## CONTENTS:

1 Quickstart
   1.1 Installation ................................................................. 3  
   1.2 Naive Discriminative Learning ....................................... 3  
   1.3 Correct Data Format ..................................................... 4  
   1.4 Learn the associations .................................................. 5  
   1.5 Save and load a weight matrix ........................................ 5  

2 Installation ........................................................................ 7  
   2.1 Supported systems and versions ................................. 7  
   2.2 Linux ...................................................................... 7  
   2.3 MacOS ................................................................. 8  
   2.4 Windows 10 .............................................................. 8  

3 Naive Discriminative Learning ..................................... 9  
   3.1 Terminology ............................................................... 9  
   3.2 Rescorla Wagner learning rule ....................................... 9  
   3.3 Other Learning Algorithms ........................................... 11  
   3.4 References ............................................................... 12  

4 More Examples ........................................................... 13  
   4.1 Lexical example ........................................................... 13  
   4.2 A minimal workflow example .......................................... 16  

5 Package overview .......................................................... 21  
   5.1 Pyndl - Naive Discriminative Learning in Python .............. 21  
   5.2 Submodules ............................................................... 21  

6 Development ......................................................................... 31  
   6.1 Getting Involved ............................................................ 31  
   6.2 Workflow .................................................................. 31  
   6.3 Running tests .............................................................. 32  
   6.4 Building documentation ................................................. 32  
   6.5 Continuous Integration .................................................. 33  
   6.6 Licensing ................................................................. 33  
   6.7 Release Process ........................................................... 33  
   6.8 Versioning ................................................................. 33  

7 Miscellaneous ..................................................................... 35  
   7.1 Running pyndl within R code .......................................... 35  
   7.2 Local testing with conda ................................................. 36  
   7.3 Memory profiling .......................................................... 36
pyndl is an implementation of Naive Discriminative Learning in Python. It was created to analyse huge amounts of text file corpora. Especially, it allows to efficiently apply the Rescorla-Wagner learning rule to these corpora.
1.1 Installation

First, you need to install `pyndl`. The easiest way to do this is using `pip`:

```
pip install --user pyndl
```

**Warning:** If you are using any other operating system than Linux this process can be more difficult. Check out *Installation* for more detailed installation instruction. However, currently we can only ensure the expected behaviour on Linux system. Be aware that on other operating system some functionality may not work.

1.2 Naive Discriminative Learning

Naive Discriminative Learning, henceforth NDL, is an incremental learning algorithm based on the learning rule of Rescorla and Wagner\(^1\), which describes the learning of direct associations between cues and outcomes. The learning is thereby structured in events where each event consists of a set of cues which give hints to outcomes. Outcomes can be seen as the result of an event, where each outcome can be either present or absent. NDL is naive in the sense that cue-outcome associations are estimated separately for each outcome.

The Rescorla-Wagner learning rule describes how the association strength $\Delta V_i^t$ at time $t$ changes over time. Time is here described in form of learning events. For each event the association strength is updated as

$$V_i^{t+1} = V_i^t + \Delta V_i^t$$

Thereby, the change in association strength $\Delta V_i^t$ is defined as

$$\Delta V_i^t = \begin{cases} 
0 & \text{if ABSENT}(C_i, t) \\
\alpha_i \beta_1 (\lambda - \sum_{\text{PRESENT}(C_j, t)} V_j) & \text{if PRESENT}(C_j, t) \& \text{PRESENT}(O, t) \\
\alpha_i \beta_2 (0 - \sum_{\text{PRESENT}(C_j, t)} V_j) & \text{if PRESENT}(C_j, t) \& \text{ABSENT}(O, t) 
\end{cases}$$

with

- $\alpha_i$ being the salience of the cue $i$
- $\beta_1$ being the salience of the situation in which the outcome occurs
- $\beta_2$ being the salience of the situation in which the outcome does not occur

---

\( \lambda \) being the the maximum level of associative strength possible

**Note:** Usually, the parameters are set to \( \alpha_i = \alpha_j \forall i, j, \beta_1 = \beta_2 \) and \( \lambda = 1 \)

### 1.3 Correct Data Format

#### 1.3.1 From Wide to Long Format

Often data which should be analysed is not in the right format to be processed with *pyndl*. To illustrate how to get the data in the right format we use data from Baayen, Milin, Đurđević, Hendrix & Marelli\(^2\) as an example:

<table>
<thead>
<tr>
<th>Word</th>
<th>Frequency</th>
<th>Lexical Meaning</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>hand</td>
<td>10</td>
<td>HAND</td>
<td></td>
</tr>
<tr>
<td>hands</td>
<td>20</td>
<td>PLURAL</td>
<td></td>
</tr>
<tr>
<td>land</td>
<td>8</td>
<td>LAND</td>
<td></td>
</tr>
<tr>
<td>lands</td>
<td>3</td>
<td>PLURAL</td>
<td></td>
</tr>
<tr>
<td>and</td>
<td>35</td>
<td>AND</td>
<td></td>
</tr>
<tr>
<td>sad</td>
<td>18</td>
<td>SAD</td>
<td></td>
</tr>
<tr>
<td>as</td>
<td>35</td>
<td>AS</td>
<td></td>
</tr>
<tr>
<td>lad</td>
<td>102</td>
<td>LAD</td>
<td></td>
</tr>
<tr>
<td>lads</td>
<td>54</td>
<td>PLURAL</td>
<td></td>
</tr>
<tr>
<td>lass</td>
<td>134</td>
<td>LASS</td>
<td></td>
</tr>
</tbody>
</table>

Table 1 shows some words, their frequencies of occurrence and their meanings as an artificial lexicon in the wide format. In the following, the letters (unigrams and bigrams) of the words constitute the cues, whereas the meanings represent the outcomes.

To analyse any data using *pyndl* requires them to be in the long format as an utf-8 encoded tab delimited gzipped text file with a header in the first line and two columns:

1. the first column contains an underscore delimited list of all cues
2. the second column contains an underscore delimited list of all outcomes
3. each line therefore represents an event with a pair of a cue and an outcome (occurring one time)
4. the events (lines) are ordered chronologically

As the data in table 1 are artificial we can generate such a file for this example by expanding table 1 randomly regarding the frequency of occurrence of each event. The resulting event file `lexample.tab.gz` consists of 420 lines (419 = sum of frequencies + 1 header) and looks like the following (nevertheless you are encouraged to take a closer look at this file using any text editor of your choice):

<table>
<thead>
<tr>
<th>Cues</th>
<th>Outcomes</th>
</tr>
</thead>
<tbody>
<tr>
<td>#h_ha_an_nd_ds_s#</td>
<td>hand_plural</td>
</tr>
<tr>
<td>#l_la_ad_d#</td>
<td>lad</td>
</tr>
<tr>
<td>#l_la_as_ss_s#</td>
<td>lass</td>
</tr>
</tbody>
</table>

---

1.3.2 From Corpus to Long Format

Often the corpus which should be analysed is only a raw utf-8 encoded text file that contains huge amounts of text. From here on we will refer to such a file as a corpus file. In the corpus files several documents can be stored with a `---end.of.document---` or `---END.OF.DOCUMENT---` string marking where an old document finished and a new document starts.

The `pyndl.preprocess` module (besides other things) provides the functionality to directly generate an event file based on a raw corpus file and filter it:

```python
>>> from pyndl import preprocess
>>> preprocess.create_event_file(corpus_file='doc/data/lcorpus.txt',
...                               event_file='doc/data/levent.tab.gz',
...                               context_structure='document',
...                               event_structure='consecutive_words',
...                               event_options=(1, ),
...                               cue_structure='bigrams_to_word')
```

Here we use the example corpus `lcorpus.txt` to produce an event file `levent.tab.gz` which (uncompressed) looks like this:

<table>
<thead>
<tr>
<th>Cues</th>
<th>Outcomes</th>
</tr>
</thead>
<tbody>
<tr>
<td>an_#h_ha_d#_nd</td>
<td>hand</td>
</tr>
<tr>
<td>ot_fo_oo_#f_t#</td>
<td>foot</td>
</tr>
<tr>
<td>ds_s#<em>an</em>#h_ha_nd</td>
<td>hands</td>
</tr>
</tbody>
</table>

Note: `pyndl.corpus` allows you to generate such a corpus file from a bunch of gunzipped xml subtitle files filled with words.

