factor_analyzer Documentation

Release 0.2.2

Jeremy Biggs

Jun 07, 2018

Contents:

This is Python module to perform exploratory factor analysis, with optional varimax and promax rotations. Estimation can be performed using a minimum residual (minres) solution, or maximum likelihood estimation (MLE).

Portions of this code are ported from the excellent R library psych.

CHAPTER 1

Description

Exploratory factor analysis (EFA) is a statistical technique used to identify latent relationships among sets of observed variables in a dataset. In particular, EFA seeks to model a large set of observed variables as linear combinations of some smaller set of unobserved, latent factors.

The matrix of weights, or factor loadings, generated from an EFA model describes the underlying relationships between each variable and the latent factors. Typically, a number of factors (K) is selected such that is substantially smaller than the number of variables. The factor analysis model can be estimated using a variety of standard estimation methods, including but not limited to OLS, minres, or MLE.

This package includes a stand-alone Python module with a `FactorAnalyzer()` class. The class includes an `analyze()` method that allows users to perform factor analysis using either minres or MLE, with optional rotations on the factor loading matrices. The package also offers a stand-alone `Rotator()` class to perform common rotations on an unrotated loading matrix.

The `factor_analyzer` package offers the following rotation methods:

- The **varimax** (orthogonal)
- The **promax** (oblique)
- The **quartimax** (orthogonal)
- The **quartimin** (oblique)
- The **obliimax** (orthogonal)
- The **oblimin** (oblique)
- The **obliimax** (orthogonal)
- The **equamax** (orthogonal)

This package includes a stand-alone Python module with a `FactorAnalyzer()` class. The class includes an `analyze()` method that allows users to perform factor analysis using either minres or MLE, with optional promax or varimax rotations on the factor loading matrices. The package also offers a stand-alone `Rotator()` class to perform common rotations on an unrotated loading matrix.

CHAPTER 2

Requirements

- Python 3.4 or higher
- `numpy`
- `pandas`
- `scipy`

CHAPTER 3

Installation

You can install this package via `pip` with:

```
$ pip install factor_analyzer
```

Alternatively, you can install via `conda` with:

```
$ conda install -c desilinguist factor_analyzer
```

3.1 factor_analyzer package

3.1.1 `factor_analyzer.analyze` Module

Factor analysis command line script.

author Jeremy Biggs (jbiggs@ets.org)

date 12/13/2017

organization ETS

`factor_analyzer.analyze.`**`main`**`()`

Run the script.

3.1.2 `factor_analyzer.factor_analyzer` Module

Factor analysis using MINRES or ML, with optional rotation using Varimax or Promax.

author Jeremy Biggs (jbiggs@ets.org)

date 10/25/2017

organization ETS

class `factor_analyzer.factor_analyzer.`**`FactorAnalyzer`**(*log_warnings=False*)

Bases: `object`

A FactorAnalyzer class, which -

1. Fits a factor analysis model using minres or maximum likelihood, and returns the loading matrix
2. Optionally performs a rotation, with method including:
   (a) varimax (orthogonal rotation)
   (b) promax (oblique rotation)
   (c) oblimin (oblique rotation)
   (d) oblimax (orthogonal rotation)
   (e) quartimin (oblique rotation)
   (f) quartimax (orthogonal rotation)
   (g) equamax (orthogonal rotation)

Parameters **log_warnings** (*bool*) – Whether to log warnings, such as failure to converge. Defaults to False.

loadings
pd.DataFrame – The factor loadings matrix. Default to None, if *analyze()* has not been called.

corr
pd.DataFrame – The original correlation matrix. Default to None, if *analyze()* has not been called.

rotation_matrix
np.array – The rotation matrix, if a rotation has been performed.

Notes

This code was partly derived from the excellent R package *psych*.

