Pymer4: Connecting R and Python for Linear Mixed Modeling

Eshin Jolly

1 Dartmouth College

Summary

In the social sciences the prevalence of clustered, correlated, and/or repeatedly measured data is very common. Modeling such data have classically involved the use of techniques such as the repeated measured analysis of variance (ANOVA) with correction factors applied to account for violations of model assumptions. More recently multi-level or hierarchical linear models have been brought to bear on modeling these kind of data given their flexibility. One of the most popular packages used for this purpose is the powerful lme4 package in the R statistical programming language (Bates, Mächler, Bolker, & Walker, 2015). While this package is highly favored by individuals that utilize R, to date there exist few comparable packages in the scientific Python community. While some solutions do exist they are often more complicated to use and sometimes less flexible (e.g. statsmodels (Seabold & Perktold, 2010) or require users to adopt a different statistical framework (e.g. Bayesian modeling, bambi (Yarkoni & Westfall, 2016)). This leaves Python users in want for a tool that: a) is highly compatible with existing scientific python tools (e.g. pandas, numpy, matplotlib, seaborn (McKinney, 2012)), b) has an API that is easy to use but not unfamiliar to those who use to lme4, c) offers additional functionality that anticipates users’ needs when analyzing real data (e.g. significance testing, simulating data, post-hoc analyses, etc).

For these reasons pymer4 was developed to offer users a way to work with multilevel models within the scientific Python ecosystem. To accomplish this pymer4 leverages the existing prowess of lme4, through the use of the rpy2 (Gautier, 2008) cross-language compatibility library. This offers analysts a tool at the level of abstraction they expect, without needing to worry about writing code specific to interfacing between languages (i.e. custom rpy2 scripts, functions, etc.). Beyond making lme4 functionality available in python, pymer4 makes working with multilevel models feel native to working in scientific Python and offers several key features users are most likely to utilize. This includes tight integration with pandas and numpy for data representation and manipulation, as well as matplotlib (Hunter, 2007) and seaborn (Waskom et al., 2014) for data visualization. A key contribution of pymer4 is its role in providing a unified interface to tasks that are spread over several different R-packages. For example, p-value computation lacking in lme4 is instead provided using the lmerTest (Kuznetsova, Brockhoff, & Christensen, 2017) library, while post-hoc subgroup comparisons are provided using the lmeans (Lenth & others, 2016) package. This obviates the need for users to move between libraries, remember idiosyncratic commands, and allows them to instead focus on the analysis at hand.

Additionally, pymer4 offers several extra features implemented at a high-level of abstraction that would otherwise require users to write custom functions and scripts. These include: simulating various types of data, visualizing model estimates, changing statistical inference procedures (e.g. parametric to non-parametric), automatically storing model
fits and residuals, and estimating ordinary-least-squares regression models with various types of robust standard error estimators (Huber & others, 1967). Overall, the key contribution of `pymer4` is the development of a consistent and intuitive API by which users can accomplish analytic tasks in a simple and style familiar to users of the scientific Python. We hope this reduces the development “switch-cost” many users experience when moving between tools to accomplish a single goal.

**Examples**

Documentation for `pymer4` includes numerous example analyses that demonstrate various functionality. Below we demonstrate how easily `pymer4` integrates into a scientific Python analysis workflow:

```python
# imports
import os
import pandas as pd
import seaborn as sns
from pymer4.models import Lmer
from pymer4.utils import get_resource_path

# Load included example data with pandas
df = pd.read_csv(os.path.join(get_resource_path(),'sample_data.csv'))

# Fit a multi-level model with 1 categorical predictor using dummy-codes with '1.0' as the reference level
model = Lmer('DV ~ IV3 + (IV3|Group)',data=df)
model.fit(factors={'IV3':
['1.0','0.5','1.5']})

# Perform post-hoc comparisons on the fitted model
model.post_hoc(marginal_vars='IV3')

# Plot model coefficients
model.plot_summary()

# Visualize residuals with seaborn; residuals are conveniently stored in the model
sns.regplot(x= 'IV2',
y= 'residuals',
data= model.data,
fit_reg= False
)
```

**Acknowledgements**

The author thanks Luke J. Chang, Jin Hyun Cheong, and Pin-hao Chen for testing and feedback. The author also thanks Jeroen van Baar for assistance in testing installations on different platforms.

**References**


