Harry Potter and a Network Analysis

based on Sentiment Analysis & Topic Modeling using SpaCy

by Merle-Sophie Thoma and Teresa Wendel

Abstract

Natural Language Processing - Applied NLP Thinking - Network Analysis - Sentiment Analysis - Topic Modeling - Media Studies - Harry Potter - Learning Reflection

This paper describes a first approach to the topic of NLP, more precisely Applied NLP. With the help of software products of the company explosion (spaCy and Prodigy), the basic features of NLP are to be comprehended and applied to an exemplary project topic.

For this purpose, data from the Harry Potter saga in the form of its film scripts was used in order to achieve initial analysis results in the question area of social networks with the help of sentiment analysis and topic modeling.

The project focuses on the learning experience by documenting and reflecting on the project and its implementations as well as the visualization of the analysis results. Thus, the implementation of the NLP question is based on rather rudimentary Python knowledge and the desire to gain first basic knowledge in the application and understanding of NLP software. For this purpose, first an outline of theoretical basics is recorded, in a further step the implementation is documented and finally the project experience is reflected.

The corresponding repository contains the data (hp_data), a folder with Python and Markdown basics (project_basics), a notebook with code, as well as the visualizations in separate folders (hp_sentimentdistributions, hp_network, hp_wordclouds) and the Reademe (readme.txt).

I. What is Network Analysis & Why Network Analysis with Harry Potter?

In terms of social network analysis, we want to understand it here as follows:

> *Network analysis can be considered as a set of techniques with a common methodological perspective that allow researchers to represent the relationships between actors and to analyze the social structures that result from the recurrence of these relationships. The basic assumption is that a better explanation of social phenomena is achieved by analyzing the relationships between actors. This analysis is done by collecting relationship data organized in the form of a matrix. When actors are represented as nodes and their relationships are represented as lines between pairs of nodes, the concept of a social network is transformed from a metaphor to an operational analytical tool that uses the mathematical language of graph theory and the linear assumptions of matrix algebra. (Chiesi 2015, 518)*

The forText digital research environment defines network analysis in the context of literary analysis and visualization as follows:

> *In network analysis, certain entities (e.g., figures, authors, places) are previously examined in their relationship to each other as a network of nodes and connecting edges or relations. In this way, quantitative aspects of the relational system,*

> *primarily the number of nodes, edges, and links, first become clear and can serve as the basis for a qualitative analysis. (Schumacher 2018)*

Our goal is to use natural language processing (NLP) and network analysis tools to provide a representation of the social relationships of the main characters in the Harry Potter films. Based on a database containing the dialog scripts of the eight films and various views created by queries, we first want to pre-select the characters and create views with relevant dialog-sequences with the help of SQL queries. Then, based on the results of Sentiment Analysis and Topic Modeling, we want to map their relationships.

We think that this approach could have exemplary value for the analysis of more complex (fictional) social networks. In particular, we can imagine that following such a basic consideration, especially the change of such relationships over time could be of interest for both film and literature or, for example, theatre studies. In this fields, for example, the comparison of certain patterns of relationships in certain periods, concerning certain authors, or certain genres might also be of interest.

II. NLP Basics (SpaCy, Prodigy, Sentiment Analysis & Topic Modeling)

II.I What is SpaCy & What can we use it for?

SpaCy is a free, open-source library for advanced Natural Language Processing (NLP) in Python. SpaCy is designed specifically for production use and helps you build applications that process and *understand* large volumes of text. It can be used to build information extraction or natural language understanding systems, or to pre-process text for deep learning.

Among other things, SpaCy can be used for Tokenization, Part-of-Speech Tagging, Dependency Parsing, Rule-based matching, training statistical models and Named Entity Recognition (NER).

II.II What is Prodigy & What can we use it for?

Prodigy is a modern annotation tool for creating, training and evaluating data for machine learning models. Prodigy is designed to help you inspect and clean your data, do error analysis and develop rule-based systems to use in combination with your statistical models. Among other things, Prodigy can be used for Span Categorization, Named Entity Recognition and Text Classification. Prodigy can be used as both a command-line tool and web application.

II.III What is Sentiment Analysis & Basic Sentiment Analysis using the SpaCy Pipeline Extension Textblob

Sentiment analysis or Sentiment Detection is a subfield of text mining and refers to the automatic evaluation of texts with the goal of identifying an expressed attitude as positive, negative or neutral.

The textblob tool evaluates the sentiment of a text based on the parameters polarity and subjectivity and outputs the former as a value between -1 and 1, the latter as a value from 0 to 1. Other machinelearning based models are furthermore able to distinguish between different emotions, but for our purposes the results of SpaCy's Textblob prove to be sufficient.

Another way to analyze sentiments could be via the text classification function of Prodigy. In this way, a model could be trained to categorize text according to different emotions (e.g. according to W. James 1890: fear, grief, love, rage). However, the amount of data seems to be too small to train a meaningful model.

II.IV What is Topic Modeling? & Basic Python Topic Modeling with Gensim

Topic modeling is a probability-based text-mining tool that is used to find "topics" by mapping frequent common occurrences of words in natural language texts. Topic models generate topics, which are collecions of words. Words may belong to several topics. In order to use topic modeling it is important to perpare the coprus by cleaning, tokenizing and stemming it. For this it is useful to create a manually annotated training and testing corpus.

For our attempt at topic modeling we used Gensim. Gensim is an open source library that allows you to do topic modeling by analysing large corpora. As a model, we used the *Latent Dirichlet allocation.*

III. Implementation

III.I Project Goal, Ideas and Structure

Our goal is to depict the social relations of the Harry Potter world using NLP based on data from the movie scripts. To do this, we will first take a closer look at the given data and select main characters based on certain parameters. Then, we will analyze the reciprocal relationship of these characters to the main protagonist Harry Potter. Sentiment analysis and topic modeling will be used for this purpose. Finally, sentiment and topics will be visualized in a descriptive way to make them presentable. This will be followed by a critical reflection of the project.

To develop this project idea, we started with research on corpora that would be suitable for NLP and chose the subject matter based on personal interest. After that, we started to familiarize ourselves with the basics of Git, GitHub and Python (Jupyter Notebook). We created a project notebook and decided to use the following data as the basis of our project.

III.II Data

We use the data from the GitHub account Kornflex28, which we found via Kaggle. The repository contains the .csv files of the eight-part film saga that we use, as well as a table on their metadata.