References

[1] https://github.com/cran/psych/blob/master/R/fa.R

Examples

```
>>> import pandas as pd
>>> from factor_analyzer import FactorAnalyzer
>>> df_features = pd.read_csv('test02.csv')
>>> fa = FactorAnalyzer()
>>> fa.analyze(df_features, 3, rotation=None)
>>> fa.loadings
           Factor1   Factor2   Factor3
sex      -0.129912 -0.163982  0.738235
zygosity  0.038996 -0.046584  0.011503
moed      0.348741 -0.614523 -0.072557
faed      0.453180 -0.719267 -0.075465
faminc    0.366888 -0.443773 -0.017371
english   0.741414  0.150082  0.299775
math      0.741675  0.161230 -0.207445
socsci    0.829102  0.205194  0.049308
```

(continues on next page)

(continued from previous page)

```
natsci    0.760418  0.237687 -0.120686
vocab     0.815334  0.124947  0.176397
```

analyze (*data*, *n_factors=3*, *rotation='promax'*, *method='minres'*, *use_smc=True*, *bounds=(0.005, 1)*, *normalize=True*, *impute='median'*, ***kwargs*)

Fit the factor analysis model using either minres or ml solutions. By default, use SMC as starting guesses and perform Kaiser normalization.

Parameters

- **data** (*pd.DataFrame*) – The data to analyze.
- **n_factors** (*int, optional*) – The number of factors to select. Defaults to 3.
- **rotation** (*str, optional*) – The type of rotation to perform after fitting the factor analysis model. If set to None, no rotation will be performed, nor will any associated Kaiser normalization.

  Methods include:

  1. varimax (orthogonal rotation)
  2. promax (oblique rotation)
  3. oblimin (oblique rotation)
  4. oblimax (orthogonal rotation)
  5. quartimin (oblique rotation)
  6. quartimax (orthogonal rotation)
  7. equamax (orthogonal rotation)

  Defaults to 'promax'.
- **method** (*{'minres', 'ml'}, optional*) – The fitting method to use, either MINRES or Maximum Likelihood. Defaults to 'minres'.
- **use_smc** (*bool, optional*) – Whether to use squared multiple correlation as starting guesses for factor analysis. Defaults to True.
- **bounds** (*tuple, optional*) – The lower and upper bounds on the variables for "L-BFGS-B" optimization. Defaults to (0.005, 1).
- **normalize** (*bool, optional*) – Whether to perform Kaiser normalization and de-normalization prior to and following rotation. Defaults to True.
- **impute** (*{'drop', 'mean', 'median'}, optional*) – If missing values are present in the data, either use list-wise deletion ('drop') or impute the column median ('median') or column mean ('mean'). Defaults to 'median'.
- **optional** (*kwargs,*) – Additional key word arguments are passed to the rotation method.

Raises

- ValueError – If rotation not *None* or in *POSSIBLE_ROTATIONS*.
- ValueError – If missing values present and *missing_values* is not set to either 'drop' or 'impute'.

Notes

varimax is an orthogonal rotation, while promax is an oblique rotation. For more details on promax rotations, see here:

References

[1] https://www.rdocumentation.org/packages/psych/versions/1.7.8/topics/Promax

fit_factor_analysis (*data*, *n_factors*, *use_smc=True*, *bounds=(0.005, 1)*, *method='minres'*)
Fit the factor analysis model using either minres or ml solutions.

Parameters

- **data** (*pd.DataFrame*) – The data to fit.
- **n_factors** (*int*) – The number of factors to select.
- **use_smc** (*bool*) – Whether to use squared multiple correlation as starting guesses for factor analysis. Defaults to True.
- **bounds** (*tuple*) – The lower and upper bounds on the variables for "L-BFGS-B" optimization. Defaults to (0.005, 1).
- **method** (*{'minres', 'ml'}*) – The fitting method to use, either MINRES or Maximum Likelihood. Defaults to 'minres'.

Returns **loadings** – The factor loadings matrix.

Return type pd.DataFrame

Raises ValueError – If any of the correlations are null, most likely due to having zero standard deviation.

get_communalities()
Calculate the communalities, given the factor loading matrix.

Returns **communalities** – A dataframe with communalities information.

