ParseEval*in R

Documentation & Reference

Bastian Auris

18th March 2013

ParseEval is a function for assessing certain aspects of articulatory data in phonetic research. It is intended to be used to evaluate the fit between measurements taken from experimental data and measurements taken from simulated data generated under different syllable parses.
The purpose of this document is twofold. First off to act as a reference for using the ParseEval function and secondly to provide a hands-on guide (including a walk-through of example parses, etc.) to the possibilities when working with ParseEval in R [1]. (This documentation assumes a basic level of familiarity with R.)

The ParseEval function on the other hand should be working with any recent version of R (R version $\geq$ 2.14). Although it is recommended to stay up-to-date on your R-installation and any additional packages you might be using. Furthermore for the basic use of ParseEval it is *not* required to have any additional packages installed; a basic, working install of R should be sufficient to produce results.

Although when it comes to plotting diagnostics (see section 2.7), it is preferred to utilize the `lattice` framework and the `gridExtra` package. In those cases it may be required to install additional packages. How to get this worked out is included in the description.
Skip to section 2 for the 'User Guide'. If you feel comfortable enough to get in the middle of things right away, follow this link to the Quick start guide.

ParseEval is the name of the original MATLAB implementation introduced by Shaw et al. (2009). Since this is only an implementation of a similar functionality on a different platform i decided to keep it for practical reasons.

[1] R Core Team (2013). *R: A Language and Environment for Statistical Computing*. ISBN 3-900051-07-0. R Foundation for Statistical Computing. Vienna, Austria. URL: `http://www.R-project.org/` (visited on 18/03/2013)

Contents

List of Figures

List of Tables

1 Introduction

1.1 License

1.2 Feedback

Please use the ParseEval project page on GitHub to report bugs and submit feature requests[2]. Before making a feature request, please ensure that you have thoroughly studied this manual. If you do not want to report a bug or request a feature but are simply in need of assistance, you might want to consider doing a web-search[3], visit Stack Overflow[4] or posting your question on a relevant newsgroup.

1.3 Contributing

Contributing to any part of the project is strongly encouraged. Contributions could range from proposed changes/improvements regarding the source-code, to work on the documentation (e.g. translating it), to being willing to share phonetic data for (reproducible) examples to be added to this document in future releases.

1.4 Acknowledgements

I would like to thank Jason A. Shaw, whose publications (Shaw et al. 2009; Shaw et al. 2011) inspired my work on ParseEval in `R`. He also provided me with the original `MATLAB` script and an extensive documentation accompanying it (on which I am going to draw on in parts during the course of this documentation). Moreover his helpful suggestions during the early stages of this project were essential in getting things rolling.

Furthermore I would like to thank Doris Mücke, who supervised my BA thesis, which resulted in ParseEval for `R`.

Thanks to Anne Hermes and Martine Grice, for providing me with the Italian corpus-data that was analysed as part of the BA thesis (additionally used for the examples in the Quick start guide) and for sharing their phonetic insight on Italian on numerous occasions whenever questions arose.

[2] `https://github.com/phncgn-ba/parseeval_in_R`
[3] e.g., via Google
[4] `http://stackoverflow.com/questions/tagged/r`

1.5 Prerequisites

This section provides an overview of all resources required to achieve basic results running ParseEval in R. In case you find yourself struggling with R I would like to recommend the following:

Impatient R click on the link to → Impatient-R which is a brief yet insightful intro to working with R.

The R Inferno excellent, (somewhat) humorous, book-sized guide → The R Inferno. To quote the author:'If you are using R and you think you're in hell, this is a map for you.'.
This book aspires to be your guide while you venture through R, adapting Dante Alighieri's famous work, 'Divine Comedy'.

Onto the requirements...

Requirements

R **software:** first and foremost you need to be able to access a running installation of the R software. You can download R from `http://www.r-project.org/` for the operating system of your choice. This website also offers extensive manuals for R, FAQ's and HOWTO's.
Basic introduction to working with R cannot be covered by this documentation, but even if you have never seen or used R, the resources available on the website above should get you started.

ParseEval (function): Get the latest version of the ParseEval function (and possibly this documentation) from the ParseEval project-page on Github.

2 User Guide

The following sections try to mimic the documents that one usually finds accompanying `R` packages. Although at this point in time ParseEval is not part a package, this might change in the future. The following sections are taken from Jason A. Shaw's original documentation of the MATLAB script, and adjusted for the purpose of this document.

