maxent.ot
A package for doing Maximum Entropy Optimality Theory in R

Connor Mayer, University of California, Irvine
and
Kie Zuraw, University of California, Los Angeles
with
Adeline Tan, University of California, Los Angeles

Who this software is for

- Anyone who uses R (R Core Team 2021) and uses MaxEnt constraint grammars (Goldwater & Johnson 2003)

- Are you tired of writing R scripts that contain comments like

```
# leave R here and go fit MaxEnt grammar to file output4.csv
```

?

- Do you wish there was a way to run MaxEnt analyses from inside your R script or R markdown file?

MaxEnt constraint grammars

- For those who don’t already use MaxEnt constraint grammars (including to argue against them!), what are they?
- They’re a way to model variation, attaching a probability to each output candidate in a tableau
  - by attaching a number (weight) to each constraint
- There are two main places where math happens
  - Finding the best constraint weights, given the training data
  - Seeing what the resulting model predicts, for both the training data and potentially new testing data
- Our package allows you to do both of those (and more!) in R

Goal of this software

- Make our/your research life easier
- Make it easier to evaluate and build on each other’s work

Reproducible research (Stodden, Leisch & Peng 2014)

- Reproducibility means making it easy for someone (including your future self!) to re-run your analysis using the same input data
    - They can check for mistakes and try out different analyses
- As much as possible, you want to run everything from one script
    - The script should take you from raw, unprocessed data…
        - e.g., a public database, or the results file that LabVanced writes when it runs your experiment
    - … to final results, ideally including figures and text
    - See also “literate programming” (Knuth 1992)
- If you make a change, you can just press one button to re-run the script
- Typical tools
    - RMarkdown files
    - Jupyter notebooks for Python

Example of R markdown and its output

Work with markdown file in RStudio

````
---
title: "MaxEnt package demo"
author: "Connor Mayer, Kie Zuraw, and Adeline Tan"
date: "10/3/2022"
output: html_document
---

# Introduction

We analyze a (constructed) child's acquisition of onset consonant clusters, based loosely on Rose 2002. For this imaginary child...

* cluster simplification is more likely in unstressed syllables
  + e.g. /gry.'o/ vs. /'grav/
* cluster simplification is more likely for s-stop than for stop-liquid
  + e.g. /'stad/ vs. /'grav/

We plot the overall pattern:

```{r plot}
#Create the data frame
simplification <- data.frame(cluster_type=c("ST","TR","ST","TR"), stress=c("stressed","stressed","unstressed","unstressed"), simplification_rate=c(0.6,0.4,0.9,0.7))

barplot(simplification$simplification_rate, ylab="simplification rate", xlab="cluster type", col=c("blue","blue","gold","gold"), ylim=c(0,1))
axis(1, at = c(0.7, 1.9, 3.1, 4.3), labels = c("s-stop", "stop-liq", "s-stop", "stop-liq"))
legend("topright", fill=c("blue","gold"), legend=c("stressed", "unstressed"))

```

# Reading in the tableaux
````

Click "knit" button to create html file (or PDF, etc.)

Introduction

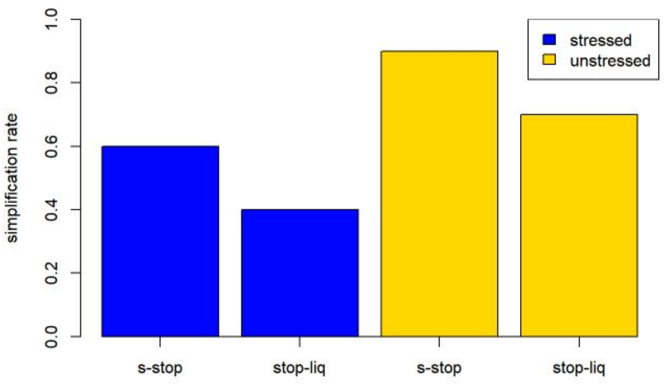
We analyze a (constructed) child's acquisition of onset consonant clusters, based loosely on Rose 2002. For this imaginary child...