Return type pd.DataFrame

Examples

```
>>> import pandas as pd
>>> from factor_analyzer import FactorAnalyzer
>>> df_features = pd.read_csv('test02.csv')
>>> fa = FactorAnalyzer()
>>> fa.analyze(df_features, 3, rotation=None)
>>> fa.get_communalities()
           Communalities
sex             0.588758
zygosity        0.003823
moed            0.504524
faed            0.728412
faminc          0.331843
english         0.662084
math            0.619110
socsci          0.731946
```

(continues on next page)

(continued from previous page)

```
natsci          0.649296
vocab           0.711497
```

get_eigenvalues()

Calculate the eigenvalues, given the factor correlation matrix.

Returns

- **e_values** (*pd.DataFrame*) – A dataframe with original eigenvalues.
- **values** (*pd.DataFrame*) – A dataframe with common-factor eigenvalues.

Examples

```
>>> import pandas as pd
>>> from factor_analyzer import FactorAnalyzer
>>> df_features = pd.read_csv('test02.csv')
>>> fa = FactorAnalyzer()
>>> fa.analyze(df_features, 3, rotation=None)
>>> ev, v = fa.get_eigenvalues()
>>> ev
   Original_Eigenvalues
0              3.510189
1              1.283710
2              0.737395
3              0.133471
4              0.034456
5              0.010292
6             -0.007400
7             -0.036948
8             -0.059591
9             -0.074281
>>> v
   Common_Factor_Eigenvalues
0                   3.510189
1                   1.283710
2                   0.737395
3                   0.133471
4                   0.034456
5                   0.010292
6                  -0.007400
7                  -0.036948
8                  -0.059591
9                  -0.074281
```

get_factor_variance()

Calculate the factor variance information, including variance, proportional variance and cumulative variance.

Returns **variance_info** – A dataframe with variance information.

Return type pd.DataFrame

Examples

```
>>> import pandas as pd
>>> from factor_analyzer import FactorAnalyzer
>>> df_features = pd.read_csv('test02.csv')
>>> fa = FactorAnalyzer()
>>> fa.analyze(df_features, 3, rotation=None)
>>> fa.get_factor_variance()
                 Factor1   Factor2   Factor3
SS Loadings     3.510189  1.283710  0.737395
Proportion Var  0.351019  0.128371  0.073739
Cumulative Var  0.351019  0.479390  0.553129
```

get_scores(*data*)

Get the factor scores, given the data.

Parameters **data** (*pd.DataFrame*) – The data to calculate factor scores.

Returns **scores** – The factor scores.

Return type pd.DataFrame

Examples

```
>>> import pandas as pd
>>> from factor_analyzer import FactorAnalyzer
>>> df_features = pd.read_csv('tests/data/test02.csv')
>>> fa = FactorAnalyzer()
>>> fa.analyze(df_features, 3, rotation='varimax')
>>> fa.get_scores(df_features).head()
    Factor1   Factor2   Factor3
0 -1.158106  0.081212  0.342195
1 -1.799933  0.155316  0.311530
2 -0.557422 -1.596457  0.548574
3 -0.973182 -1.530071  0.543792
4 -1.450108 -1.553214  0.446574
```

get_uniqueness()

Calculate the uniquenesses, given the factor loading matrix.

Returns **uniqueness** – A dataframe with uniqueness information.

Return type pd.DataFrame

Examples

```
>>> import pandas as pd
>>> from factor_analyzer import FactorAnalyzer
>>> df_features = pd.read_csv('test02.csv')
>>> fa = FactorAnalyzer()
>>> fa.analyze(df_features, 3, rotation=None)
>>> fa.get_uniqueness()
           Uniqueness
sex          0.411242
zygosity     0.996177
```

(continues on next page)

(continued from previous page)

```
moed        0.495476
faed        0.271588
faminc      0.668157
english     0.337916
math        0.380890
socsci      0.268054
natsci      0.350704
vocab       0.288503
```

remove_non_numeric(*data*)

Remove non-numeric columns from data, as these columns cannot be used in factor analysis.

Parameters **data** (*pd.DataFrame*) – The dataframe from which to remove non-numeric columns.

Returns **data** – The dataframe with non-numeric columns removed.

Return type pd.DataFrame

static smc(*data*, *sort=False*)

Calculate the squared multiple correlations. This is equivalent to regressing each variable on all others and calculating the r-squared values.

Parameters

- **data** (*pd.DataFrame*) – The dataframe used to calculate SMC.
- **sort** (*bool, optional*) – Whether to sort the values for SMC before returning. Defaults to False.