2.1 Description

(from Jason Shaw's original documentation)

ParseEval is the `R` implementation of a MATLAB script that evaluates the fit between measurements taken from experimental data and measurements taken from simulated data generated under different syllable parses. A description of the evaluation procedure is provided in Shaw and Gafos (2010). ParseEval simulates data based upon either a tautosyllabic or heterosyllabic parse of initial consonant clusters. On the tautosyllabic parse of a CCVX sequence both consonants are parsed into a single syllable ('X' indicates a don't care symbol standing in for any string of consonants and vowels). On a heterosyllabic parse of the same string, a syllable boundary is inserted between initial consonants, e.g. [C.CVX-], where '.' indicates a syllable boundary. ParseEval returns a summary of the fit between these parses of initial consonant clusters and measurements taken from the experimental data.

In the case of the heterosyllabic [C.CVX] parse, ParseEval does not assume anything about the syllabic role of the first consonant (or any consonants that may precede it as in [CC.CVX]). That consonant may be a coda of a preceding syllable or a syllable by itself as has been claimed in some Maghrebian (Dell & Elmedlaoui, 2002) or Southeast Asian languages (Svantesson, 1983). More specifically, the script cannot be used to make inferences about the syllabic role of that consonant. The script calculates statistics summarizing the goodness of fit of a syllable parse to the experimental data and returns a quantitative index of syllable parse evaluation. This index can be used to compare competing hypotheses about the syllabic organization of the experimental data.

2.2 Usage

```
parseEval( c1, c2, c3, cipi12, cipi23, voweld, anchor = 15, var_anchor = 3,
          types, words = 30, simN = 1000, RE_rsd, CC_rsd, LE_rsd)
```

2.3 Arguments

Common sense dictates input of articulatory durations (plateau durations, vowel duration) in the same unit coherently (e.g. milliseconds).

c1 – represents the articulatory data for the immediate pre-vocalic consonant

- must be a numeric vector with two elements
- must be specified in every instance
- the first element equals the (mean) gestural *plateau duration*, the second element equals the respective *standard deviation*
- Example: `c1=c(45, 14)`

c2 – represents the articulatory data for the second consonant in tri-consonantal, or the first consonant in bi-consonantal clusters

- must be a numeric vector with two elements
- must be specified in every instance
- specified analogous to `c1` (see Example under **c1**)

c3 – represents the articulatory data of the initial consonant in tri-consonantal clusters

- must be a numeric vector with two elements
- must be specified for every instance
- specified analogous to `c1` (see Example under **c1**)

cipi12 – represents the duration of the interval between the plateaus of the first two consonants in tri-consonantal clusters (right-edge of C3 to left-edge of C2)

- must be a numeric vector with two elements
- must be specified for every instance
- the first element equals the (mean) *inter-plateau duration*, the second element equals the respective *standard deviation*
- Example: `cipi12=c(0, 12)`

cipi23 – represents the duration of the interval between the plateaus of the first two consonants in bi-consonantal clusters or the duration of the interval between the middle and the pre-vocalic consonants in tri-consonantal clusters (right-edge C2 to left-edge C1)

- must be a numeric vector with two elements
- must be specified for every instance
- specified analogous to `cipi12` (see Example under **cipi12**)

voweld – represents the articulatory duration of the vocalic gesture

- must be a numeric input
- must be specified for every instance
- Example: `voweld=230`

anchor – represents the number of stepwise increases in variability simulated by the model

- must be an integer input
- can be specified optionally
- defaults to `anchor=15`

var_anchor – represents the size (in milliseconds) of each stepwise increase in variability

- must be an integer input
- can be specified optionally
- defaults to `var_anchor=3`

types – represents the number of different word types, e.g. #CV-, #CCV-, #CCCV-

- must be an integer input
- must be either
  2 (for bi-consonantal clusters) or
  3 for tri-consonantal clusters
- must be specified for every instance
- Example: `types=2`

words – represents the number of words (stimuli) per word type

- must be an integer input
- can be specified optionally
- due to internal calculations this input must be divisible by the value of `types` without a remainder
- defaults to `words=12`

simN – represents the number of simulation runs over which hit rate is calculated

- must be an integer input
- can be specified optionally
- impacts the duration of the function call the most

- defaults to `simN=1000`
- see section 2.5 for notes on computational performance

