

# Homework 2:

## Paraphrase Recognition; Matrix Representations

Dr. Aikaterini-Lida Kalouli  
Computerlinguistische Anwendungen

Due: Wednesday May 18, 2022, 14:00

In the first exercise, you will use the perceptron to implement a paraphrase classifier and design some features for that. In the second exercise, you will practice the use of Numpy arrays and the Scikit-Learn CountVectorizer.

### Exercise 1: Paraphrase Detection with the Perceptron

Get the paraphrases folder (*paraphrases*) from the data folder of your GitLab project. Have a look at the training, development and test files. They contain pairs of (tab-separated) tweets and a label of the tweets being paraphrases, or not.

Take a look at the file `paraphrases.py`. In this script, the perceptron implementation of the first exercise is used to solve the paraphrase detection task. To make it work, you need to complete some functions, which you can verify with unittests:

```
python3 -m unittest -v hw02_paraphrases/test_paraphrases.py
```

Note: we are providing the files `utils.py` and `perceptron_classifier.py` of the first exercise as compiled files. These files are normally imported in the script. If you get an error in importing, make sure you use version Python 3.8 (because this is the version used to compile these files).

#### Exercise 1.1: Token N-grams [2 Points]

Complete the function `token_ngrams`. Given a list of tokens, the function should return a list of all ngrams ( $n$  consecutive tokens). Each ngram is a string and the concatenated tokens are separated by whitespace (" ").

**Attention:** The function should not only work for 3-grams, instead the function should use the parameter  $n$  as a generic number.

### Exercise 1.2: Token Features [1 Points]

Complete the function `token_features`. Given two **sets of tokens** (without repetitions)  $A$  and  $B$ , the following features should be added to a new dictionary:

- `features[WORD_OVERLAP]`: The number of tokens common to both sets (the intersection  $A \cap B$ )
- `features[WORD_UNION]`: The number of tokens in the union  $A \cup B$

Then return the dictionary.

### Exercise 1.3: Word Ngram Features [1 points]

Complete the function `ngram_features`. Given two sets of token ngrams, the following features should be added to a new dictionary:

- `features[WORD_NGRAM_OVERLAP]`: The number of token ngrams common to both texts (the intersection of the sets).
- `features[WORD_NGRAM_UNION]`: The number of token ngrams in the union of both sets.

Then return the dictionary.

### Exercise 1.4: Character Ngram Features [1 points]

Complete the function `ngram_features`. Given two sets of character ngrams, the following features should be added to a new dictionary:

- `features[CHARACTER_NGRAM_OVERLAP]`: The number of character ngrams common to both texts (the intersection of the sets).
- `features[CHARACTER_NGRAM_UNION]`: The number of character ngrams in the union of both sets.

Then return the dictionary.

### Exercise 1.5: Wordpair Features [1 points]

Complete the function `wordpair_features`. Given two sets of tokens, a feature for **every** wordpair  $(u, v)$  should be added, where  $u$  appears in the first set of tokens, and  $v$  in the second one. To represent each feature, use the symbol `#` as infix between  $u$  and  $v$ . (For example, there are 4 wordpair features for  $\{\text{'hello'}, \text{'greetings'}\}$  and  $\{\text{'hi'}, \text{'bye'}\}$ , one of which is `hello#bye`). Return a dictionary that maps each wordpair feature (for the given token sets) to the value 1.

### Exercise 1.6: Feature Comparison [0 points]

This exercise is a small check whether your features work with the real data. Call the script from the *src-folder* with:

```
python3 -m hw02_paraphrases.paraphrases -t data/paraphrases/train.txt \
-d data/paraphrases/dev.txt -e data/paraphrases/test.txt
```

and see how it performs. You might be interested in how much each of your features contributed to this result. For that, a *feature comparison mode* was implemented! Simply add the flag `-fc` to the command above. This is also a great way to check if all of your functions in the previous exercises on this sheet are working correctly. You should receive something like this:

FEATURE COMPARISON MODE

---

Only wordpair features  
Dev acc: 0.6484789956542734

---

Only character ngram features  
Dev acc: 0.7349106711733462

---

Only word ngram features  
Dev acc: 0.6875905359729599

---

Only token features  
Dev acc: 0.739014968614196

...

Note that there are some non-deterministic parts in the code, so the scores may vary a little bit! (You get points if your features work correctly with the dataset.)

### Exercise 2: Numpy

Complete the following functions in `small_functions.py`. You can check your progress using the doctests:

```
python3 -m doctest -v hw02_paraphrases/small_functions.py
```

and unittests:

```
python3 -m unittest -v hw02_paraphrases/test_small_functions.py
```

#### Exercise 2.1: Creating a 1d Numpy Array [4 points]

Complete the function `square_roots(start, end, length)`, that returns a 1d (vector shaped) numpy array with the specified `length`. It should contain the square roots of equally spaced input values between `start` and `end` (both included). Look at the doctest for an example. Have a look at the numpy function `np.linspace` to get some help: <https://numpy.org/doc/stable/reference/generated/numpy.linspace.html>

### Exercise 2.2: Creating a 2d Numpy Array [4 points]

Complete the function `odd_ones_squared(rows, cols)`, that returns a 2d numpy array with shape `(rows, cols)`. The matrix cells should contain increasing integer values (create a range and reshape), where all odd numbers are squared. Look at the doctest for an example. You can also use the numpy function `np.putmask()` for this exercise (<https://numpy.org/doc/stable/reference/generated/numpy.putmask.html>).