pipeasy-spark Documentation

Release 0.2.1

Benjamin Habert

Feb 06, 2019

Contents:

pipeasy-spark is a Python package designed to help you prepare your pyspark dataframe for a machine learning task performed by the spark ML library.

Check out the repository on Github

CHAPTER 1

Installation

1.1 Stable release

(test: addig stuff manually)

To install pipeasy-spark, run this command in your terminal:

```
$ pip install pipeasy_spark
```

This is the preferred method to install pipeasy-spark, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

1.2 From sources

The sources for pipeasy-spark can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/Quantmetry/pipeasy-spark
```

Or download the tarball:

```
$ curl  -OL https://github.com/Quantmetry/pipeasy-spark/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

CHAPTER 2

Usage

- basic example in the package README.
- basic demo (notebook): how to pre-process a pyspark dataframe.
- advanced (notebook): use a preprocessing pipeline for maching learning.

CHAPTER 3

pipeasy_spark

3.1 pipeasy_spark package

3.1.1 Submodules

3.1.2 pipeasy_spark.convenience module

pipeasy_spark.convenience.**build_default_pipeline**(*dataframe*, *exclude_columns=()*)

Build simple transformation pipeline (untrained) for the given dataframe.

By defaults numeric columns are processed with StandardScaler and string columns are processed with StringIndexer + OneHotEncoderEstimator

dataframe: pyspark.sql.Dataframe only the schema of the dataframe is used, not actual data.

exclude_columns: list of str name of columns for which we want no transformation to apply.

pipeline: pyspark.ml.Pipeline instance (untrained)

pipeasy_spark.convenience.**build_pipeline_by_dtypes**(*dataframe*, *exclude_columns=()*, *string_transformers=()*, *numeric_transformers=()*)

Build simple transformation pipeline (untrained) for the given dataframe.

dataframe: pyspark.sql.Dataframe only the schema of the dataframe is used, not actual data.

exclude_columns: list of str name of columns for which we want no transformation to apply.

string_transformers: list of transformer instances The successive transformations that will be applied to string columns Each element is an instance of a pyspark.ml.feature transformer class.

numeric_transformers: list of transformer instances The successive transformations that will be applied to numeric columns Each element is an instance of a pyspark.ml.feature transformer class.

pipeline: pyspark.ml.Pipeline instance (untrained)

3.1.3 pipeasy_spark.core module

pipeasy_spark.core.**build_pipeline**(*column_transformers*)

Create a dataframe transformation pipeline.

The created pipeline can be used to apply successive transformations on a spark dataframe. The transformations are intended to be applied per column.

```
>>> df = titanic.select('Survived', 'Sex', 'Age').dropna()
>>> df.show(2)
+--------+------+----+
|Survived|   Sex| Age|
+--------+------+----+
|       0|  male|22.0|
|       1|female|38.0|
+--------+------+----+
>>> pipeline = build_pipeline({
        # 'Survived' : this variable is not modified, it can also be omitted from␣
↪the dict
        'Survived': [],
        'Sex': [StringIndexer(), OneHotEncoderEstimator(dropLast=False)],
        # 'Age': a VectorAssembler must be applied before the StandardScaler
        # as the latter only accepts vectors as input.
        'Age': [VectorAssembler(), StandardScaler()]
    })
>>> trained_pipeline = pipeline.fit(df)
>>> trained_pipeline.transform(df).show(2)
+--------+-------------+--------------------+
|Survived|          Sex|                 Age|
+--------+-------------+--------------------+
|       0|(2,[0],[1.0])|[1.5054181442954726]|
|       1|(2,[1],[1.0])| [2.600267703783089]|
+--------+-------------+--------------------+
```

column_transformers: dict(str -> list) key (str): column name; value (list): transformer instances (typically instances of pyspark.ml.feature transformers)

pipeline: a pyspark.ml.Pipeline instance

3.1.4 pipeasy_spark.transformers module

class pipeasy_spark.transformers.**ColumnDropper**(*inputCols=None*)

Bases: pyspark.ml.base.Transformer, pyspark.ml.param.shared.HasInputCols

Transformer to drop several columns from a dataset.

transform(*dataset*)

Transforms the input dataset with optional parameters.

Parameters

- **dataset** – input dataset, which is an instance of pyspark.sql.DataFrame
- **params** – an optional param map that overrides embedded params.

Returns transformed dataset

New in version 1.3.0.

class pipeasy_spark.transformers.**ColumnRenamer**(*inputCol=None*, *outputCol=None*)
Bases: pyspark.ml.base.Transformer, pyspark.ml.param.shared.HasInputCol, pyspark.ml.param.shared.HasOutputCol

Transformer to rename a column to another.

transform(*dataset*)
Transforms the input dataset with optional parameters.

Parameters

- **dataset** – input dataset, which is an instance of pyspark.sql.DataFrame
- **params** – an optional param map that overrides embedded params.

Returns transformed dataset

New in version 1.3.0.

pipeasy_spark.transformers.**set_transformer_in_out**(*transformer*, *inputCol*, *outputCol*)
Set input and output column(s) of a transformer instance.