RE_rsd – represents *experimental data*, here the *relative standard deviation* of the right edge to anchor interval

- must be a numeric input
- must be specified for every instance
- Reasons for using relative standard deviation (RSD) and details on how to calculate RSD are provided in Shaw et al. (2009, section 3:3)

CC_rsd – represents *experimental data*, here the *relative standard deviation* of the center to anchor interval

- must be a numeric input
- must be specified for every instance

LE_rsd – represents *experimental data*, here the *relative standard deviation* of the left edge to anchor interval

- must be a numeric input
- must be specified for every instance

2.4 Value

A call to parseEval returns a visible output to the R console which offers a brief description on the overall performance of the simulation. In addition to that the function also invisibly returns a list-object, containing information about the simulation and assorted data for diagnostic plotting. This necessitates that you save the function output to a variable in your workspace (see Examples).

```
parse1 <- parseeval(...)
```

Given that you ran the above code (the ... are place holders for whichever arguments you may have chosen), the (list-)object `parse1` contains the following information and you can access it by typing:

`parse1[[1]]` returns a summary of model performance

`parse1[[2]]` returns the information of the type of simulation run (bi- or tri-consonantal clusters)

`parse1[[3]]` contains a data frame to plot RSD to anchor diagnostics

`parse1[[4]]` contains a data frame to plot number of Hits to anchor diagnostics

`parse1[[5]]` contains a data frame to plot median R-squared to anchor across the whole simulation diagnostics

`parse1[[6]]` contains another list object, that contains the $2 * 1000$ outcomes of the regression analyses

`parse1[[7]]` contains another list object, that in turn contains two data frames (one for simple parse, one for complex parse) that each contain (in the default case) 15,000 observations (covering the whole data of the simulation) on

- F-statistics, R-squared and
- SigFit (which determines either a 'Hit' in which case it equals 1 or 'no Hit' then it equals 0.

2.5 Note

2.5.1 Computational Performance

This sections tries to give a brief overview about computational performance when you use ParseEval in `R`.

Running a simulation of `simN=1000` trials, for `types=3` (tri-consonantal data) and `words=30` tokens takes about 2.5 minutes on a contemporary multi-core laptop (Intel I3 cpu @ 2.4GHz).

The source-code makes use of vectorization from the `*pply`-family of functions on multiple occasions, which could be the bases for potential parallel computations.

In its current state the output of a call to ParseEval is rather big in terms of occupying system memory space. A single simulation output takes up roughly 27Mbyte of RAM. Depending on your hardware you will experience sluggishness when there is a large number of simulation outcomes sitting in your `R` workspace.

2.6 Examples

(The following examples ([a] and [b]) make use of data originally collected and analysed by Jason Shaw and are taken from his original documentation of the `MATLAB` script)

[a]

American English, lot – pot – sot – plot – spot – splot (Browman and Goldstein 1988)

This example tries to replicate the findings of data based on an analyses of tri-consonantal clusters in American English. Copy & paste the example code to `R` to start the

simulation.

```
set.seed(2954) # make examples reproducible

parse1<-parseeval.test(c1=c(45,14), c2=c(45,14), c3=c(45,14), cipi12=c(0,12),
    cipi23=c(0,12), voweld=230, var_anchor=5, types=3, words=30, RE_rsd=0.230, CC_
    rsd=.080, LE_rsd=.140)
```

Running the code should result in:

```
Overall Quality of Modell-Performance (Testing Triads)
(Ratio of:  Total Number of (any Anchor-)Hits / Number of Simulations)
------------------------
Simple Modelling: 0.442       442 / 1000

Complex Modelling:  1.424       1424 / 1000
```

The results indicate that based on the experimental data, American English tri-consonantal clusters are more likely organized based on the assumptions of the complex onset hypothesis. Inspecting the diagnostic plots created with the example code in 2.7 should clarify the interpretation of the results.

[b]

Moroccan Arabic, bulha – sbulha – ksbulha (Shaw et al. 2009)

This example tries to replicate the findings of data based on an analyses of tri-consonantal clusters in Moroccan Arabic. Copy & paste the example code to R to start the simulation.

```
set.seed(2954) # make examples reproducible

parse2<-parseeval.test(c1=c(42,20), c2=c(42,20), c3=c(42,20), cipi12=c(66,20),
    cipi23=c(66,20), voweld=196, var_anchor=5, types=3, words=30, RE_rsd=0.112, CC
    _rsd=.159, LE_rsd=.246)
```

Running the code should result in:

```
Overall Quality of Modell-Performance (Testing Triads)
(Ratio of:  Total Number of (any Anchor-)Hits / Number of Simulations)
------------------------
Simple Modelling: 2.572       2572 / 1000

Complex Modelling:  0           0 / 1000
```

The results indicate that based on the experimental data, Moroccan Arabic tri-consonantal clusters are likely organized based on the assumptions of the simple onset hypothesis. Inspecting the diagnostic plots created with the example code in 2.7 should clarify the interpretation of the results.