1.4 Learn the associations

The strength of the associations for the data can now easily be computed using the `pyndl.ndl.ndl` function from the `pyndl.ndl` module:

```python
>>> from pyndl import ndl
>>> weights = ndl.ndl(events='doc/data/levent.tab.gz',
...                    alpha=0.1, betas=(0.1, 0.1), method="threading")
```

1.5 Save and load a weight matrix

To save time in the future, we recommend saving the weights. For compatibility reasons we recommend saving the weight matrix in the netCDF format:

```python
>>> weights.to_netcdf('doc/data/weights.nc')
```

Now, the saved weights can later be reused or be analysed in Python or R. In Python the weights can simply be loaded with the `xarray` module:

---


1.4. Learn the associations
In R you need the `ncdf4` package to load a in netCDF format saved matrix:

```r
#install.packages("ncdf4") # uncomment to install
library(ncdf4)
weights_nc <- nc_open(filename = "doc/data/weights.nc")
weights_read <- t(as.matrix(ncvar_get(nc = weights_nc, varid = "__xarray_dataarray_variable__")))
rownames(weights_read) <- ncvar_get(nc = weights_nc, varid = "outcomes")
colnames(weights_read) <- ncvar_get(nc = weights_nc, varid = "cues")
nc_close(nc = weights_nc)
rm(weights_nc)
```
2.1 Supported systems and versions

*pyndl* currently is only tested and mainly used on 64-bit Linux systems. However, it is possible to install it on other operating systems, but be aware that some functionality might not work or will not work as intended. Therefore be extra careful and run the test suite after installing it on a non Linux system.

**Note:** We recommend to install Minicoda before installing *pyndl* or to create a virtualenv within your personal folder.

2.2 Linux

If you want to install *pyndl* on Linux the easiest way is to install it from pypi with:

```
pip install --user pyndl
```

2.3 MacOS

If you want to install *pyndl* on MacOS you can also install it from pypi. However, you need xcode and gcc/g++ 6.3 installed. As gcc/g++ might be outdated as xcode provides only 4.X, it might be necessary to update gcc first, before installing *pyndl*:

1. Download and install x-code or for safe-guarding redo the xcode install in the Terminal if you have already installed it:

   ```
xcode-select --install
   ```

2. download gcc from Mac OSX High Performance Computing then run these commands in Terminal:

   ```
gunzip gcc-6.X-bin.tar.gz
sudo tar -xvf gcc-6.X-bin.tar -C /
```

3. finally, install pyndl:
pip install --user pyndl

**Warning:** This procedure is experimental and might not work. As long as we do not actively support MacOS be aware that these installation instructions can fail or the installed package does not always work as intended!

## 2.4 Windows 10

**Note:** You might need to enable the *bash* within Windows 10 first to be able to follow the following instructions.

After installing Anaconda or Miniconda, first install the dependencies with the `conda` command in the bash or the Ana/Miniconda terminal:

```
conda update --all
conda install numpy cython pandas xarray netCDF4 numpydoc pip
```

After the installation of the dependencies finished successfully you should be able to install `pyndl` with `pip`:

```
pip install --user pyndl
```

**Warning:** This procedure is experimental and might not work. As long as we do not actively support Windows 10 be aware that these installation instructions can fail or the installed package does not always work as intended!
3.1 Terminology

Before explaining Naive Discriminative Learning (NDL) in detail, we want to give you a brief overview over important notions:

**cue**: A cue is something that gives a hint on something else. The something else is called outcome. Examples for cues in a text corpus are trigrams or preceding words for the word or meaning of the word.

**outcome**: The outcome is the result of an event. Examples are words, the meaning of the word, or lexomes.

**event**: An event connects cues with outcomes. In any event one or more unordered cues are present and one or more outcomes are present.

**weights**: The weights represent the learned experience / association between all cues and outcomes of interest. Usually, some meta data is stored alongside the learned weights.

3.2 Rescorla Wagner learning rule

In order to update the association strengths (weights) between cues and outcomes we do for each event the following:

We calculate the activation (prediction) $a_j$ for each outcome $o_j$ by using all present cues $C_{\text{PRESENT}}$:

$$a_j = \sum_{i \text{ for } c_i \in C_{\text{PRESENT}}} w_{ij}$$

After that, we calculate the update $\Delta w_{ij}$ for every cue-outcome-combination:

$$\Delta w_{ij} \begin{cases} 0 & \text{if cue } c_i \text{ is absent} \\ \alpha_i \beta_1 \cdot (\lambda - a_j) & \text{if outcome } o_j \text{ and cue } c_i \text{ is present.} \\ \alpha_i \beta_2 \cdot (0 - a_j) & \text{if outcome } o_j \text{ is absent and cue } c_i \text{ is present.} \end{cases}$$

In the end, we update all weights according to $w_{ij} = w_{ij} + \Delta w_{ij}$.

**Note**: If we set all the $\alpha$’s and $\beta$’s to a fixed value we can replace them in the equation with a general learning parameter $\eta = \alpha \cdot \beta$. 
3.2.1 In matrix notation

We can rewrite the Rescorla-Wagner learning rule into matrix notation with a binary cue (input) vector \( \vec{c} \), which is one for each cue present in the event and zero for all other cues. Respectively, we define a binary outcome (output) vector \( \vec{o} \), which is one for each outcome present in the event and zero if the outcome is not present. In order to stick close to the definition above we can define the activation vector as \( \vec{a} = W^T \vec{c} \). Here \( W^T \) denotes the transposed matrix of the weight matrix \( W \).

For simplicity let us assume we have a fixed learning rate \( \eta = \alpha \beta \). We will relax this simplification in the end. We can rewrite the above rule as:

\[
\Delta = \eta \vec{c} \cdot (\lambda \vec{o} - \lambda \vec{a} - W^T \vec{c})^T
\]

Let us first check the dimensionality of the matrices:

\( \Delta \) is the update of the weight matrix \( W \) and therefore needs to have the same dimensions \( n \times m \) where \( n \) denotes the number of cues (inputs) and \( m \) denotes the number of outcomes (outputs).

The cue vector \( \vec{c} \) can be seen as a matrix with dimensions \( n \times 1 \) and the outcome vector can be seen as a matrix with dimensions \( m \times 1 \). Let us tabulate the dimensions:

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>( \lambda \vec{o} )</td>
<td>( m \times 1 )</td>
<td></td>
</tr>
<tr>
<td>( W^T )</td>
<td>( m \times n )</td>
<td></td>
</tr>
<tr>
<td>( \vec{c} )</td>
<td>( n \times 1 )</td>
<td></td>
</tr>
<tr>
<td>( W^T \cdot \vec{c} )</td>
<td>( m \times 1 = (m \times n) \cdot (n \times 1) )</td>
<td></td>
</tr>
<tr>
<td>( \lambda \vec{o} - W^T \cdot \vec{c} )</td>
<td>( m \times 1 = (m \times 1) - (m \times 1) )</td>
<td></td>
</tr>
<tr>
<td>( (\lambda \vec{o} - W^T \cdot \vec{c})^T )</td>
<td>( 1 \times m = (m \times 1)^T )</td>
<td></td>
</tr>
<tr>
<td>( \eta \vec{c} \cdot (\lambda \vec{o} - W^T \cdot \vec{c}) )</td>
<td>( n \times m = (n \times 1) \cdot (1 \times n) )</td>
<td></td>
</tr>
</tbody>
</table>

We therefore end with the right set of dimensions. We now can try to simplify / rewrite the equation.

\[
\Delta = \eta \vec{c} \cdot ((\lambda \vec{o})^T - (W^T \cdot \vec{c})^T)
\]

\[
= \eta \vec{c} \cdot (\lambda \vec{o}^T - \vec{c}^T \cdot W)
\]

\[
= \eta \lambda \vec{c} \cdot \vec{o}^T - \eta \vec{c} \cdot \vec{c}^T \cdot W
\]

If we now look at the full update:

\[
W_{t+1} = W_t + \Delta_t
\]

\[
= W + \Delta
\]

\[
= W + \eta \lambda \vec{c} \cdot \vec{o}^T - \eta \vec{c} \cdot \vec{c}^T \cdot W
\]

\[
= \eta \lambda \vec{c} \cdot \vec{o}^T + W - \eta \vec{c} \cdot \vec{c}^T \cdot W
\]

\[
= \eta \lambda \vec{c} \cdot \vec{o}^T + (1 - \eta \vec{c} \cdot \vec{c}^T) \cdot W
\]

We therefore see that the Rescorla-Wagner update is an affine (linear) transformation\(^1\) in the weights \( W \) with an intercept of \( \eta \lambda \vec{c} \cdot \vec{o}^T \) and a slope of \( 1 - \eta \vec{c} \cdot \vec{c}^T \).

