LingPy Documentation

Release 1.0

Johann-Mattis List

April 16, 2012

CONTENTS

1 Basic Classes **1**
- 1.1 Sequence Modeling (`Model`) . . . 1
- 1.2 Sequence Analysis (`Sequence`) . . . 2
- 1.3 Pairwise Sequence Comparison (`Pairwise`) . . . 3
- 1.4 Multiple Sequence Comparison (`Multiple`) . . . 5
- 1.5 Lexicostatistical Analyses (`LexStat`) . . . 7

2 Specific Modules **9**
- 2.1 Predefined Datasets (`data`) . . . 9
- 2.2 Customizing Sound-Class Models (`derive`) . . . 9
- 2.3 Cluster Algorithms (`cluster`) . . . 11
- 2.4 Miscellaneous Functions (`misc`) . . . 14
- 2.5 Testing the Algorithms (`test`) . . . 19
- 2.6 Evaluation of Automatic Analyses (`evaluate`) . . . 20
- 2.7 Data Plotting (`plot`) . . . 28
- 2.8 What's Next? . . . 30

Python Module Index **31**

CHAPTER

ONE

BASIC CLASSES

1.1 Sequence Modeling (Model)

class lingpy.data.model.**Model**(*model*)

Class for the handling of sound-class models.

Parameters **model** : { 'sca', 'dolgo', 'asjp', 'art', 'color' }

A string indicating the name of the model which shall be loaded. Select between:

- 'sca' - the SCA sound-class model (see List forthcoming),
- 'dolgo' - the DOLGO sound-class model (see: :evobib:'Dolgopolsky1986'),
- 'asjp' - the ASJP sound-class model (see Brown et al. 2008 and Brown, Holman, and Wichmann 2011),
- 'art - the sound-class model which is used for the calculation of sonority profiles and prosodic strings (see List 2012), and
- 'color" - the sound-class model which is used for the coloring of sound-tokens when creating html-output.

See Also:

lingpy.data.derive.compile_model
lingpy.data.derive.compile_diacritics_and_vowels

Notes

Models are loaded from binary files which can be found in the data/models/ folder of the LingPy package. A model has two essential attributes:

- converter – a dictionary with IPA-tokens as keys and sound-class characters as values, and
- scorer – a scoring dictionary with tuples of sound-class characters as keys and scores (integers or floats) as values.

Examples

When loading LingPy, the models sca, asjp, dolgo, and art are automatically loaded:

```
>>> from lingpy import *
```

Check, how the letter a is converted in the various models:

```
>>> for m in [asjp,sca,dolgo,art]:
>>> for m in [asjp,sca,dolgo,art]:
...     print('{0} > {1} ({2})'.format('a',m.converter['a'],m.name))
...
a > a (asjp)
a > A (sca)
a > V (dolgo)
a > 7 (art)
```

Retrieve basic information of a given model:

```
>>> print(sca)
Model:    sca
Info:     Extended sound class model based on Dolgopolsky (1986)
Source:   List (2012)
Compiler: Johann-Mattis List
Date:     2012-03
```

Attributes

converter	dict	A dictionary with IPA tokens as keys and sound-class characters as values.
scorer	dict	A scoring dictionary with tuples of sound-class characters as keys and similarity scores as values.
info	dict	A dictionary storing the key-value pairs defined in the `INFO`.
name	str	The name of the model which is identical with the name of the folder from wich the model is loaded.

1.2 Sequence Analysis (Sequence)

class `lingpy.sequence.`**Sequence**(*seq*, *model=sca*, *merge_vowels=True*)

Basic class for handling sound-class sequences.

Parameters **seq** : str

The input sequence in IPA format.

model : `Model`

A `Model` object. Three models are predefined and automatically loaded when loading LingPy:

- 'dolgo' – a model which is based on the sound-class model of `Dolgopolsky 1964`,
- 'sca' – an extension of the "dolgo" sound-class based on `List forthcoming`, and
- 'asjp' – an independent sound-class model which is based on the sound-class model of `Brown et al. 2008` and the empirical data of `Brown, Holman, and Wichmann 2011`.

merge_vowels : bool (default=True)

Indicate, whether neighboring vowels should be merged into diphtongs, or whether they should be kept separated during the analysis.

Examples

Initialize a sound-class sequence.

```
>>> from lingpy import *
>>> sca = Sequence('t͡sɔygə')
```

Print out its tokens.

```
>>> for token in sca.tokens: print(token)
...
t͡s
ɔy
g
ə
```

Print out its class-string:

```
>>> print(sca.classes)
CUKE
```

Compare the length of the IPA-string with that of the sound-class string.

```
>>> len(sca) == len(sca.ipa)
False
```

Access the third element of the sound-class sequence and the IPA-string.

```
>>> print(sca[3],sca.ipa[3])
ə s
```

Access the prosodic string of the sequence.

```
>>> sca.prostring
'#vC>'
```

Access a trigram representation of the sequence.

```
>>> sca.trigram
['#CU', 'CUK', 'UKE', 'KE$']
```

Attributes

ipa	str	The original format of the input sequence.
tokens	list	A tokenized version of the input sequence.
classes	str	A sound-class representation of the input sequence.
prostring	str	A string-representation of the prosodic environment of the segments.
trigram	list	A list representing the sequence as a trigram.

1.3 Pairwise Sequence Comparison (Pairwise)

class lingpy.compare.**Pairwise**(*infile*, *merge_vowels=True*, *comment='#'*)

Basic class for dealing with the pairwise alignment of sequences.

Parameters **infile** : file

A file in psq-format.

merge_vowels : bool (default=True)

Indicate, whether neighboring vowels should be merged into diphtongs, or whether they should be kept separated during the analysis.

comment : char (default='#')

The comment character which, inserted in the beginning of a line, prevents that line from being read.

Notes

In order to read in data from text files, two different file formats can be used along with this class:

psq-format The `psq`-format is a specific format for text files containing unaligned sequence pairs. Files in this format should have the extension `psq`.

The first line of a `psq`-file contains information regarding the dataset. The sequence pairs are given in triplets, with a sequence identifier in the first line of a triplet (containing the meaning, or orthographical information) and the two sequences in the second and third line, whereas the first column of each sequence line contains the name of the taxon and the second column the sequence in IPA format. All triplets are divided by one empty line. As an example, consider the file `test.psq`:

```
Harry Potter Testset
Woldemort in German and Russian
German  waldemar
Russian vladimir

Woldemort in English and Russian
English woldemort
Russian vladimir

Woldemort in English and German
English woldemort
German  waldemar
```

psa-format The `psa`-format is a specific format for text files containing already aligned sequence pairs. Files in this format should have the extension `psq`.