2.7 Diagnostic plots with `lattice`

This section covers one of many possible approaches for plotting diagnostics after running ParseEval. Due to personal preference I will be using the graphical framework provided by the `lattice`-package[5].

Installing packages

- `install.packages("lattice")`
  - copy&paste the above line to your `R` command-line and hit the 'Return'-key
  - this should prompt you with a new window, where you can choose a CRAN mirror
  - after choosing the mirror, it should take a short time for the installation to finish
- `install.packages("gridExtra", dependencies = TRUE)`
  - copy&paste the above line to your `R` command-line and hit the 'Return'-key
  - again, it should take a short time for the installation to finish

Please make sure, that you run the example code in [a] *before* you try to run the diagnostic plot script. Since the example code provides the object `parse1` in your working environment in `R`, which is needed for the diagnostic plots.
After this you should be ok to try the code below for various formats of diagnostic plots.

[5] Deepayan Sarkar (2008). *Lattice: Multivariate Data Visualization with R*. New York: Springer. URL: `http://lmdvr.r-forge.r-project.org` (visited on 18/03/2013)

```
require(lattice, quietly=T)
require(latticeExtra, quietly=T)
require(gridExtra, quietly=T)
p1_s<-xyplot((parse_s*100)~as.factor(anchorindex), groups=edge, type="o", pch=20,
    cex=0.4, lwd=.5, main="(a.1)", xlab="Anchorindex", ylab="Median RSD (%)",
    scales=list(x=list(cex=.48)), auto.key=list(space="top", text=c("CC to anchor"
    , "LE to anchor", "RE to anchor"), points=F, lines=T, columns=2, cex=.7,
    padding.text=2), data=parse1$Plot_1)
p1_c<-xyplot((parse_c*100)~as.factor(anchorindex), groups=edge, type="o", pch=20,
    cex=0.4, lwd=.5, main="(b.1)",xlab="Anchorindex", ylab="Median RSD (%)",
    scales=list(x=list(cex=.48)), auto.key=list(space="top", text=c("CC to anchor"
    , "LE to anchor", "RE to anchor"), points=F, lines=T, columns=2, cex=.7,
    padding.text=2), data=parse1$Plot_1)
p2_s<-xyplot(parse_s~as.factor(anchorindex), type="o", pch=20, lty=1, cex=0.5,
    main="(a.2)", xlab="Anchorindex", ylab="Hits (F-Statistics > 98.503)", scales=
    list(x=list(cex=.48)), data=parse1$Plot_2)
p2_c<-xyplot(parse_c~as.factor(anchorindex), type="o", pch=20, lty=1, cex=0.5,
    main="(b.2)", xlab="Anchorindex", ylab="Hits (F-Statistics > 98.503)", scales=
    list(x=list(cex=.48)), data=parse1$Plot_2)
rs.3<-xyplot(rs_median~as.factor(anchorindex),
             panel=function(...) {
               panel.xyplot(...)
               panel.abline(h = 1.0, lty = 2, lwd = .5)
             },
             type="o", pch=20, cex=0.4, lwd=.5, subset=parse=="simp", main="(a.3)"
                 , xlab="Anchorindex", ylab=expression("Median"~R^2), scales=list(
                 x=list(cex=.48)), data=parse1$Plot_3$median)
rs.4<-xyplot(rs_median~as.factor(anchorindex),
             panel=function(...) {
               panel.xyplot(...)
               panel.abline(h = 1.0, lty = 2, lwd = .5)
             },
             type="o", pch=20, cex=0.4, lwd=.5, subset=parse=="comp", main="(b.3)"
                 , xlab="Anchorindex", ylab=expression("Median"~R^2), scales=list(
                 x=list(cex=.48)), data=parse1$Plot_3$median)
# open pdf-device simplex diagnostics
pdf(file="diag_simple.pdf", paper="a4", width=0, height=0)
grid.arrange(p1_s, p2_s, rs.3, as.table=FALSE, ncol=1)
dev.off()
# open pdf-device complex diagnostics
pdf(file="diag_complex.pdf", paper="a4", width=0, height=0)
grid.arrange(p1_c, p2_c, rs.4, as.table=FALSE, ncol=1)
dev.off()
# open pdf-device both diagnostics
pdf(file="diag_allh.pdf", paper="a4r", width=0, height=0)
grid.arrange(p1_s, p2_s, rs.3, p1_c, p2_c, rs.4, as.table=FALSE, ncol=3)
dev.off()
# open pdf-device both diagnostics
pdf(file="diag_allv.pdf", paper="a4", width=0, height=0)
grid.arrange(p1_s, p1_c, p2_s, p2_c, rs.3, rs.4, as.table=FALSE, ncol=2)
dev.off()
```

