mlqa

Release 0.1.1

Dogan Askan

Feb 05, 2023

CONTENTS:

Welcome to mlqa's documentation. Get started with Introduction and then get an overview with the Quickstart. The rest of the documentation describes each component of MLQA in detail in the *API Reference* section.

CHAPTER

ONE

USER'S GUIDE

1.1 Introduction

1.1.1 What is it?

MLQA is a Python package that is created to help data scientists, analysts and developers to perform quality assurance (i.e. QA) on pandas dataframes and 1d arrays, especially for machine learning modeling data flows. It's designed to work with logging library to log and notify QA steps in a descriptive way. It includes stand alone functions (i.e. checkers) for different QA activities and DiffChecker class for integrated QA capabilities on data.

1.1.2 Installation

You can install MLQA with pip.

```
pip install mlqa
```

MLQA depends on Pandas and Numpy.

1.2 Quickstart

Here, you can see some quick examples on how to utilize the package. For more details, refer to API Reference.

1.2.1 DiffChecker Basics

DiffChecker is designed to perform QA on data flows for ML. You can easily save statistics from the origin data such as missing value rate, mean, min/max, percentile, outliers, etc., then to compare against the new data. This is especially important if you want to keep the prediction data under the same assumptions with the training data.

Below is a quick example on how it works, just initiate and save statistics from the input data.

```
>>> from mlqa.identifiers import DiffChecker
>>> import pandas as pd
>>> dc = DiffChecker()
>>> dc.fit(pd.DataFrame({'mean_col':[1, 2]*50, 'na_col':[None]*50+[1]*50}))
```

Then, you can check on new data if it's okay for given criteria. Below, you can see some data that is very similar in column *mean_col* but increased NA count in column *na_col*. The default threshold is 0.5 which means it should be okay if NA rate is 50% more than the origin data. NA rate is 50% in the origin data so up to 75% (i.e. 50*(1+0.5)) should be okay. NA rate is 70% in the new data and, as expected, the QA passes.

```
>>> dc.check(pd.DataFrame({'mean_col':[.99, 2.1]*50, 'na_col':[None]*70+[1]*30}))
True
```

If you think the threshold is too loose, you can adjust as you wish with set_threshold method. And, now the same returns *False* indicating the QA has failed.

```
>>> dc.set_threshold(0.1)
>>> dc.check(pd.DataFrame({'mean_col':[.99, 2.1]*50, 'na_col':[None]*70+[1]*30}))
False
```

1.2.2 DiffChecker Details

As default, DiffChecker is initialized with *qa_level='loose'*. Different values can also be given.

```
>>> from mlqa.identifiers import DiffChecker
>>> dc = DiffChecker()
>>> dc.threshold
0.5
>>> dc = DiffChecker(qa_level='mid')
>>> dc.threshold
0.2
>>> dc = DiffChecker(qa_level='strict')
>>> dc.threshold
0.1
```

To be more precise, you can set both threshold and stats individually.

```
>>> import pandas as pd
>>> import numpy as np
>>> dc = DiffChecker()
>>> dc.set_threshold(0.2)
>>> dc.set_stats(['mean', 'max', np.sum])
>>> dc.fit(pd.DataFrame({'col1':[1, 2, 3, 4], 'col2':[1]*4}))
>>> dc.check(pd.DataFrame({'col1':[1, 2, 3, 4], 'col2':[0]*4}))
False
>>> dc.check(pd.DataFrame({'col1':[1, 2.1, 3.2, 4.2], 'col2':[1.1]*4}))
True
```

You can even be more detailed in set_threshold.

```
>>> dc = DiffChecker()
>>> dc.set_stats(['mean', 'max'])
>>> dc.set_threshold(0.1) # to reset all thresholds
>>> print(dc.threshold)
0.1
>>> dc.fit(pd.DataFrame({'col1':[1, 2, 3, 4], 'col2':[0]*4}))
>>> dc.set_threshold({'col1':0.2, 'col2':0.1}) # to set in column level
>>> print(dc.threshold_df)
      col1  col2
mean   0.2   0.1
max    0.2   0.1
>>> dc.set_threshold({'col1':{'mean':0.1}}) # to set in column-stat level
>>> print(dc.threshold_df)
      col1  col2
mean   0.1   0.1
max    0.2   0.1
```

You can also pickle the object to be used later with to_pickle method.

```
>>> dc1 = DiffChecker()
>>> dc1.fit(pd.DataFrame({'col1':[1, 2, 3, 4], 'col2':[0]*4}))
>>> dc1.to_pickle(path='DiffChecker.pkl')
```

Then, to load the same object later.

```
>>> import pickle
>>> pkl_file = open('DiffChecker.pkl', 'rb')
>>> dc2 = pickle.load(pkl_file)
>>> pkl_file.close()
```

1.2.3 DiffChecker with Logging

If you enable logging functionality, you can get detailed description of what column failed for which stat and why. You can even log DiffChecker steps.