The first line of a `psa`-file contains information regarding the dataset. The sequence pairs are given in triplets, with a sequence identifier in the first line of a triplet (containing the meaning, or orthographical information) and the aligned sequences in the second and third line, whith the name of the taxon in the first column and all aligned segments in the following columns, separated by tabstops. All triplets are divided by one empty line. As an example, consider the file `test.psa`:

```
Harry Potter Testset
Woldemort in German and Russian
German.    w    a    l    -    d    e    m    a    r
Russian    v    -    l    a    d    i    m    i    r

Woldemort in English and Russian
English    w    o    l    -    d    e    m    o    r    t
Russian    v    -    l    a    d    i    m    i    r    -

Woldemort in English and German
English    w    o    l    d    e    m    o    r    t
German.    w    a    l    d    e    m    a    r    -
```

Attributes

taxa	list	A list of tuples containing the taxa of all sequence pairs.
seqs	list	A list of tuples containing all sequence pairs.
tokens	list	A list of tuples containing all sequence pairs in a tokenized form.

Methods

`align`([model, mode, gop, gep_scale, scale, ...])	Align two sequences or a list of sequence pairs pairwise.
`output`([fileformat, filename])	Write the results of the analyses to a text file.

1.4 Multiple Sequence Comparison (`Multiple`)

class `lingpy.compare.`**`Multiple`**(*infile*, *merge_vowels=True*, *comment='#'*)

Basic class for carrying out multiple sequence alignment analyses.

Parameters **infile** : file

A file in `msq`-format or `msa`-format.

merge_vowels : bool (default=True)

Indicate, whether neighboring vowels should be merged into diphtongs, or whether they should be kept separated during the analysis.

comment : char (default='#')

The comment character which, inserted in the beginning of a line, prevents that line from being read.

Notes

In order to read in data from text files, two different file formats can be used along with this class:

msq-format The `msq`-format is a specific format for text files containing unaligned sequences. Files in this format should have the extension `msq`. The first line of an `msq`-file contains information regarding the dataset. The second line contains information regarding the sequence (meaning, identifier), and the following lines contain the name of the taxa in the first column and the sequences in IPA format in the second column, separated by a tabstop. As an example, consider the file `test.msq`:

```
Harry Potter Testset
Woldemort (in different languages)
German  waldemar
English woldemort
Russian vladimir
```

msa-format The `msa`-format is a specific format for text files containing already aligned sequence pairs. Files in this format should have the extension `msa`.

The first line of a `msa`-file contains information regarding the dataset. The second line contains information regarding the sequence (its meaning, the protoform corresponding to the cognate set, etc.). The aligned sequences are given in the following lines, whereas the taxa are given in the first column and the aligned segments in the following columns. Additionally, there may be a specific line indicating the presence of swaps and a specific line indicating highly consistent sites (local peaks) in the MSA. The line for swaps starts with the headword `SWAPS` whereas a plus character (`+`) marks the beginning of a swapped region, the dash

character (-) its center and another plus character the end. All sites which are not affected by swaps contain a dot. The line for local peaks starts with the headword LOCAL. All sites which are highly consistent are marked with an asterisk (*), all other sites are marked with a dot (.). As an example, consider the file test.msa:

```
Harry Potter Testset
Woldemort (in different languages)
English     w    o    l    -    d    e    m    o    r    t
German.     w    a    l    -    d    e    m    a    r    -
Russian     v    -    l    a    d    i    m    i    r    -
SWAPS..     .    +    -    +    .    .    .    .    .    .
LOCAL..     *    *    *    .    *    *    *    *    *    .
```

Examples

Get the path to a file from the testset.

```
>>> from lingpy import *
>>> seq_file = get_file('test.seq')
```

Load the file into the Multiple class.

```
>>> mult = Multiple(seq_file)
```

Carry out a progressive alignment analysis of the sequences.

```
>>> mult.prog_align()
```

Print the result to the screen:

```
>>> print(mult)
w    o    l    -    d    e    m    o    r    t
w    a    l    -    d    e    m    a    r    -
v    -    l    a    d    i    m    i    r    -
```

Methods

get_pairwise_alignments([new_calc, model, ...])	Function creates a dictionary of all pairwise alignments scores.
get_peaks([gap_weight])	Calculate the profile score for each column of the alignment.
get_pid([mode])	Return the Percentage Identity (PID) score of the calculated MSA.
ipa2cls([model])	Retrieve sound-class strings from aligned IPA sequences.
iterate_all_sequences([check, mode, gop, ...])	Iterative refinement based on a complete realignment of all sequences.
iterate_clusters(threshold[, check, mode, ...])	Iterative refinement based on a flat cluster analysis of the data.
iterate_orphans([check, mode, gop, ...])	Iterate over the most divergent sequences in the sample.
iterate_similar_gap_sites([check, mode, ...])	Iterative refinement based on the *Similar Gap Sites* heuristic.
lib_align([model, mode, modes, scale, ...])	Carry out a library-based progressive alignment analysis of the sequences.
output([fileformat, filename, sorted_seqs, ...])	Write data to file.

Continued on next page

Table 1.2 – continued from previous page

prog_align([model, mode, gop, gep_scale, ...])	Carry out a progressive alignment analysis of the input sequences.
sum_of_pairs([alm_matrix, mat, gap_weight])	Calculate the sum-of-pairs score for a given alignment analysis.
swap_check([swap_penalty, score_mode])	Check for possibly swapped sites in the alignment.

1.5 Lexicostatistical Analyses (LexStat)

class lingpy.lexstat.**LexStat**(*infile*)

Basic class for handling lexicostatistical datasets.

Parameters **infile** : file

A file in lxs-format.

Notes

The LexStat class serves as the base class for the handling of lexicostatistical datasets (see Swadesh 1955 for a detailed description of the method of lexicostatistics). It provides methods for data conversion, when analyses on cognacy have been conducted in a qualitative way, and also allows to carry out cognate judgments automatically, based on the different methods described in List forthcoming.

