pyLPA: A Package for Latent Profile Analysis in Python

Johanna Einsiedler
University of Copenhagen

Abstract

Latent Profile Analysis (LPA) is a method to find clusters of individuals with similar characteristics based on a set of observable, continuous variables. LPA is commonly used in various disciplines within the Social Sciences to identify latent group structures in big data sets. This article describes **pyLPA**, a Python package that facilitates easy implementations of LPA models while allowing researchers to specify relevant parameters in the fitting process. We further present two worked examples of LPAs in **pyLPA** and showcase multiple tools to assess the goodness-of-fit of the resulting models.

Keywords: Python, gaussian mixture models, latent profile analysis, multiple-group analysis, structural equation modelling, .

1. Introduction

LPA is a model-based method to identify sub-groups of individuals with similar characteristics in large data sets. It is frequently used in the Social Sciences to explain population heterogeneity and uncover underlying typologies and profiles based on continuous, observed variables.

Applications span a wide range of fields: LPA has been used to study volunteer motivation (Geiser *et al.* 2014), development of burnout (Leiter and Maslach 2016; Mäkikangas *et al.* 2021), school disengagement (Fredricks *et al.* 2019), neurogenetic syndrome (Prince and Fidler 2021), the influence or race on socio-economic status on child development (Witherspoon *et al.* 2019), the impact of anxiety and working memory on algebraic reasoning (Trezise and Reeve 2017), fetal alcohol spectrum disorders (Mattson *et al.* 2010, 2013), personality disorders (Rufino *et al.* 2017) or psychopathy (Mokros *et al.* 2015; Hare 2016).
From a methodological viewpoint, LPAs are a class of finite mixture models, applied to static, continuous-valued response variables. Related concepts are latent class analysis, latent transition analysis and growth mixture modelling (McLachlan *et al.* 2019; Bauer 2022).
There are existing software implementations of LPA in the structural equation modelling program Mplus (Muthén and Muthén 2017) as well as several R libraries (**mclust**, Scrucca *et al.* 2016; **lavaan**, Rosseel 2012; **tidyLPA**, Rosenberg *et al.* 2019). However, so far there

doesn't exist a specialized package in Python.

A number of existing Python packages (e.g. **PyMix**, Georgi *et al.*; **PyPR**, Petersen; **gmr**, Fabisch 2021, Petersen and **scikit-learn**, Pedregosa *et al.* 2011) provide high-level functions for finite mixture modelling. However, all of these packages lack features that are often needed or desired in the case of LPA.

Specifically,

1. **Multiple-Group Analysis:** To systematically investigate the generalizability of latent structures and compare them across sub-samples, social science researchers have proposed methods that rely on sub-sample analyses (Morin *et al.* 2016). These are used to, for example, assess whether the same profiles are applicable for survey respondents with different nationalities (Morin *et al.* 2016). This in turn requires the mixture model to take into account sub-sample structures during the fitting process which is functionality currently not available in any open-source mixture model implementation.

2. **Standard Errors for Parameter Estimates:** Often it can be relevant for researchers to quantify the uncertainty with regards to the estimated parameters. This is a functionality that R packages focused on latent variable analysis offer but which is not available in existing Python packages.

The package **pyLPA** tries to fill the above described "gap" by offering high-level Python functions tailored to LPA, the first open-source framework to run multiple-group analysis and functionality to calculate standard errors for parameters in Python. The package can be downloaded from PyPi[1] and is also available on Github[2].

2. General Framework

2.1. LPA Models

The underlying assumption of LPA models is that each latent profile is characterized by a multivariate Gaussian distribution over the observed variables.[3] Thus, each latent class $(1, ..., K)$ is normally distributed with $\mathbf{x} \sim N(\mu_k, \Sigma_k)$, where $\mathbf{x}$ is a $M \times N$ matrix, representing N observations (i.e. typically the number of individuals in the data set) of M variables. μ_k and Σ_k are the mean vector and variance-covariance matrix for class k.

The joint probability density model is then given by

$$p(\mathbf{x}|\Theta) = \sum_k \alpha_k N(\mathbf{x}; \mu_k, \Sigma_k)$$

where α_k is the probability of the k^{th} class, or equivalently, the k^{th} multivariate Gaussian distribution.

[1]https://pypi.org/project/py-latent-profiles/

[2]https://github.com/johanna-einsiedler/PyLPA

[3]Theoretically it would be possible to use other distributions as well, e.g. Possion for count data. However this feature is currently not implemented in **pyLPA**.

Before the analysis μ_k, Σ_k and α_k are unknown and thus need to be estimated as part of the model fitting process. With increasing numbers of classes and response variables, the number of parameters to be estimated proliferates quickly. As this introduces more instability in the estimation, it is common in LPA to impose additional restrictions during the fitting process (Bauer 2022). Frequently, one assumes *local independence* which means that the response variables are constrained to be uncorrelated within a class. To enforce this restriction, the off-diagonal elements of the variance-covariance matrices are set to zero. Another common restriction is *cross-class equality* or *homogeneity* of the response variable variances (Vermunt and Magidson 2002). This means the variance co-variance matrices of all classes are constrained to be equal.