In index notation we can write:

\[
W_{t+1}^{i} = W_t^{i} + \eta \vec{c}_i \cdot (\lambda \vec{o}^T - \vec{c}^T \cdot W)
\]

\[
W_{ij}^{t+1} = W_{ij}^{t} + \eta c_i (\lambda o_j - \sum_k c_k W_{kj})
\]

\(^1\) [https://en.wikipedia.org/wiki/Affine_transformation](https://en.wikipedia.org/wiki/Affine_transformation)
Note: Properties of the transpose\(^4\) with \(A\) and \(B\) matrices and \(\alpha\) skalar:

\[
\begin{align*}
(A^T)^T &= A \\
(A + B)^T &= A^T + B^T \\
(\alpha A)^T &= \alpha A^T \\
(A \cdot B)^T &= B^T \cdot A^T
\end{align*}
\]

### 3.3 Other Learning Algorithms

#### 3.3.1 Delta rule

The delta rule\(^2\) is a gradient descent learning rule for updating the weights of the inputs to artificial neurons in a single-layer neural network. It is a special case of the more general backpropagation algorithm\(^3\).

The delta rule can be expressed as:

\[
\Delta_{ij} = \alpha(t_j - y_j)\partial_{h_j}g(h_j)x_i
\]

In the terminology above we can identify the actual output with \(y_j = g(h_j) = g\left(\sum_i w_{ij}c_i\right)\), the cues with \(x_i = c_i\), under the assumption that \(o_j\) is binary (i.e. either zero or one) we can write \(t_j = \lambda o_j\), the learning rate \(\alpha = \eta = \alpha \beta\).

Substituting this equalities results in:

\[
\Delta_{ij} = \eta(\lambda o_j - g\left(\sum_i w_{ij}c_i\right))\partial_{h_j}g(h_j)c_i
\]

In order to end with the Rescorla-Wagner learning rule we need to set the neuron’s activation function \(g(h_j)\) to the identity function, i.e. \(g(h_j) = 1 \cdot h_j + 0 = h_j \sum_i w_{ij}c_i\). The derivative in respect to \(h_j\) is \(\partial_{h_j}g(h_j) = 1\) for any input \(h_j\).

We now have:

\[
\begin{align*}
\Delta_{ij} &= \eta(\lambda o_j - \sum_i w_{ij}c_i) \cdot 1 \cdot c_i \\
&= \eta(\lambda o_j - \sum_i w_{ij}c_i)c_i \\
&= \eta c_i(\lambda o_j - \sum_i w_{ij}c_i)
\end{align*}
\]

Assuming the cue vector is binary the vector \(c_i\) effectively filters those updates of the present cues and sets all updates of the cues that are not present to zero. Additionally, we can rewrite the equation above into vector notation (without indices):

\[
\Delta_{ij} = \eta c_i(\lambda o_j - \sum_i w_{ij}c_i)
\]

\(^4\) https://en.wikipedia.org/wiki/Transpose

\(^2\) https://en.wikipedia.org/wiki/Delta_rule

\(^3\) https://en.wikipedia.org/wiki/Backpropagation
\[ \Delta = \eta \mathbf{c} \cdot (\lambda \mathbf{o}^T - W^T \cdot \mathbf{c})^T \]

This is exactly the form of the Rescorla-Wagner rule rewritten in matrix notation.

**Conclusion**

In conclusion, the Rescorla-Wagner learning rule, which only allows for one \( \alpha \) and one \( \beta \) and therefore one learning rate \( \eta = \alpha \beta \) is exactly the same as a single layer backpropagation gradient decent method (the delta rule) where the neuron’s activation function \( g(h_j) \) is set to the identity \( g(h_j) = h_j \) and the inputs \( x_i = c_i \) and target outputs \( t_j = \lambda o_j \) to be binary.

### 3.4 References
4.1 Lexical example

The lexical example illustrates the Rescorla-Wagner equations\(^1\). This example is taken from Baayen, Milin, Đurđević, Hendrix and Marelli\(^2\).

4.1.1 Premises

1. Cues are associated with outcomes and both can be present or absent
2. Cues are segment (letter) unigrams, bigrams, . . .
3. Outcomes are meanings (word meanings, inflectional meanings, affixal meanings), . . .
4. \(\text{PRESENT}(X, t)\) denotes the presence of cue or outcome \(X\) at time \(t\)
5. \(\text{ABSENT}(X, t)\) denotes the absence of cue or outcome \(X\) at time \(t\)
6. The association strength \(V_{i+1}^{t}\) of cue \(C_i\) with outcome \(O\) at time \(t + 1\) is defined as \(V_{i+1}^{t} = V_i^t + \Delta V_i^t\)
7. The change in association strength \(\Delta V_i^t\) is defined as in (4.1) with
   - \(\alpha_i\) being the salience of the cue \(i\)
   - \(\beta_1\) being the salience of the situation in which the outcome occurs
   - \(\beta_2\) being the salience of the situation in which the outcome does not occur
   - \(\lambda\) being the the maximum level of associative strength possible
8. Default settings for the parameters are: \(\alpha_i = \alpha_j \forall i, j, \beta_1 = \beta_2\) and \(\lambda = 1\)

\[
\Delta V_i^t = \begin{cases} 
0 & \text{if } \text{ABSENT}(C_i, t) \\
\alpha_i \beta_1 (\lambda - \sum_{\text{PRESENT}(C_j,t)} V_j) & \text{if } \text{PRESENT}(C_j, t) \& \text{PRESENT}(O, t) \\
\alpha_i \beta_2 (0 - \sum_{\text{PRESENT}(C_j,t)} V_j) & \text{if } \text{PRESENT}(C_j, t) \& \text{ABSENT}(O, t)
\end{cases}
\] (4.1)

See \textit{Delta rule} for alternative formulations of the Rescorla Wagner learning rule.

---


4.1.2 Data

Table 1

<table>
<thead>
<tr>
<th>Word</th>
<th>Frequency</th>
<th>Lexical Meaning</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>hand</td>
<td>10</td>
<td>HAND</td>
<td></td>
</tr>
<tr>
<td>hands</td>
<td>20</td>
<td>HAND PLURAL</td>
<td></td>
</tr>
<tr>
<td>land</td>
<td>8</td>
<td>LAND</td>
<td></td>
</tr>
<tr>
<td>lands</td>
<td>3</td>
<td>LAND PLURAL</td>
<td></td>
</tr>
<tr>
<td>and</td>
<td>35</td>
<td>AND</td>
<td></td>
</tr>
<tr>
<td>sad</td>
<td>18</td>
<td>SAD</td>
<td></td>
</tr>
<tr>
<td>as</td>
<td>35</td>
<td>AS</td>
<td></td>
</tr>
<tr>
<td>lad</td>
<td>102</td>
<td>LAD</td>
<td></td>
</tr>
<tr>
<td>lads</td>
<td>54</td>
<td>LADS PLURAL</td>
<td></td>
</tr>
<tr>
<td>lass</td>
<td>134</td>
<td>LASS</td>
<td></td>
</tr>
</tbody>
</table>

Table 1 shows some words, their frequencies of occurrence and their meanings as an artificial lexicon in the wide format. In the following, the letters (unigrams and bigrams) of the words constitute the cues, the meanings represent the outcomes.