The input data for LexStat is a simple tab-delimited text file with the language names in the first row, an ID in the first column, and the data in the columns corresponding to the language names. Additionally, the file can contain headwords corresponding to the IDs and cognate-IDs, specifying which words in the data are thought to be cognate. This structure is almost the same as the one employed in the Starling database program (see http://starling.rinet.ru). Synonyms are also specified in the same way by simply adding additional rows with the same ID. The following is an example for the possible structure of an input file:

```
ID  Word    German  COG   English   COG ...
1   hand    hantʰ   1     hæ:nd     1   ...
2   fist    faustʰ  2     fist      2   ...
... ...     ...     ...   ...       ... ...
```

Methods

analyze(threshold[, score_mode, model, ...])	Conduct automatic cognate judgments following the method of List forthcoming.
output([fileformat, filename])	Write the data to file.
pairwise_distances()	Calculate the lexicostatistical distance between all taxa.

CHAPTER

TWO

SPECIFIC MODULES

2.1 Predefined Datasets (data)

LingPy comes along with many different kinds of predefined data. When loading the library, the following data are automatically loaded and can be used in all applications:

ipa_diacritics : unicode
The default string of IPA diacritics which is used for the tokenization of IPA strings.

ipa_vowels : unicode
The default string of IPA vowels which is used for the tokenization of IPA strings.

sca : Model
The SCA sound-class Model (see List 2012).

dolgo : Model
The DOLGO sound-class Model (see Dolgopolsky 1964).

asjp : Model
The ASJP sound-class Model (see Brown et al. 2008 and Brown, Holman, and Wichmann 2011).

art : Model
The ART sound-class Model which is used for the calculation of sonority profiles and prosodic strings (see List 2012).

2.2 Customizing Sound-Class Models (derive)

The module provides functions for the customized compilation of sound-class models. All models are defined in simple text files. In order to guarantee their quick access when loading the library, the models are compiled and stored in binary files.

2.2.1 Customizing Diacritics and Vowels

compile_diacritics_and_vowels()	Function compiles diacritics and vowels.

lingpy.data.derive.compile_diacritics_and_vowels

lingpy.data.derive.**compile_diacritics_and_vowels**()
Function compiles diacritics and vowels.

See Also:

`lingpy.data.model.Model`
`lingpy.data.derive.compile_model`

Notes

Diacritics and vowels are defined in the `data/models/dv/` directory of the LingPy package and automatically loaded when loading the LingPy library. The values are defined as the constants `ipa_diacritics` and `ipa_vowels`. Their core purpose is to guide the tokenization of IPA strings (cf. `ipa2tokens()`). In order to change the variables, one simply has to change the text files `diacritics` and `vowels` in the `data/models/dv` directory. The structure of these files is fairly simple: Each line contains a vowel or a diacritic character, whereas diacritics are preceded by a dash.

2.2.2 Customizing Sound-Class Models

`compile_model`(model)	Function compiles customized sound-class models.

lingpy.data.derive.compile_model

`lingpy.data.derive.`**`compile_model`**(*model*)

Function compiles customized sound-class models.

Parameters **model** : str

A string indicating the name of the model which shall be created.

See Also:

`lingpy.data.model.Model`
`lingpy.data.derive.compile_diacritcs_and_vowels`

Notes

A model is defined by a folder placed in `data/models` directory of the LingPy package. The name of the folder reflects the name of the model. It contains three files: the file `converter`, the file `INFO`, and the optional file `scorer`. The format requirements for these files are as follows:

`INFO` The `INFO`-file serves as a reference for a given sound-class model. It can contain arbitrary information (and also be empty). If one wants to define specific characteristics, like the `source`, the `compiler`, the `date`, or a `description` of a given model, this can be done by employing a key-value structure in which the key is preceded by an `@` and followed by a colon and the value is written right next to the key in the same line, e.g.:

```
@source: Dolgopolsky (1986)
```

This information will then be read from the `INFO` file and rendered when printing the model to screen with help of the `print()` function.

`converter` The `converter` file contains all sound classes which are matched with their respective sound values. Each line is reserved for one class, precede by the key (preferably an ASCII-letter) representing the class:

```
B : ɸ̪, β, f, p͡f, p□f, ɓ
E : ɛ, æ, ɜ, ɐ, ʌ, e, □, ə, ɘ, ɤ, è, é, ē, ě, ê, ɚ
```

```
D : θ, ð, ŧ, þ, đ
G : x, ɣ, χ
...
```

scorer The `scorer` file (which is optional) contains the graph of class-transitions which is used for the calculation of the scoring dictionary. Each class is listed in a separate line, followed by the symbols `v`,"c", or `t` (indicating whether the class represents vowels, consonants, or tones), and by the classes it is directly connected to. The strength of this connection is indicated by digits (the smaller the value, the shorter the path between the classes):

```
A : v, E:1, O:1
C : c, S:2
B : c, W:2
E : v, A:1, I:1
D : c, S:2
...
```

The information in such a file is automatically converted into a scoring dictionary (see `List forthcoming` for details).

Based on the information provided by the files, a dictionary for the conversion of IPA-characters to sound classes and a scoring dictionary are created and stored as a binary. The model can be loaded with help of the `Model` class and used in the various classes and functions provided by the library.

2.3 Cluster Algorithms (`cluster`)

This module provides functions for basic cluster algorithms.

2.3.1 Flat Cluster Algorithms

`flat_upgma`(matrix, threshold[, taxa])	Carry out a flat cluster analysis based on the UPGMA algorithm (`Sokal and Michener 1958`).

lingpy.algorithm.cluster.flat_upgma

`lingpy.algorithm.cluster.`**`flat_upgma`**(*matrix*, *threshold*, *taxa=None*)

Carry out a flat cluster analysis based on the UPGMA algorithm (`ibid.`).

Parameters **matrix** : list or `numpy.array`

A two-dimensional list containing the distances.

threshold : float

The threshold which terminates the algorithm.

taxa : list

A list containing the names of the taxa. If set to `None`, the indices of the taxa will be returned instead of their names.

Returns **clusters** : dict

A dictionary with cluster-IDs as keys and a list of the taxa corresponding to the respective ID as values.