2.2. Steps in LPA

Model Specification. In the first step, researchers need to choose a model specification, i.e. which indicator variables to include in the model, if and how to pre-process those variables and which restrictions to impose. More information on choosing a suitable model specification can be found in previous literature e.g. Bauer (2022).

The default in **pyLPA** is a model without restrictions. However, restrictions on mean, variances, covariances and class probabilities can be implemented very easily as outlined in Chapter 4. In Mplus the default is a model with local independendence and cross-class equality, in the R library **mclust** the default is a data driven model choice.[4]

Choosing the Number of Classes. In the next step, researchers needs to choose how many latent classes the model should fit. To do so, one usually fits a number of models with different class numbers and chooses based on the comparative model fit. Frequently used fit indicators are the Bayesian Information Criterion (BIC), the sample-size adjusted BIC (SABIC), the Akaike Information Criterion (AIC) and the consistent AIC. With these, a lower value points to a better fit and ideally, the best solution can be identified based on the minimum value. Computation of all of these indicators is implemented in **pyLPA** as demonstrated in Chapter 4.2. In practice, fit indicators often continuously improve as more classes are added to the model. In this case, researchers often rely on an 'elbow plot' to identify at which number of classes the improvement reaches a plateau (Bauer 2022).

Model Fitting. The model fitting is usually done using the iterative expectation- maximization training algorithm. In the first step, the parameters (i.e. $\mu_{\mathbf{k}}$, Σ_k, α_k) are initialized, either with some random values from the data or based on k-means clustering results. In **pyLPA** the mean vectors of each class are initialized with random draws from the sample data, the variance-covariance matrix is initialized with the $M \times K$ identity matrix and initial class probabilities are set to be the same for each class (i.e. $\alpha_k = 1/K, \forall k$).

Conditional on these parameter values, the probability of each observation x_i to belong to a class k is calculated (expectation step). Next, these probabilities are used to compute better estimates for the parameters (maximization step). The expectation and maximization steps

[4]Different models with different sets of restrictions are estimated and the one with the best fit is chosen automatically.

are then performed iteratively until convergence is reached. A more detailed description of the algorithm can be found in previous literature, e.g. Ng *et al.* (2012). Lastly, researchers needs to evaluate the model using goodness of fit measures, standard errors of the parameters and theory.

3. A Simple Example

For this starting example we use a simulated dataset with two-dimensional data, generated from three different Gaussian distributions. The goal of the following analysis is to recover those underlying Gaussian distributions (i.e. classes) and their parameters (i.e. mean, covariance matrix and class probabilities) from the two observed variables.
This is a very general setup that covers many use cases. To give some examples:

- The observed variables could represent individuals' scores from a personality questionnaire and the latent classes could be different personality profiles that the researcher is interested in studying.
- The observed variables could be financial indicators (e.g. inflation rate) while the latent classes represent different financial regimes (e.g. crisis vs. growth) that the researcher wants to uncover.
- The observed variables could be psychosis symptoms and the latent classes could represent distinct groups of patients.

For **pyLPA** to be able to work with the data, it has to be represented as a **numpy** array with the rows corresponding to the number of observations and one column for each observed variables. In our example case, the data has the shape of a 500 x 2 array:

```
>>> data
array([[ 2.68626042e+00, -2.96125622e-01],
       [ 6.36611831e-01,  5.54466473e-02],
       [-4.80347537e+00,  2.48126851e-01],
       [ 3.21786130e+00, -1.14034736e+00],
      ...
       [-4.50526086e+00, -7.02371550e-01],
       [-9.66496279e+00, -3.51732660e+00],
       [ 1.94617345e+00,  5.87075970e-01]])
>>> data.shape
(500, 2)
```

3.1. Specifying and Fitting the Model

We start by importing the **pyLPA** package and initializing an object instance of the Gaussian mixture class that comes with **pyLPA**. Since we aim to identify three latent profiles, i.e. fit three Gaussian distributions to the data, the parameter `K` is set to three.

```
>>> import py_lpa as pylpa
>>> lpa = pylpa.GaussianMixture(K=3)
```

Next we simply call the `fit` method on the newly created object and the data to fit a simple Gaussian mixture model. **pyLPA** generates random initialization values from the data for the mean as a starting point for for the expectation-maximization algorithm. As this can result in the algorithm converging to different local optima, it is recommendable to conduct multiple runs and subsequently choose the solution with the highest likelihood. With **pyLPA** one can specify the number of random starts by setting the `rstarts`-parameter and the `fit` method will automatically return the solution with the highest log-likelihood.

```
>>> lpa.fit(data, rstarts = 50)
```