Returns **smc** – The squared multiple correlations matrix.

Return type pd.DataFrame

Examples

```
>>> import pandas as pd
>>> from factor_analyzer import FactorAnalyzer
>>> df_features = pd.read_csv('test02.csv')
>>> FactorAnalyzer.smc(df_features)
                SMC
sex        0.212047
zygosity   0.010857
moed       0.385399
faed       0.453161
faminc     0.273753
english    0.566065
math       0.547790
socsci     0.677035
natsci     0.576016
vocab      0.660264
```

factor_analyzer.factor_analyzer.**calculate_bartlett_sphericity**(*data*)

Test the hypothesis that the correlation matrix is equal to the identity matrix.identity

H0: The matrix of population correlations is equal to I. H1: The matrix of population correlations is not equal to I.

The formula for Bartlett's Sphericity test is:

$$-1 * (n - 1 - ((2p + 5)/6)) * ln(det(R))$$

Where R det(R) is the determinant of the correlation matrix, and p is the number of variables.

Parameters **data** (*pd.DataFrame*) – The data frame from which to calculate sphericity.

Returns

- **statistic** (*float*) – The chi-square value.
- **p_value** (*float*) – The associated p-value for the test.

`factor_analyzer.factor_analyzer.`**`calculate_kmo`**(*data*)

Calculate the Kaiser-Meyer-Olkin criterion for items and overall. This statistic represents the degree to which each observed variable is predicted, without error, by the other variables in the dataset. In general, a KMO < 0.6 is considered inadequate.

Parameters **data** (*pd.DataFrame*) – The data frame from which to calculate KMOs.

Returns

- **kmo_per_variable** (*pd.DataFrame*) – The KMO score per item.
- **kmo_total** (*float*) – The KMO score overall.

`factor_analyzer.factor_analyzer.`**`covariance_to_correlation`**(*m*)

This is a port of the R *cov2cor* function.

Parameters **m** (*numpy array*) – The covariance matrix.

Returns **retval** – The cross-correlation matrix.

Return type numpy array

Raises `ValueError` – If the input matrix is not square.

`factor_analyzer.factor_analyzer.`**`partial_correlations`**(*data*)

This is a python port of the *pcor* function implemented in the *ppcor* R package, which computes partial correlations of each pair of variables in the given data frame *data*, excluding all other variables.

Parameters **data** (*pd.DataFrame*) – Data frame containing the feature values.

Returns **df_pcor** – Data frame containing the partial correlations of of each pair of variables in the given data frame *df*, excluding all other variables.

Return type pd.DataFrame

3.1.3 `factor_analyzer.rotator` Module

Rotator class to perform various rotations of factor loading matrices.

author Jeremy Biggs (jbiggs@ets.org)

date 05/21/2018

organization ETS

class `factor_analyzer.rotator.`**`Rotator`**

Bases: `object`

The Rotator class takes an (unrotated) factor loading matrix and performs one of several rotations.

Notes

Most of the rotations in this class are ported from R's *GPARotation* package.

References

[1] https://cran.r-project.org/web/packages/GPArotation/index.html

Examples

```
>>> import pandas as pd
>>> from factor_analyzer import Rotator
>>> unrotated_loadings = pd.read_csv('loading_uls_none_3_test01.csv')
>>> rotator = Rotator()
>>> loadings, rotate_mtx = rotator.rotate(unrotated_loadings, 'varimax')
>>> loadings
           Factor1   Factor2   Factor3
sex      -0.076925  0.044992  0.762026
zygosity  0.018420  0.057579  0.012978
moed      0.060674  0.706943 -0.033120
faed      0.113147  0.845224 -0.034069
faminc    0.153070  0.555351 -0.001220
english   0.774515  0.147466  0.201190
math      0.706296  0.172295 -0.300973
socsci    0.839906  0.150589 -0.061835
natsci    0.766202  0.104519 -0.226524
vocab     0.813730  0.209159  0.074794
```

oblique (*loadings*, *objective*, *max_iter=1000*, *tolerance=1e-05*, ***kwargs*)

A generic function for performing all oblique rotations, except for promax, which is implemented separately.