Analyzing any data using pyndl requires them to be in the long format as an utf-8 encoded tab delimited gzipped text file with a header in the first line and two columns:

1. the first column contains an underscore delimited list of all cues
2. the second column contains an underscore delimited list of all outcomes
3. each line therefore represents an event with a pair of a cue and an outcome (occurring one time)
4. the events (lines) are ordered chronologically

As the data in table 1 are artificial we can generate such a file for this example by expanding table 1 randomly regarding the frequency of occurrence of each event. The resulting event file lexample.tab.gz consists of 420 lines (419 = sum of frequencies + 1 header) and looks like the following (nevertheless you are encouraged to take a closer look at this file using any text editor of your choice):

<table>
<thead>
<tr>
<th>Cues</th>
<th>Outcomes</th>
</tr>
</thead>
<tbody>
<tr>
<td>#h_ha_an_nd_ds_s#</td>
<td>hand_plural</td>
</tr>
<tr>
<td>#l_la_ad_d#</td>
<td>lad</td>
</tr>
<tr>
<td>#l_la_as_ss_s#</td>
<td>lass</td>
</tr>
</tbody>
</table>

4.1.3 pyndl.ndl module

We can now compute the strength of associations (weights or weight matrix) after the presentation of the 419 tokens of the 10 words using pyndl.ndl. pyndl.ndl provides the two functions pyndl.ndl.ndl and pyndl.ndl.dict_ndl to calculate the weights for all outcomes over all events. pyndl.ndl.ndl itself provides to methods regarding estimation, openmp and threading. We have to specify the path of our event file lexample.tab.gz and for this example set \( \alpha = 0.1, \beta_1 = 0.1, \beta_2 = 0.1 \) with leaving \( \lambda = 1.0 \) at its default value. You can use pyndl directly in a Python3 Shell or you can write an executable script, this is up to you. For educational purposes we use a Python3 Shell in this example.

pyndl.ndl.ndl

pyndl.ndl.ndl is a parallel Python implementation using numpy, multithreading and a binary format which is created automatically. It allows you to choose between the two methods openmp and threading, with the former
one using openMP and therefore being expected to be much faster when analyzing larger data. Besides, you can set three technical arguments which we will not change here:

1. `number_of_threads` (int) giving the number of threads in which the job should be executed (default=2)
2. `sequence` (int) giving the length of sublists generated from all outcomes (default=10)
3. `remove_duplicates` (logical) to make cues and outcomes unique (default=None; which will raise an error if the same cue is present multiple times in the same event)

Let's start:

```python
>>> from pyndl import ndl
>>> weights = ndl.ndl(events='doc/data/lexample.tab.gz', alpha=0.1, ... betas=(0.1, 0.1), method='openmp')
>>> weights
<xarray.DataArray (outcomes: 8, cues: 15)>
...
```


```python
>>> weights[1, 5]
<xarray.DataArray ()>
...
>>> weights.loc[{'outcomes': 'plural', 'cues': 's#'}]
<xarray.DataArray ()>
array(0.07698883)
Coordinates:
  outcomes <U6 'plural'
  cues <U2 's#'
...
>>> weights.loc[{'outcomes': 'plural', 'cues': 's#'}]
<xarray.DataArray ()>
array(0.07698883)
Coordinates:
  outcomes <U6 'plural'
  cues <U2 's#'
...
```

return the weight of the cue ‘s#’ (the unigram ‘s’ being the word-final) for the outcome ‘plural’ (remember counting in Python does start at 0) as ca. 0.077 and hence indicate ‘s#’ being a marker for plurality.

`pyndl.ndl.ndl` also allows you to continue learning from a previous weight matrix by specifying the weight argument:

```python
>>> weights2 = ndl.ndl(events='doc/data/lexample.tab.gz', alpha=0.1, ... betas=(0.1, 0.1), method='openmp', weights=weights)
>>> weights2
<xarray.DataArray (outcomes: 8, cues: 15)>
array([...]
...
...
)
Coordinates:
  * outcomes (outcomes) <U6 ...
  * cues (cues) <U2 ...
Attributes:
...
```

4.1. Lexical example
As you may have noticed already, `pyndl.ndl.ndl` provides you with meta data organized in a `dict` which was collected during your calculations. Each entry of each list of this meta data therefore references one specific moment of your calculations:

```python
>>> weights2.attrs
OrderedDict(...)
```

### `pyndl.ndl.dict_ndl`

`pyndl.ndl.dict_ndl` is a pure Python implementation, however, it differs from `pyndl.ndl.ndl` regarding the following:

1. there are only two technical arguments: `remove_duplicates` (logical) and `make_data_array` (logical)
2. by default, no longer an `xarray.DataArray` is returned but a `dict` of `dicts`
3. however, you are still able to get an `xarray.DataArray` by setting `make_data_array=True`
4. the case \( \alpha_i \neq \alpha_j \) can be handled by specifying a `dict` consisting of the cues as keys and corresponding \( \alpha \)'s

Therefore

```python
>>> weights = ndl.dict_ndl(events='doc/data/lexample.tab.gz',
...                       alphas=0.1, betas=(0.1, 0.1))
>>> weights['plural']['s#'] # doctests: +ELLIPSIS
0.076988227...
```

yields approximately the same results as before, however, you now can specify different \( \alpha \)'s per cue and as in `pyndl.ndl.ndl` continue learning or do both:

```python
>>> alphas_cues = dict(zip(['#h', 'ha', 'an', 'nd', 'ds', 's#', '#l', 'la', 'as', 'ss →', 'ad', 'd#', '#a', '#s', 'sa'], [0.1, 0.2, 0.3, 0.4, 0.1, 0.2, 0.3, 0.1, 0.2, 0.1, 0.2, 0.1, 0.2, 0.1]))
>>> weights = ndl.dict_ndl(events='doc/data/lexample.tab.gz',
...                       alphas=alphas_cues, betas=(0.1, 0.1))
>>> weights2 = ndl.dict_ndl(events='doc/data/lexample.tab.gz',
...                          alphas=alphas_cues, betas=(0.1, 0.1),
...                          weights=weights)
```

If you prefer to get a `xarray.DataArray` returned you can set the flag `make_data_array=True`:

```python
>>> weights = ndl.dict_ndl(events='doc/data/lexample.tab.gz',
...                         alphas=alphas_cues, betas=(0.1, 0.1),
...                         make_data_array=True)
>>> weights
<xarray.DataArray (outcomes: 8, cues: 15)>
...```

### 4.2 A minimal workflow example

As you should have a basic understanding of `pyndl.ndl` by now, the following example will show you how to:

1. generate an event file based on a raw corpus file
2. count cues and outcomes
3. filter the events
4. learn the weights as already shown in the lexical learning example
5. save and load a weight matrix (netCDF format)
6. load a weight matrix (netCDF format) into R for further analyses

4.2.1 Generate an event file based on a raw corpus file

Suppose you have a raw utf-8 encoded corpus file (by the way, `pyndl.corpus` allows you to generate such a corpus file from a bunch of gunzipped xml subtitle files filled with words, which we will not cover here). For example take a look at `lcorpus.txt`.

To analyse the data, you need to convert the file into an event file similar to `lexample.tab.gz` in our lexical learning example, as currently there is only one word per line and neither is there the column for cues nor for outcomes:

```
hand
foot
hands
```

The `pyndl.preprocess` module (besides other things) allows you to generate an event file based on a raw corpus file and filter it:

```python
>>> import pyndl
>>> from pyndl import preprocess
>>> preprocess.create_event_file(corpus_file='doc/data/lcorpus.txt',
...                               context_structure='document',
...                               event_structure='consecutive_words',
...                               event_options=(1, ),
...                               cue_structure='bigrams_to_word')
```

The function `pyndl.preprocess.create_event_file` has several arguments which you might have to change to suit them your data, so you are strongly recommended to read its documentation. We set `context_structure='document'` as in this case the context is the whole document, `event_structure='consecutive_words'` as these are our events, `event_options=(1, )` as we define an event to be one word and `cue_structure='bigrams_to_word'` to set cues being bigrams. There are also several technical arguments you can specify, which we will not change here. Our generated event file `levent.tab.gz` now looks (uncompressed) like this:

```
Cues               Outcomes
an_#h_ha_d#_nd    hand
ot_fo_oo_#f_t#    foot
ds_s#_an_#h_ha_nd hands
```

4.2.2 Count cues and outcomes

We can now count the cues and outcomes in our event file using the `pyndl.count` module and also generate id maps for cues and outcomes:

```python
>>> from pyndl import count
>>> freq, cue_freq_map, outcome_freq_map = count.cues_outcomes(event_file_name='doc/data/levent.tab.gz')
>>> freq
```

(continues on next page)
12