3.2. Analysing the Results

The fitted object now stores information on the means, covariances, and shares of each Gaussian mixture, which can be outputted as follows:

```
>>> lpa.means_
array([[[ 1.90899883,  0.41483735],
        [-9.00065343, -5.4069592 ],
        [-4.49077845,  0.06660923]]])
>>> lpa.covariances_
array([[[[ 1.03008561,  0.05223688],
         [ 0.05223688,  0.84939269]],

        [[ 1.06940312, -0.11680927],
         [-0.11680927,  0.96695214]],

        [[ 0.99232853,  0.05998318],
         [ 0.05998318,  0.99528599]]]])
>>> lpa.pi_
array([[0.33207204, 0.33398048, 0.33394748]])
```

We can further use the fitted object instance to predict the probability of each data point for belonging to a specific class, or equivalently, Gaussian.

```
>>> lpa.predict_proba(data)
[array([[9.11494152e-02, 1.50948903e-38, 6.23259132e-13],
        [7.37774074e-02, 4.77330495e-30, 2.69011534e-07],
        [5.31450703e-11, 1.30255247e-13, 1.49664431e-01],
        ...,
        [2.49791775e-10, 8.92967704e-12, 1.19144452e-01],
        [1.32726606e-32, 2.27215541e-02, 1.00860669e-09],
        [1.67407116e-01, 1.26203427e-37, 1.36339110e-10]])]
```

From this we can also infer which Gaussian has the highest likelihood of having generated a specific data point, or alternatively we can simply call `predict` which returns an array with the highest likelihood Gaussian for each data point:

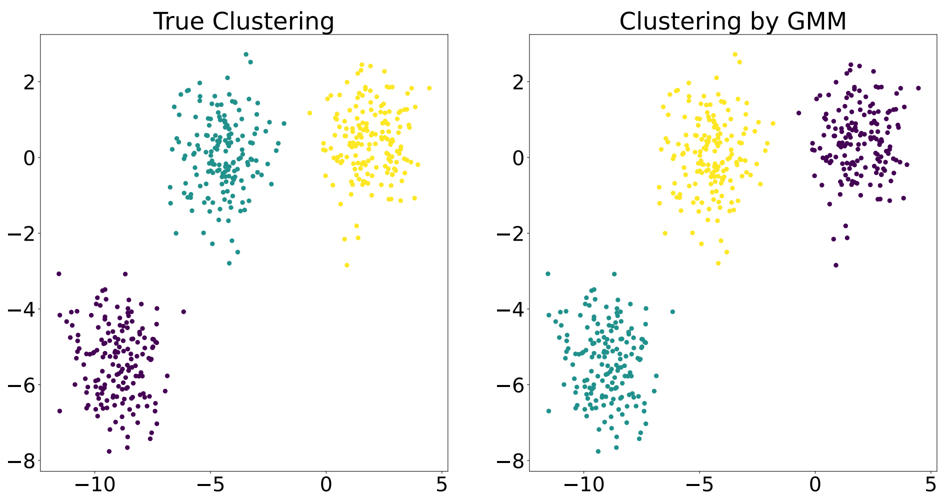

Figure 1: True clusters compared to clustering solution found by the **pyLPA** package

```
>>> lpa.predict(data)
[array([0, 0, 2, 0, 1, 2, 0, 2, 1, 1, 2, 1, 0, 1, 2, 0, 2, 0, 0, 2, 0, 2,
        1, 2, 2, 2, 0, 2, 2, 0, 1, 0, 0, 2, 2, 1, 2, 2, 2, 1, 2, 2, 0, 1,
        ...
        0, 1, 1, 1, 0, 1, 0, 0, 2, 1, 0, 1, 2, 0, 0, 0, 0, 1, 2, 2, 1, 2,
        1, 2, 0, 2, 2, 2, 1, 0, 0, 1, 2, 1, 0, 2, 1, 0])]
```

The results can be easily visualised using **matplotlib**. Below, we are plotting the true assignment of data points (i.e. which Gaussian distribution the data point has originally been generated from) as well as the clustering found by **pyLPA**.

```
>>> y_predicted = lpa.predict(data)
>>> fig, axs = plt.subplots(nrows=1,ncols=2,figsize=(24,12))
>>> axs[0].scatter(data[:, 0], data[:, 1], c=y)
>>> axs[0].set_title('True Clustering')
>>> axs[1].scatter(data[:, 0], data[:, 1], c=y_predicted)
>>> axs[1].set_title('Clustering by LPA')
```

So in this simple case, where we know that the underlying data comes from three different Gaussian distributions, the algorithm perfectly recovers the underlying data structure.