To gain better insight, we loaded our data into the SQLite Browser DB and performed queries (more on this in the digressions: Hands on Data - SQLite Queries).

https://www.kaggle.com/kornflex/harry-potter-movies-dataset?select=datasets

https://github.com/kornflex28/hp-dataset

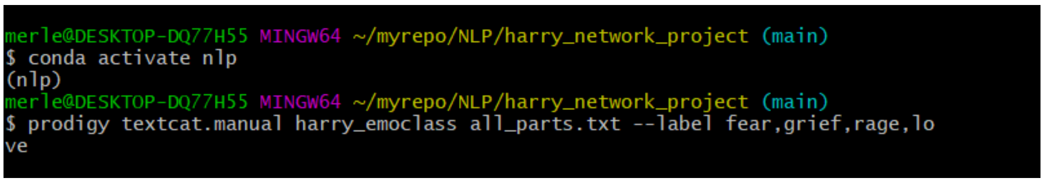
III.III Implementation

How to: Sentiment Analysis Attempt I

Step One: Preprocessing the Data

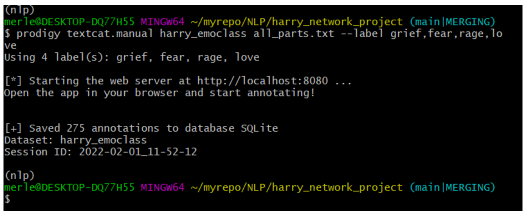
Exporting all dialogfields from SQLite to .txt and using the all_parts.txt-file to train a text classification using Prodigy.

Step Two: Feeding Data to Prodigy

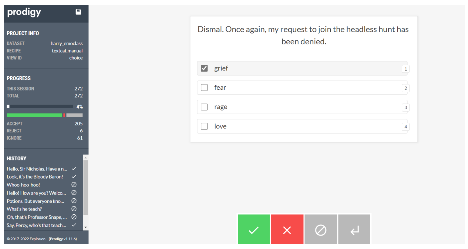


```
merle@DESKTOP-DQ77H55 MINGW64 ~/myrepo/NLP/harry_network_project (main)
$ conda activate nlp
(nlp)
merle@DESKTOP-DQ77H55 MINGW64 ~/myrepo/NLP/harry_network_project (main)
$ prodigy textcat.manual harry_emoclass all_parts.txt --label fear,grief,rage,lo
ve
```

Step Three: Text Categorization using Prodigy



```
(nlp)
merle@DESKTOP-DQ77H55 MINGW64 ~/myrepo/NLP/harry_network_project (main|MERGING)
$ prodigy textcat.manual harry_emoclass all_parts.txt --label grief,fear,rage,lo
ve
Using 4 label(s): grief, fear, rage, love

[*] Starting the web server at http://localhost:8080 ...
Open the app in your browser and start annotating!


[+] Saved 275 annotations to database SQLite
Dataset: harry_emoclass
Session ID: 2022-02-01_11-52-12

(nlp)
merle@DESKTOP-DQ77H55 MINGW64 ~/myrepo/NLP/harry_network_project (main|MERGING)
$
```



prodigy

PROJECT INFO

DATASET	harry_emoclass
RECIPE	textcat.manual
VIEW ID	choice

PROGRESS

THIS SESSION	272
TOTAL	272
	4%
ACCEPT	205
REJECT	6
IGNORE	61

HISTORY

Hello, Sir Nicholas. Have a n...	✓
Look, it's the Bloody Baron!	✓
Whoo-hoo-hoo!	⊘
Hello! How are you? Welco...	⊘
Potions. But everyone kno...	⊘
What's he teach?	⊘
Oh, that's Professor Snape, ...	⊘
Say, Percy, who's that teach...	✓

© 2017-2022 Explosion (Prodigy v1.11.6)

Dismal. Once again, my request to join the headless hunt has been denied.

- [x] grief 1
- [ ] fear 2
- [ ] rage 3
- [ ] love 4

Obstacles

Problem! Corpus too small to train a model? To perform the sentiment analysis, we wanted to try to train our own classification model using Prodigy as mentioned earlier. However, it quickly became clear that the data basis would not be sufficient to train a meaningful classification model.

Alternative? Using the Textblob SpaCy Pipeline Extension for Sentiment Analysis

What is Textblob?

Textblob is a pipeline extension for spaCy that performs sentiment analysis using the TextBlob library. Adding spaCyTextBlob to a spaCy nlp pipeline provides access to three extension attributes: polarity, subjectivity and assessment. Polarity is a float within the range [-1.0, 1.0], subjectivity is a float within the range [0.0, 1.0] where 0.0 is very objective and 1.0 is very subjective, and assessments is a list of polarity and subjectivity scores for the assessed tokens.

How to: Sentiment Analysis Attempt II

Step One: Preprocessing the Data and Connecting to the Database (see EXTRA: SQLite Queries)

```
import sqlite3 #database connection
#cursor-object connection to database
conn = sqlite3.connect('hp-dataset-with-relations.sqlite')
c = conn.cursor()
```

Step Two: Adding the TextBlob Extension

```
nlp.add_pipe("spacytextblob")
```

Step Three: Using TextBlob to assign sentiments

```
# define function to get the sentiments of a table
def calculate_sentiment(text):
  analysed = [nlp(line) for line in text]
  polarities = [l._.polarity for l in analysed]
  assessments = [l._.assessments for l in analysed]
  #return assessments -> see relevant words for assignment, example for presentation & reflexion
return polarities #returns sentiment between -1 and 1
```

How to: Topic Modeling - Attempt I (Gensim)

Step One: Preparing corpus by cleaning and tokenizing the movie scripts

```
# load file and create doc-object
with open("all_parts.txt", "r", encoding='utf-8') as tf:
  text = tf.readlines()
  text = ''.join(text).replace('\n',' ')
  text = ''.join(text)
  doc = nlp(text.lower())

# load and add stopwords
stopwords = STOP_WORDS  STOP_WORDS.add("'mr','ms','mrs','say','\s', 'said', 'says', 'saying', 'be', ' ', 'uh', 'oh', 'ah', 'ow','hey', 'whoa', 'ah
h', 'sir', 'ohh' 'e', 'eh', 'whew', 'ahhhh', 'ahhh'")
for word in STOP_WORDS:
  lexeme = nlp.vocab[word]
  lexeme.is_stop = True

# tokenize corpus and remove stopwords and punctuation
processed_corpus = [[token.text for token in doc if token.is_stop == False and  token.text.isalpha() == True and not token.is_punct and not token.
like_num]]
```

Step Two: Create Bag of Words (occurrence of each word)

```
# map words to ids
dictionary = corpora.Dictionary(processed_corpus)
# create bag of words
corpus = [dictionary.doc2bow(txt) for txt in processed_corpus]
```