./diagnostic_plot.R

You can either copy & paste the code from the box, or alternatively get the script-file from github.

If the code executes without errors, there should be four PDF-files saved to your working directory. All the file-names should have the format `diag_*.pdf`.

`diag_simple.pdf` provides three diagnostic plots for the part of the simulation that assumes *Simple Onset Hypothesis*

`diag_complex.pdf` provides three diagnostic plots for the part of the simulation that assumes *Complex Onset Hypothesis*

`diag_allh.pdf` combines the above two categories in one landscape oriented plot

- plots marked with 'a' denote assumption *Simple Onset Hypothesis*
- plots marked with 'b' denote assumption *Complex Onset Hypothesis*

`diag_allv.pdf` combines the first two categories in a portrait oriented plot

- plots marked with 'a' denote assumption *Simple Onset Hypothesis*
- plots marked with 'b' denote assumption *Complex Onset Hypothesis*

2.8 Evaluating the Results

> „In evaluating the fit of a syllable parse to experimental data, ParseEval simulates the target syllabic parse at different levels of variability and evaluates the experimental data at each variability level.
> This technique is motivated by the demonstration that the phonetic indices of any given syllable parse can change under different levels of variability – more generally, the correspondence between phonological organization (syllable parse) and phonetic manifestations of that organization is not one to one."
>
> Shaw et al. 2009, section 4.2

The 'hit rate' output by the model offers a convenient summary of model performance, but it does not reveal how temporal stability patterns change as a function of variability, how changes in measurements (induced by increases in variability) affect the fit between simulated data and experimental data or the frequency with which a given level of variability maximizes the fit of a syllable parse to the experimental data. This information can be derived by combining the information from the hit rate-plot and the other diagnostic plots. The names of the figures mentioned below correspond to the diagnostic plots:

- `diag_allh.pdf` or
- `diag_allv.pdf`)

