mdmpy Documentation

Release 0.0.15.9

mdmpy Authors

Feb 21, 2019

Contents:

Please note that this is a work in progress.

To solve the Maximum Likelihood problem in Marginal Distribution Models, this package leverages on Pyomo for interfacing with solvers and NumPy for calculations of a gradient based method.

CHAPTER 1

Installation

pip can be used to install *mdmpy*.

```
pip install mdmpy
```

CHAPTER 2

Overview

We follow *[MNP+14]* and wish to solve the following Maximum Likelihood problem:

$$\begin{aligned} \max_{\mathbf{\Lambda},\mathbf{B}} \; & \sum_{i\in\mathcal{I}} \sum_{k\in\mathcal{K}} z_{ik} \ln\left(1 - F_{ik}\left(\lambda_i - \mathbf{B}'\mathbf{x_{ik}}\right)\right) \\ \text{s.t.} \; & \sum_{k\in\mathcal{K}} \left(1 - F_{ik}\left(\lambda_i - \mathbf{B}'\mathbf{x_{ik}}\right)\right) = 1, \quad \forall i \in \mathcal{I} \end{aligned}$$

Note: Due to limitations with MathJax, Greek lower-case letters cannot be bolded. Hence, we use upright Greek bolded letters to denote vectors.

To do so, we can use the `mdmpy.mdm.MDM` class, which includes a method for initialising the model in *Pyomo*, and another to solve it. In the next section, we will explore a few examples to see this in action.

CHAPTER 3

Examples

3.1 Travel Mode Choice

We look at the Travel Choice data as examined in Chapter 18 of *[Gre11]*.

3.1.1 Preprocessing

We will use `requests` to get the dataset, and then process it with `csv` and `pandas`.

```
import csv
import requests
import pandas as pd
import numpy as np

url = "http://pages.stern.nyu.edu/~wgreene/Text/Edition7/TableF18-2.csv"
# Preallocate 840 rows and 7 columns, which is the size of the data
# We will skip the header row
df = pd.DataFrame(index=np.arange(0, 840), columns=np.arange(0,7))
with requests.Session() as s:
    download = s.get(url)
    cr = csv.reader(download.content.decode().splitlines())
    next(cr) # skip header
    for ix, row in enumerate(cr):
        df.loc[ix] = [int(x) for x in row]
```

3.1.2 Solving for the MLE

We can initialise the model using the `model_init()` method.

```
import mdmpy
mdm0 = mdmpy.MDM(df,    # input dataframe
```

(continues on next page)

(continued from previous page)

```
                    0,    # the choice is index 0
                    4,    # there are 4 possible choices
                    [2,3])
mdm0.model_init()
```

Assuming that the *IPOPT* solver is in *PATH*, we can choose to simply put *"IPOPT"* as an argument to the `model_solve()` method.

```
mdm0.model_solve("ipopt")
beta0 = [mdm0.m.beta[idx].value for idx in mdm0.m.beta]
print(beta0)
```

We can also get the loglikelihood of such an outcome.

```
ll0 = mdm0.ll(beta0)
print(ll0)
```

Alternatively, we can use all of the data of the choices. Since we are not changing much, we can define it as a function.

```
def print_beta_vals(attr_col_indices):
    mdm = mdmpy.MDM(df,    # input dataframe
                    0,    # the choice is index 0
                    4,    # there are 4 possible choices
                    attr_col_indices # now instead we use all 4 choice-specific-data␣
↪columns
                    )
    mdm.model_init()
    mdm.model_solve("ipopt")
    beta = [mdm.m.beta[idx].value for idx in mdm.m.beta]
    print(beta)
    ll = mdm.ll(beta)
    print(ll)

print_beta_vals([1, 2, 3, 4])
```

3.1.3 Full Code

```
import csv
import requests
import pandas as pd
import numpy as np
import mdmpy

url = "http://pages.stern.nyu.edu/~wgreene/Text/Edition7/TableF18-2.csv"

df = pd.DataFrame(index=np.arange(0, 840), columns=np.arange(0,7))
with requests.Session() as s:
    download = s.get(url)
    cr = csv.reader(download.content.decode().splitlines())
    next(cr) # skip header
    for ix, row in enumerate(cr):
        df.loc[ix] = [int(x) for x in row]

def print_beta_vals(attr_col_indices):
```

(continues on next page)

(continued from previous page)

```
    mdm = mdmpy.MDM(df,    # input dataframe
                    0,     # the choice is index 0
                    4,     # there are 4 possible choices
                    attr_col_indices # now instead we use all 4 choice-specific-data␣
↪columns
                    )
    mdm.model_init()
    mdm.model_solve("ipopt")
    beta = [mdm.m.beta[idx].value for idx in mdm.m.beta]
    print(beta)
    ll = mdm.ll(beta)
    print(ll)

print_beta_vals([2, 3])

print_beta_vals([1, 2, 3, 4])
```

CHAPTER 4

mdmpy

4.1 mdmpy package

4.1.1 Submodules

4.1.2 mdmpy.mdm module

This module contains the main class, MDM, in this package.