```python
>>> cue_freq_map
Counter({...})
>>> outcome_freq_map
Counter({...})
>>> cues = list(cue_freq_map.keys())
>>> cues.sort()
>>> cue_id_map = {cue: ii for ii, cue in enumerate(cues)}
>>> cue_id_map
{...}
>>> outcomes = list(outcome_freq_map.keys())
>>> outcomes.sort()
>>> outcome_id_map = {outcome: nn for nn, outcome in enumerate(outcomes)}
>>> outcome_id_map
{...}
```

### 4.2.3 Filter the events

As we do not want to include the outcomes ‘foot’ and ‘feet’ in this example as well as their cues ‘#f’, ‘fo’ ‘oo’, ‘ot’, ‘t#’, ‘fe’, ‘ee’ ‘et’, we use the `pyndl.preprocess` module again, filtering our event file and update the id maps for cues and outcomes:

```python
>>> preprocess.filter_event_file(input_event_file='doc/data/levent.tab.gz',
...                               output_event_file='doc/data/levent.tab.gz.filtered',
...                               remove_cues=('#f', 'fo', 'oo', 'ot', 't#', 'fe', 'ee', 'et'),
...                               remove_outcomes=('foot', 'feet'))
>>> freq, cue_freq_map, outcome_freq_map = count.cues_outcomes(event_file_name='doc/data/levent.tab.gz.filtered')
>>> cues = list(cue_freq_map.keys())
>>> cues.sort()
>>> cue_id_map = {cue: ii for ii, cue in enumerate(cues)}
>>> cue_id_map
{...}
>>> outcomes = list(outcome_freq_map.keys())
>>> outcomes.sort()
>>> outcome_id_map = {outcome: nn for nn, outcome in enumerate(outcomes)}
>>> outcome_id_map
{...}
```

Alternatively, using `pyndl.preprocess.filter_event_file` you can also specify which cues and outcomes to keep (`keep_cues` and `keep_outcomes`) or remap cues and outcomes (`cue_map` and `outcomes_map`). Besides, there are also some technical arguments you can specify, which will not discuss here.

Last but not least `pyndl.preprocess` does provide some other very useful functions regarding preprocessing of which we did not make any use here, so make sure to go through its documentation.

### 4.2.4 Learn the weights

Computing the strength of associations for the data is now easy, using for example `pyndl.ndl.ndl` from the `pyndl.ndl` module like in the lexical learning example:
>>> from pyndl import ndl
>>> weights = ndl.ndl(events='doc/data/levent.tab.gz.filtered',
...                   alpha=0.1, betas=(0.1, 0.1), method="threading")

4.2.5 Save and load a weight matrix

is straight forward using the netCDF format³

```python
>>> import xarray
>>> weights.to_netcdf('doc/data/weights.nc')
>>> with xarray.open_dataarray('doc/data/weights.nc') as weights_read:
...     weights_read
```

In order to keep everything clean we might want to remove all the files we created in this tutorial:

```python
>>> import os
>>> os.remove('doc/data/levent.tab.gz')
>>> os.remove('doc/data/levent.tab.gz.filtered')
```

4.2.6 Load a weight matrix to R⁴

We can load a in netCDF format saved matrix into R:

```r
#install.packages("ncdf4") # uncomment to install
library(ncdf4)
weights_nc <- nc_open(filename = "doc/data/weights.nc")
weights_read <- t(as.matrix(ncvar_get(nc = weights_nc, varid = "__xarray_dataarray__variable__")))
rownames(weights_read) <- ncvar_get(nc = weights_nc, varid = "outcomes")
colnames(weights_read) <- ncvar_get(nc = weights_nc, varid = "cues")
nc_close(nc = weights_nc)
rm(weights_nc)
```


4.2. A minimal workflow example
5.1 Pyndl - Naive Discriminative Learning in Python

`pyndl` is an implementation of Naive Discriminative Learning in Python. It was created to analyse huge amounts of text file corpora. Especially, it allows to efficiently apply the Rescorla-Wagner learning rule to these corpora.

5.2 Submodules

5.2.1 pyndl.activation

`pyndl.activation` provides the functionality to estimate activation of a trained ndl model for given events. The trained ndl model is thereby represented as the outcome-cue weights.

`pyndl.activation.activation(events, weights, number_of_threads=1, remove_duplicates=None, ignore_missing_cues=False)`

Estimate activations for given events in event file and outcome-cue weights.

Memory overhead for multiprocessing is one copy of weights plus a copy of cues for each thread.

Parameters

- **events** [generator or str] generates cues, outcomes pairs or the path to the event file
- **weights** [xarray.DataArray or dict[dict[float]]] the xarray.DataArray needs to have the dimensions ‘outcomes’ and ‘cues’ the dictionaries hold weight[outcome][cue].
- **number_of_threads** [int] a integer giving the number of threads in which the job should executed
- **remove_duplicates** [{None, True, False}] if None raise a ValueError when the same cue is present multiple times in the same event; True make cues unique per event; False keep multiple instances of the same cue (this is usually not preferred!)
- **ignore_missing_cues** [{True, False}] if True function ignores cues which are in the test dataset but not in the weight matrix if False raises a KeyError for cues which are not in the weight matrix

Returns

- **activations** [xarray.DataArray] with dimensions ‘outcomes’ and ‘events’. Contains coords for the outcomes. returned if weights is instance of xarray.DataArray

or

\[
activations [dict of numpy.arrays] the first dict has outcomes as keys and dicts as values the list has a activation value per event returned if weights is instance of dict

5.2.2 pyndl.corpus

`pyndl.corpus` generates a corpus file (outfile) out of a bunch of gunzipped xml subtitle files in a directory and all its subdirectories.

```python
class pyndl.corpus.JobParseGz(break_duration)
    Bases: object
    Stores the persistent information over several jobs and exposes a job method that only takes the varying parts as one argument.
```

Note: Using a closure is not possible as it is not pickable / serializable.

Methods

