# Sprachverarbeitung: Übung

**SoSe 24**

Janis Pagel
Department for Digital Humanities, University of Cologne

22 April 2025

Please submit your solution via Ilias, either as a Jupyter Notebook (`.ipynb`, you can export your Notebook in Jupyter by going to File > Download) or as a Python script (`.py`) if you are not working in Jupyter.

You can write the written answers to the exercises into the source code as a comment or into a separate document (PDF). In the second case, please submit both your PDF file and your Python source code.

Exercise 6 is a bonus exercise that you can solve if you want to, but don't have to.

## Exercise 1.

Download the following corpus files from the *Deutsches Text Archiv* (DTA) into a directory called `dta` and extract them (if you are working on `compute.spinfo.uni-koeln.de`) in Jupyter, it is probably easier to extract the files on your local machine and then upload them to Jupyter via the upload button).

- `https://media.dwds.de/dta/download/dtae/2020-10-23/original/1400-1499.zip`
- `https://media.dwds.de/dta/download/dtae/2020-10-23/original/1500-1599.zip`

This should give you the sub-directories `1400-1499` and `1500-1599`. Write a Python script that reads in all `.txt` files from each directory and stores the content into a dictionary that contains the century ("1400" and "1500") as a key and all tokens from the text files for this century as the value in a list. Hence, your dictionary should look something like the following:

```
{1400: ["token1", "token2", "token3", ...],
 1500: ["token1", "token2", "token3", ...]}
```

In order to achieve proper tokenization, you can use the NLTK library, as shown on today's slides.

Create a pandas dataframe out of this dictionary. To achieve this, first convert your dictionary into a pandas `Series` object and use the `explode()` and `reset_index()` functions like so:

```
pd.DataFrame(pd.Series(dictionary).explode().reset_index().rename(columns={"index": "
                                                century", 0: "token"}))
```

Your dataframe should contain two columns, one with the century and one with the respective tokens:

| century | token |
|---|---|
| 1400 | token1 |
| 1400 | token2 |
| 1400 | token3 |
| ... | |
| 1500 | token1 |
| 1500 | token2 |
| 1500 | token3 |

Hint: In order to get a list of all the files in the directories, you can import the `glob` Library and use the following code: `list(glob.iglob("dta/**/*.txt", recursive=True))`. This will give you a list of the file names and paths for each txt-File. You can then iterate over this list to read in all the files' content.

## Solution 1.

```
import glob
import pandas as pd
from nltk.tokenize import word_tokenize
import nltk
nltk.download("punkt_tab")

def read_files(file_list):
    file_dict = {}
    for f in file_list:
        with open(f, "r") as fo:
            if "1400-1499" in f:
                if 1400 in file_dict:
                    file_dict[1400].extend(word_tokenize(fo.read()))
                else:
                    file_dict[1400] = word_tokenize(fo.read())
            elif "1500-1599" in f:
                if 1500 in file_dict:
                    file_dict[1500].extend(word_tokenize(fo.read()))
                else:
                    file_dict[1500] = word_tokenize(fo.read())
    return file_dict

file_dict = read_files(list(glob.iglob("dta/**/*.txt", recursive=True)))
```

```
df = pd.DataFrame(pd.Series(file_dict).explode().reset_index().rename(columns={"index":
                                                "century", 0: "token"}))
```

## Exercise 2.

On the dataframe from Exercise 1, calculate Type-Token Ratio (TTR) for the 1400 and the 1500 corpus, respectively. To do this, look at the pandas methods `drop_duplicates()` and `size()`. What do you observe? What do the numbers tell you about the vocabulary distribution in the two corpora?

## Solution 2.

```
df.drop_duplicates().groupby("century").size() / df.groupby("century").size()
```

The TTR for the 1400 sub-corpus is 0.1356 and for the 1500 sub-corpus, the TTR is 0.0428. This means that the ratio of types per tokens is higher in the 1400 corpus and thus the 1400 corpus has a "richer" vocabulary with more unique words than the 1500 corpus. Note however that this approach is not taking into account the lengths of the texts.

The 1500 corpus uses the same words a lot, there is only a 4% chance to encounter a new word; compared to the 1400 corpus, where there is a 13.5% chance to encounter a new word when going through the texts. This could mean that the 1500 corpus has a lot of texts that deal with the same or very similar topics.

## Exercise 3.

You have the hypothesis that the 1500s use much more words related to the topic of "family" than compared to the 1400s. Using the dataframe from Exercise 1, retrieve the absolute counts for German family-related tokens (try at least "kind", "mutter" and "vater") and compare them between the 1400 and 1500 corpora. Do the numbers corroborate or disprove your hypothesis?