Step Three: Run Model

```
# run model with 5 topics
lda_model = LdaModel(corpus=corpus, num_topics=5, id2word=dictionary)
# show topics
lda_model.show_topics()
```

Obstacles

Problem! When trying to implement our code for topic modeling, we encountered some obstacles. The most likely cause for the inconclusive output was either a mistake in our code or that the corpus was not large enough to train a model. Since we were not able to produce meaningful results (see below), we decided to take a different approach.

```
[(0,
  '0.021*"harry" + 0.015*"know" + 0.009*"potter" + 0.008*"think" + 0.008*"got" + 0.007*"come" + 0.007*"oh" + 0.006*"like" + 0.006*"right" +
0.006*"yes"'),
 (1,
  '0.021*"harry" + 0.011*"know" + 0.008*"think" + 0.008*"come" + 0.008*"potter" + 0.007*"oh" + 0.007*"right" + 0.006*"yes" + 0.006*"profess
or" + 0.006*"got"'),
 (2,
  '0.015*"harry" + 0.011*"know" + 0.010*"potter" + 0.009*"come" + 0.008*"like" + 0.007*"think" + 0.007*"right" + 0.007*"oh" + 0.007*"dumble
dore" + 0.006*"got"'),
 (3,
  '0.014*"harry" + 0.013*"know" + 0.008*"come" + 0.007*"potter" + 0.007*"like" + 0.007*"think" + 0.006*"yes" + 0.005*"right" + 0.005*"hermi
one" + 0.004*"oh"'),
 (4,
  '0.023*"harry" + 0.009*"know" + 0.009*"think" + 0.008*"potter" + 0.007*"come" + 0.007*"like" + 0.007*"right" + 0.006*"oh" + 0.006*"sir" +
0.005*"yes"')]
```

Alternative? Using the Python Counter to determine the frequency of words and visualizing the output with the help of WordClouds

What is Python's Counter and what are WordClouds?

Python's Counter (class `collections.Counter`) is a subclass that counts objects. We used the Counter to determine the frequency of words in our relationship-based tables. Identifying frequently occurring words enabled us to find possible recurring themes within the relationships. To visualize the results, we used WordClouds and passed the counted words to the provided function `generate_from_frequencies`. To have a more conclusive result we decides to create WordClouds that contained all words as well as only nouns, only adjectives and only verbs.

How to: Topic Modeling Attempt II

Step One: Loading and preparing the dialog of our movie scripts

```
# load tables
for table in views_names:
    sql = c.execute("SELECT dialog FROM " + table).fetchall()
    dialog = [i[0] for i in sql]
```

```
  # create doc-object without whitespaces and paragraphs and in lowercase as a single string
  text = ''.join(dialog).replace('\n',' ')
  text = ''.join(text)
  doc = nlp(text.lower())
```

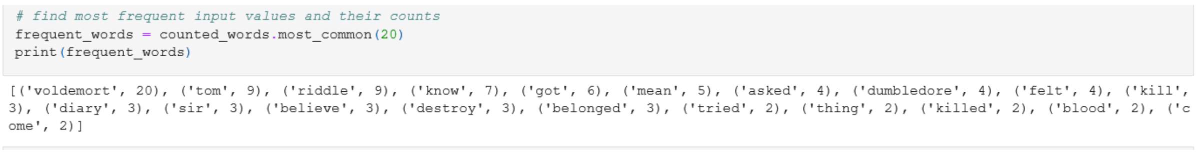
Step Two: Cleaning and Tokenizing the dialog and creating a list that contains tokenized words

```
# create list that contains tokenized words
    all_words = []
    for token in doc:
        if not token.is_stop and not token.is_punct and token.text.isalpha():
            all_words.append(token.text)
```

Step Three: Pass created list to Counter

```
# pass list to instance of Counter class
    counted_words = Counter(all_words)
```

Result:



```
# find most frequent input values and their counts
frequent_words = counted_words.most_common(20)
print(frequent_words)
```

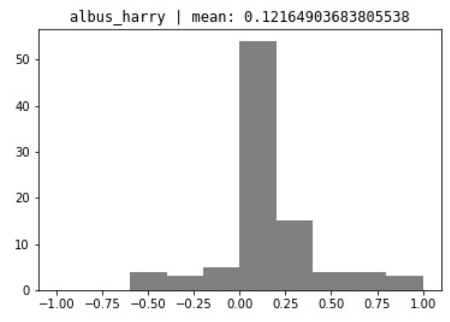
```
[('voldemort', 20), ('tom', 9), ('riddle', 9), ('know', 7), ('got', 6), ('mean', 5), ('asked', 4), ('dumbledore', 4), ('felt', 4), ('kill',
3), ('diary', 3), ('sir', 3), ('believe', 3), ('destroy', 3), ('belonged', 3), ('tried', 2), ('thing', 2), ('killed', 2), ('blood', 2), ('c
ome', 2)]
```

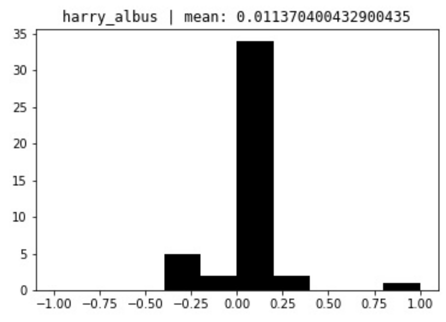
Potential problems: Not all words are recognized as the type of word they are supposed to be (e.g. harry is assigned as a adjective and a verb). Moreover, if you apply the Counter to a specific relationship-based table and filter for a specific word type (e.g. nouns), the results are very limited because many of the words occur only once, but are interpreted as frequent words.