```python
run (filename)
```

`pyndl.corpus.create_corpus_from_gz(directory, outfile, *, n_threads=1, verbose=False)`
Create a corpus file from a set of gunzipped (.gz) files in a directory.

Parameters

- directory [str] use all gz-files in this directory and all subdirectories as input.
- outfile [str] name of the outfile that will be created.
- n_threads [int] number of threads to use.
- verbose [bool]

`pyndl.corpus.read_clean_gzfile(gz_file_path, *, break_duration=2.0)`
Generator that opens and reads a gunzipped xml subtitle file, while all xml tags and timestamps are removed.

Parameters

- break_duration [float] defines the amount of time in seconds that need to pass between two subtitles in order to start a new paragraph in the resulting corpus.

Yields

- line [non empty, cleaned line out of the xml subtitle file]

Raises

- FileNotFoundError [if file is not there.]

5.2.3 pyndl.count

`pyndl.count` provides functions in order to count

- words and symbols in a corpus file
- cues and outcomes in an event file
pyndl.count.cues_outcomes(event_file_name, *, number_of_processes=2, verbose=False)

Counts cues and outcomes in event_file_name using number_of_processes processes.

Returns

(n_events, cues, outcomes) [(int, collections.Counter, collections.Counter)]

pyndl.count.load_counter(filename)

Loads a counter out of a tab delimited text file.

pyndl.count.save_counter(counter, filename, *, header='key	freq
')

Saves a counter object into a tab delimited text file.

pyndl.count.words_symbols(corpus_file_name, *, number_of_processes=2, lower_case=False, verbose=False)

Counts words and symbols in corpus_file_name using number_of_processes processes.

Returns

(words, symbols) [(collections.Counter, collections.Counter)]

5.2.4 pyndl.io

pyndl.io provides functions to create event generators from different sources in order to use them with pyndl.ndl to train NDL models or to save existing events from a DataFrame or a list to a file.

pyndl.io.events_from_dataframe(df, columns=('cues', 'outcomes'))

Yields events for all events in a pandas dataframe.

Parameters

df [pandas.DataFrame] a pandas DataFrame with one event per row and one column with the cues and one column with the outcomes.

columns [tuple] a tuple of column names

Yields

cues, outcomes [list, list] a tuple of two lists containing cues and outcomes

pyndl.io.events_from_file(event_path, compression='gzip')

Yields events for all events in a gzipped event file.

Parameters

event_path [str] path to gzipped event file

compression [str] indicates whether the events should be read from gunzip file or not can be {"gzip" or None}

Yields

cues, outcomes [list, list] a tuple of two lists containing cues and outcomes

pyndl.io.events_from_list(lst)

Yields events for all events in a list.

Parameters

lst [list of list of str or list of str] a list either containing a list of cues as strings and a list of outcomes as strings or a list containing a cue and an outcome string, where cues respectively outcomes are seperated by an underscore

Yields
cues, outcomes [list, list] a tuple of two lists containing cues and outcomes

pyndl.io.events_to_file(events, file_path, delimiter='\t', compression='gzip', columns=('cues', 'outcomes'))

Writes events to a file

Parameters

events [pandas.DataFrame or Iterator or Iterable] a pandas DataFrame with one event per row and one column with the cues and one column with the outcomes or a list of cues and outcomes as strings or a list of a list of cues and a list of outcomes which should be written to a file

file_path: str path to where the file should be saved
delimiter: str Separator which should be used. Default is a tab
compression [str] indicates whether the events should be read from gzip file or not can be {"gzip" or None}
columns: tuple a tuple of column names

5.2.5 pyndl.ndl

pyndl.ndl provides functions in order to train NDL models

class pyndl.ndl.WeightDict (*args, **kwargs)

Subclass of defaultdict to represent outcome-cue weights.

Notes

Weight for each outcome-cue combination is 0 per default.

Attributes

attrs

default_factory Factory for default value called by __missing__().

Methods

clear()
copy()
fromkeys($type, iterable[, value]) Create a new dictionary with keys from iterable and values set to value.
get($self, key[, default]) Return the value for key if key is in the dictionary, else default.
items()
keys()
pop(k[,d]) If key is not found, d is returned if given, otherwise KeyError is raised
popitem() 2-tuple; but raise KeyError if D is empty.
setdefault($self, key[, default]) Insert key with a value of default if key is not in the dictionary.

Continued on next page
Table 1 – continued from previous page

| update([E,**F]) | If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k] |
| values()       |                                                                 |

attrs

pyndl.ndl.dictndl(events, alphas, betas, lambda_=1.0, *, weights=None, inplace=False, remove_duplicates=None, make_data_array=False, verbose=False)

Calculate the weights for all_outcomes over all events in event_file.

This is a pure python implementation using dicts.

Parameters

- events [generator or str] generates cues, outcomes pairs or the path to the event file
- alphas [dict or float] a (default) dict having cues as keys and a value below 1 as value
- betas [(float, float)] one value for successful prediction (reward) one for punishment
- lambda_ [float]
- weights [dict of dicts or xarray.DataArray or None] initial weights
- inplace: [True, False] if True calculates the weight matrix inplace if False creates a new weight matrix to learn on
- remove_duplicates [(None, True, False)] if None though a ValueError when the same cue is present multiple times in the same event; True make cues and outcomes unique per event; False keep multiple instances of the same cue or outcome (this is usually not preferred!)
- make_data_array [(False, True)] if True makes a xarray.DataArray out of the dict of dicts.
- verbose [bool] print some output if True.

Returns

- weights [dict of dicts of floats] the first dict has outcomes as keys and dicts as values the second dict has cues as keys and weights as values weights[outcome][cue] gives the weight between outcome and cue.

or

- weights [xarray.DataArray] with dimensions ‘outcomes’ and ‘cues’. You can lookup the weights between a cue and an outcome with weights.loc[['outcomes': outcome, 'cues': cue]] or weights.loc[outcome].loc[cue].

Notes

The metadata will only be stored when make_data_array is True and then dictndl cannot be used to continue learning. At the moment there is no proper way to automatically store the meta data into the default dict.

pyndl.ndl.events_from_file(event_path)

pyndl.ndl.ndl(events, alpha, betas, lambda_=1.0, *, method='openmp', weights=None, number_of_threads=8, len_sublists=10, remove_duplicates=None, verbose=False, temporary_directory=None, events_per_temporary_file=10000000)

Calculate the weights for all_outcomes over all events in event_file given by the files path.
This is a parallel python implementation using numpy, multithreading and the binary format defined in preprocess.py.

**Parameters**

- **events** (str) path to the event file
- **alpha** (float) saliency of all cues
- **betas** ((float, float)) one value for successful prediction (reward) one for punishment
- **lambda_** (float)
- **method** ({'openmp', 'threading'})
- **weights** [None or xarray.DataArray] the xarray.DataArray needs to have the dimensions ‘cues’ and ‘outcomes’
- **number_of_threads** (int) a integer giving the number of threads in which the job should executed
- **len_sublists** (int) a integer giving the length of sublists generated from all outcomes
- **remove_duplicates** (None, True, False)] if None though a ValueError when the same cue is present multiple times in the same event; True make cues and outcomes unique per event; False keep multiple instances of the same cue or outcome (this is usually not preferred!)
- **verbose** (bool) print some output if True.
- **temporary_directory** (str) path to directory to use for storing temporary files created; if none is provided, the operating system’s default will be used (/tmp on unix)
- **events_per_temporary_file** (int) Number of events in each temporary binary file. Has to be larger than 1

**Returns**

- **weights** [xarray.DataArray] with dimensions ‘outcomes’ and ‘cues’. You can lookup the weights between a cue and an outcome with weights.loc[['outcomes': outcome, 'cues': cue]] or weights.loc[outcome].loc[cue].

```python
pyndl.slice_list(list_, len_sublists)
```

Slices a list in sublists with the length len_sublists.

**Parameters**

- **list_** (list] list which should be sliced in sublists
- **len_sublists** (int] integer which determines the length of the sublists

**Returns**

- **seq_list** [list of lists] a list of sublists with the length len_sublists

### 5.2.6 pyndl.preprocess

`pyndl.preprocess` provides functions in order to preprocess data and create event files from it.

**class** `pyndl.preprocess.JobFilter` *(keep_cues, keep_outcomes, remove_cues, remove_outcomes, cue_map, outcome_map)*

**Bases:** `object`

Stores the persistent information over several jobs and exposes a job method that only takes the varying parts as one argument.
Note: Using a closure is not possible as it is not pickable / serializable.

## Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>job(line)</code></td>
<td></td>
</tr>
<tr>
<td><code>process_cues(cues)</code></td>
<td></td>
</tr>
<tr>
<td><code>process_cues_all(cues)</code></td>
<td></td>
</tr>
<tr>
<td><code>process_cues_keep(cues)</code></td>
<td></td>
</tr>
<tr>
<td><code>process_cues_map(cues)</code></td>
<td></td>
</tr>
<tr>
<td><code>process_cues_remove(cues)</code></td>
<td></td>
</tr>
<tr>
<td><code>process_outcomes(outcomes)</code></td>
<td></td>
</tr>
<tr>
<td><code>process_outcomes_all(outcomes)</code></td>
<td></td>
</tr>
<tr>
<td><code>process_outcomes_keep(outcomes)</code></td>
<td></td>
</tr>
<tr>
<td><code>process_outcomes_map(outcomes)</code></td>
<td></td>
</tr>
<tr>
<td><code>process_outcomes_remove(outcomes)</code></td>
<td></td>
</tr>
</tbody>
</table>

### pyndl.preprocess.bandsample(population, sample_size=50000, *, cutoff=5, seed=None, verbose=False)

Creates a sample of size `sample_size` out of the population using band sampling.

### pyndl.preprocess.create_binary_event_files(event_file, path_name, cue_id_map, outcome_id_map, *, sort_within_event=False, number_of_processes=2, events_per_file=10000000, overwrite=False, remove_duplicates=None, verbose=False)

Creates the binary event files for a tabular cue outcome corpus.

**Parameters**

- **event_file** [str] path to tab separated text file that contains all events in a cue outcome table.
- **path_name** [str] folder name where to store the binary event files
- **cue_id_map** [dict (str -> int)] cue to id map

5.2. Submodules
outcome_id_map [dict (str -> int)] outcome to id map
sort_within_event [bool] should we sort the cues and outcomes within the event
number_of_processes [int] number of threads to use
events_per_file [int] Number of events in each binary file. Has to be larger than 1
overwrite [bool] overwrite files if they exist
remove_duplicates [{None, True, False}] if None though a ValueError when the same cue is present multiple times in the same event; True make cues and outcomes unique per event; False keep multiple instances of the same cue or outcome (this is usually not preferred!)
verbose [bool]

Returns
number_events [int] sum of number of events written to binary files

pyndl.preprocess.create_event_file (corpus_file, event_file, symbols='abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ', *, context_structure='document', event_structure='consecutive_words', event_options=(3,), cue_structure='trigrams_to_word', lower_case=False, remove_duplicates=True, verbose=False)

Create an text based event file from a corpus file.

Parameters
corpus_file [str] path where the corpus file is
event_file [str] path where the output file will be created
symbols [str] string of all valid symbols
context_structure [{“document”, “paragraph”, “line”}]
event_structure [{“line”, “consecutive_words”, “word_to_word”, “sentence”}]
event_options [None or (number_of_words,) or (before, after) or None] in “consecutive words” the number of words of the sliding window as an integer; in “word_to_word” the number of words before and after the word of interest each as an integer.
cue_structure: {“trigrams_to_word”, “word_to_word”, “bigrams_to_word”}
lower_case [bool] should the cues and outcomes be lower cased
remove_duplicates [bool] create unique cues and outcomes per event
verbose [bool]

Notes

Breaks / Separators :

What marks parts, where we do not want to continue learning?

• ---end.of.document--- string?
• line breaks?
• empty lines?

What do we consider one event?
• three consecutive words?
• one line of the corpus?
• everything between two empty lines?
• everything within one document?

Should the events be connected to the events before and after?

No.

Context:

A context is a whole document or a paragraph within which we will take (three) consecutive words as occurrences or events. The last words of a context will not form an occurrence with the first words of the next context.

Occurrence:

An occurrence or event is will result in one event in the end. This can be (three) consecutive words, a sentence, or a line in the corpus file.

pyndl.preprocess.event_generator(event_file, cue_id_map, outcome_id_map, *, sort_within_event=False)

pyndl.preprocess.filter_event_file(input_event_file, output_event_file, *, keep_cues='all', keep_outcomes='all', remove_cues=None, remove_outcomes=None, cue_map=None, outcome_map=None, number_of_processes=1, chunk_size=100000, verbose=False)

Filter an event file by a list or a map of cues and outcomes.

Parameters

You can either use keep_*, remove_*, or map_*.

input_event_file [str] path where the input event file is
output_event_file [str] path where the output file will be created
keep_cues ['all' or sequence of str] list of all cues that should be kept
keep_outcomes ['all' or sequence of str] list of all outcomes that should be kept
remove_cues [None or sequence of str] list of all cues that should be removed
remove_outcomes [None or sequence of str] list of all outcomes that should be removed
cue_map [dict] maps every cue as key to the value. Removes all cues that do not have a key. This can be used to map several different cues to the same cue or to rename cues.
outcome_map [dict] maps every outcome as key to the value. Removes all outcome that do not have a key. This can be used to map several different outcomes to the same outcome or to rename outcomes.
number_of_processes [int] number of threads to use
chunksize [int] number of chunks per submitted job, should be around 100000

Notes

It will keep all cues that are within the event and that (for a human reader) might clearly belong to a removed outcome. This is on purpose and is the expected behaviour as these cues are in the context of this outcome.

5.2. Submodules
If an event has no cues it gets removed, but if an event has no outcomes it is still present in order to capture the background rate of that cues.