Just initiate the class with *logger='<your-logger-name>.log'* argument.

```
>>> from mlqa.identifiers import DiffChecker
>>> import pandas as pd
>>> dc = DiffChecker(logger='mylog.log')
>>> dc.fit(pd.DataFrame({'mean_col':[1, 2]*50, 'na_col':[None]*50+[1]*50}))
>>> dc.set_threshold(0.1)
>>> dc.check(pd.DataFrame({'mean_col':[1, 1.5]*50, 'na_col':[None]*70+[1]*30}))
False
```

If you open *mylog.log*, you'll see something like below.

```
WARNING|2020-05-31 15:56:48,146|mean value (i.e. 1.25) is not in the range of [1.35,␣
↪1.65] for mean_col
WARNING|2020-05-31 15:56:48,147|na_rate value (i.e. 0.7) is not in the range of [0.45,
↪ 0.55] for na_col
```

If you initiate the class with also *log_info=True* argument, then the other class steps (e.g. set_threshold, check) would be logged, too.

Note: Although DiffChecker is able to create a Logger object by just passing a file name (i.e. *logger='mylog.log'*), creating the Logger object externally then passing accordingly (i.e. *logger=<mylogger>*) is highly recommended.

1.2.4 Checkers with Logging

There are also checkers to provide other kind of QA functionalities such as outliers detection, pd.DataFrame comparison or some categorical value QA. You can use these individually or combining with DiffChecker's logger.

Let's say you initiated DiffChecker with some logger already.

```
>>> from mlqa.identifiers import DiffChecker
>>> dc = DiffChecker(logger='mylog.log')
```

Then, you can just pass *logger* attribute of the object when calling checkers. Here is an example of qa_outliers.

```
>>> import mlqa.checkers as ch
>>> import numpy as np
>>> import pandas as pd
>>> np.random.seed(123)
>>> df = pd.DataFrame({
...     'col1':np.random.normal(0, 0.1, 100),
...     'col2':np.random.normal(0, 1.0, 100)})
>>> ch.qa_outliers(df, std=0.5, logger=dc.logger)
False
```

This should log something like below.

```
WARNING|2020-05-31 17:54:13,426|70 outliers detected within inlier range (i.e. [-0.
↪053985309527773806, 0.059407124225845764]) for col1
WARNING|2020-05-31 17:54:13,428|53 outliers detected within inlier range (i.e. [-0.
↪5070058315486367, 0.46793470772834406]) for col2
```

You can also compare multiple datasets from the same population with qa_df_set.

```
>>> df1 = pd.DataFrame({'col1':[1, 2]*10, 'col2':[0, 4]*10})
>>> df2 = pd.DataFrame({'col1':[1, 9]*10, 'col2':[0, -4]*10})
>>> ch.qa_df_set([df1, df2], logger=dc.logger)
False
```

This should log something like below.

```
INFO|2020-05-31 18:09:47,581|df sets QA initiated with threshold 0.1
WARNING|2020-05-31 18:09:47,598|mean of col1 not passed. Values are 1.5 and 5.0
WARNING|2020-05-31 18:09:47,599|mean of col2 not passed. Values are 2.0 and -2.0
WARNING|2020-05-31 18:09:47,599|std of col1 not passed. Values are 0.51299 and 4.10391
WARNING|2020-05-31 18:09:47,599|min of col2 not passed. Values are 0.0 and -4.0
WARNING|2020-05-31 18:09:47,599|25% of col2 not passed. Values are 0.0 and -4.0
WARNING|2020-05-31 18:09:47,599|50% of col1 not passed. Values are 1.5 and 5.0
WARNING|2020-05-31 18:09:47,600|50% of col2 not passed. Values are 2.0 and -2.0
WARNING|2020-05-31 18:09:47,600|75% of col1 not passed. Values are 2.0 and 9.0
WARNING|2020-05-31 18:09:47,600|75% of col2 not passed. Values are 4.0 and 0.0
WARNING|2020-05-31 18:09:47,600|max of col1 not passed. Values are 2.0 and 9.0
WARNING|2020-05-31 18:09:47,600|max of col2 not passed. Values are 4.0 and 0.0
INFO|2020-05-31 18:09:47,600|df sets QA done with threshold 0.1
```

For categorical values, you can check its distribution on a numeric column with qa_category_distribution_on_value.

```
>>> df1 = pd.DataFrame({'Gender': ['Male', 'Male', 'Female', 'Female'],
...                     'Weight': [200, 250, 100, 125]})
>>> ch.qa_category_distribution_on_value(df1,
...                                      'Gender',
...                                      {'Male':.5, 'Female':.5},
...                                      'Weight',
...                                      logger=dc.logger)
False
```

This should log something like below.

```
WARNING|2020-05-31 18:21:20,019|Gender distribution looks wrong, check Weight for␣
↪Gender=Male. Expected=0.5, Actual=0.6666666666666666
WARNING|2020-05-31 18:21:20,019|Gender distribution looks wrong, check Weight for␣
↪Gender=Female. Expected=0.5, Actual=0.3333333333333333
```

Note: Although DiffChecker is able to create a Logger object by just passing a file name (i.e. *logger='mylog.log'*), creating the Logger object externally then passing accordingly (i.e. *logger=<mylogger>*) is highly recommended.

CHAPTER

TWO

API REFERENCE

2.1 mlqa.checkers

2.2 mlqa.identifiers

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search