IV Representation and Visualization

How to: Visualize Distributions

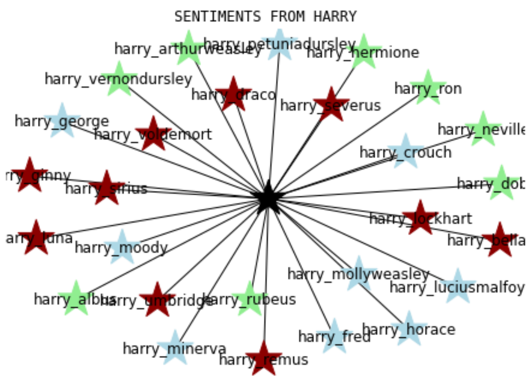
```
#call function with every view
for table in views_names:
  sql = c.execute("SELECT dialog FROM " + table).fetchall()
  dialog = [i[0] for i in sql] #iterate on dialogrows
  result = calculate_sentiment(dialog)
  narr = np.array(result)
  mean = np.mean(narr)

  #plot sentimentdistribution and mean as histogram for each table
  plt.title(table + ' | mean: ' + str(mean), family='monospace')
  x_axis = (-1, 1)
  if table.startswith('harry'):
    plt.hist(narr, range = x_axis, facecolor='black')
  else:
    plt.hist(narr, range = x_axis, facecolor='grey')

  #save distributions as .jpg files
  plt.savefig(table + '.jpg')
  #clear current figure
  plt.clf()
  #print(table + ': || ' + str(mean) + ' ||' + str(result))
```

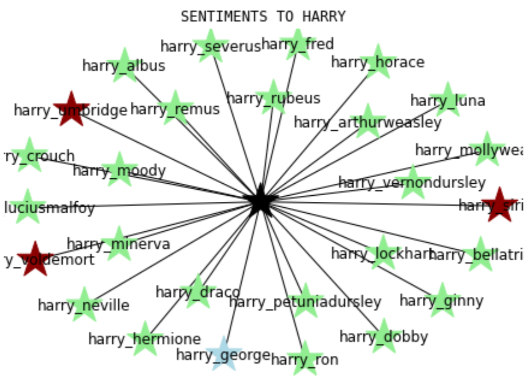




How to: Visualize Networks (networkx)

```
#from Harry
tab = pd.DataFrame(columns={'from': [], 'name':[], 'mean':[]})
#create dataframe
print(tab)

for table in views_names:
  sql = c.execute("SELECT dialog FROM " + table).fetchall()
  dialog = [i[0] for i in sql]
  result = calculate_sentiment(dialog)
  narr = np.array(result)
  mean = np.mean(narr)
  #ignore NaN fields
    if mean ==0 or mean <0 or mean >0:
    # =from Harry
      if table.startswith('harry'):
        #add tablenames and means to dataframe
        tab.loc[len(tab.index)]=["H", table, mean]

#use df to create networkgraph
G = nx.from_pandas_edgelist(tab, source='from', target='name', edge_attr='mean')
colors = ["black", "darkred", "lightgreen", "lightgreen", "darkred", "lightblue", "lightgreen", "darkred", "lightblue", "lightblue", "lightgreen",
"lightblue","darkred", "lightblue","darkred", "lightblue", "lightblue", "lightblue", "lightgreen", "lightblue", "darkred", "lightgreen", "lightgre
en", "darkred", "darkred", "darkred", "lightgreen", "darkred"]
nx.draw_spring(G, with_labels=True, node_shape='*', node_color = colors, node_size = 1000, edge_color='black', font_color="black")
plt.title("SENTIMENTS FROM HARRY")plt.show()


#to Harry
tab = pd.DataFrame(columns={'from': [], 'name':[], 'mean':[]})
print(tab)
for table in views_names:
  sql = c.execute("SELECT dialog FROM " + table).fetchall()
  dialog = [i[0] for i in sql]
  result = calculate_sentiment(dialog)
  narr = np.array(result)
  mean = np.mean(narr)
    if mean ==0 or mean <0 or mean >0:
      if table.endswith('_harry'):
        tab.loc[len(tab.index)]=["H", table, mean]

#use df to create networkgraph
G=nx.from_pandas_edgelist(tab, source='from', target='name', edge_attr='mean')
colors = ["black", "lightgreen", "lightgreen","lightgreen", "lightgreen", "lightgreen", "lightgreen", "lightgreen", "lightgreen", "lightblue", "li
ghtgreen", "lightgreen", "lightgreen","lightgreen", "lightgreen", "lightgreen","lightgreen","lightgreen","lightgreen","lightgreen","lightgreen","li
ghtgreen","lightgreen","lightgreen", "darkred", "darkred", "lightgreen", "darkred"]
nx.draw_spring(G, with_labels=True, node_shape='*', node_color = colors, node_size = 1000, edge_color='black', font_color="black")
plt.show()
```



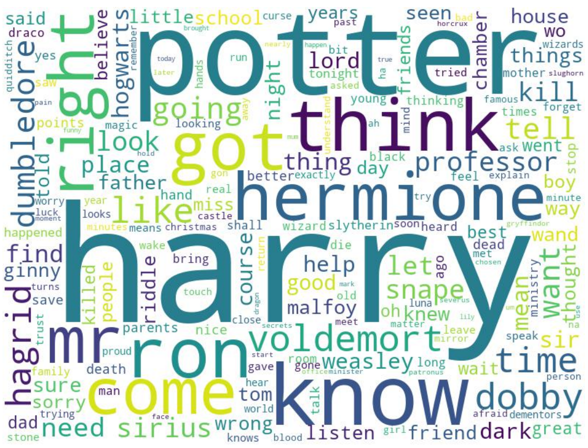


To visualize the Harry Potter network, i.e. the sentiments directed towards and away from Harry, we decided to use different colors for the nodes: blue = neutral, red = negative, green = positive. For more meaningful visualization, it would have been useful to weight the edges, and for a more complete network, to analyze all relationships (not only those concerning Harry).

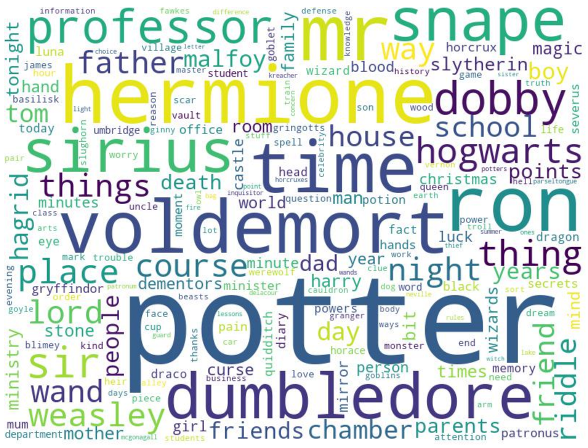
How to: Visualize topics (wordclouds)

```
# call function with every view
for table in views_names:
    sql = c.execute("SELECT dialog FROM " + table).fetchall()
    dialog = [i[0] for i in sql]

    # create doc-object without whitespaces and paragraphs and in lowercase as a single string
    text = ''.join(dialog).replace('\n',' ')
    text = ''.join(text)
    doc = nlp(text.lower())
    # create list that contains tokenized words
    all_words = []
    for token in doc:
        if not token.is_stop and not token.is_punct and token.text.isalpha():
            all_words.append(token.text)

    # pass list to instance of Counter class
    counted_words = Counter(all_words)

    # check length
    if len(all_words) > 0:
        # create wordcloud object and call the generate_from_frequencies function
        if table.startswith('harry'):
            wc = WordCloud(background_color="black", width=800,height=600)
            wc.generate_from_frequencies(counted_words)
            plt.imshow(wc, interpolation="bilinear")
            plt.axis("off")
        else:
            wc = WordCloud(background_color="white", width=800,height=600)
            wc.generate_from_frequencies(counted_words)
            plt.imshow(wc, interpolation="bilinear")
            plt.axis("off")

        wc.to_file(table + '_allwords.jpg')
```