3.3. Getting Standard Errors for the Estimates

pyLPA also allows to easily compute bootstrapped standard errors for the estimated mean, covariance parameters, and share of each Gaussian. `bootstrap_fitted()` uses draws from the fitted distribution (also sometimes referred to as 'parametric bootstrap') to derive standard errors. This method has been shown to be more robust compared to boostrapping errors based on draws from the observed data (O'Hagan *et al.* 2019). The function takes one additional parameter `nbs` that determines the number of bootstrap samples to use for the estimation. The results are returned as a nested vector that contains the standard errors for class proportions, means and covariances in this order.

```
>>> lpa.bootstrap_fitted(data,nbs=100)
[array([[0.00117432, 0.00050639, 0.0011551 ]]),
 array([[[0.2529193 , 0.23876297],
         [0.26663347, 0.26379016],
         [0.25685989, 0.24323757]]]),
 array([[[[0.14510702, 0.08042116],
          [0.08042116, 0.08566152]],

         [[0.1163188 , 0.08690474],
          [0.08690474, 0.10199451]],

         [[0.11082559, 0.07474173],
          [0.07474173, 0.10278483]]]])]
```

4. A More Complex Example - LPA with Multiple-Group Analysis

As the EM-algorithm is not deterministic, solutions can be different depending on the starting values. Further, contrary to the above example the underlying true solution is mostly unknown in real-world applications of LPAs. This makes it hard to evaluate the goodness of fit of the model.
Researchers thus need to make an educated decision about the number of latent structures they aim to model and, to a large degree, have to rely on theory and intuition to draw conclusions about the generalizability of the solutions.
In this example, we show how **pyLPA** can be used to easily compute various statistical indicators that facilitate model selection and perform multiple-group similarity analysis (Morin *et al.* 2016) to assess the generalizability of solutions.

For this example we have created a synthetic data set using kernel density estimation that is based on an actual sample of replies to the HEXACO-60 personality questionnaire (Ashton and Lee 2009). The data contains 1,000 'observations'. Each observation consists of normalized scores for each of the personality dimensions (Honesty-Humility, Emotionality, Extraversion, Agreeableness vs. Anger, Conscientiousness, and Openness to Experience) as well as a variable indicating the biological gender of the fictional survey taker. In this case, the goal of the latent profile analysis is to identify common personality profiles in the population and assess whether they are similar for both genders. Each personality profile thus corresponds to one multi-variate Gaussian distribution.

4.1. Running the Latent Profile Analysis

As this example already has a fairly large number of parameters that need to be estimated, we will impose local independence an cross-class equality (as described in Section 2.1).
These restrictions can easily be implemented in **pyLPA** using the `var_constraint` and

`cov_constraint` parameters. To impose local independence we set `cov_constraint` to `'zero'` which means that the off-diagonal elements of the covariance matrix will be zero and thus different personality dimensions are not allowed to covary. To impose cross-class equality, we set `var_constraint` to `'classes'` which translates to forcing all Gaussians to have the exact same variance.
We can further manually set the maximum number of iterations of the EM-algorithm as well as the tolerance threshold. The algorithm will stop if either the difference between the log-likelihoods from the solutions of the current iteration compared to the previous iteration is smaller than `tol` or the maximum number of iterations has been reached.
Additional fine-tuning can be done using the parameters `first_stage_iter` and `n_final_stage`. Specifying those breaks the optimization process into two stages, the initial and the final stage. This setup follows the methodology implemented in Mplus (Muthén and Muthén 2017). In our example, we set `first_stage_iter` to 200 which means 200 iterations of the EM-algorithm will be performed for each of the `rstarts` (i.e. 2000 in this case). We also set `n_final_stage` to 200 which means that from the initial 2000 starts, the 200 solutions with the highest log-likelihood after 200 iterations of the EM-algorithm will be further optimized in the final stage. In the final stage, the optimization process will continue until either the tolerance threshold or the overall number of maximum iterations is reached (i.e. in the final stage there will be maximum `max_iter - first_stage_iter` iterations, in this case 300). The numbers chosen in this example correspond to the default values implemented in Mplus (Muthén and Muthén 2017).

```
# example of lpa with 3 clusters
>>> lpa = pylpa.GaussianMixture(K=3)
>>> lpa.fit(X,rstarts=2000,max_iter=500, tol=0.0000001,
first_stage_iter=200, n_final_stage=200, var_constraint='classes',
cov_constraint='zero')
```

4.2. Assessing Goodness of Fit

Next to the whole vector of loglikelihoods of all intermediate solutions, other commonly used statistical indicators can be easily obtained from the fitted model object. Specifically, **pyLPA** supports the calculation of the Akaike Information Criterion (AIC; Akaike 1974), the Bayesian Information Criterion (BIC; Schwarz 1978), the Sample-Size Adjusted BIC (SABIC; Sclove 1987) and the Consistent Akaike Information Criterion (CAIC; Bozdogan 1987).