See Also:

```
lingpy.algorithm.clusters.upgma
lingpy.algorithm.clusters.neighbor
```

Examples

The function is automatically imported along with LingPy.

```
>>> from lingpy import *
```

Create a list of arbitrary taxa.

```
>>> taxa = ['German','Swedish','Icelandic','English','Dutch']
```

Create an arbitrary distance matrix.

```
>>> matrix = squareform([0.5,0.67,0.8,0.2,0.4,0.7,0.6,0.8,0.8,0.3])
>>> matrix
array([[ 0.  ,  0.5 ,  0.67,  0.8 ,  0.2 ],
       [ 0.5 ,  0.  ,  0.4 ,  0.7 ,  0.6 ],
       [ 0.67,  0.4 ,  0.  ,  0.8 ,  0.8 ],
       [ 0.8 ,  0.7 ,  0.8 ,  0.  ,  0.3 ],
       [ 0.2 ,  0.6 ,  0.8 ,  0.3 ,  0.  ]])
```

Carry out the flat cluster analysis.

```
>>> flat_upgma(clusters,matrix,0.5)
{0: ['German', 'Dutch', 'English'], 1: ['Swedish', 'Icelandic']}
```

2.3.2 Deep Cluster Algorithms

`upgma`(matrix, taxa[, distances])	Carry out a cluster analysis based on the UPGMA algorithm (`Sokal and Michener 1958`).
`neighbor`(matrix, taxa[, distances])	Function clusters data according to the Neighbor-Joining algorithm (`Saitou and Nei 1987`).

lingpy.algorithm.cluster.upgma

`lingpy.algorithm.cluster.`**`upgma`**(*matrix*, *taxa*, *distances=True*)

Carry out a cluster analysis based on the UPGMA algorithm (`Sokal and Michener 1958`).

Parameters **matrix** : list or `numpy.array`

A two-dimensional list containing the distances.

taxa : list

An list containing the names of all taxa corresponding to the distances in the matrix.

distances : bool

If set to `False`, only the topology of the tree will be returned.

Returns **newick** : str

A string in newick-format which can be further used in biological software packages to view and plot the tree.

See Also:

lingpy.algorithm.cluster.neighbor
lingpy.algorithm.cluster.flat_upgma

Examples

Function is automatically imported when importing lingpy.

```
>>> from lingpy import *
```

Create an arbitrary list of taxa.

```
>>> taxa = ['German','Swedish','Icelandic','English','Dutch']
```

Create an arbitrary matrix.

```
>>> matrix = squareform([0.5,0.67,0.8,0.2,0.4,0.7,0.6,0.8,0.8,0.3])
```

Carry out the cluster analysis.

```
>>> upgma(matrix,taxa,distances=False)
'((Swedish,Icelandic),(English,(German,Dutch)));'
```

lingpy.algorithm.cluster.neighbor

lingpy.algorithm.cluster.**neighbor**(*matrix*, *taxa*, *distances=True*)

Function clusters data according to the Neighbor-Joining algorithm (Saitou and Nei 1987).

Parameters **matrix** : list or numpy.array

A two-dimensional list containing the distances.

taxa : list

An list containing the names of all taxa corresponding to the distances in the matrix.

distances : bool

If set to False, only the topology of the tree will be returned.

Returns **newick** : str

A string in newick-format which can be further used in biological software packages to view and plot the tree.

See Also:

lingpy.algorithm.cluster.upgma
lingpy.algorithm.cluster.flat_upgma

Examples

Function is automatically imported when importing lingpy.

```
>>> from lingpy import *
```

Create an arbitrary list of taxa.

```
>>> taxa = ['Norwegian','Swedish','Icelandic','Dutch','English']
```

Create an arbitrary matrix.

```
>>> matrix = squareform([0.5,0.67,0.8,0.2,0.4,0.7,0.6,0.8,0.8,0.3])
```

Carry out the cluster analysis.

```
>>> neighbor(matrix,taxa)
'(((Norwegian,(Swedish,Icelandic)),English),Dutch);'
```

2.4 Miscellaneous Functions (misc)

This module provides miscellaneous functions which are mostly used internally.

2.4.1 Sequence Modeling

ipa2tokens(seq[, diacritics, vowels, ...])	Tokenize IPA-encoded strings.
tokens2class(tokens, model)	Convert tokenized IPA strings into their respective class strings.
class2tokens(tokens, classes[, gap_char])	Turn aligned sound-class sequences into an aligned sequences of IPA tokens.
prosodic_string(seq)	Create a prosodic string of the sonority profile of a sequence.
prosodic_weights(prostring[, scale, factor])	Calculate prosodic weights for each position of a sequence.

lingpy.algorithm.misc.ipa2tokens

lingpy.algorithm.misc.**ipa2tokens**(*seq*, *diacritics=None*, *vowels=None*, *merge_vowels=True*)

Tokenize IPA-encoded strings.

Parameters **seq** : string or unicode

The input sequence that shall be tokenized.

diacritics : unicode

A string containing all diacritics which shall be considered in the respective analysis. When set to *None*, the default diacritic string will be used.

vowels : unicode

A string containing all vowel symbols which shall be considered in the respective analysis. When set to *None*, the default vowel string will be used.

merge_vowels : bool

Indicate, whether vowels should be merged into diphtongs (default=True), or whether each vowel symbol should be considered separately.

Returns **tokens** : list

A list of IPA tokens.

See Also:

tokens2class
class2tokens

Examples

```
>>> from lingpy import *
>>> myseq = 't͡sɔygə'
>>> ipa2tokens(myseq)
[u't\u0361s', u'\u0254y', u'\u0261', u'\u0259']
>>> for t in ipa2tokens(myseq): print t
t͡s
ɔy
g
ə
```

lingpy.algorithm.misc.tokens2class

lingpy.algorithm.misc.**tokens2class**(*tokens*, *model*)

Convert tokenized IPA strings into their respective class strings.

Parameters **tokens** : list

A list of tokens as they are returned from `ipa2tokens()`.

model : `Model`

A `Model` object.

Returns **classes** : string

A sound-class representation of the tokenized IPA string.

See Also:

`ipa2tokens`
`class2tokens`

Examples

```
>>> from lingpy import *
>>> tokens = ipa2tokens('t͡sɔygə')
>>> classes = tokens2class(tokens,sca)
>>> print(classes)
CUKE
```

lingpy.algorithm.misc.class2tokens

lingpy.algorithm.misc.**class2tokens**(*tokens*, *classes*, *gap_char='-'*)

Turn aligned sound-class sequences into an aligned sequences of IPA tokens.

Parameters **tokens** : list

The list of tokens corresponding to the unaligned IPA string.

classes : string or list

The aligned class string.

gap_char : string

The character which indicates gaps in the aligned class string (defaults to "-").

Returns **alignment** : list

A list of tokens with gaps at the positions where they occured in the alignment of the class string.