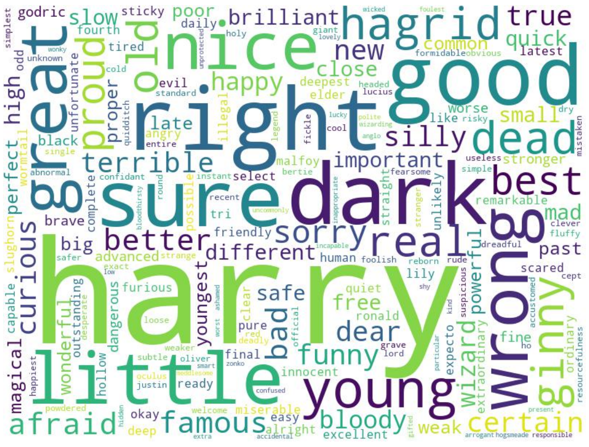
Different outputs:



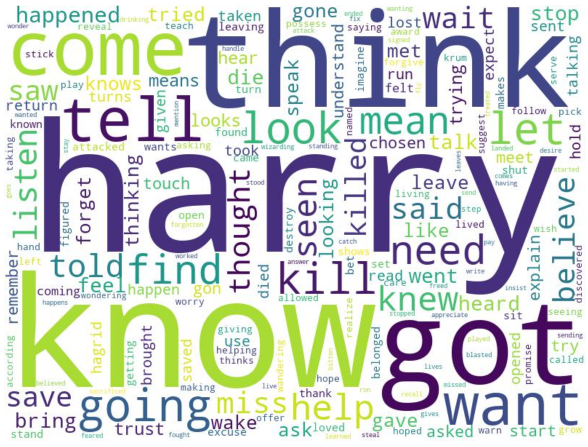
all_words



nouns



adjectives



verbs

V Reflection and Discussion

Documentation of our steps

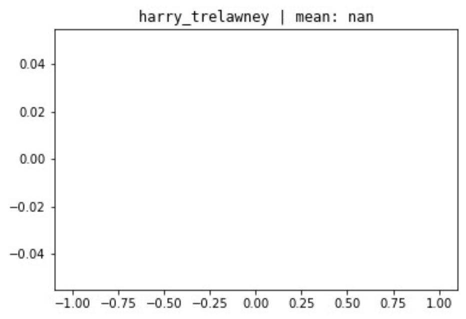
1. Get Inspiration (Data, Purpose)
2. Create GitHub Project
3. Create Project Schedule (Milestones)

4. Hands on Data (.csv to .sqlite?)
5. Assign fundamentals (Merle: network analysis, sentiment analysis / Teresa: NER, topic modeling)
6. Decision-Making: use Prodigy to train models?
7. Create plain text data using SQLite DBBrowser
8. Use SQLite within JupyterNotebook to shorten process
9. Loop over dialogdata (SQL views) to assign sentiments
10. Visualize sentiment distribution for further data understanding
11. Topic Modeling with Gensim (alternative: Counter)
12. Visualisation: networkx & wordclouds?
13. Complete jupyter notebook (add CodeNotebook)
14. Complete GIT Repository (add readme, visualizations, data, ...)
15. Create Presentation (PP)
16. Write Paper (structure, assign topics, etc.)

Obstacles

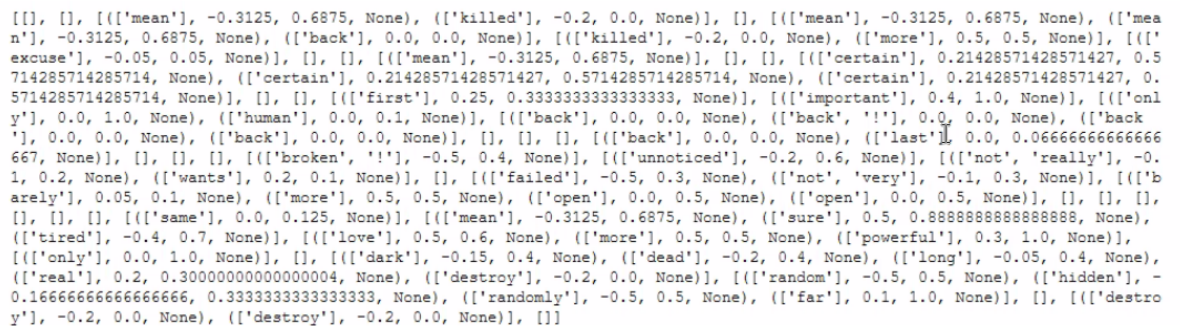
Amount of Data

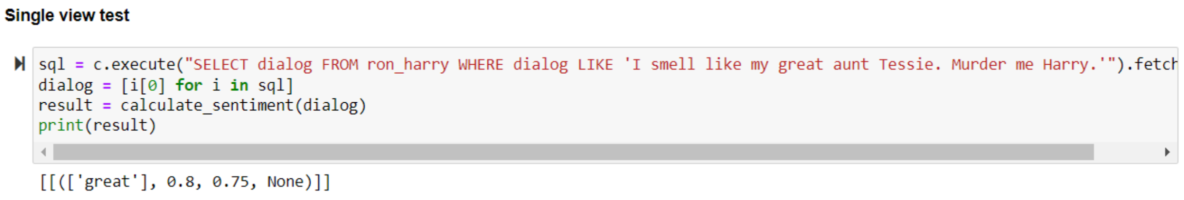
→ different approach on data selection (SQL-queries taking further dialogrows into account)