Parameters

- **loadings** (`pd.DataFrame`) – The original loadings matrix
- **objective** (`function`) – The function for a given orthogonal rotation method. Must return a dictionary with *grad* (gradient) and *criterion* (value of the objective criterion).
- **max_iter** (`int, optional`) – The maximum number of iterations. Defaults to *1000*.
- **tolerance** (`float, optional`) – The convergence threshold. Defaults to *1e-5*.
- **kwargs** – Additional key word arguments are passed to the *objective* function.

Returns

- **loadings** (*pd.DataFrame*) – The loadings matrix (n_cols, n_factors)
- **rotation_mtx** (*np.array*) – The rotation matrix (n_factors, n_factors)

orthogonal (*loadings*, *objective*, *max_iter=1000*, *tolerance=1e-05*, ***kwargs*)

A generic function for performing all orthogonal rotations, except for varimax, which is implemented separately.

Parameters

- **loadings** (`pd.DataFrame`) – The original loadings matrix

- **objective** (*function*) – The function for a given orthogonal rotation method. Must return a dictionary with *grad* (gradient) and *criterion* (value of the objective criterion).
- **max_iter** (*int, optional*) – The maximum number of iterations. Defaults to *1000*.
- **tolerance** (*float, optional*) – The convergence threshold. Defaults to *1e-5*.
- **kwargs** – Additional key word arguments are passed to the *objective* function.

Returns

- **loadings** (*pd.DataFrame*) – The loadings matrix (n_cols, n_factors)
- **rotation_mtx** (*np.array*) – The rotation matrix (n_factors, n_factors)

promax (*loadings*, *normalize=False*, *power=4*)

Perform promax (oblique) rotation, with optional Kaiser normalization.

Parameters

- **data** (*pd.DataFrame*) – The loadings matrix to rotate.
- **normalize** (*bool, optional*) – Whether to perform Kaiser normalization and de-normalization prior to and following rotation. Defaults to False.
- **power** (*int, optional*) – The power to which to raise the varimax loadings (minus 1). Numbers should generally range form 2 to 4. Defaults to 4.

Returns

- **loadings** (*pd.DataFrame*) – The loadings matrix (n_cols, n_factors)
- **rotation_mtx** (*np.array*) – The rotation matrix (n_factors, n_factors)

rotate (*loadings*, *method='varimax'*, ***kwargs*)

Rotate the factor loading matrix.

Parameters

- **loadings** (*pd.DataFrame*) – The loadings matrix from your factor analysis.
- **method** (*str, optional*) – The factor rotation method. Options include:

  1. varimax (orthogonal rotation)
  2. promax (oblique rotation)
  3. oblimin (oblique rotation)
  4. oblimax (orthogonal rotation)
  5. quartimin (oblique rotation)
  6. quartimax (orthogonal rotation)
  7. equamax (orthogonal rotation)

  Defaults to 'varimax'.
- **kwargs** – Additional key word arguments are passed to the rotation method.

Returns

- **loadings** (*pd.DataFrame*) – The loadings matrix (n_cols, n_factors)
- **rotation_mtx** (*np.array*) – The rotation matrix (n_factors, n_factors)

Raises ValueError – If the *method* is not in the list of acceptable methods.

varimax (*loadings*, *normalize=True*, *max_iter=500*, *tolerance=1e-05*)

Perform varimax (orthogonal) rotation, with optional Kaiser normalization.

Parameters

- **loadings** (*pd.DataFrame*) – The loadings matrix to rotate.
- **normalize** (*bool, optional*) – Whether to perform Kaiser normalization and de-normalization prior to and following rotation. Defaults to True.
- **max_iter** (*int, optional*) – Maximum number of iterations. Defaults to 500.
- **tolerance** (*float, optional*) – The tolerance for convergence. Defaults to 1e-5.

Returns

- **loadings** (*pd.DataFrame*) – The loadings matrix (n_cols, n_factors)
- **rotation_mtx** (*np.array*) – The rotation matrix (n_factors, n_factors)

CHAPTER 4

Indices and tables

- genindex
- search

Python Module Index

f

Index