Within the class are methods related to solving for the Maximum Likelihood Estimate (MLE) of coefficients. Two methods are used to initialise a model using pyomo, which can then be passed onto a solver to be solved. Another way to solve for the coefficients is to use a gradient ascent.

class mdmpy.mdm.**MDM**(*input_dataframe*, *ch_id: int*, *num_choices: int*, *list_of_coefs*, *input_cdf=<function exp_cdf>*, *input_pdf=<function exp_pdf>*)

Bases: object

The main class of the package. This models the Marginal Distribution Models (MDM)

add_conv(*conv_min: float = 0*)

This function restricts the argument of the CDF and PDF such that they are above a set limit, commonly zero. This restricts the domain to a region whereby the 1-CDF is convex.

grad_desc(*initial_beta*, *max_steps: int = 50*, *grad_mult=1*, *eps: float = 1e-07*, *verbosity=0*)

Starts a gradient-descent based method using the CDF and PDF. Requires a starting beta iterate.

TODO: to add f_arg_min which will be pass onto the gradient calculators and use grad_lambda_beta to move towards a convex region, which is above f_arg_min

TODO: Add some sort of linesearch method. That is, a method that uses a direction and tries varies stepsizes to check what happens to the loglikelihood with those stepsizes and uses some rules or heuristics to decide which stepsize to choose

ll(*input_beta*, *corr_lambs=None*) → float

This function gets the log-likelihood using the current beta. If the corresponding lambdas for each individual are given, then it will use those, rather than re-computing them, which saves computations

model_init(*heteroscedastic=False*, *alpha_to_be_fixed=0*, *alpha_cons_limits=(0, 1)*, *use_ASCs=False*, *ASC_fixed_to_zero=None*, *model_seed=None*, *v=0*)
This method initializes the pyomo model as an instance attribute m. Values can later be taken directly from m.

The various keyword arguments of this method are used to customise the resulting model. They include alternative-specific constants (ASCs) which act as y-intercepts depending on which choice was made.

Another keyword argument is involved in deciding whether the model will handle heteroscedasticity.

model_solve(*solver*, *solver_exec_location=None*, *tee: bool = False*, ***kwargs*)
Start a solver to solve the model

4.1.3 mdmpy.util module

This module contains utility functions which are called by the main class.

mdmpy.util.**default_bisect_func**(*input_cdf*, *input_beta*, *input_x*, *lambda_: float*) → float
This is the default bisection function. The last input will be varied during the bisection search using partial from functools.

mdmpy.util.**exp_cdf**(*x*, *lambda_: float = 1*)
Exponential Cumulative Distribution Function (CDF), with default parameter 1

mdmpy.util.**exp_pdf**(*x*, *lambda_: float = 1*)
Exponential Probability Density Function (PDF), with default parameter 1

mdmpy.util.**find_corresponding_lambda**(*input_cdf*, *input_beta*, *input_x*, *bisect_func=<function default_bisect_func>*, *lamb_const: float = 50000*, *max_lamb_retries: int = 1000*, *lamb_coef: float = 1.4*) → float
This function is called to find lambda given the model, input beta and input x_i. It does this by first using a large number which may be too big and causes overflows, but then reduces the positive and negative parts separately until the valid gives a valid output. Then, with a positive output and a negative output, a bisection search for the root is performed.

mdmpy.util.**gumbel_cdf**(*x*, *beta: float = 1*)
Gumbel Cumulative Distribution Function (CDF), with default parameter 1

mdmpy.util.**gumbel_pdf**(*x*, *beta: float = 1*)
Gumbel Probability Density Function (PDF), with default parameter 1

4.1.4 Module contents

This package revolves around the implementation of Marginal Distribution Models, which is implemented as a class MDM

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Bibliography

[MNP+14] Vinit Kumar Mishra, Karthik Natarajan, Dhanesh Padmanabhan, Chung-Piaw Teo, and Xiaobo Li. On theoretical and empirical aspects of marginal distribution choice models. *Management Science*, 60(6):1511–1531, June 2014. URL: https://doi.org/10.1287/mnsc.2014.1906, doi:10.1287/mnsc.2014.1906.

[Gre11] William H. Greene. *Econometric Analysis (7th Edition)*. Pearson, 2011. ISBN 0131395386. URL: https://www.amazon.com/Econometric-Analysis-7th-William-Greene/dp/0131395386.

Python Module Index

m

Index