See Also:

`ipa2tokens`
`tokens2class`

Examples

```
>>> from lingpy import *
>>> tokens = ipa2tokens('t͡sɔygə')
>>> aligned_sequence = 'CU-KE'
>>> print ', '.join(class2tokens(tokens,aligned_sequence))
t͡s, ɔy, -, g, ə
```

lingpy.algorithm.misc.prosodic_string

`lingpy.algorithm.misc.`**`prosodic_string`**(*seq*)

Create a prosodic string of the sonority profile of a sequence.

Returns **prostring** : string

A prosodic string corresponding to the sonority profile of the underlying sequence.

Notes

A prosodic string is a sequence of specific characters which indicating their resprective prosodic context (see `List 2012` or `List forthcoming` for a detailed description).

Examples

```
>>> profile = [int(i) for i in tokens2class(ipa2tokens('t͡sɔygə'),art)]
>>> prosodic_string(profile)
'#vC>'
```

lingpy.algorithm.misc.prosodic_weights

`lingpy.algorithm.misc.`**`prosodic_weights`**(*prostring*, *scale=(1.2, 1.0, 1.1000000000000001)*, *factor=0.29999999999999999*)

Calculate prosodic weights for each position of a sequence.

Parameters **prostring** : string

A prosodic string as it is returned by **:py:function:‘prosodic_string‘**.

scale : tuple or list

A tuple or list of floats indicating the degree by which the gaps in the environment of ascending, maximum, and descending should be decreased or increased.

factor : float

A scaling factor by which the specific positions of initial and final should be increased and decreased.

Returns **weights** : list

A list of floats reflecting the modification of the weight for each position.

See Also:

`prosodic_string`

Notes

Prosodic weights are specific scaling factors which decrease or increase the gap score of a given segment in alignment analyses (see `List 2012` or `List forthcoming` for a detailed description).

Examples

```
>>> from lingpy import *
>>> prostring = '#vC>'
>>> prosodic_weights(prostring)
[1.5600000000000001, 1.0, 1.2, 0.69999999999999996]
>>> prosodic_weights(prostring,scale=(4,1,2),factor=0.5)
[6.0, 1, 4, 0.5]
```

2.4.2 Internal Data Handling

`squareform`(x)	A simplified version of the `scipy.spatial.distance.squareform()` function.
`loadtxt`(infile)	Function imitates the `numpy.loadtxt()` function.
`LingpyArray`(input_list)	An extension of the numpy array object which allows the storage of lists in two-dimensional arrays.

lingpy.algorithm.misc.squareform

`lingpy.algorithm.misc.`**`squareform`**(*x*)

A simplified version of the `scipy.spatial.distance.squareform()` function.

Parameters **x** : `numpy.array` or list

The one-dimensional flat representation of a symmetrix distance matrix.

Returns **matrix** : `numpy.array`

The two-dimensional redundant representation of a symmetric distance matrix.

lingpy.algorithm.misc.loadtxt

`lingpy.algorithm.misc.`**`loadtxt`**(*infile*)

Function imitates the `numpy.loadtxt()` function.

Parameters **infile** : file

The input file from which the data is read.

Returns **data** : list

A list object which renders the dimensions of the input file.

lingpy.algorithm.misc.LingpyArray

class lingpy.algorithm.misc.**LingpyArray**(*input_list*)

An extension of the numpy array object which allows the storage of lists in two-dimensional arrays.

Parameters **input_list** : list

The list which shall be converted in an array-like object.

__init__(*input_list*)

Methods

__init__(input_list)	

2.4.3 Sequence Comparison

pid(almA, almB[, mode])	Calculate the Percentage Identity (PID) score for aligned sequence pairs.

lingpy.algorithm.misc.pid

lingpy.algorithm.misc.**pid**(*almA*, *almB*, *mode=1*)

Calculate the Percentage Identity (PID) score for aligned sequence pairs.

Parameters **almA, almB** : string or list

The aligned sequences which can be either a string or a list.

mode : { 1, 2, 3, 4, 5 }

Indicate which of the four possible PID scores described in Raghava and Barton 2006 should be calculated, the fifth possibility is added for linguistic purposes:

1. identical positions / (aligned positions + internal gap positions),
2. identical positions / aligned positions,
3. identical positions / shortest sequence, or
4. identical positions / shortest sequence (including internal gap pos.)
5. identical positions / (aligned positions + 2 * number of gaps)

Returns **score** : float

The PID score of the given alignment as a floating point number between 0 and 1.

See Also:

lingpy.compare.Multiple.get_pid

Notes

The PID score is a common measure for the diversity of a given alignment. The implementation employed by LingPy follows the description of ibid. where four different variants of PID scores are distinguished. Essentially, the PID score is based on the comparison of identical residue pairs with the total number of residue pairs in a given alignment.

Examples

Load an alignment from the test suite.

```
>>> from lingpy import *
>>> pairs = PSA(get_file('test.psa'))
```

Extract the alignments of the first aligned sequence pair.

```
>>> almA,almB,score = pairs.alignments[0]
```

Calculate the PID score of the alignment.

```
>>> pid(almA,almB)
0.44444444444444442
```

2.5 Testing the Algorithms (test)

The module provides functions to handle the testset provided by LingPy. The current testset consists of test files stored in the folder `lingpy/test/tests/`. These files are formatted according to the requirements for the different methods. Each specific format comes along with a specific file extension. Currently, there are the following five extensions:

1. `lxs` – input files for the `LexStat` class.
2. `psq`, `psq` – input files for the `Pairwise` class.
3. `msq`, `msa` – input files for the `Multiple` class.

All these files can be easily accessed with help of some specific functions defined in this module.

2.5.1 Basic Functions

`get_file`(filename)	Return a path to the filename in the testset.
`list_files`(filetype[, dataset])	List all files in the testset that correspond to a certain filetype.

lingpy.test.test.get_file

lingpy.test.test.**get_file**(*filename*)

Return a path to the filename in the testset.

Parameters **filename** : str

The name of the file (with extension) in the testset.

Examples

```
>>> from lingpy import *
>>> get_file('SLAV.lxs')
'/usr/local/lib/python2.6/dist-packages/lingpy/test/tests/lxs/SLAV.lxs'
```

lingpy.test.test.list_files

lingpy.test.test.**list_files**(*filetype*, *dataset='*'*)

List all files in the testset that correspond to a certain filetype.

Parameters **filetype** : { 'lxs', 'msa', 'msq', 'psa', 'psq' }

The extension of the files that shall be listed.

dataset : str (default='*')

A string which can be used to specify the dataset closer. One can use the Unix wildcard syntax in order to narrow down which files to look for.