Sentiment Assessment

→ different model


```
[[], [], [(['mean'], -0.3125, 0.6875, None), (['killed'], -0.2, 0.0, None)], [], [(['mean'], -0.3125, 0.6875, None), (['mea
n'], -0.3125, 0.6875, None), (['back'], 0.0, 0.0, None)], [(['killed'], -0.2, 0.0, None), (['more'], 0.5, 0.5, None)], [(['
excuse'], -0.05, 0.05, None)], [], [], [(['mean'], -0.3125, 0.6875, None)], [], [], [(['certain'], 0.21428571428571427, 0.5
714285714285714, None), (['certain'], 0.21428571428571427, 0.5714285714285714, None), (['certain'], 0.21428571428571427, 0.
5714285714285714, None)], [], [], [(['first'], 0.25, 0.3333333333333333, None)], [(['important'], 0.4, 1.0, None)], [(['onl
y'], 0.0, 1.0, None), (['human'], 0.0, 0.1, None)], [(['back'], 0.0, 0.0, None), (['back', '!'], 0.0, 0.0, None), (['back
'], 0.0, 0.0, None), (['back'], 0.0, 0.0, None)], [], [], [], [(['back'], 0.0, 0.0, None), (['last'], 0.0, 0.06666666666666
667, None)], [], [], [], [(['broken', '!'], -0.5, 0.4, None)], [(['unnoticed'], -0.2, 0.6, None)], [(['not', 'really'], -0.
1, 0.2, None), (['wants'], 0.2, 0.1, None)], [], [(['failed'], -0.5, 0.3, None), (['not', 'very'], -0.1, 0.3, None)], [(['b
arely'], 0.05, 0.1, None), (['more'], 0.5, 0.5, None), (['open'], 0.0, 0.5, None), (['open'], 0.0, 0.5, None)], [], [], [],
[], [], [], [(['same'], 0.0, 0.125, None)], [(['mean'], -0.3125, 0.6875, None), (['sure'], 0.5, 0.8888888888888888, None),
(['tired'], -0.4, 0.7, None)], [(['love'], 0.5, 0.6, None), (['more'], 0.5, 0.5, None), (['powerful'], 0.3, 1.0, None)],
[(['only'], 0.0, 1.0, None)], [], [(['dark'], -0.15, 0.4, None), (['dead'], -0.2, 0.4, None), (['long'], -0.05, 0.4, None),
(['real'], 0.2, 0.30000000000000004, None), (['destroy'], -0.2, 0.0, None)], [(['random'], -0.5, 0.5, None), (['hidden'], -
0.16666666666666666, 0.3333333333333333, None), (['randomly'], -0.5, 0.5, None), (['far'], 0.1, 1.0, None)], [], [(['destro
y'], -0.2, 0.0, None), (['destroy'], -0.2, 0.0, None)], []]
```

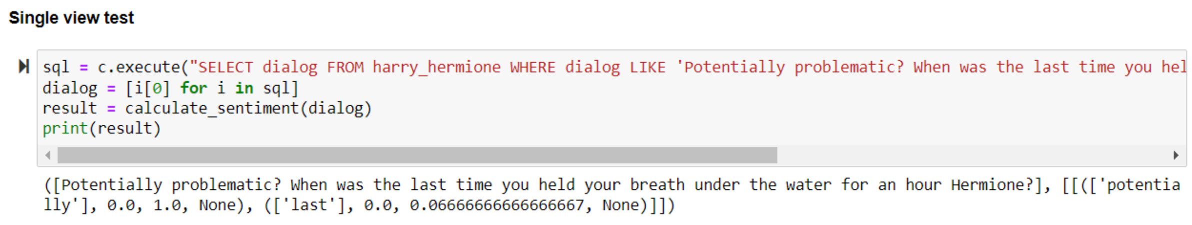

Single view test

```
sql = c.execute("SELECT dialog FROM ron_harry WHERE dialog LIKE 'I smell like my great aunt Tessie. Murder me Harry.'").fetch
dialog = [i[0] for i in sql]
result = calculate_sentiment(dialog)
print(result)
```

```
[[(['great'], 0.8, 0.75, None)]]
```

Sarcasm

→ sarcasm & sentiment analysis



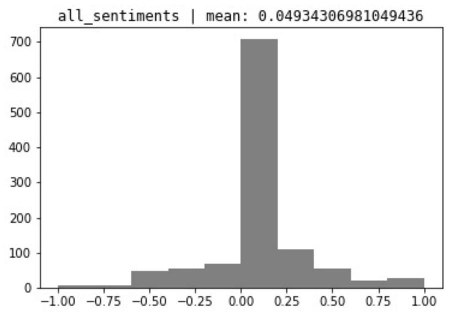
Single view test

```
sql = c.execute("SELECT dialog FROM harry_hermione WHERE dialog LIKE 'Potentially problematic? When was the last time you hel
dialog = [i[0] for i in sql]
result = calculate_sentiment(dialog)
print(result)
```

```
([Potentially problematic? When was the last time you held your breath under the water for an hour Hermione?], [[(['potentia
lly'], 0.0, 1.0, None), (['last'], 0.0, 0.06666666666666667, None)]])
```

Distributionrange

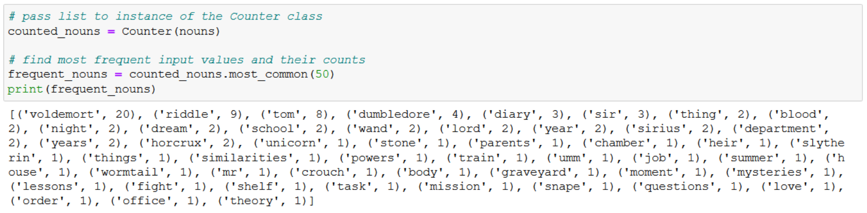
→ relative distribution



Counter

Words with a frequency of 1 that are interpreted as frequent words due to the limted data

