Market Mix Modeling

Release 0.6.1

Jun 30, 2022

Contents

CHAPTER 1

Overview

docs	
tests	
package	

Market Mix Modeling

- Free software: GNU Lesser General Public License v3 or later (LGPLv3+)

1.1 Installation

```
pip install mrktmix
```

You can also install the in-development version with:

```
pip install https://github.com/540pd/mrktmix/archive/master.zip
```

1.2 Documentation

https://mrktmix.readthedocs.io/

1.3 Development

To run all the tests run:

```
tox
```

Note, to combine the coverage data from all the tox environments run:

Windows	`set PYTEST_ADDOPTS=--cov-append` `tox`
Other	`PYTEST_ADDOPTS=--cov-append tox`

CHAPTER 2

Installation

At the command line:

```
pip install mrktmix
```

To use optimization function, one needs to install Numpy+MKL. Please follow installation process on Install Python 3.6 with Numpy + MKL, SciPy , Matplotlib and Pandas on Windows. Download and install Numpy+MKL. Ater installing Numpy+MKL, install gpkit:

```
pip install gpkit
```

CHAPTER 3

Quick Start

Assuming you have Python already, install mrktmix:

```
import pandas as pd
import numpy as np
import mrktmix as mmm
```

Modeling dataset is tabular format dataframe with variable in column and panel in row index. To create modeling dataset from long format:

```
# Create long format data
input_df1={'Panel': {0: 'panel1', 1: 'panel1', 2: 'panel2', 3: 'panel2', 4: 'panel2', 5: 'panel3', 6: 'panel3'},
           'Channel': {0: 'TV_Free', 1: 'TV_Free', 2: 'TV_Free', 3: 'TV_Free', 4: 'TV_Free', 5: 'TV_Free', 6: 'TV_Free'},
           'Metric': {0: 'GRp', 1: 'Spend', 2: 'GRP', 3: 'SpenD', 4: 'Spend', 5: 'gRP', 6: 'gRP'},
           'Metric_Value': {0: 33, 1: 102, 2: 45, 3: 129, 4: 170, 5: 24, 6: 49}}
input_df1=pd.DataFrame.from_dict(input_df1)
input_df2={'Panel': {0: 'panel1', 1: 'panel1', 2: 'panel2', 3: 'panel2', 4: 'panel2', 5: 'panel3', 6: 'panel3'},
           'Channel': {0: 'TV_Free', 1: 'TV_Free', 2: 'TV_Free', 3: 'TV_Free', 4: 'TV_Free', 5: 'TV_Free', 6: 'TV_Free'},
           'Metric': {0: 'GRp', 1: 'Spend', 2: 'GRP', 3: 'SpenD', 4: 'Spend', 5: 'gRP', 6: 'gRP'},
           'Metric_Value': {0: 33, 1: 102, 2: 45, 3: 129, 4: 170, 5: 24, 6: 49}}
input_df2=pd.DataFrame.from_dict(input_df2)
input_files={"source1":input_df1, "source2":input_df2}
# Create modeling data
mdl_data, code_mapping, file_mapping = mmm.create_mdldata(input_files, ['Panel'], ["Channel", "Metric"] ,"Metric_Value", description2code={'GRP':'GRP',"Spend":'SPD',"TV_Free":"TV"})
```

To imput missing value where non missing value represents sum of preceding missing values:

```
# Spreads non missing values to missing values
input_df=pd.DataFrame([[34., np.nan], [np.nan, np.nan], [np.nan,  6.], [30., np.nan],␣
↪[13., np.nan], [np.nan, np.nan], [20.,  7.], [np.nan, np.nan], [40., np.nan]],␣
↪columns=["Spend","Volume"])
mmm.spread_notna(input_df1)
```

While building models, one often have to create baseline which represents unexplained or exogenous part of response variable. To crease base for model:

```
# Create base for dates specified i.e. date input is list
mmm.create_base("var1", ["2020-03-10", "2020-04-07"], "7D", increasing=False,␣
↪negative=False, panel=None))

# Create base for given range of dates i.e. date input is tuple
mmm.create_base("var1", ("2020-03-10", "2020-03-21"), "7D", increasing=False,␣
↪negative=False, panel=None))

# Create base from given date i.e. date input is string
mmm.create_base("var1", "2020-03-10", "7D", increasing=False, negative=False,␣
↪periods=3, panel=None))

# Create base from pandas series
df = pd.DataFrame([["var2", ["2020-03-10", "2020-04-07"]], ["var2", ("2020-03-10",
↪"2020-04-07")], ["var3", ("2020-03-10", "2020-04-06")], ["var4", "2020-03-10"]])
mmm.create_base(df[0].values, df[1].values, "7D", increasing=False, negative=False,␣
↪periods=5, panel=["panel1", "panel1", "panel2", "panel1"]))
```