Examples

```
>>> from lingpy import *
>>> list_files('msa','sindial*')
sindial_3_1.msa
sindial_3_2.msa
sindial_3_3.msa
sindial_6_1.msa
```

2.6 Evaluation of Automatic Analyses (evaluate)

This is the basic module for the evaluation of automatic analyses. The module consists of three classes which deal with the evaluation of automatic analyses (alignments, cognate judgments). The evaluation is based on the comparison of a gold standard (reference set) with a test set. The different evaluation measures which can be calculated with help of the different classes are essentially all based on the calculation of the proportion to which the test set is similar to the reference set.

The evaluation measures implemented in this module can be divided into two parts: Those measures which deal with the comparison of automatic alignments, and those which deal with the comparison of automatic cognate judgments.

2.6.1 Evaluation of Automatic Sequence Analyses

EvalPSA(gold, test)	Base class for the evaluation of automatic pairwise sequence analyses.
EvalMSA(gold, test)	Base class for the evaluation of automatic multiple sequence analyses.

lingpy.test.evaluate.EvalPSA

class lingpy.test.evaluate.**EvalPSA**(*gold*, *test*)

Base class for the evaluation of automatic pairwise sequence analyses.

Parameters **gold, test** : lingpy.compare.Pairwise

The Pairwise objects which shall be compared. The first object should be the gold standard and the second object should be the test set.

See Also:

lingpy.test.evaluate.EvalMSA

Notes

Moste of the scores which can be calculated with help of this class are standard evaluation scores in evolutionary biology. For a close description on how these scores are calculated, see, for example, Thompson, Plewniak, and Poch 1999, List 2012, and Rosenberg and Ogden 2009.

Methods

c_score()	Calculate column (C) score.
diff([filename])	Write all differences between two sets to a file.
jc_score()	Calculate the Jaccard (JC) score.
pir_score([mode])	Compute the percentage of identical rows (PIR) score.
sp_score()	Calculate the sum-of-pairs (SP) score.

lingpy.test.evaluate.EvalPSA.c_score

EvalPSA.**c_score**()

Calculate column (C) score.

Returns **score** : float

The C score for reference and test alignments.

See Also:

lingpy.test.evaluate.EvalMSA.c_score

Notes

The C score, as it is described in Thompson, Plewniak, and Poch 1999, is calculated by dividing the number of columns which are identical in the gold standarad and the test alignment by the total number of columns in the test alignment.

lingpy.test.evaluate.EvalPSA.diff

EvalPSA.**diff**(*filename=None*)

Write all differences between two sets to a file.

Parameters **filename** : str (default='eval_psa_diff')

Default

lingpy.test.evaluate.EvalPSA.jc_score

EvalPSA.**jc_score**()

Calculate the Jaccard (JC) score.

Returns **score** : float

The JC score.

See Also:

lingpy.test.evaluate.EvalMSA.jc_score

Notes

The Jaccard score (see List 2012) is calculated by dividing the size of the intersection of residue pairs in reference and test alignment by the size of the union of residue pairs in reference and test alignment.

lingpy.test.evaluate.EvalPSA.pir_score

EvalPSA.**pir_score**(*mode=1*)

Compute the percentage of identical rows (PIR) score.

Parameters **mode** : { 1, 2 }

Select between mode 1, where all sequences are compared with each other, and mode 2, where only whole alignments are compared.

Returns **score** : float

The PIR score.

See Also:

lingpy.test.evaluate.EvalMSA.pir_score

Notes

The PIR score is the number of identical rows (sequences) in reference and test alignment divided by the total number of rows.

lingpy.test.evaluate.EvalPSA.sp_score

EvalPSA.**sp_score**()

Calculate the sum-of-pairs (SP) score.

Returns **score** : float

The SP score for reference and test alignments.

See Also:

lingpy.test.evaluate.EvalMSA.sp_score

lingpy.test.evaluate.EvalMSA

class lingpy.test.evaluate.**EvalMSA**(*gold*, *test*)

Base class for the evaluation of automatic multiple sequence analyses.

Parameters **gold, test** : Multiple

The Multiple objects which shall be compared. The first object should be the gold standard and the second object should be the test set.

See Also:

lingpy.test.evaluate.EvalPSA

Notes

Moste of the scores which can be calculated with help of this class are standard evaluation scores in evolutionary biology. For a close description on how these scores are calculated, see, for example, Thompson, Plewniak, and Poch 1999, List 2012, and :evobib:‘Rosenberg2009b.

Methods

c_score([mode])	Calculate the column (C) score.
check_swaps()	Check for possibly identical swapped sites.
jc_score()	Calculate the Jaccard (JC) score.
pir_score()	Compute the percentage of identical rows (PIR) score.
sp_score([mode])	Calculate the sum-of-pairs (SP) score.

lingpy.test.evaluate.EvalMSA.c_score

EvalMSA.**c_score**(*mode=1*)

Calculate the column (C) score.

Parameters **mode** : { 1, 2, 3, 4 }

Indicate, which mode to compute. Select between:

1. divide the number of common columns in reference and test alignment by the total number of columns in the test alignment (the traditional C score described in Thompson, Plewniak, and Poch 1999, also known as “precision” score in applications of information retrieval),
2. divide the number of common columns in reference and test alignment by the total number of columns in the reference alignment (also known as “recall” score in applications of information retrieval),
3. divide the number of common columns in reference and test alignment by the average number of columns in reference and test alignment, or
4. combine the scores of mode 1 and mode 2 by computing their F-score, using the formula $2 * \frac{pr}{p+r}$, where p is the precision (mode 1) and r is the recall (mode 2).

Returns **score** : float

The C score for reference and test alignments.

See Also:

lingpy.test.evaluate.EvalPSA.c_score

Notes

The different c-

lingpy.test.evaluate.EvalMSA.check_swaps

EvalMSA.**check_swaps**()

Check for possibly identical swapped sites.

Returns **swap** : { -2, -1, 0, 1, 2 }

Information regarding the identity of swap decisions is coded by integers, whereas

1 – indicates that swaps are detected in both gold standard and testset, whereas a negative value indicates that the positions are not identical,

2 – indicates that swap decisions are not identical in gold standard and testset, whereas a negative value indicates that there is a false positive in the testset, and

0 – indicates that there are no swaps in the gold standard and the testset.

lingpy.test.evaluate.EvalMSA.jc_score

`EvalMSA.`**`jc_score`**`()`

Calculate the Jaccard (JC) score.