Now calulate the relative counts for each century for the same tokens and compare. How have the numbers changed? Do they still corroborate/disprove your hypothesis? Do you think it is generally valid to investigate change in vocabulary usage this way?

## Solution 3.

```
df_counts = df.groupby("century").value_counts().reset_index()
df_counts_norm = df.groupby("century").value_counts(normalize=True).reset_index()
print(df_counts[(df_counts["token"] == "kind") | (df_counts["token"] == "mutter") | (
                                                df_counts["token"] == "vater")])
print(df_counts_norm[(df_counts_norm["token"] == "kind") | (df_counts_norm["token"] == "
                                                mutter") | (df_counts_norm["token"] == "
                                                vater")])
```

The absolute counts corroborate the hypothesis that the 1500s use more family-related words, since the absolute counts for "kind", "vater" and "mutter" are much higher for the 1500 corpus:

| centruy | token | absolute count |
|---|---|---|
| 1400 | kind | 109 |
| 1400 | vater | 16 |
| 1400 | mutter | 9 |
| 1500 | kind | 361 |
| 1500 | vater | 234 |
| 1500 | mutter | 96 |

However, the absolute counts are not very meaningful for a comparison, since the 1400 and 1500 corpora have very different lengths and the words had much more chances to appear in the 1500 corpus a priori. Therefore, it is more meaningful to look at the relative counts:

| centruy | token | relative count |
|---|---|---|
| 1400 | kind | 0.000866 |
| 1400 | vater | 0.000127 |
| 1400 | mutter | 0.000072 |
| 1500 | kind | 0.000079 |
| 1500 | vater | 0.000052 |
| 1500 | mutter | 0.000021 |

Here we can see that relative to the size of the corpora, the three tokens appear more frequently in the 1400 corpus. This would disprove the hypothesis that the 1500s contain more family-related words. In general, one can problematize the approach that was taken here in several ways. In order to make a general statement about the 1400s and 1500s, the textual basis can be considered too small (note however that for previous centuries there often simply is not much more data available). Furthermore, only looking at a handful of hand-picked words to represent the topic of "family" is problematic, one would need to find a more general approach to operationalize/represent the topic of "family" quantitatively. Lastly, the hypothesis of "do these words appear more frequently" is probably not very interesting to begin with, usually one would be interest in more general statements such as "the concept of 'family' is more important in century x compared to century y". The approach taken here is in this form not able to investigate such a hypothesis, only approximate it.

### Exercise 4.

Retrieve the absolute and relative counts for the token "gott" in both centuries. What does it tell you about the usage of this token across the centuries? Now retrieve absolute and relative counts for the token "Gott" in both centuries. What do these counts tell you, also in regards to the results for the token "gott"?

### Solution 4.

```
df_counts = df.groupby("century").value_counts().reset_index()
df_counts_norm = df.groupby("century").value_counts(normalize=True).reset_index()
print(df_counts[(df_counts["token"] == "gott") | (df_counts["token"] == "Gott")])
print(df_counts_norm[(df_counts_norm["token"] == "gott") | (df_counts_norm["token"] == "
                                                    Gott")])
```

The absolute and relative counts for the token "gott" are as follows:

| century | token | absolute count | relative count |
|---|---|---|---|
| 1400 | gott | 31 | 0.000246 |
| 1500 | gott | 99 | 0.000022 |

The relative count would suggest that the token is much more prevelant in the 1400 corpus. However, when looking at the counts for the token "Gott":

| century | token | absolute count | relative count |
|---|---|---|---|
| 1400 | Gott | 2 | 0.000016 |
| 1500 | Gott | 14133 | 0.003111 |

it becomes clear that there is also an effect of the capitalized version being favored in the 1500 corpus and almost not present in the 1400 corpus. This problem of counting the different versions of words as the same word can not be solved by counting tokens, but can be addressed via so called *lemmatization* (i.e. finding the base form (or lemma) of tokens).

### Exercise 5.

Using the dataframe from Exercise 1, calculate the absolute counts for each token per century and store in a new column. Based on these counts, retrieve the rank of each token per century, such that the token with the highest absolute count has the highest rank (rank 1), decreasing with frequency. To achieve this, look into the usage of the pandas method called `cumcount()`. Use seaborn's `lineplot` to plot the absolute count on the y axis and the rank on the x axis and the century as a group to create two lines, one for 1400 and one for 1500. What do you observe? Can you see a Zipf curve?

