

n_lvl1_rules_
The number of level 1 rules.

Type int

n_lvl2_rules_
The number of level 2 rules.

Type int

fit(*X*, *y*, *column_names=None*, *save_human_readable=False*, *remove_files=True*)
A reference implementation of a fitting function for a classifier.

Parameters

- **X** (*array-like, shape (n_samples, n_features)*) – The training input samples. No numerical inputs are allowed it.
- **y** (*array-like, shape (n_samples,)*) – The target values. An array of int.
- **column_names** (*list, default=None*) – A list containing the names to assign to columns in the dataset. They will be used when printing the human readable format of the rules.
- **remove_files** (*bool, default=True*) – Use this parameter to remove all the file generated by the original L3 implementation at training time.

Returns **self** – Returns self.

Return type object

predict(*X*)
Predict the class labels for each sample in X.

Additionally, the method helps to characterize the rules used during the inference. Each record to be predicted is converted into a Transaction and the list of transactions is saved. From the transactions one can retrieve:

- which level was used to classify it (level=-1 means that no rule has covered the record)
- which Rule (or rules) was used to classify it.

Parameters **X** (*array-like, shape (n_samples, n_features)*) – The input samples.

Returns **y** – The label for each sample.

Return type ndarray, shape (n_samples,)

1.6.2 l3wrapper.validation

This module provides several validation functions used by the estimator.

l3wrapper.validation.**check_column_names**(*X*, *column_names*)
Check the column names specified by the user.

By design, the character ':' is not allowed in any column name.

l3wrapper.validation.**check_dtype**(*array*)
Check the type of input values given by the user.

No subclasses on numpy.number are allowed.

CHAPTER 2

Indices and tables

- genindex
- search

Python Module Index

l

Index