- cluster simplification is more likely in unstressed syllables
  - e.g. /gry.'o/ vs. /'grav/
- cluster simplification is more likely for s-stop than for stop-liquid
  - e.g. /stad/ vs. /dra/

We plot the overall pattern:

```
#Create the data frame
simplification <- data.frame(cluster_type=c("ST","TR","ST","TR"),  stress=c("stressed","stressed","unstressed","unstressed"), simplification_rate=c(0.6,0.4,0.9,0.7))

barplot(simplification$simplification_rate, ylab="simplification rate", xlab="cluster type", col=c("blue","blue","gold","gold"), ylim=c(0,1))
axis(1, at = c(0.7, 1.9, 3.1, 4.3), labels = c("s-stop", "stop-liq", "s-stop", "stop-liq"))
legend("topright", fill=c("blue","gold"), legend=c("stressed", "unstressed"))
```



Benefits of reproducible research

- Easier for others to understand or build on your work
  - They can easily run and check your script
  - They can understand how to modify your script, or copy chunks of code
  - If they want to do a full *replication* (with new data), they can keep the analysis the same for better comparison
- Easier to return to a project after a break
  - e.g., after getting reviews back!
  - No need to hunt for multiple files or remember procedures for analysis–everything is organized in one file
- Easier to prevent, catch, and fix errors
  - All steps of the analysis are right there in the file for your inspection
  - Easy to make changes and re-run analysis
- By contrast, switching back and forth between R and an external MaxEnt tool makes it harder to keep things tidy
  - E.g., switching to MaxEnt Grammar Tool (Hayes, Wilson & George 2009) or Excel Solver
  - When you go back to a project, you have to remember where all your stuff is, which files to use in which program, and what settings you used or what cells you clicked

What the software can do: overview

- Read input files in MaxEnt Grammar Tool/OTSoft (Hayes & al. 2014) format
- Fit a MaxEnt model to training data
- Produce model predictions for training and test data
- Compare how well different models fit the data
- Use prior terms to encode bias or avoid overfitting (μ and σ)

Tutorial

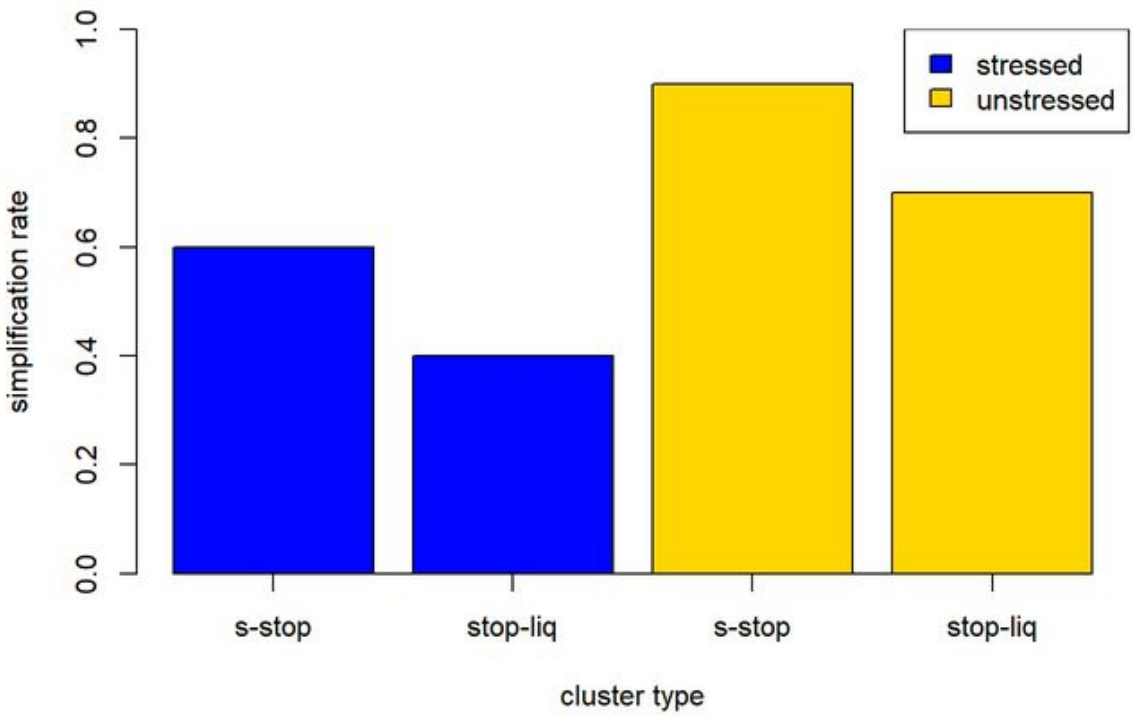
- Download from connormayer.com/misc/amp_2022_tutorial.zip
- We'll show you screenshots from the tutorial