Now, only use the 50 highest ranked tokens from each century and plot again. What is the difference? Can you observe a Zipf curve? If not, what is the problem? How could you solve this problem?

Lastly, create a mini corpus with only two texts, `dta/1400-1499/nn_almanach05_1473.txt` and `dta/1500-1599/rechnungsbuch01_1500.txt`. Create the same plots with only these two texts. What do you observe? How do you explain?

## Solution 5.

```
df_zipf = df.groupby("century").value_counts().reset_index()
df_zipf["rank"] = df_zipf.groupby("century").cumcount()+1
sns.lineplot(data=df_zipf, x = "rank", y = "count", hue = "century")

# Only the 50 highest ranked tokens
df_zipf = df.groupby("century").value_counts().reset_index()
df_zipf["rank"] = df_zipf.groupby("century").cumcount()+1
df_zipf = df_zipf.groupby("century").head(50)
sns.lineplot(data=df_zipf, x = "rank", y = "count", hue = "century")

# Using normalized counts instead of absolute
df_zipf = df.groupby("century").value_counts(normalize=True).reset_index()
df_zipf["rank"] = df_zipf.groupby("century").cumcount()+1
df_zipf = df_zipf.groupby("century").head(50)
sns.lineplot(data=df_zipf, x = "rank", y = "proportion", hue = "century")

# Experiments with the mini corpus with only two texts
mini_file_dict = read_files(["dta/1400-1499/nn_almanach05_1473.txt", "dta/1500-1599/
                                        rechnungsbuch01_1500.txt"])
mini_df = pd.DataFrame(pd.Series(mini_file_dict).explode().reset_index().rename(columns=
                                        {"index": "century", 0: "token"}))
mini_df_zipf = mini_df.groupby("century").value_counts(normalize=True).reset_index()
mini_df_zipf["rank"] = mini_df_zipf.groupby("century").cumcount()+1
mini_df_zipf = mini_df_zipf.groupby("century").head(50)
sns.lineplot(data=mini_df_zipf, x = "rank", y = "proportion", hue = "century")
```

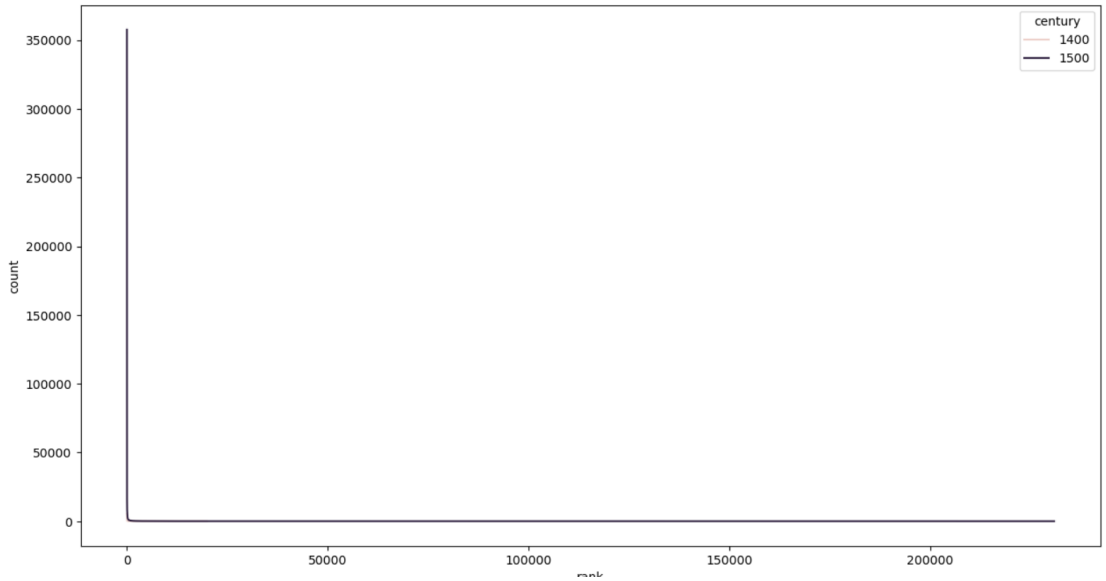


Figure 1:

The first plot (Figure 1) does not clearly show the expected Zipf curve, since there are too many words ranked on the x-axis and the curve becomes almost rectangular. This can be solved by reducing the number of highest ranked tokens to 50.