Returns **score** : float

The JC score.

See Also:

`lingpy.test.evaluate.EvalPSA.jc_score`

Notes

The Jaccard score (see `List 2012`) is calculated by dividing the size of the intersection of residue pairs in reference and test alignment by the size of the union of residue pairs in reference and test alignment.

lingpy.test.evaluate.EvalMSA.pir_score

`EvalMSA.`**`pir_score`**`()`

Compute the percentage of identical rows (PIR) score.

Returns **score** : float

The PIR score.

See Also:

`lingpy.test.evaluate.EvalPSA.pir_score`

Notes

The PIR score is the number of identical rows (sequences) in reference and test alignment divided by the total number of rows.

lingpy.test.evaluate.EvalMSA.sp_score

`EvalMSA.`**`sp_score`**`(`*`mode=1`*`)`

Calculate the sum-of-pairs (SP) score.

Parameters **mode** : { 1, 2, 3 }

Indicate, which mode to compute. Select between:

1. divide the number of common residue pairs in reference and test alignment by the total number of residue pairs in the test alignment (the traditional SP score described in `Thompson, Plewniak, and Poch 1999`, also known as “precision” score in applications of information retrieval),

2. divide the number of common residue pairs in reference and test alignment by the total number of residue pairs in the reference alignment (also known as "recall" score in applications of information retrieval),
3. divide the number of common residue pairs in reference and test alignment by the average number of residue pairs in reference and test alignment.

Returns **score** : float

The SP score for gold standard and test alignments.

See Also:

`lingpy.test.evaluate.EvalPSA.sp_score`

Notes

The SP score (see `ibid.`) is calculated by dividing the number of identical residue pairs in reference and test alignment by the total number of residue pairs in the reference alignment.

2.6.2 Evaluation of Lexicostatistical Analyses

`EvalLXS`(gold, test)	Basic class for comparing automatic lexicostatistical analyses.

lingpy.test.evaluate.EvalLXS

class `lingpy.test.evaluate.`**`EvalLXS`**(*gold*, *test*)

Basic class for comparing automatic lexicostatistical analyses.

Parameters **gold, test** : `lingpy.lexstat.LexStat`

The `LexStat` objects which shall be compared. The first object should be the gold standard and the second object should be the test set.

Methods

`compare_pairwise_decisions`(threshold)	Compute the number of identical decisions for pairwise scores.
`pid_score`()	Compute the pairwise identical decisions (PID) score.
`pind_score`()	Compute the pairwise identical negative decisions (PIND) score.
`pipd_score`()	Compute the pairwise identical positive decisions (PIPD) score.
`set_fscore`()	Compute the set f-score (FSC).
`set_precision`()	Compute the set precision (SPR).
`set_recall`()	Compute the set recall (SRE).

lingpy.test.evaluate.EvalLXS.compare_pairwise_decisions

`EvalLXS.`**`compare_pairwise_decisions`**(*threshold*)

Compute the number of identical decisions for pairwise scores.

Parameters **threshold** : float

The threshold which determines the cognacy decisions in the test set.

Returns **scores** : tuple

A tule containing the scores for true positives, true negatives and the general percentage of identical decisions (PID) score.

lingpy.test.evaluate.EvalLXS.pid_score

EvalLXS.**pid_score**()

Compute the pairwise identical decisions (PID) score.

Returns **score** : float

The PID score for reference and test set.

See Also:

`lingpy.test.evaluate.EvalLXS.pind_score`, `lingpy.test.evaluate.EvalLXS.pipd_score`

Notes

The PID score (see `List forthcoming`), is calculated by dividing the number of identical pairwise decisions in reference and test set by the total number of pairwise decisions.

lingpy.test.evaluate.EvalLXS.pind_score

EvalLXS.**pind_score**()

Compute the pairwise identical negative decisions (PIND) score.

Returns **score** : float

The PIND score for reference and test set.

See Also:

```
lingpy.test.evaluate.EvalLXS.pid_score
lingpy.test.evaluate.EvalLXS.pipd_score
```

Notes

The PIND score (see `ibid.`), is calculated by dividing the number of identical pairwise negative decisions in reference and test set by the total number of pairwise negative decisions in the reference set.

lingpy.test.evaluate.EvalLXS.pipd_score

EvalLXS.**pipd_score**()

Compute the pairwise identical positive decisions (PIPD) score.

Returns **score** : float

The PIPD score for reference and test set.

See Also:

```
lingpy.test.evaluate.EvalLXS.pid_score
lingpy.test.evaluate.EvalLXS.pind_score
```

Notes

The PIPD score (see ibid.), is calculated by dividing the number of identical pairwise positive decisions in reference and test set by the total number of pairwise positive decisions in the reference set.

lingpy.test.evaluate.EvalLXS.set_fscore

EvalLXS.**set_fscore**()

Compute the set f-score (FSC).

Returns **score** : float

The FSC for reference and test set.

See Also:

lingpy.test.evaluate.EvalLXS.set_precision
lingpy.test.evaluate.EvalLXS.set_recall

Notes

The set f-score (see Bergsma and Kondrak 2007) is calculated with help of the formula

$$2\frac{pr}{p+r},$$

where p is the set precision and r is the set recall.

lingpy.test.evaluate.EvalLXS.set_precision

EvalLXS.**set_precision**()

Compute the set precision (SPR).

Returns **score** : float

The SPR for reference and test set.

See Also:

lingpy.test.evaluate.EvalLXS.set_recall
lingpy.test.evaluate.EvalLXS.set_fscore

Notes

The set precision (see ibid.) is defined as the number of identical cognate sets in reference and test set divided by the total number of cognate sets in the test set.

lingpy.test.evaluate.EvalLXS.set_recall

EvalLXS.**set_recall**()

Compute the set recall (SRE).

Returns **score** : float

The SRE for reference and test set.

See Also:

`lingpy.test.evaluate.EvalLXS.set_precision`
`lingpy.test.evaluate.EvalLXS.set_fscore`

Notes

The set recall (see `Bergsma and Kondrak 2007`) is defined as the number of identical cognate sets in reference and test set divided by the total number of cognate sets in the reference set.

2.7 Data Plotting (`plot`)

The Module provides different functions for the transformation of text data into a visually appealing format.