3.1.5 Module contents

Top-level package for pipeasy-spark.

The pipeasy-spark package provides a set of convenience functions that make it easier to map each column of a Spark dataframe (or subsets of columns) to user-specified transformations.

pipeasy_spark.**build_pipeline**(*column_transformers*)
Create a dataframe transformation pipeline.

The created pipeline can be used to apply successive transformations on a spark dataframe. The transformations are intended to be applied per column.

```
>>> df = titanic.select('Survived', 'Sex', 'Age').dropna()
>>> df.show(2)
+--------+------+----+
|Survived|   Sex| Age|
+--------+------+----+
|       0|  male|22.0|
|       1|female|38.0|
+--------+------+----+
>>> pipeline = build_pipeline({
        # 'Survived' : this variable is not modified, it can also be omitted from␣
↪the dict
        'Survived': [],
        'Sex': [StringIndexer(), OneHotEncoderEstimator(dropLast=False)],
        # 'Age': a VectorAssembler must be applied before the StandardScaler
        # as the latter only accepts vectors as input.
        'Age': [VectorAssembler(), StandardScaler()]
    })
>>> trained_pipeline = pipeline.fit(df)
>>> trained_pipeline.transform(df).show(2)
+--------+-------------+--------------------+
|Survived|          Sex|                 Age|
+--------+-------------+--------------------+
|       0|(2,[0],[1.0])|[1.5054181442954726]|
|       1|(2,[1],[1.0])| [2.600267703783089]|
+--------+-------------+--------------------+
```

column_transformers: dict(str -> list) key (str): column name; value (list): transformer instances (typically instances of pyspark.ml.feature transformers)

pipeline: a pyspark.ml.Pipeline instance

pipeasy_spark.**build_pipeline_by_dtypes**(*dataframe*, *exclude_columns=()*, *string_transformers=()*, *numeric_transformers=()*)

Build simple transformation pipeline (untrained) for the given dataframe.

dataframe: pyspark.sql.Dataframe only the schema of the dataframe is used, not actual data.

exclude_columns: list of str name of columns for which we want no transformation to apply.

string_transformers: list of transformer instances The successive transformations that will be applied to string columns Each element is an instance of a pyspark.ml.feature transformer class.

numeric_transformers: list of transformer instances The successive transformations that will be applied to numeric columns Each element is an instance of a pyspark.ml.feature transformer class.

pipeline: pyspark.ml.Pipeline instance (untrained)

pipeasy_spark.**build_default_pipeline**(*dataframe*, *exclude_columns=()*)

Build simple transformation pipeline (untrained) for the given dataframe.

By defaults numeric columns are processed with StandardScaler and string columns are processed with StringIndexer + OneHotEncoderEstimator

dataframe: pyspark.sql.Dataframe only the schema of the dataframe is used, not actual data.

exclude_columns: list of str name of columns for which we want no transformation to apply.

pipeline: pyspark.ml.Pipeline instance (untrained)

CHAPTER 4

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

4.1 Types of Contributions

4.1.1 Report Bugs

Report bugs at https://github.com/Quantmetry/pipeasy-spark/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

4.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

4.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

4.1.4 Write Documentation

pipeasy-spark could always use more documentation, whether as part of the official pipeasy-spark docs, in docstrings, or even on the web in blog posts, articles, and such.

4.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/Quantmetry/pipeasy-spark/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

4.2 Get Started!

Ready to contribute? Here's how to set up *pipeasy_spark* for local development.

1. Fork the *pipeasy_spark* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/pipeasy_spark.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv pipeasy_spark
$ cd pipeasy_spark/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 pipeasy_spark tests
$ python setup.py test or py.test
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

4.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.7, 3.4, 3.5 and 3.6, and for PyPy. Check https://travis-ci.org/BenjaminHabert/pipeasy_spark/pull_requests and make sure that the tests pass for all supported Python versions.

4.4 Tips

To run a subset of tests:

```
$ py.test tests.test_pipeasy_spark
```

4.5 Deploying

A reminder for the maintainers on how to deploy. Make sure all your changes are committed (including an entry in HISTORY.rst). Then run:

```
$ bumpversion patch # possible: major / minor / patch
$ git push
$ git push --tags
```

Travis will then deploy to PyPI if tests pass.

CHAPTER 5

History

5.1 0.2.0 (2019-01-06)

First usable version of the package. We decided on the api:

- `pipeasy_spark.build_pipeline(column_transformers={'column': []})` is the core function where you can define a list of transormers for each columns.
- `pipeasy_spark.build_pipeline_by_dtypes(df, string_transformers=[])` allows you to define a list of transormers for two types of columns: `string_` and `numeric_`.
- `pipeasy_spark.build_default_pipeline(df, exclude_columns=['target'])` builds a default transformer for the `df` dataframe.

5.2 0.1.2 (2018-10-12)

- I am still learning how all these tools interact with each other

5.3 0.1.1 (2018-10-12)

- First release on PyPI.

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

Index