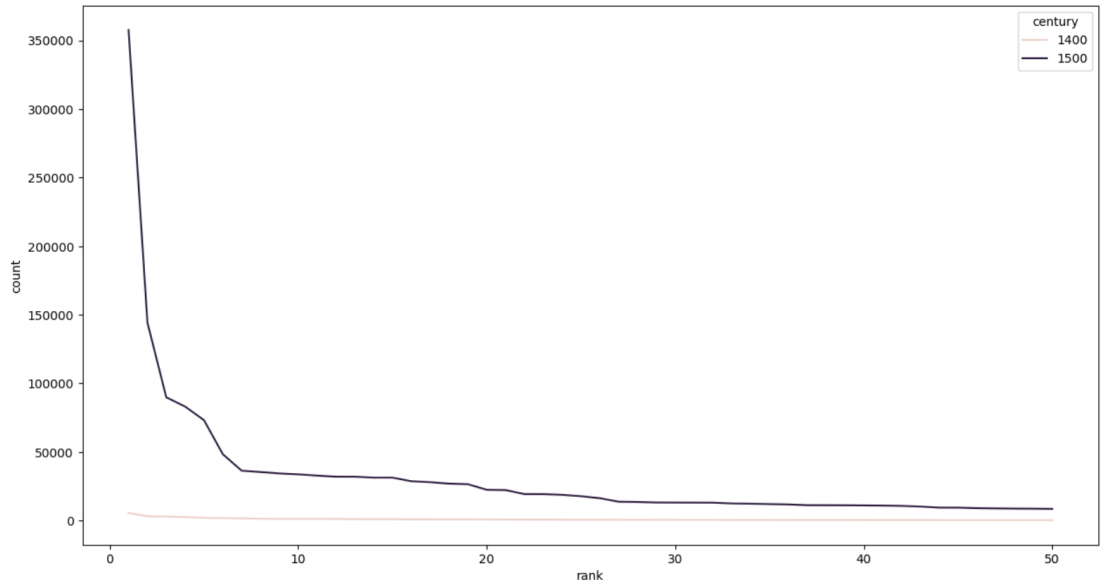


Figure 2:

In the resulting plot (Figure 2), the Zipfian curve is recognizable for the 1500 corpus,

but for the 1400 corpus, it is almost a flat line. This is because the absolute counts of the 1500 corpus are so much higher than for the 1400 corpus and a comparsion is therefore difficult. One solution to solve this problem is to use relative counts instead of absolute counts for the y-axis.

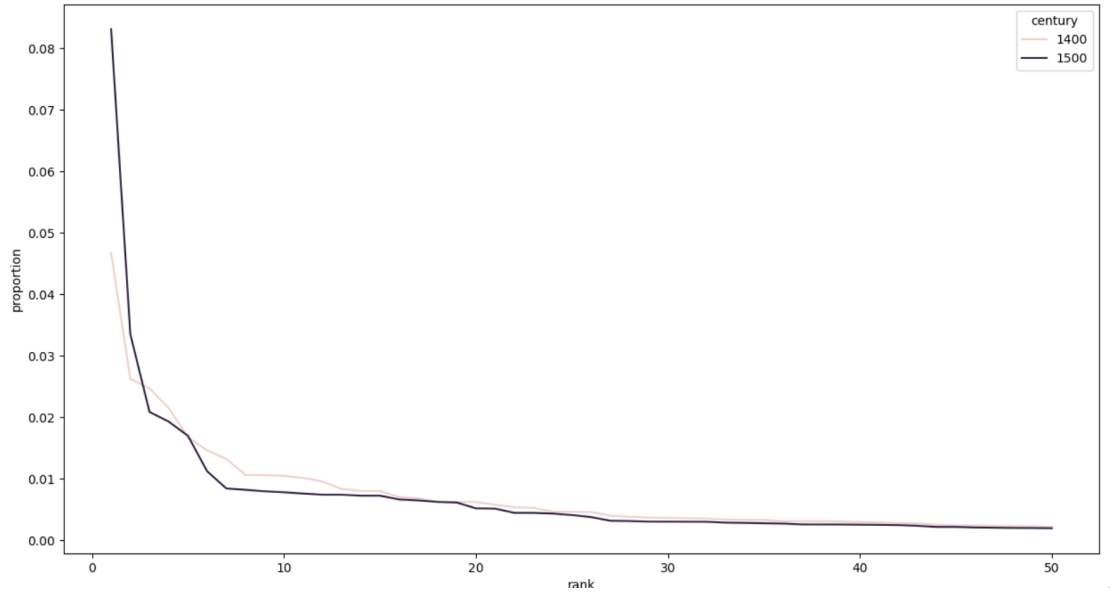


Figure 3:

The result can be seen in this plot (Figure 3), where the Zipf curve can now be seen for both corpora (at least in tendency).

