plspm Documentation

Release 0.4

Google LLC

Aug 06, 2020

Contents:

PLSPM: A library implementing Partial Least Squares Path Modeling .. image:: https://badge.fury.io/py/plspm.svg

target https://badge.fury.io/py/plspm

alt PyPI version

Please note: This is not an officially supported Google product.

plspm is a Python 3 package dedicated to Partial Least Squares Path Modeling (PLS-PM) analysis. It is a port of the R package plspm, with additional features adopted from the R package seminr

PLSPM (partial least squares path modeling) is a correlation-based structural equation modeling (SEM) algorithm. It allows for estimation of complex cause-effect or prediction models using latent/manifest variables.

PLSPM may be preferred to other SEM methods for several reasons: it is a method that is appropriate for exploratory research, can be used with small-to-medium sample sizes (as well as large data sets), and does not require assumptions of multivariate normality. (See Hulland, J. (1999). Use of partial least squares (PLS) in strategic management research: a review of four recent studies. Strategic management journal, 20(2), 195-204.) In contrast to covariance-based SEM (CBSEM), goodness of fit is less important, because the purpose of the algorithm is to optimize for prediction of the dependent variable vs. fit of data to a predetermined model. (See "goodness of fit" vs "goodness of model" in Chin, W. W. (2010). How to write up and report PLS analyses. In Handbook of partial least squares (pp. 655-690). Springer, Berlin, Heidelberg.)

- Uses variance-based PLS esimation to model composite constructs using Mode A and Mode B
- Uses a natural-feeling, domain specific language to build and estimate structural equation models, including second-order constructs
- Supports centroid, factorial, and path schemes
- Supports metric and non-metric numerical data (including nominal and ordinal)
- Handles missing data
- Bootstrapping with multi-core support
- Tested against seminr, which is, in turn, tested against SmartPLS (Ringle et al., 2015) and ADANCO (Henseler and Dijkstra, 2015), as well as other R packages such as semPLS (Monecke and Leisch, 2012) and matrixpls (Rönkkö, 2016).

CHAPTER 1

Planned but not yet implemented

- Native modeling of moderation
- Improved assessment measures such as HTMT, VIF, f^2, Q^2, and q^2
- Modeling formative constructs using the PLS consistent (PLSc) algorithm

1.1 Installation

You can install the latest version of this package using pip:

```
python3 -m pip install --user plspm
```

It's hosted on pypi: https://pypi.org/project/plspm/

1.2 Use

plspm expects to get a Pandas DataFrame containing your data. You start by creating a Config object with the details of the model, and then pass it, along with the data and optionally some further configuration, to an instance of Plspm. Use the examples below to get started, or browse the documentation (start with Config and Plspm)

1.3 Examples

CHAPTER 2

PLS-PM with metric data

Typical example with a Customer Satisfaction Model

```
#!/usr/bin/env python3
import pandas as pd, plspm.config as c
from plspm.plspm import Plspm
from plspm.scheme import Scheme
from plspm.mode import Mode

satisfaction = pd.read_csv("file:tests/data/satisfaction.csv", index_col=0)

structure = c.Structure()
structure.add_path(["IMAG"], ["EXPE", "SAT", "LOY"])
structure.add_path(["EXPE"], ["QUAL", "VAL", "SAT"])
structure.add_path(["QUAL"], ["VAL", "SAT"])
structure.add_path(["VAL"], ["SAT"])
structure.add_path(["SAT"], ["LOY"])

config = c.Config(structure.path(), scaled=False)
config.add_lv_with_columns_named("IMAG", Mode.A, satisfaction, "imag")
config.add_lv_with_columns_named("EXPE", Mode.A, satisfaction, "expe")
config.add_lv_with_columns_named("QUAL", Mode.A, satisfaction, "qual")
config.add_lv_with_columns_named("VAL", Mode.A, satisfaction, "val")
config.add_lv_with_columns_named("SAT", Mode.A, satisfaction, "sat")
config.add_lv_with_columns_named("LOY", Mode.A, satisfaction, "loy")

plspm_calc = Plspm(satisfaction, config, Scheme.CENTROID)
print(plspm_calc.inner_summary())
print(plspm_calc.path_coefficients())
```

This will produce the output:

```
            type  r_squared  block_communality  mean_redundancy       ave
EXPE  Endogenous   0.335194           0.616420         0.206620  0.616420
IMAG   Exogenous   0.000000           0.582269         0.000000  0.582269
```

(continues on next page)

(continued from previous page)

```
LOY   Endogenous   0.509923          0.639052         0.325867  0.639052
QUAL  Endogenous   0.719688          0.658572         0.473966  0.658572
SAT   Endogenous   0.707321          0.758891         0.536779  0.758891
VAL   Endogenous   0.590084          0.664416         0.392061  0.664416

          IMAG      EXPE      QUAL       VAL       SAT  LOY
IMAG  0.000000  0.000000  0.000000  0.000000  0.000000    0
EXPE  0.578959  0.000000  0.000000  0.000000  0.000000    0
QUAL  0.000000  0.848344  0.000000  0.000000  0.000000    0
VAL   0.000000  0.105478  0.676656  0.000000  0.000000    0
SAT   0.200724 -0.002754  0.122145  0.589331  0.000000    0
LOY   0.275150  0.000000  0.000000  0.000000  0.495479    0
```