→ more extensive corpus or different approach to data selection



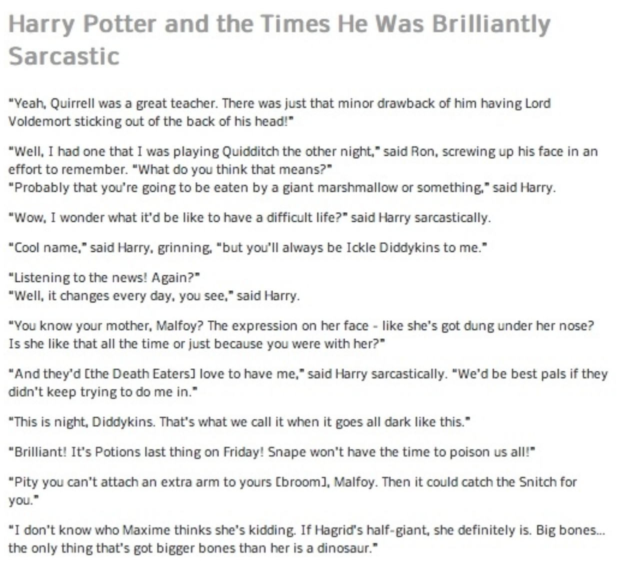
```
# pass list to instance of the Counter class
counted_nouns = Counter(nouns)

# find most frequent input values and their counts
frequent_nouns = counted_nouns.most_common(50)
print(frequent_nouns)
```

```
[('voldemort', 20), ('riddle', 9), ('tom', 8), ('dumbledore', 4), ('diary', 3), ('sir', 3), ('thing', 2), ('blood',
2), ('night', 2), ('dream', 2), ('school', 2), ('wand', 2), ('lord', 2), ('year', 2), ('sirius', 2), ('department',
2), ('years', 2), ('horcrux', 2), ('unicorn', 1), ('stone', 1), ('parents', 1), ('chamber', 1), ('heir', 1), ('slythe
rin', 1), ('things', 1), ('similarities', 1), ('powers', 1), ('train', 1), ('umm', 1), ('job', 1), ('summer', 1), ('h
ouse', 1), ('wormtail', 1), ('mr', 1), ('crouch', 1), ('body', 1), ('graveyard', 1), ('moment', 1), ('mysteries', 1),
('lessons', 1), ('fight', 1), ('shelf', 1), ('task', 1), ('mission', 1), ('snape', 1), ('questions', 1), ('love', 1),
('order', 1), ('office', 1), ('theory', 1)]
```

Future Options

Data and Analysis

1. Use different SQL Queries for data selection
2. Train specific model
3. Use different analyse tool (extended sentiments: classified emotions instead of polarity)
4. Take time-factor into account
5. Take sarcasm into account



Harry Potter and the Times He Was Brilliantly Sarcastic

"Yeah, Quirrell was a great teacher. There was just that minor drawback of him having Lord Voldemort sticking out of the back of his head!"

"Well, I had one that I was playing Quidditch the other night," said Ron, screwing up his face in an effort to remember. "What do you think that means?"
"Probably that you're going to be eaten by a giant marshmallow or something," said Harry.

"Wow, I wonder what it'd be like to have a difficult life?" said Harry sarcastically.

"Cool name," said Harry, grinning, "but you'll always be Ickle Diddykins to me."

"Listening to the news! Again?"
"Well, it changes every day, you see," said Harry.

"You know your mother, Malfoy? The expression on her face – like she's got dung under her nose? Is she like that all the time or just because you were with her?"

"And they'd [the Death Eaters] love to have me," said Harry sarcastically. "We'd be best pals if they didn't keep trying to do me in."

"This is night, Diddykins. That's what we call it when it goes all dark like this."

"Brilliant! It's Potions last thing on Friday! Snape won't have the time to poison us all!"

"Pity you can't attach an extra arm to yours [broom], Malfoy. Then it could catch the Snitch for you."

"I don't know who Maxime thinks she's kidding. If Hagrid's half-giant, she definitely is. Big bones... the only thing that's got bigger bones than her is a dinosaur."

Visualization

1. Optimize visualization of sentiment distribution (different diagramtype, relative sentiments, etc.)
2. Optimize visualization of sentiment network (extended, edge-weights by sentiment-polarity, etc.)
3. Optimize visualization of topics (more meaningful output)

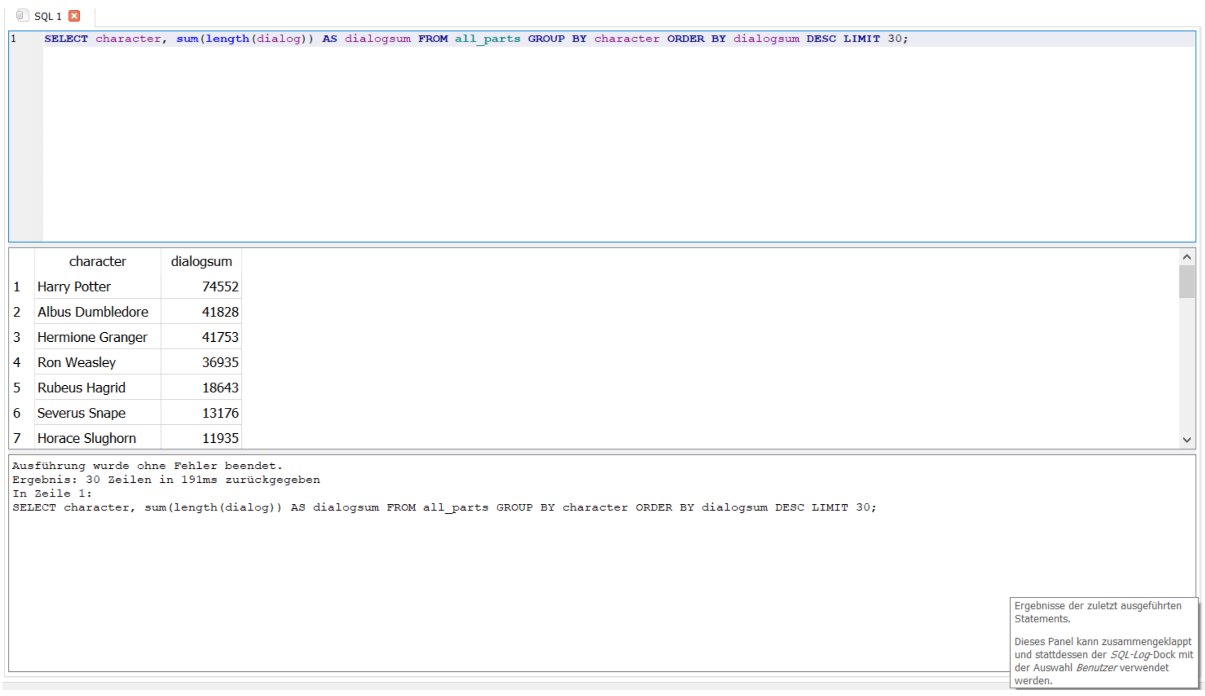
Hands on Data - SQLite Queries

Task: Create plain text data to use for NLP

First Step: Character Selection (factor: sum of spoken word)

How much do the character speak by length of dialog?