```python
pyndl.preprocess.ngrams_to_word(occurrences, n_chars, outfile, remove_duplicates=True)
```

Process the occurrences and write them to outfile.

**Parameters**

- `occurrences` [sequence of (cues, outcomes) tuples] cues and outcomes are both strings where underscores and # are special symbols.
- `n_chars` [number of characters (e.g. 2 for bigrams, 3 for trigrams, ...)]
- `outfile` [file handle]
- `remove_duplicates` [bool] if True make cues and outcomes per event unique

```python
pyndl.preprocess.process_occurrences(occurrences, outfile, *,
    cue_structure=’trigrams_to_word’,
    remove_duplicates=True)
```

Process the occurrences and write them to outfile.

**Parameters**

- `occurrences` [sequence of (cues, outcomes) tuples] cues and outcomes are both strings where underscores and # are special symbols.
- `outfile` [file handle]
- `cue_structure` [{‘bigrams_to_word’, ‘trigrams_to_word’, ‘word_to_word’}]
- `remove_duplicates` [bool] if True make cues and outcomes per event unique

```python
pyndl.preprocess.read_binary_file(binary_file_path)
```

```python
pyndl.preprocess.to_bytes(int_)
```

```python
pyndl.preprocess.to_integer(byte_)
```

```python
pyndl.preprocess.write_events(events, filename, *,
    start=0,
    stop=4294967295,
    remove_duplicates=None)
```

Write out a list of events to a disk file in binary format.

**Parameters**

- `events` [iterator of (cue_ids, outcome_ids) tuples called event]
- `filename` [string]
- `start` [first event to write (zero based index)]
- `stop` [last event to write (zero based index; excluded)]
- `remove_duplicates` [{None, True, False}] if None though a ValueError when the same cue is present multiple times in the same event; True make cues and outcomes unique per event; False keep multiple instances of the same cue or outcome (this is usually not preferred!)

**Returns**

- `number_events` [int] actual number of events written to file

**Raises**

- `StopIteration` [events generator is exhausted before stop is reached]
6.1 Getting Involved

The pyndl project welcomes help in the following ways:

- Making Pull Requests for code, tests or documentation.
- Commenting on open issues and pull requests.
- Helping to answer questions in the issue section.
- Creating feature requests or adding bug reports in the issue section.

6.2 Workflow

1. Fork this repository on Github. From here on we assume you successfully forked this repository to https://github.com/yourname/pyndl.git

2. Get a local copy of your fork and install the package in ‘development’ mode, which will make changes in the source code active immediately, by running

   ```
   git clone https://github.com/yourname/pyndl.git
   cd pyndl
   python setup.py develop --user
   ```

3. Add code, tests or documentation.

4. Test your changes locally by running within the root folder (pyndl/)

   ```
   make checkstyle
   make test
   ```

5. Add and commit your changes after tests run through without complaints.

   ```
   git add -u
   git commit -m 'fixes #42 by posing the question in the right way'
   ```
You can reference relevant issues in commit messages (like #42) to make GitHub link issues and commits together, and with phrase like “fixes #42” you can even close relevant issues automatically.

6. Push your local changes to your fork:

```
git push git@github.com:yourname/pyndl.git
```


**Note:** If you want to develop *pyndl* you should have installed:

```
pip install --user tox pylint pytest pycodestyle sphinx
```

### 6.3 Running tests

We use `make` and `tox` to manage testing. You can run the tests by executing the following within the repository’s root folder (`pyndl/`):

```
make test
```

For manually checking coding guidelines run:

```
make checkstyle
```

There is an additional way to invoke `pylint` as a linter with `tox` by running

```
tox -e lint
```

The linting gives still a lot of complaints that need some decisions on how to fix them appropriately.

### 6.4 Building documentation

Building the documentation requires some extra dependencies. Therefore, run

```
pip install -e .[docs]
```

in the project root directory. This command will install all required dependencies. The projects documentation is stored in the `pyndl/doc/` folder and is created with `sphinx`. You can rebuild the documentation by either executing

```
make documentation
```

in the repository’s root folder (`pyndl`) or by executing