CHAPTER 3

Specifying higher-order constructs

Example using seminr's mobile industry data set:

```
mobi = pd.read_csv("file:tests/data/mobi.csv", index_col=0)

structure = c.Structure()
structure.add_path(["Expectation", "Quality"], ["Satisfaction"])
structure.add_path(["Satisfaction"], ["Complaints", "Loyalty"])

config = c.Config(structure.path(), default_scale=Scale.NUM)
config.add_higher_order("Satisfaction", Mode.A, ["Image", "Value"])
config.add_lv_with_columns_named("Expectation", Mode.A, mobi, "CUEX")
config.add_lv_with_columns_named("Quality", Mode.B, mobi, "PERQ")
config.add_lv_with_columns_named("Loyalty", Mode.A, mobi, "CUSL")
config.add_lv_with_columns_named("Image", Mode.A, mobi, "IMAG")
config.add_lv_with_columns_named("Complaints", Mode.A, mobi, "CUSCO")
config.add_lv_with_columns_named("Value", Mode.A, mobi, "PERV")

mobi_pls = Plspm(mobi, config, Scheme.PATH, 100, 0.00000001)

print(plspm_calc.inner_model())
```

This will produce the output:

```
                                  from            to  estimate  std error          ⌐
↪t         p>|t|
index
Quality -> Satisfaction           Quality  Satisfaction  0.743041   0.046318  16.
↪042102  3.633866e-40
Expectation -> Satisfaction   Expectation  Satisfaction  0.089572   0.046318   1.
↪933832  5.427626e-02
Satisfaction -> Loyalty      Satisfaction       Loyalty  0.627940   0.049420  12.
↪706272  7.996788e-29
Satisfaction -> Complaints   Satisfaction    Complaints  0.486696   0.055472   8.
↪773752  2.841768e-16
```

CHAPTER 4

PLS-PM with nonmetric data

Example with the classic Russett data (original data set)

```
#!/usr/bin/env python3
import pandas as pd, plspm.config as c
from plspm.plspm import Plspm
from plspm.scale import Scale
from plspm.scheme import Scheme
from plspm.mode import Mode

russa = pd.read_csv("file:tests/data/russa.csv", index_col=0)

structure = c.Structure()
structure.add_path(["AGRI", "IND"], ["POLINS"])

config = c.Config(structure.path(), default_scale=Scale.NUM)
config.add_lv("AGRI", Mode.A, c.MV("gini"), c.MV("farm"), c.MV("rent"))
config.add_lv("IND", Mode.A, c.MV("gnpr"), c.MV("labo"))
config.add_lv("POLINS", Mode.A, c.MV("ecks"), c.MV("death"), c.MV("demo"), c.MV("inst
↪"))

plspm_calc = Plspm(russa, config, Scheme.CENTROID, 100, 0.0000001)
print(plspm_calc.inner_summary())
print(plspm_calc.effects())
```

This will produce the output:

```
              type  r_squared  block_communality  mean_redundancy       ave
AGRI     Exogenous   0.000000           0.739560         0.000000  0.739560
IND      Exogenous   0.000000           0.907524         0.000000  0.907524
POLINS  Endogenous   0.592258           0.565175         0.334729  0.565175

   from      to    direct  indirect     total
0  AGRI  POLINS  0.225639       0.0  0.225639
1   IND  POLINS  0.671457       0.0  0.671457
```

PLS-PM using data set `russa`, and different scaling

```
#!/usr/bin/python3
import pandas as pd, plspm.config as c, plspm.util as util
from plspm.plspm import Plspm
from plspm.scale import Scale
from plspm.scheme import Scheme
from plspm.mode import Mode

russa = pd.read_csv("file:tests/data/russa.csv", index_col=0)

structure = c.Structure()
structure.add_path(["AGRI", "IND"], ["POLINS"])
config = c.Config(structure.path(), default_scale=Scale.NUM)
config.add_lv("AGRI", Mode.A, c.MV("gini"), c.MV("farm"), c.MV("rent"))
config.add_lv("IND", Mode.A, c.MV("gnpr", Scale.ORD), c.MV("labo", Scale.ORD))
config.add_lv("POLINS", Mode.A, c.MV("ecks"), c.MV("death"), c.MV("demo", Scale.NOM),␣
↪c.MV("inst"))

plspm_calc = Plspm(russa, config, Scheme.CENTROID, 100, 0.0000001)
```

```
#!/usr/bin/env python3
import pandas as pd, plspm.config as c
from plspm.plspm import Plspm
from plspm.scale import Scale
from plspm.scheme import Scheme
from plspm.mode import Mode

russa = pd.read_csv("file:tests/data/russa.csv", index_col=0)
russa.iloc[0, 0] = np.NaN
russa.iloc[3, 3] = np.NaN
russa.iloc[5, 5] = np.NaN

structure = c.Structure()
structure.add_path(["AGRI", "IND"], ["POLINS"])
config = c.Config(structure.path(), default_scale=Scale.NUM)
config.add_lv("AGRI", Mode.A, c.MV("gini"), c.MV("farm"), c.MV("rent"))
config.add_lv("IND", Mode.A, c.MV("gnpr"), c.MV("labo"))
config.add_lv("POLINS", Mode.A, c.MV("ecks"), c.MV("death"), c.MV("demo"), c.MV("inst
↪"))

plspm_calc = Plspm(russa, config, Scheme.CENTROID, 100, 0.0000001)
```

4.1 Maintainers

Jez Humble (humble at google.com)

Nicole Forsgren (nicolefv at google.com)

4.1.1 API Documentation

The two classes below are the main ones you'll need to use to perform the calculations. Anything else you need is linked from the documentation for these two classes. Documentation for all modules can be found in the modindex.

- `plspm.config.Config` - specify the model.

- `plspm.plspm.Plspm` - perform the computation.

See the examples above for more on how to use this library.

4.1.2 Indices and tables

- genindex
- modindex
- search