These statistical indicators can be used to facilitate the choice of the number of latent structures. To do so, we run LPA models with different values of k and compare the fit statistics of the resulting solutions. Figure 2 shows AIC, BIC, CAIC and SABIC scores for models with two, three, four and five latent profiles. In our case, we find that AIC and SABIC are decreasing with increasing numbers of profiles, while showing something like an 'elbow point' at three. BIC and CAIC are actually higher for the four and five profile solutions compared to the three profile solution. This would indicate, that the three profile solution should be our preferred solution.

```
# statistical indicators for three profile solution
>>> lpa.AIC()
```

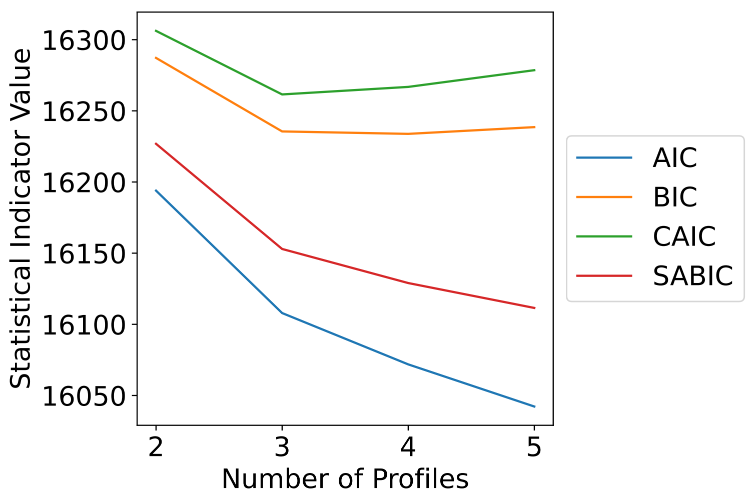

Figure 2: Statistical indicators for solutions ranging from two to five profiles.

```
16107.9302

>>> lpa.BIC()
16235.5319

>>> lpa.CAIC()
16261.5319

>>> lpa.SABIC()
16152.9544
```

4.3. Comparing pyLPA with Mplus

To compare performance and output, we ran the same model in MPlus (Muthén and Muthén 2017). The resulting parameter estimates were identical up to at least two decimal points. The Mplus output as well as the solution provided by **pyLPA** can be found in the Supplementary Material.

4.4. Multiple-Group Analysis of Similarity

In a next step, researchers might be interested in assessing the generalizability of the derived profile solutions across sub-groups. In this example case, we aim to examine the similarity of profiles by gender.
Morin *et al.* (2016) developed an approach to test for different types of similarity in profiles based on fitting LPA models jointly to sub-samples while imposing increasingly restrictive equality constraints on the parameters of the model.

Configural Similarity

The first step is to test for *configural similarity* which is equivalent to testing, whether the

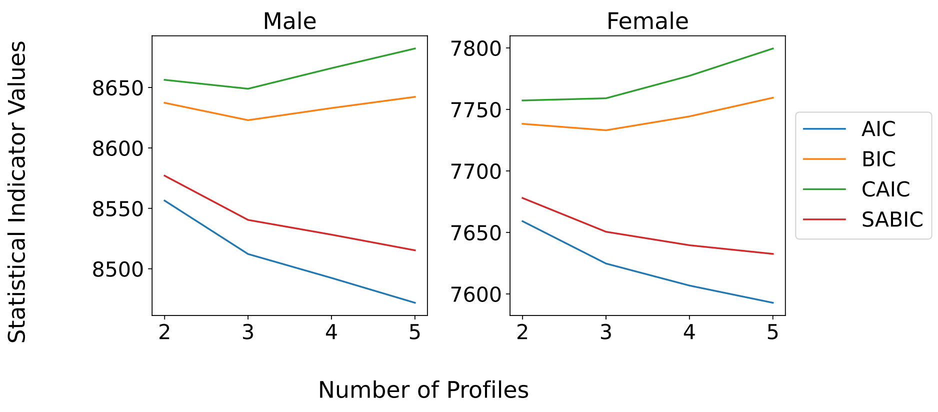

Figure 3: Statistical indicators for solutions ranging from two to five profiles for the male and the female sub-samples.

same number of latent profiles is identified in both sub-samples (Morin *et al.* 2016). To do so, separate latent profile analyses are conducted and statistical indicators are used to assess, which solution fits best. Figure 3 shows the resulting values of AIC, BIC, CAIC and SABIC for the male sub-sample and the female sub-sample. We see the same tendency in both groups as in the overall sample. We can thus conclude that the three-profile solution passes the test for configural similarity.

Structural Similarity

For the following steps, the number of latent profiles is fixed (i.e. in our case at three) and joint models with increasing restrictions on the parameters are estimated. To test for *structural similarity* we estimate a LPA where the means of the profiles are constrained to be the same across sub-groups and compare the fit statistics of this model to a joint model without constraints (referred to as the 'configural model').