The main idea is to render alignments in colored tables where the colors of the cells are chosen with respect to the sound-class of the sound value of each given cell, such as in the following example:

Taxon	Alignment								
English	w	o	l	-	d	e	m	o	r	t
German	w	a	l	-	d	e	m	a	r	-
Russian	v	-	l	a	d	i	m	i	r	-

Here, the coloring of sounds follows the sound-class model of `Dolgopolsky 1964`, the black margin around the cells in the second, the third, and the fourth column indicates a swapped site. The benefit of this way to display alignments is that differences and similarities between the sequences become visible at once, making it easy to check the correctness of a given alignment analysis.

2.7.1 Plotting Alignments

`msa2html`(infile[, shorttitle, filename])	Convert files in `msa`-format into colored `html`-format.

lingpy.output.plot.msa2html

`lingpy.output.plot.`**`msa2html`**(*infile*, *shorttitle=None*, *filename=None*)

Convert files in `msa`-format into colored `html`-format.

Parameters **shorttitle** : str

Define the shorttitle of the `html`-page. If no title is provided, the default title `LexStat` will be used.

filename : str

Define the name of the output file. If no name is defined, the name of the input file will be taken as a default.

See Also:

`lingpy.output.plot.alm2html`

Notes

The coloring of sound segments with respect to the sound class they belong to is based on the definitions given in the `colorModel`. It can easily be changed and adapted.

Examples

Load the libary.

```
>>> from lingpy import *
```

Load an `msq`-file from the test-sets.

```
>>> msa = Multiple(get_file('test.msq'))
```

Align the data progressively and carry out a check for swapped sites.

```
>>> msa.prog_align()
>>> msa.swap_check()
>>> print(msa)
w    o    l    -    d    e    m    o    r    t
w    a    l    -    d    e    m    a    r    -
v    -    l    a    d    i    m    i    r    -
```

Save the data to the file `test.msa`.

```
>>> msa.output('msa')
```

Convert the `msa`-file to `html`.

```
>>> msa2html('test.msa')
```

2.7.2 Plotting Lexicostatistic Wordlists

`alm2html`(infile[, title, shorttitle])	Convert files in `alm`-format into colored `html`-format.

lingpy.output.plot.alm2html

`lingpy.output.plot.`**`alm2html`**(*infile*, *title=None*, *shorttitle=None*)

Convert files in `alm`-format into colored `html`-format.

Parameters **title** : str

Define the title of the output file. If no title is provided, the default title `LexStat - Automatic Cognate Judgments` will be used.

shorttitle : str

Define the shorttitle of the `html`-page. If no title is provided, the default title `LexStat` will be used.

See Also:

`lingpy.output.plot.msa2html`

Notes

The coloring of sound segments with respect to the sound class they belong to is based on the definitions given in the `colorModel`. It can easily be changed and adapted.

2.8 What's Next?

2.8.1 Download

Source Code and Binaries

http://pypi.python.org/pypi/lingpy

Documentation

PDF

http://lingulist.de/lingpy/lingpy_doc.pdf

HTML (zip)

http://lingulist.de/lingpy/lingpy_doc.zip

PYTHON MODULE INDEX

a

lingpy.algorithm.cluster, 11
lingpy.algorithm.misc, 14

d

lingpy.data, 9
lingpy.data.derive, 9

o

lingpy.output.plot, 28

t

lingpy.test.evaluate, 20
lingpy.test.test, 19

BIBLIOGRAPHY

Bergsma, S. and G. Kondrak (2007). "Multilingual Cognate Identification using Integer Linear Programming". In: *RANLP Workshop on Acquisition and Management of Multilingual Lexicons*. Ed. by The International Conference on Recent Advances in Natural Language Processing. Borovets, Bulgaria. URL: \url{http://pers-www.wlv.ac.uk/~in8113/amml07/papers/7.pdf}.

Brown, C. H., E. W. Holman, and S. Wichmann (2011). *Sound correspondences in the world's languages*. URL: http://wwwstaff.eva.mpg.de/~wichmann/wwcPaper23.pdf.

Brown, C. H. et al. (2008). "Automated classification of the world's languages. A description of the method and preliminary results". In: *Sprachtypologie und Universalienforschung* 61.4, 285–308.

Dolgopolsky, A. B. (1964). "Gipoteza drevnejšego rodstva jazykovych semej Severnoj Evrazii s verojatnostej točky zrenija [A probabilistic hypothesis concering the oldest relationships among the language families of Northern Eurasia]". In: *Voprosy Jazykoznanija* 2, 53–63; English translation: Dolgopolsky, A. B. (1986). "A probabilistic hypothesis concerning the oldest relationships among the language families of northern Eurasia". In: *Typology, relationship and time. A collection of papers on language change and relationship by Soviet linguists*. Ed. and trans. from the Russian by V. V. Shevoroshkin. Ann Arbor: Karoma Publisher, 27–50.

List, J.-M. (2012). "Multiple Sequence Alignment in Historical Linguistics. A Sound Class Based Approach". In: *Proceedings of ConSOLE XIX (2011)*, 241–260. PDF: http://media.leidenuniv.nl/legacy/console19-proceedings-list.pdf.

– (forthcoming[a]). "LexStat. Automatic Detection of Cognates in Multilingual Wordlists". In:

– (forthcoming[b]). "SCA: Phonetic alignment based on sound classes". In: *New directions in logic, language, and computation*. Ed. by M. Slavkovik and D. Lassiter. Berlin and Heidelberg: Springer.

Raghava, G. P. S. and G. J. Barton (2006). "Quantification of the variation in percentage identity for protein sequence alignments". In: *BMC Bioinformatics* 7.415.

Rosenberg, M. S. and T. H. Ogden (2009). "Simulation approaches to evaluating alignment error and methods for comparing alternate alignments". In: *Sequence alignment. Methods, models, concepts, and strategies*. Ed. by M. S. Rosenberg. Berkeley, Los Angeles, and London: University of California Press, 179–207.

Saitou, N. and M. Nei (1987). "The neighbor-joining method: A new method for reconstructing phylogenetic trees". In: *Molecular Biology and Evolution* 4.4, 406–425.

Sokal, R. R. and C. D. Michener (1958). "A statistical method for evaluating systematic relationships". In: *University of Kansas Scientific Bulletin* 28, 1409–1438.

Swadesh, M. (1955). "Towards greater accuracy in lexicostatistic dating". In: *International Journal of American Linguistics* 21.2, 121–137. JSTOR: 1263939.

Thompson, J. D., F. Plewniak, and O. Poch (1999). "A comprehensive comparison of multiple sequence alignment programs". In: *Nucleic Acids Research* 27.13, 2682–2690. PMID: 10373585.