Simple, fabricated dataset

- Very loosely based on Rose (2002)
- Fictionalized acquisition of onset consonant clusters in French

- cluster simplification is more likely in unstressed syllables
    - e.g. /gry.'o/ vs. /'grav/
- cluster simplification is more likely for s-stop than for stop-liquid
    - e.g. /'stad/ vs. /'grav/

We plot the overall pattern below:



Reading a file

- Must be in OTSoft tableau-like format
    - Same format as MaxEnt Grammar Tool
- *Future work: make readable from R data frame*

outputs

constraints

			StarComplex	Max
			StarComplex	Max
'stad	'stad	40	1	
	'tad	60		1
'grav	'grav	60	1	
	'gav	40		1
spa.gə.'ti	spa.gə.'ti	10	1	
	pa.gə.'ti	90		1
gry.'o	gry.'o	30	1	
	gy.'o	70		1

inputs

frequency of each candidate

Fitting a grammar

- Function to create model is `optimize_weights()`

- Then we can extract various parts of the fitted model

```r
base_file <- "amp_demo_grammar_base.csv"
base_model <- optimize_weights(base_file, in_sep=',')

# Get the weights of each constraint
base_model$weights
```

```
## StarComplex          Max
##   1.6088911    0.9898518
```

```r
# Get the log likelihood assigned to the training data under these weight
s
base_model$loglik
```

```
## [1] -258.9787
```

```r
# Get the number of free parameters (i.e. number of constraints)
base_model$k
```

```
## [1] 2
```

```r
# Get the number of data points
base_model$n
```

```
## [1] 400
```

Looking at model predictions with `predict_probabilities()`

- In this case, we want want to see what model predicts for the training tableaux themselves
  - But we could also see what it predicts for a file with different tableaux
- Function shows us same tableaux as were read in, but now with predicted probabilities
  - And comparisons to observed probabilities
- This grammar treats all four words the same

name of model we just fitted

```
predict_probabilities(base_file, base_model$weights, in_sep=',')
```

```
## $loglik
## [1] -258.9787
##
## $predictions
##              UR           SR Freq StarComplex Max Predicted Probability Observed Probability       Error
## 1:        'stad        'stad   40           1   0                  0.35                  0.4 -0.05000002
## 2:        'stad         'tad   60           0   1                  0.65                  0.6  0.05000002
## 3:        'grav        'grav   60           1   0                  0.35                  0.6 -0.25000002
## 4:        'grav         'gav   40           0   1                  0.65                  0.4  0.25000002
## 5: spa.gə.'ti spa.gə.'ti   10           1   0                  0.35                  0.1  0.24999998
## 6: spa.gə.'ti  pa.gə.'ti   90           0   1                  0.65                  0.9 -0.24999998
## 7:     gry.'o     gry.'o   30           1   0                  0.35                  0.3  0.04999998
## 8:     gry.'o      gy.'o   70           0   1                  0.65                  0.7 -0.04999998
```

Reading a new file

- New constraints that care about stress and sonority
  - MaxStressed
  - SSP: Sonority Sequencing Principle (st, sp are bad)

outputs

constraints

			*Complex	Max	MaxStressed	SSP
			*Complex	Max	MaxStressed	SSP
'stad	'stad	40	1			1
	'tad	60		1	1	
'grav	'grav	60	1			
	'gav	40		1	1	
spa.gə.'ti	spa.gə.'ti	10	1			1
	pa.gə.'ti	90		1		
gry.'o	gry.'o	30	1			
	gy.'o	70		1		

inputs

frequency of each candidate

- Fit the new grammar

```
full_file <- "amp_demo_grammar_full.csv"
full_model <- optimize_weights(full_file, in_sep=',')
```