In **pyLPA** a structural model can be estimated by simply setting `mean_constraint` to `'groups'` in the `fit`-method. The resulting fit statistics are displayed in Table 1. Compared with the baseline configural similarity model, the structural model resulted in lower values for all statistical indicators. The strucutural similarity of the three-profile solution across gender is thus supported.

```
# baseline model
>>> lpa_multi = pylpa.GaussianMixture(K=3)
>>> lpa_multi.fit(data_male,data_female,rstarts=2000,max_iter=500,
tol=0.0000001, first_stage_iter=200, n_final_stage=200,
var_constraint='classes',cov_constraint='zero')

# structural model
>>> lpa_struct = pylpa.GaussianMixture(K=3)
>>> lpa_struct.fit(data_male,data_female,rstarts=2000,max_iter=500,
tol=0.0000001, first_stage_iter=200, n_final_stage=200,
var_constraint='classes',cov_constraint='zero',mean_constraint='groups')
```

Table 1: 3 Profile Solution

Cross-Sample Similarity	LL	AIC	BIC	CAIC	SABIC
Configural	-8,016.454	16,132.908	16,345.887	16,295.887	16,187.175
Structural	-8,025.800	16,115.599	16,251.906	16,283.906	16,150.331
Dispersion	-8,027.894	16,107.787	16,218.536	16,244.536	16,136.006
Distributional	-8,027.983	16,107.966	16,218.715	16,244.715	16,136.185

Dispersion Similarity

To estimate a model of *disperson similarity*, the within-profile variability is restricted to be equal across gender. This means, the variances of each profile are the same for both genders. In **pyLPA** this translates to setting the parameter `var_constraint` in the `fit`-method to `'groups-classes'`.
From the resulting statistical indicators (see Table 1) we can infer that the three profile solution also satisfies dispersion similarity.

```
# dispersion model
>>> lpa_disp = pylpa.GaussianMixture(K=3)
>>> lpa_disp.fit(data_male,data_female,rstarts=2000,max_iter=500,
tol=0.0000001, first_stage_iter=200,
n_final_stage=200, var_constraint='classes-groups',cov_constraint='zero',
mean_constraint='groups')
```

Distributional Similarity

In the final step, we constrain the sizes of the latent profiles to be equal across genders. This means the share of each personality profile is set to be the same for both genders. In **pyLPA** this can be done by setting `pi_constraint` to `'groups'`.
Here we find, as reported in Table 1, that the fit statistics have increased slightly. Consequently, one would argue that the three profile solution does not achieve distributional similarity.

```
# distributional model
>>> lpa_dist = gmm.GaussianMixture(K=3)
>>> lpa_dist.fit(data_male,data_female,rstarts=2000,max_iter=500,
tol=0.0000001, first_stage_iter=200, n_final_stage=200,
var_constraint='classes-groups',cov_constraint='zero',mean_constraint='groups',
pi_constraint='groups')
```

In this example case, the three profile solution comes out as the best fit and does achieve configural, structural and dispersion similarity. Researchers working on this - fictional - example could thus conclude, that there are three distinct personality profiles present in the population which could be further compared and interpreted with regards to their mean scores across the various personality dimensions. Further, we can conclude that the same profiles seem to applicable for both men and women although the frequency of occurrence of the profiles is varies across sex.

5. Conclusion

This article provides an introduction and overview of the Python package **pyLPA**. The main goal of the package is to provide high-level functions that allow researchers to easily run latent profile analysis in Python while being able to specify various constraints for the estimation method and compute relevant goodness-of-fit measures.

With more (social science) researchers using computational methods in their work, **pyLPA** aims to contribute to making those methods more accessible and further the usage of open-source software in science. To our knowledge it is the first Python package specifically focused on LPA. It further provides the first open-source implementation of multiple-group analysis of similarity for LPAs (Morin *et al.* 2016).

However, while **pyLPA** offers a comprehensive implementation of LPA, it does not extend to other forms of structural equation models as the more general R packages **lavaan** and **mclust** or the analysis program Mplus. Possible further developments include extensions towards other types of mixture distributions (e.g. Poisson) or other types of latent variable analysis (e.g. latent class analysis).

All in all, **pyLPA** makes LPA as a research method more accessible to researchers and provides tools to critically assess the results. It adds to the methodological toolbox available to social science researchers working in Python and hopefully can serve as an entry point for further implementations of similar models.

References

Akaike H (1974). "A new look at the statistical model identification." *IEEE Transactions on Automatic Control*, **19**(6), 716–723. ISSN 1558-2523. doi:10.1109/TAC.1974.1100705. Conference Name: IEEE Transactions on Automatic Control.

Ashton MC, Lee K (2009). "The HEXACO-60: a short measure of the major dimensions of personality." *Journal of Personality Assessment*, **91**(4), 340–345. ISSN 1532-7752. doi:10.1080/00223890902935878.

Bauer J (2022). "A Primer to Latent Profile and Latent Class Analysis." In M Goller, E Kyndt, S Paloniemi, C Damşa (eds.), *Methods for Researching Professional Learning and Development: Challenges, Applications and Empirical Illustrations*, Professional and Practice-based Learning, pp. 243–268. Springer International Publishing, Cham. ISBN 978-3-031-08518-5. doi:10.1007/978-3-031-08518-5_11.