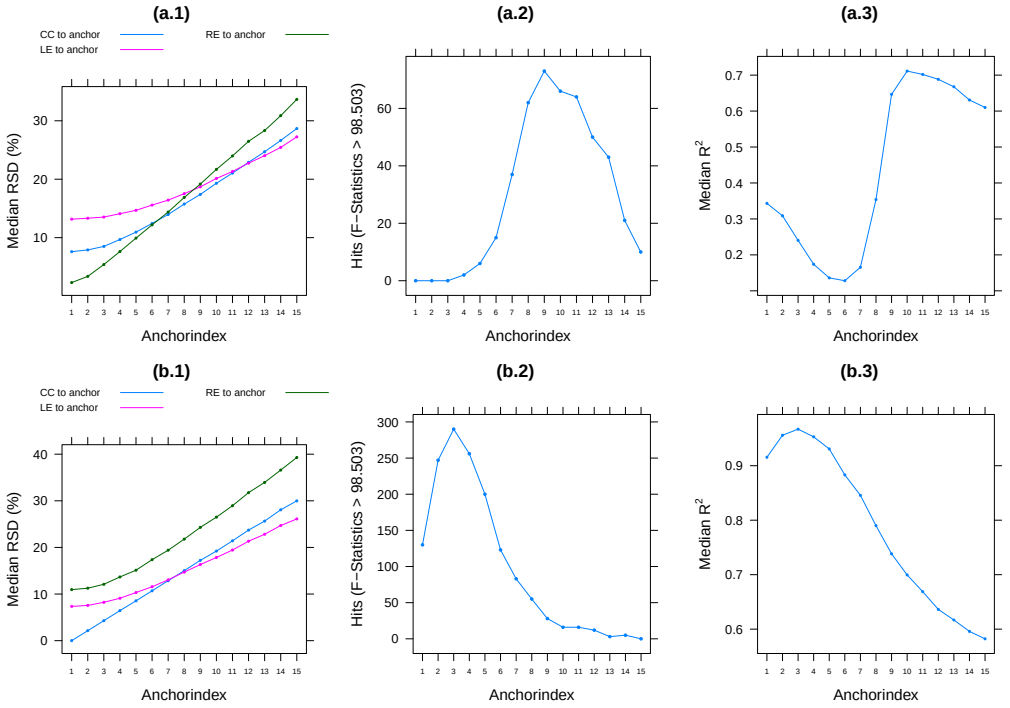

Figure 1: This figure represents the the diagnosticplots for the example given in section 2.6. It shows the outcome of the simulation testing for complex onset hypotheses.

Figure (a/b.1): change in stability pattern as a function of anchor variability. The first figure shows how the relative standard deviation of each interval measured in the simulated data changes as a function of the level of variability in the intervals. This type of figure is used in Shaw et al. (2009, section 4.2) to show that stability patterns change under different conditions of overall variability. In ParseEval, variability is infused in the simulated data through the standard deviation of the anchor point. The size of the variability range considered by the model and the size of the stepwise increase in variability is specified in the function call.

Figure (a/b.2): line-plot of best-fitting variability level. The second figure shows the total number of 'Hits' achieved for each anchor. Each 'Hit' indicates a simulation for the particular anchor, for which the OLS regression yielded F-statistics that were greater than 98.503[6]. The total number of 'Hits' for a given anchor can be used as an indicator to assess how well the experimental data was fit by the simulated data. Anchor-

[6]This particular value for the F-statistics marks a threshold of statistical significance

indices with high numbers of 'Hits' indicate variability-ranges where the simulated data matched the experimental data very well.

Figure (a/b.3): The third figure shows that median R-squared values across all simulations in relation to the anchorindex. In combining the information from Figure 2 and 3 this should further indicate which variability-ranges provided a match between the experimental and the simulated data.

3 Quick start guide

This section skips any information and only represents example code. The data, code and diagnostic plots are taken from Auris (2013).

```
set.seed (2342) # make examples reproducible
parse_mina_smina <- parseEval(c1=c(82.637,9.332),c2=c(84.610,11.070),
c3=c(84.610,11.070),cipi12=c(71.186,15.798),cipi23=c(71.186,15.798),
voweld=307,types=2,RE_rsd=0.311,CC_rsd=0.263,LE_rsd=0.232)
```

Running the above code should produce an outcome looking similar to:

```
Overall Quality of Modell-Performance (Testing Dyads)
(Ratio of:  Total Number of Hits / Number of Simulations)
------------------------
Simple Modelling: 5.832       5832 / 1000

Complex Modelling:  0       0 / 1000
```

With corresponding diagnostic plots looking similar to the following figure.