- And have a look
  - Now it captures both stress and sonority effects

```
predict_probabilities(full_file, full_model$weights, in_sep=',')
```

```
##             UR         SR Freq *Complex Max MaxStressed SSP Predicted Probability Observed Probability       Error
## 1:       'stad      'stad   40        1   0           0   1             0.3769828                  0.4 -0.02301723
## 2:       'stad       'tad   60        0   1           1   0             0.6230172                  0.6  0.02301723
## 3:       'grav      'grav   60        1   0           0   0             0.6230123                  0.6  0.02301233
## 4:       'grav       'gav   40        0   1           1   0             0.3769877                  0.4 -0.02301233
## 5: spa.gə.'ti spa.gə.'ti   10        1   0           0   1             0.1230143                  0.1  0.02301433
## 6: spa.gə.'ti  pa.gə.'ti   90        0   1           0   0             0.8769857                  0.9 -0.02301433
## 7:      gry.'o     gry.'o   30        1   0           0   0             0.2769861                  0.3 -0.02301393
## 8:      gry.'o      gy.'o   70        0   1           0   0             0.7230139                  0.7  0.02301393
```

Model comparison

- Does the full model's better fit justify its greater complexity?
- `compare_models()` function will tell you, under various measures
  - Here, we show BIC
- Answer: yes
  - Full grammar's BIC is much lower than base grammar's
  - Also lower than an intermediate grammar that we didn't show: MaxStress but no SSP)
  - Lower BIC means better grammar, even taking complexity into account

```
compare_models(base_model, stressed_model, full_model, method='bic')
```

```
##                        model k   n      bic bic.delta       bic.wt    cum.wt
## 1     amp_demo_grammar_full 4 400 481.5879   0.00000 9.989964e-01 0.9989964
## 2 amp_demo_grammar_stressed 3 400 495.3942  13.80633 1.003595e-03 1.0000000
## 3     amp_demo_grammar_base 2 400 529.9402  48.35233 3.162196e-11 1.0000000
```

What about an even more complex model?

- DoTheRightThing
  - Penalizes the forms that our full model was slightly over-predicting

			*Complex	Max	MaxStressed	SSP	DoTheRightThing
			*Complex	Max	MaxStressed	SSP	DoTheRightThing
'stad	'stad	40	1			1	
	'tad	60		1	1		1
'grav	'grav	60	1				1
	'gav	40		1	1		
spa.gə.'ti	spa.gə.'ti	10	1			1	1
	pa.gə.'ti	90		1			
gry.'o	gry.'o	30	1				
	gy.'o	70		1			1

Yes, it fits even better…

```
overfit_file <- "amp_demo_grammar_overfit.csv"
overfit_model <- optimize_weights(overfit_file, in_sep=',')
predict_probabilities(overfit_file, overfit_model$weights, in_sep=',')
```

```
## $loglik
## [1] -228.1971
##
## $predictions
##              UR          SR Freq *Complex Max MaxStressed SSP DoTheRightThing Predicted Probability Observed Probability         Error
## 1:        'stad       'stad   40        1   0           0   1               0             0.3999982                  0.4 -1.838151e-06
## 2:        'stad        'tad   60        0   1           1   0               1             0.6000018                  0.6  1.838151e-06
## 3:        'grav       'grav   60        1   0           0   0               1             0.6000000                  0.6  6.001617e-09
## 4:        'grav        'gav   40        0   1           1   0               0             0.4000000                  0.4 -6.001617e-09
## 5: spa.gə.'ti spa.gə.'ti   10        1   0           0   1               1             0.1000001                  0.1  1.120851e-07
## 6: spa.gə.'ti  pa.gə.'ti   90        0   1           0   0               0             0.8999999                  0.9 -1.120851e-07
## 7:       gry.'o      gry.'o   30        1   0           0   0               0             0.2999981                  0.3 -1.860377e-06
## 8:       gry.'o       gy.'o   70        0   1           0   0               1             0.7000019                  0.7  1.860377e-06
```

…but the fit didn't improve enough to justify the additional constraint

best BIC is still the "full" model (4 constraints)

```
compare_models(base_model, stressed_model, full_model, overfit_model, method='bic')
```

```
##                          model k   n      bic bic.delta       bic.wt    cum.wt        ll
## 1       amp_demo_grammar_full 4 400 481.5879  0.000000 9.145853e-01 0.9145853 -228.8110
## 2    amp_demo_grammar_overfit 5 400 486.3514  4.763534 8.449594e-02 0.9990812 -228.1971
## 3   amp_demo_grammar_stressed 3 400 495.3942 13.806325 9.187953e-04 1.0000000 -238.7099
## 4       amp_demo_grammar_base 2 400 529.9402 48.352330 2.895004e-11 1.0000000 -258.9787
```