Bozdogan H (1987). "Model selection and Akaike's Information Criterion (AIC): The general theory and its analytical extensions." *Psychometrika*, **52**(3), 345–370. ISSN 1860-0980. doi:10.1007/BF02294361. URL https://doi.org/10.1007/BF02294361.

Fabisch A (2021). "gmr: Gaussian Mixture Regression." *Journal of Open Source Software*, **6**(62), 3054. doi:10.21105/joss.03054. Publisher: The Open Journal, URL https://doi.org/10.21105/joss.03054.

Fredricks JA, Ye F, Wang MT, Brauer S (2019). "Chapter 3 - Profiles of School Disengagement: Not All Disengaged Students are Alike." In JA Fredricks, AL Reschly, SL Christenson (eds.), *Handbook of Student Engagement Interventions*, pp. 31–43. Academic Press. ISBN 978-0-12-813413-9. doi:10.1016/B978-0-12-813413-9.00003-6. URL https://www.sciencedirect.com/science/article/pii/B9780128134139000036.

Geiser C, Okun MA, Grano C (2014). "Who is motivated to volunteer? A latent profile analysis linking volunteer motivation to frequency of volunteering." *Psychological Test and Assessment Modeling*, **56**, 3–24. ISSN 2190-0507. Place: Germany Publisher: Pabst Science Publishers.

Georgi B, Costa IG, Schliep A (????). "PyMix - The Python Mixture package | PyMix / Home." URL http://www.pymix.org/pymix/index.php?n=PyMix.Home.

Hare RD (2016). "Psychopathy, the PCL-R, and criminal justice: Some new findings and current issues." *Canadian Psychology / Psychologie canadienne*, **57**(1), 21. ISSN 1878-7304. doi:10.1037/cap0000041. Publisher: US: Educational Publishing Foundation, URL https://psycnet.apa.org/fulltext/2016-06331-003.pdf.

Leiter MP, Maslach C (2016). "Latent burnout profiles: A new approach to understanding the burnout experience." *Burnout Research*, **3**(4), 89–100. ISSN 22130586. doi:10.1016/j.burn.2016.09.001. URL https://linkinghub.elsevier.com/retrieve/pii/S2213058615300188.

Mattson SN, Roesch SC, Fagerlund Autti-Rämö I, Jones KL, May PA, Adnams CM, Konovalova V, Riley EP, Cifasd T (2010). "Toward a Neurobehavioral Profile of Fetal Alcohol Spectrum Disorders." *Alcohol: Clinical and Experimental Research*, **34**(9), 1640–1650. ISSN 1530-0277. doi:10.1111/j.1530-0277.2010.01250.x. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1530-0277.2010.01250.x, URL https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1530-0277.2010.01250.x.

Mattson SN, Roesch SC, Glass L, Deweese BN, Coles CD, Kable JA, May PA, Kalberg WO, Sowell ER, Adnams CM, Jones KL, Riley EP (2013). "Further Development of a Neurobehavioral Profile of Fetal Alcohol Spectrum Disorders." *Alcohol: Clinical and Experimental Research*, **37**(3), 517–528. ISSN 1530-0277. doi:10.1111/j.1530-0277.2012.01952.x. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1530-0277.2012.01952.x, URL https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1530-0277.2012.01952.x.

McLachlan GJ, Lee SX, Rathnayake SI (2019). "Finite Mixture Models."

Mokros A, Hare RD, Neumann CS, Santtila P, Habermeyer E, Nitschke J (2015). "Variants of psychopathy in adult male offenders: A latent profile analysis." *Journal of Abnormal Psychology*, **124**(2), 372. ISSN 1939-1846. doi:10.1037/abn0000042. Publisher: US: American Psychological Association, URL https://psycnet.apa.org/fulltext/2015-03429-001.pdf.

Morin AJ, Meyer JP, Creusier J, Biétry F (2016). "Multiple-Group Analysis of Similarity in Latent Profile Solutions." *Organizational Research Methods*, **19**(2), 231–254. ISSN 1094-4281. doi:10.1177/1094428115621148. Publisher: SAGE Publications Inc, URL https://doi.org/10.1177/1094428115621148.

Muthén B, Muthén L (2017). "Mplus User's Guide." URL https://www.statmodel.com/html_ug.shtml.

Mäkikangas A, Leiter MP, Kinnunen U, Feldt T (2021). "Profiling development of burnout over eight years: relation with job demands and resources." *European Journal of Work and Organizational Psychology*, **30**(5), 720–731. ISSN 1359-432X. doi:10.1080/1359432X.2020.1790651. Publisher: Routledge _eprint: https://doi.org/10.1080/1359432X.2020.1790651, URL https://doi.org/10.1080/1359432X.2020.1790651.