To apply adstock, power and lag transformation, input data must have sorted date in level -1 of row multiindex. Additional hiearachy like panel can be present in row multiindex. If there is only date index in data, key of dictionary input must be False. If additional hiearchy is present in data like panel, then key of dictionary input can be name of panels if transformation is applicable to selected panel or True if tranformation is applicable to all the panels present in data:

```
# Apply adstock, power and lag transformation when there is no panel in input data i.
↪e. row index have date index only
variables = {False: [('Intercept', 0, 1, 0), ('one', 0, 1, 0), ('two', 0, 1, 0), ('one
↪', 0, 1, 1)]}
df_1 = pd.DataFrame({'two': [1., 2., 3., 4.], 'one': [4., 3., 2., 1.], 'Intercept':␣
↪[4., 3., 2., 1.]}, index=['2020-01-01', '2020-01-02', '2020-01-03', '2020-01-04'])
mmm.apply_apl(df_1, variables)
```

Apply modeling/regression coefficient and create decomposition of response variable:

```
# Create model coefficient
ind = [np.repeat(["METRO","CITY"], 2), [("Intercept", 0, 1, 0), ("A", 0, 1, 0), (
↪"Intercept", 0, 1, 0), ("B", 0, 1, 0)]]
coef= pd.Series(data=[3240, 600, 10, 0.9], index=ind)
# Create data
df_ind = [np.repeat(["METRO","CITY"], 3), np.tile(pd.date_range(start='1/1/2018', end=
↪'1/03/2018'),2)]
df = pd.DataFrame({'A': [773, 137, 508, 562, 365, 500], 'B': [848, 326, 969, 730, 761,
↪ 137]}, index=df_ind)
df["Intercept"] = 1
# Create response data
dep = pd.DataFrame({'Dep': [773, 137, 508, 562, 365, 500]}, index=df_ind)
# Apply coefficient
mmm.apply_coef(df, coef, dep["Dep"])
```

Summarize decomposition of response variable:

```
# Summarise decomposition of response variables based on date filter
dep_decompose=apply_coef(df, coef, dep["Dep"])
date_dict = {'test Year': ('3/1/2018', '3/1/2018'), 'train_year': ('1/1/2018', '2/1/
↪2018')}
mmm.collapse_date(dep_decompose, date_dict)
```

Compare response series and predicted series along with errors from decomposition of response variable:

```
# Create response variable and predicted series
mmm.assess_error(apply_coef(df, coef, dep["Dep"]))
```

Optimize spend based on revenue where $Revenue = \sum_{i=0}^{N} Coefficient_i * Spend_i^{Power_i}$. Basic input parameters are coefficient, intial spend, exponent and contraints amount. Constraints can be applied on spend for spend based optimization or revenue for goal based optimization. In addition to these, one can also apply additional contrains like lower and upper bound for spend:

```
optimized_spend, optimized_revenue, optimized_status = mmm.mmm_optimize(
            [47, 75, 13, 63, 96, 25, 17],
            [806, 332, 173, 661, 286, 253, 978],
            [0.9 , 0.32, 0.97, 0.53, 0.02, 0.86, 0.67],
            3489,
            contraint_type="budget",
            lower_bound=[644.8, 265.6, 138.4, 528.8, 228.8, 202.4, 782.4],
            upper_bound=[967.2, 398.4, 207.6, 793.2, 343.2, 303.6, 1173.6])
# Optimized Spend
optimized_spend
# Optimized Revenue
optimized_revenue
# Optimized Status whether the algorithm converged or not
optimized_status
```

CHAPTER 4

Reference

4.1 mrktmix

CHAPTER 5

Contributing

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

5.1 Bug reports

When 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.

5.2 Documentation improvements

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

5.3 Feature requests and feedback

The best way to send feedback is to file an issue at https://github.com/540pd/mrktmix/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 code contributions are welcome :)

5.4 Development

To set up *mrktmix* for local development:

1. Fork mrktmix (look for the "Fork" button).
2. Clone your fork locally:

```
git clone git@github.com:YOURGITHUBNAME/mrktmix.git
```

3. Create a branch for local development:

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

   Now you can make your changes locally.

4. When you're done making changes run all the checks and docs builder with tox one command:

```
tox
```

5. 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
```

6. Submit a pull request through the GitHub website.

5.4.1 Pull Request Guidelines

If you need some code review or feedback while you're developing the code just make the pull request.

For merging, you should:

1. Include passing tests (run `tox`)[1].
2. Update documentation when there's new API, functionality etc.
3. Add a note to `CHANGELOG.rst` about the changes.
4. Add yourself to `AUTHORS.rst`.

5.4.2 Tips

To run a subset of tests:

```
tox -e envname -- pytest -k test_myfeature
```

To run all the test environments in *parallel*:

```
tox -p auto
```

[1] If you don't have all the necessary python versions available locally you can rely on Travis - it will run the tests for each change you add in the pull request.
It will be slower though ...

CHAPTER 6

Authors

- Abhimanyu Prasad Mahato - https://www.linkedin.com/in/abhimanyu-mahato-540/

CHAPTER 7

Changelog

7.1 0.0.1 (2020-09-06)

- First release on PyPI.

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

Python Module Index

m

Index

M