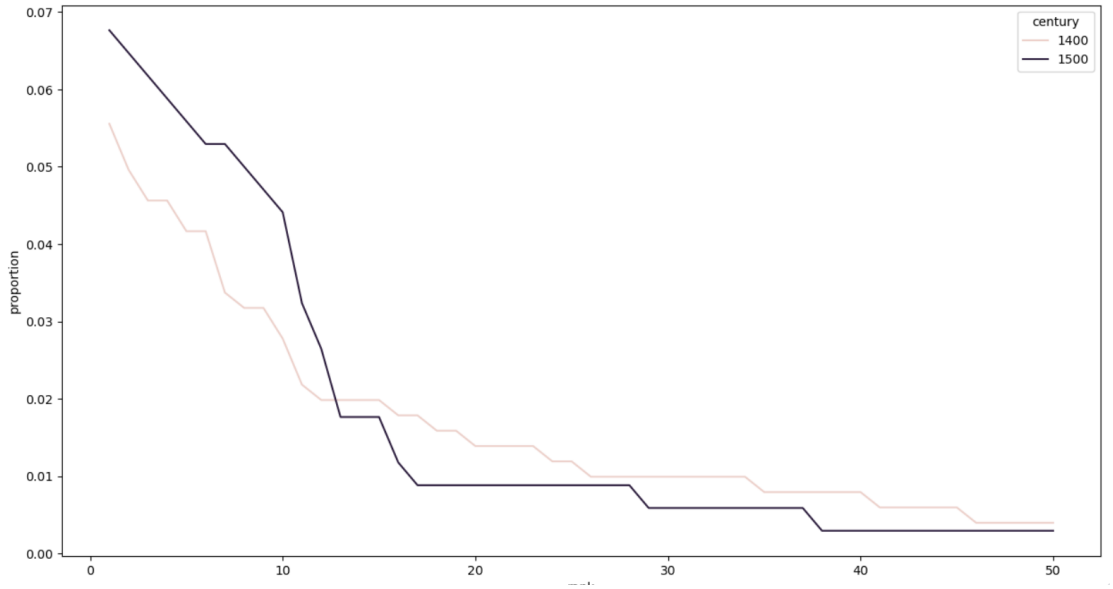


Figure 4:

Lastly, when using the mini corpus, the Zipf curves disappear completely, or at least are barely recognizable (Figure 4). This is because Zipf's law only holds starting from a certain threshold of tokens. If the texts are too short, there are not enough observations for Zipf's law to hold.

### Bonus Exercise 6.

Using the dataframe from Exercise 1, calculate Standardized TTR (STTR) for a window of 1000 words per century. Look into the usage of the pandas method `cut()` to create the windows. Compare your result to the result of Exercise 2. Are there differences? What do the numbers tell you about the distribution of the vocabulary in the texts of the two centuries?

### Solution 6.

```
window_size = 1000
df_1400 = df[df["century"] == 1400]
df_1500 = df[df["century"] == 1500]
df_1400["windows"] = pd.cut(df_1400.index+1, range(0, df_1400.index.max() + window_size,
                                                   window_size))
df_1500["windows"] = pd.cut(df_1500.index+1, range(0, df_1500.index.max() + window_size,
                                                   window_size))
token_count_1400 = df_1400.groupby(["century", "windows"], observed=True).size().
                                                  reset_index()
token_count_1500 = df_1500.groupby(["century", "windows"], observed=True).size().
                                                  reset_index()
type_count_1400 = df_1400.drop_duplicates().groupby(["century", "windows"], observed=
                                                  True).size().reset_index()
```

```
type_count_1500 = df_1500.drop_duplicates().groupby(["century", "windows"], observed=
                                                    True).size().reset_index()

print((type_count_1400[0] / token_count_1400[0]).mean())
print((type_count_1500[0] / token_count_1500[0]).mean())
```

The STTR for the 1400 corpus with a window size of 1000 is 0.4471927855031303 and for the 1500 corpus the STTR is 0.4741545732217667. These two values are much closer together than the values for the non-standardized TTR (0.169772 for 1400 and 0.053673 for 1500), which means that when disregarding the size of the corpora, the average TTR is actually very similar. When compared to the much smaller non-standardized TTR values, this also means that on average, the TTR values are relatively equally distributed, but there must be passages in the corpora where there are much less types per tokens than on average, especially for the 1500 corpus.