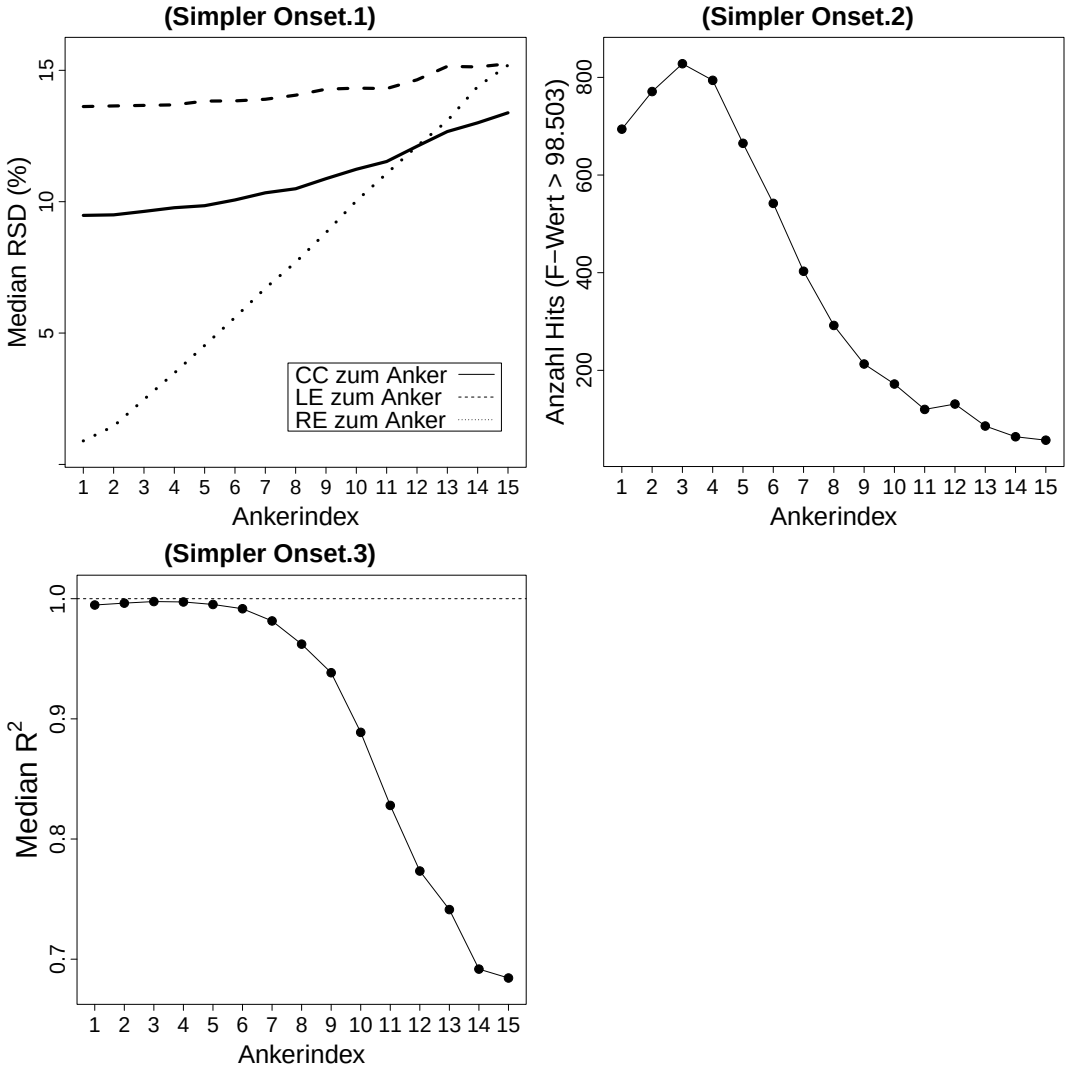

Figure 2: Diagnosticplot generated from a succesfull parse. In particular (a.1-3) showing three plots that visualize the outcome of the parse corresponding to simple onset hypotheses. This parse was taken from analysing data of targetwords /mina/ – /zmina/.

References

Auris, Bastian (2013). 'Silbifizierung von wortinitialen Clusters: Ein quantitatives Modell in R'. unpublished Bachelor thesis, University of Cologne.

Browman, C. P. and L. Goldstein (1988). 'Some Notes on Syllable Structure in Articulatory Phonology'. In: *Phonetica* 45, pp. 140–155. DOI: 10.1159/000261823.

R Core Team (2013). *R: A Language and Environment for Statistical Computing*. ISBN 3-900051-07-0. R Foundation for Statistical Computing. Vienna, Austria. URL: http://www.R-project.org/ (visited on 18/03/2013).

Sarkar, Deepayan (2008). *Lattice: Multivariate Data Visualization with R*. New York: Springer. URL: http://lmdvr.r-forge.r-project.org (visited on 18/03/2013).

Shaw, Jason A. and Adamantios I. Gafos (2010). 'Quantitative evaluation of competing syllable parses'. In: *Proceedings of the 11th Meeting of ACL Special Interest Group in Computational Morphology and Phonology, Uppsala, Sweden*.

Shaw, Jason A., Adamantios I. Gafos, Philip Hoole and Chakir Zeroual (2009). 'Syllabification in Moroccan Arabic: evidence from patterns of temporal stability in articulation'. In: *Phonology* 26 (01), pp. 187–215. DOI: 10.1017/S0952675709001754. URL: http://journals.cambridge.org/article_S0952675709001754.

– (2011). 'Dynamic invariance in the phonetic expression of syllable structure: a case study of Moroccan Arabic consonant clusters'. In: *Phonology* 28 (03), pp. 455–490. DOI: 10.1017/S0952675711000224. URL: http://journals.cambridge.org/article_S0952675711000224.