Using a prior

- People who use MaxEnt typically use a prior
  - aka *regularization*, *smoothing*, *bias*
- Rather than optimizing log likelihood (model fit), optimize log likelihood *minus* a penalty for weights that depart from their default
- You can use a very agnostic default (“weights should be zero”), which works against overfitting
  - See Martin (2011)
- Or you can use a more content-ful default to build in phonetic and other biases
  - See Wilson (2006), White (2017)

Gaussian prior using `optimize_weights()`

- In function `optimize_weights`, use the arguments `mu_scalar` and `sigma_scalar` to set same bias for all constraints
  - Here we given all constraints a default weight (μ) of 0
  - And a "willingness to depart from μ" (σ) of 0.5

```
full_model_map <- optimize_weights(full_file, in_sep=',', mu_scalar=0, sigma_scalar=0.5)
```

- You can also set different μ and σ for each constraint, or read them from a file

Additional functionality not covered here

- Save model predictions to output file
- Change parameters of optimizer
- Set a temperature parameter for predicting new data
    - Should predictions be exactly same as model predictions?
    - Or closer to 50%-50%?
    - Or closer to 100%-0%?
    - see e.g. Hayes & al. (2009)

Future plans

- Cross-validation for choosing values of μ and σ
- Read input data from R data frame
- Submit to CRAN to make it an official R package!

Reminder about where to get everything

- Package: github.com/connormayer/maxent.ot
    - But remember you can also just install it using the devtools library in R
- Tutorial: connormayer.com/misc/amp_2022_tutorial.zip

Thank you and we hope you try it out!

References

Goldwater, Sharon & Mark Johnson. 2003. Learning OT Constraint Rankings Using a Maximum Entropy Model. In *Stockholm University*, 111–120.
Hayes, Bruce, Bruce Tesar & Kie Zuraw. 2014. OTSoft 2.3.3. http://www.linguistics.ucla.edu/people/hayes/otsoft/.
Hayes, Bruce & Colin Wilson. 2008. A Maximum Entropy Model of Phonotactics and Phonotactic Learning. *Linguistic Inquiry* 39(3). 379–440.
Hayes, Bruce, Colin Wilson & Ben George. 2009. Maxent Grammar Tool. http://www.linguistics.ucla.edu/people/hayes/MaxentGrammarTool/.
Hayes, Bruce, Kie Zuraw, Zsuzsa Cziráky Londe & Peter Siptár. 2009. Natural and unnatural constraints in Hungarian vowel harmony. *Language* 85. 822–863.
Knuth, Donald E. 1992. *Literate Programming*. Cambridge University Press.
Martin, Andrew. 2011. Grammars leak: modeling how phonotactic generalizations interact within the grammar. *Language* 87(4). 751–770.
Mayer, Connor. 2021. *Issues in Uyghur Backness Harmony: Corpus, Experimental, and Computational Studies*. PhD dissertation, University of California, Los Angeles
Microsoft Corporation. 2018. Microsoft Excel. Software.

References

Moore-Cantwell, Claire & Joe Pater. 2016. Gradient exceptionality in Maximum Entropy Grammar with lexically specific constraints. *Catalan Journal of Linguistics* 15. 53–66.

Prince, Alan & Paul Smolensky. 2008. *Optimality Theory: Constraint Interaction in Generative Grammar*. John Wiley & Sons.

R Core Team. 2021. R: a language and environment for statistical computing. Vienna: R Foundation for Statistical Computing. www.R-project.org.

Rose, Yvan. 2002. Relations between segmental and prosodic structure in first language acquisition. *Annual Review of Language Acquisition*. John Benjamins 2(1). 117–155.

Stodden, Victoria, Friedrich Leisch & Roger D Peng (eds.). 2014. *Implementing Reproducible Research* (Chapman & Hall/CRC The R Series). Boca Raton, FL: Chapman and Hall/CRC.

White, James. 2017. Accounting for the learnability of saltation in phonological theory: A maximum entropy model with a P-map bias. *Language*. Linguistic Society of America 93(1). 1–36.

Wilson, Colin. 2006. Learning Phonology With Substantive Bias: An Experimental and Computational Study of Velar Palatalization. *Cognitive Science* 30(5). 945–982.

Zuraw, Kie & Bruce Hayes. 2017. Intersecting constraint families: An argument for harmonic grammar. *Language*. Linguistic Society of America 93(3). 497–548.