Ng SK, Krishnan T, McLachlan GJ (2012). "The EM Algorithm." In JE Gentle, WK Härdle, Y Mori (eds.), *Handbook of Computational Statistics: Concepts and Methods*, Springer Handbooks of Computational Statistics, pp. 139–172. Springer, Berlin, Heidelberg. ISBN 978-3-642-21551-3. doi:10.1007/978-3-642-21551-3_6. URL https://doi.org/10.1007/978-3-642-21551-3_6.

O'Hagan A, Murphy TB, Scrucca L, Gormley IC (2019). "Investigation of parameter uncertainty in clustering using a Gaussian mixture model via jackknife, bootstrap and weighted likelihood bootstrap." *Computational Statistics*, **34**(4), 1779–1813. ISSN 1613-9658. doi:10.1007/s00180-019-00897-9. URL https://doi.org/10.1007/s00180-019-00897-9.

Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, Blondel M, Prettenhofer P, Weiss R, Dubourg V, Vanderplas J, Passos A, Cournapeau D, Brucher M, Perrot M, Duchesnay E (2011). "Scikit-learn: Machine Learning in Python." *Journal of Machine Learning Research*, **12**, 2825–2830.

Petersen JP (????). "Welcome to PyPR's documentation! — PyPR v0.1rc3 documentation." URL https://pypr.sourceforge.net/.

Prince MA, Fidler DJ (2021). "Chapter Two - Analytic approaches to heterogeneity in neurogenetic syndrome research." In RM Hodapp, DJ Fidler (eds.), *International Review of Research in Developmental Disabilities*, volume 60, pp. 55–73. Academic Press. doi:10.1016/bs.irrdd.2021.08.004. URL https://www.sciencedirect.com/science/article/pii/S2211609521000129.

Rosenberg JM, Beymer PN, Anderson DJ, Lissa Cjv, Schmidt JA (2019). "tidyLPA: An R Package to Easily Carry Out Latent Profile Analysis (LPA) Using Open-Source or Commercial Software." *Journal of Open Source Software*, **3**(30), 978. ISSN 2475-9066. doi:10.21105/joss.00978. URL https://joss.theoj.org/papers/10.21105/joss.00978.

Rosseel Y (2012). "lavaan: An R Package for Structural Equation Modeling." *Journal of Statistical Software*, **48**, 1–36. ISSN 1548-7660. doi:10.18637/jss.v048.i02. URL https://doi.org/10.18637/jss.v048.i02.

Rufino KA, Ellis TE, Clapp J, Pearte C, Fowler JC (2017). "Variations of emotion dysregulation in borderline personality disorder: a latent profile analysis approach with adult psychiatric inpatients." *Borderline Personality Disorder and Emotion Dysregulation*, **4**, 17. ISSN 2051-6673. doi:10.1186/s40479-017-0068-2.

Schwarz G (1978). "Estimating the Dimension of a Model." *The Annals of Statistics*, **6**(2), 461–464. ISSN 0090-5364. Publisher: Institute of Mathematical Statistics, URL https://www.jstor.org/stable/2958889.

Sclove SL (1987). "Application of model-selection criteria to some problems in multivariate analysis." *Psychometrika*, **52**(3), 333–343. ISSN 1860-0980. doi:10.1007/BF02294360. URL https://doi.org/10.1007/BF02294360.

Scrucca L, Fop M, Murphy TB, Raftery AE (2016). "mclust 5: clustering, classification and density estimation using Gaussian finite mixture models." *The R Journal*, **8**(1), 289–317. URL https://doi.org/10.32614/RJ-2016-021.

Trezise K, Reeve RA (2017). "Chapter 5 - The Impact of Anxiety and Working Memory on Algebraic Reasoning." In U Xolocotzin Eligio (ed.), *Understanding Emotions in Mathematical Thinking and Learning*, pp. 133–158. Academic Press, San Diego. ISBN 978-0-12-802218-4. doi:10.1016/B978-0-12-802218-4.00005-4. URL https://www.sciencedirect.com/science/article/pii/B9780128022184000054.

Vermunt JK, Magidson J (2002). "Latent Class Cluster Analysis." In AL McCutcheon, JA Hagenaars (eds.), *Applied Latent Class Analysis*, pp. 89–106. Cambridge University Press, Cambridge. ISBN 978-0-521-59451-6. doi:10.1017/CBO9780511499531.004. URL https://www.cambridge.org/core/books/applied-latent-class-analysis/latent-class-cluster-analysis/8E8DEA91530F92525EAEDED4B3B8DF0F.

Witherspoon DP, May EM, McDonald A, Boggs S, Bámaca-Colbert M (2019). "Chapter Eight - Parenting within residential neighborhoods: A pluralistic approach with African American and Latino families at the center." In DA Henry, E Votruba-Drzal, P Miller (eds.), *Advances in Child Development and Behavior*, volume 57 of *Child Development at the Intersection of Race and SES*, pp. 235–279. JAI. doi:10.1016/bs.acdb.2019.05.004. URL https://www.sciencedirect.com/science/article/pii/S0065240719300217.