```
make html
```

in the documentation folder (`pyndl/doc/`).
6.5 Continuous Integration

We use several services in order to continuously monitor our project:

<table>
<thead>
<tr>
<th>Service</th>
<th>Status</th>
<th>Config file</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Travis CI</td>
<td></td>
<td>.travis.yml</td>
<td>Automated testing</td>
</tr>
<tr>
<td>Coveralls</td>
<td></td>
<td></td>
<td>Monitoring of test coverage</td>
</tr>
<tr>
<td>Landscape</td>
<td></td>
<td>.landscape.yml</td>
<td>Monitoring of code quality</td>
</tr>
</tbody>
</table>

6.6 Licensing

All contributions to this project are licensed under the MIT license. Exceptions are explicitly marked. All contributions will be made available under MIT license if no explicit request for another license is made and agreed on.

6.7 Release Process

1. Ensure, that the version of the branch to be merged, is adequately increased see Versioning below.
2. Merge Pull Requests with new features or bugfixes into pyndl's master branch.
3. Create a new release on Github of the master branch of the form vX.Y.Z (where X, Y, and Z refer to the new version). Add a description of the new feature or bugfix. For details on the version number see Versioning below.
4. Pull the repository and checkout the tag and create the distribution files using:

   ```
git pull
git checkout vX.Y.Z
python setup.py build  # to compile *.pyx -> *.c
python setup.py sdist
   ```
5. Create GPG signatures of the distribution files using:

   ```
gpg --detach-sign -a dist/pyndl-X.Y.Z.tar.gz
   ```
6. (maintainers only) Upload the distribution files to PyPI using twine.

   ```
twine upload -s dist/*
   ```
7. (maintainers only) Check if the new version is on pypi (https://pypi.python.org/pypi/pyndl/).

6.8 Versioning

We use a semvers versioning scheme. Assuming the current version is X.Y.Z than X refers to the major version, Y refers to the minor version and Z refers to a bugfix version.

6.8.1 Bugfix release

For a bugfix only merge, which does not add any new features and does not break any existing API increase the bugfix version by one (X.Y.Z -> X.Y.Z+1).
6.8.2 Minor release

If a merge adds new features or breaks with the existing API a deprecation warning has to be supplied which should keep the existing API. The minor version is increased by one \((X.Y.Z \rightarrow X.Y+1.Z)\). Deprecation warnings should be kept until the next major version. They should warn the user that the old API is only usable in this major version and will not be available any more with the next major \(X+1.0.0\) release onwards. The deprecation warning should give the exact version number when the API becomes unavailable and the way of achieving the same behaviour.

6.8.3 Major release

If enough changes are accumulated to justify a new major release, create a new pull request which only contains the following two changes:

- the change of the version number from \(X.Y.Z\) to \(X+1.0.0\)
- remove all the API with deprecation warning introduced in the current \(X.Y.Z\) release
This is a collection of more or less unrelated tips and tricks that can be helpful during development and maintenance.

### 7.1 Running pyndl within R code

In order to run pyndl within R code first install Python and pyndl as described in the install instructions. Make sure pyndl runs for your user within Python.

Now we can switch to R and install the reticulate package (https://cran.r-project.org/web/packages/reticulate/vignettes/introduction.html) After having the reticulate package installed we can run within R the following code:

```r
library(reticulate)

learn_weights <- function(event_file) {
  py_env <- py_run_string(
    "from pyndl import ndl",
    paste0("weights = ndl.ndl('", event_file, ", alpha=0.01, betas=(1.0, 1.0), remove_duplicates=True)")),
    "weight_matrix = weights.data",
    "outcome_names = weights.coords['outcomes'].values",
    "cue_names = weights.coords['cues'].values",
    sep = "\n"
  ),
  convert = FALSE
}

wm <- py_to_r(py_env$weight_matrix)
rownames(wm) <- py_to_r(py_env$outcome_names)
colnames(wm) <- py_to_r(py_env$cue_names)
py_run_string(
    paste(
        "del cue_names",
        "del outcome_names",
        "del weight_matrix",
        "del weights",
        sep = "\n"
    ),
    convert = FALSE
)
wm
```

After having defined this function a gzipped tab separated event file can be learned using:
wm <- learn_weights('event_file.tab.gz')

Note that this code needs at the moment slightly more than two times the size of the weights matrix. There might be a way to learn the weight matrix without any copying between R and Python, but this needs to be elaborated a bit further. The basic idea is

1. to create the the matrix in R (in Fortran mode),
2. borrow / make the matrix available in Python,
3. transpose the matrix in Python to get it into C mode
4. learn the weights in place,
5. Check that the matrix in R has the weights learned as a side effect of the Python code.

Further reading:
- https://cran.r-project.org/web/packages/reticulate/vignettes/introduction.html
- https://cran.r-project.org/web/packages/reticulate/vignettes/arrays.html

### 7.2 Local testing with conda

Sometimes it might be useful to test if pyndl works in a clean python environment. Besides tox this is possible with conda as well. The commands are as follows:

```
conda create -n testpyndl
conda activate testpyndl
conda install python
python -c 'from pyndl import ndl; print("success")'  # this should fail
git clone https://github.com/quantling/pyndl.git
pip install pyndl
python -c 'from pyndl import ndl; print("success")'  # this should succeed
conda deactivate
conda env remove -n testpyndl
```

### 7.3 Memory profiling

Sometimes it is useful to monitor the memory footprint of the python process. This can be achieved by using memory_profiler (https://pypi.python.org/pypi/memory_profiler).
8.1 Authors

`pyndl` was mainly developed by Konstantin Sering, Marc Weitz, David-Elias Künstle and Lennart Schneider. For the full list of contributors have a look at Github’s Contributor summary.

Currently, it is maintained by Konstantin Sering and Marc Weitz.

8.2 Contact

In case you want to contact the project maintainers, please send an email to


8.3 Citation

If this work was helpful in your work, feel free to cite it as


If you are using BibTex you may want to use this example BibTex entry:

```latex
@misc{pyndl,
  author = {Konstantin Sering and Marc Weitz and David-Elias Künstle and Lennart Schneider},
  title = {Pyndl: Naive discriminative learning in python},
  year = 2017,
  doi = {10.5281/zenodo.597964},
  url = {https://doi.org/10.5281/zenodo.597964}
}
```

Note: If you want to cite a specific version, check out the history on zenodo!
8.4 Funding

*pyndl* was partially funded by the Humboldt grant, the ERC advanced grant and by the University of Tübingen.

8.5 Acknowledgements

This package is build as a python replacement for the R ndl2 package. Some ideas on how to build the API and how to efficiently run the Rescorla Wagner iterative learning on large text corpora are inspired by the way the ndl2 package solves this problems. The ndl2 package will be published to github soon and a reference will be placed here.
Python Module Index

p
pyndl, 21
pyndl.activation, 21
pyndl.corpus, 22
pyndl.count, 22
pyndl.io, 23
pyndlndl, 24
pyndl.preprocess, 26
INDEX

A
activation() (in module pyndl.activation), 21
attrs (pyndl.ndl.WeightDict attribute), 25

B
bandsample() (in module pyndl.preprocess), 27

C
create_binary_event_files() (in module pyndl.preprocess), 27
create_corpus_from_gz() (in module pyndl.corpus), 22
create_event_file() (in module pyndl.preprocess), 28
cues_outcomes() (in module pyndl.count), 23

D
dict_ndl() (in module pyndl.ndl), 25

event_generator() (in module pyndl.preprocess), 29
events_from_dataframe() (in module pyndl.io), 23
events_from_file() (in module pyndl.ndl), 25
events_from_list() (in module pyndl.io), 23
events_to_file() (in module pyndl.io), 24

F
filter_event_file() (in module pyndl.preprocess), 29

J
job() (pyndl.preprocess.JobFilter method), 27
JobFilter (class in pyndl.preprocess), 26
JobParseGz (class in pyndl.corpus), 22

L
load_counter() (in module pyndl.count), 23

N
ndl() (in module pyndl.ndl), 25
ngrams_to_word() (in module pyndl.preprocess), 30

P
process_cues() (pyndl.preprocess.JobFilter method), 27
process_cues_all() (pyndl.preprocess.JobFilter method), 27
process_cues_keep() (pyndl.preprocess.JobFilter method), 27
process_cues_map() (pyndl.preprocess.JobFilter method), 27
process_cues_remove() (pyndl.preprocess.JobFilter method), 27
process_occurrences() (in module pyndl.preprocess), 30
process_outcomes() (pyndl.preprocess.JobFilter method), 27
process_outcomes_all() (pyndl.preprocess.JobFilter method), 27
process_outcomes_keep() (pyndl.preprocess.JobFilter method), 27
process_outcomes_map() (pyndl.preprocess.JobFilter method), 27
process_outcomes_remove() (pyndl.preprocess.JobFilter method), 27
pyndl (module), 1, 21
pyndl.activation (module), 21
pyndl.corpus (module), 22
pyndl.count (module), 22
pyndl.io (module), 23
pyndl.ndl (module), 24
pyndl.preprocess (module), 26

R
read_binary_file() (in module pyndl.preprocess), 30
read_clean_gzfile() (in module pyndl.corpus), 22
return_empty_string()
  (pyndl.preprocess.JobFilter static method),
run() (pyndl.corpus.JobParseGz method), 22

S
save_counter() (in module pyndl.count), 23
slice_list() (in module pyndl.ndl), 26

T
to_bytes() (in module pyndl.preprocess), 30
to_integer() (in module pyndl.preprocess), 30

W
WeightDict (class in pyndl.ndl), 24
words_symbols() (in module pyndl.count), 23
write_events() (in module pyndl.preprocess), 30