- SELECT DISTINCT character from all_parts; 175 results
- SELECT character, count(dialog) FROM all_parts GROUP BY character ORDER BY count(dialog)
- SELECT character, sum(length(dialog)) AS dialogsum FROM all_parts GROUP BY character ORDER BY dialogsum DESC LIMIT 30



```
SELECT character, sum(length(dialog)) AS dialogsum FROM all_parts GROUP BY character ORDER BY dialogsum DESC LIMIT 30;
```

	character	dialogsum
1	Harry Potter	74552
2	Albus Dumbledore	41828
3	Hermione Granger	41753
4	Ron Weasley	36935
5	Rubeus Hagrid	18643
6	Severus Snape	13176
7	Horace Slughorn	11935

```
Ausführung wurde ohne Fehler beendet.
Ergebnis: 30 Zeilen in 191ms zurückgegeben
In Zeile 1:
SELECT character, sum(length(dialog)) AS dialogsum FROM all_parts GROUP BY character ORDER BY dialogsum DESC LIMIT 30;
```

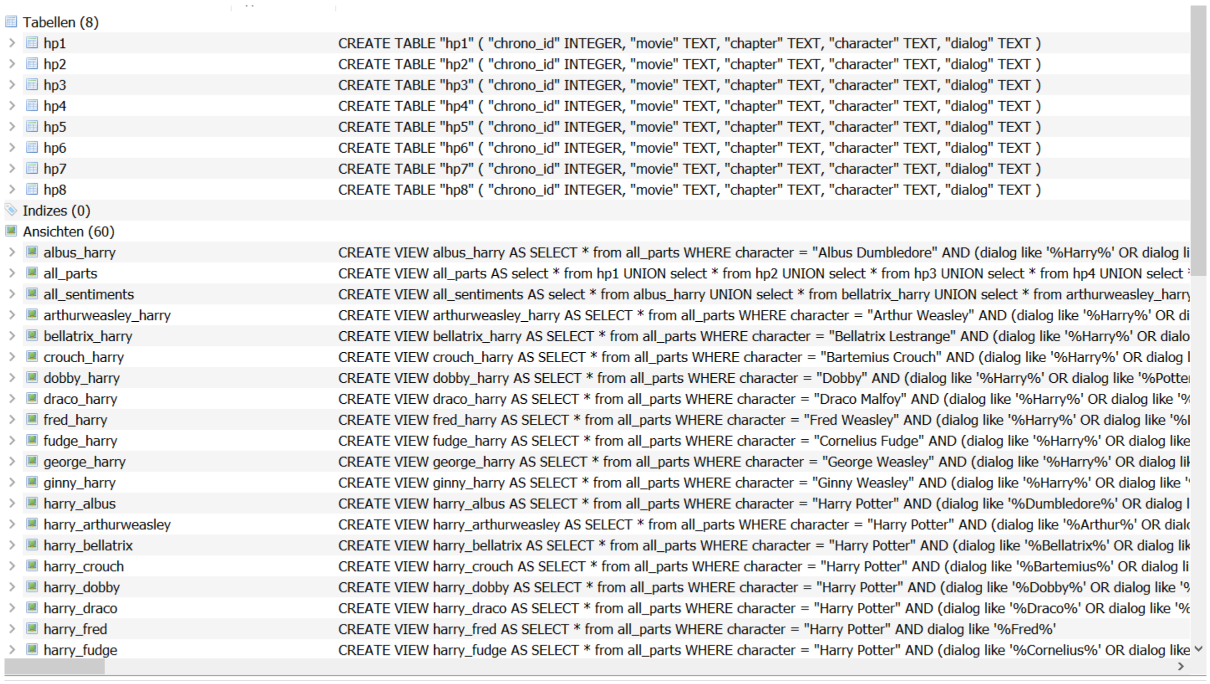
Ergebnisse der zuletzt ausgeführten Statements.

Dieses Panel kann zusammengeklappt und stattdessen der *SQL-Log*-Dock mit der Auswahl *Benutzer* verwendet werden.

Second Step: Data Preprocessing and Relationship-Building (factor: addressed by name)

Which dialogfields contain the previously selected characters being addressed by Harry or addressing Harry by name?

- CREATE VIEW harry_albus AS SELECT * from all_parts WHERE character = "Harry Potter" AND dialog like '%Dumbledore%' OR dialog like '%Albus%';
- CREATE VIEW albus_harry AS SELECT * from all_parts WHERE character = "Albus Dumbledore" AND dialog like '%Harry%' OR dialog like '%Potter%' OR dialog like '%chosen one%';



```
Tabellen (8)
> hp1                    CREATE TABLE "hp1" ( "chrono_id" INTEGER, "movie" TEXT, "chapter" TEXT, "character" TEXT, "dialog" TEXT )
> hp2                    CREATE TABLE "hp2" ( "chrono_id" INTEGER, "movie" TEXT, "chapter" TEXT, "character" TEXT, "dialog" TEXT )
> hp3                    CREATE TABLE "hp3" ( "chrono_id" INTEGER, "movie" TEXT, "chapter" TEXT, "character" TEXT, "dialog" TEXT )
> hp4                    CREATE TABLE "hp4" ( "chrono_id" INTEGER, "movie" TEXT, "chapter" TEXT, "character" TEXT, "dialog" TEXT )
> hp5                    CREATE TABLE "hp5" ( "chrono_id" INTEGER, "movie" TEXT, "chapter" TEXT, "character" TEXT, "dialog" TEXT )
> hp6                    CREATE TABLE "hp6" ( "chrono_id" INTEGER, "movie" TEXT, "chapter" TEXT, "character" TEXT, "dialog" TEXT )
> hp7                    CREATE TABLE "hp7" ( "chrono_id" INTEGER, "movie" TEXT, "chapter" TEXT, "character" TEXT, "dialog" TEXT )
> hp8                    CREATE TABLE "hp8" ( "chrono_id" INTEGER, "movie" TEXT, "chapter" TEXT, "character" TEXT, "dialog" TEXT )
Indizes (0)
Ansichten (60)
> albus_harry            CREATE VIEW albus_harry AS SELECT * from all_parts WHERE character = "Albus Dumbledore" AND (dialog like '%Harry%' OR dialog li
> all_parts              CREATE VIEW all_parts AS select * from hp1 UNION select * from hp2 UNION select * from hp3 UNION select * from hp4 UNION select
> all_sentiments         CREATE VIEW all_sentiments AS select * from albus_harry UNION select * from bellatrix_harry UNION select * from arthurweasley_harry
> arthurweasley_harry    CREATE VIEW arthurweasley_harry AS SELECT * from all_parts WHERE character = "Arthur Weasley" AND (dialog like '%Harry%' OR di
> bellatrix_harry        CREATE VIEW bellatrix_harry AS SELECT * from all_parts WHERE character = "Bellatrix Lestrange" AND (dialog like '%Harry%' OR dialo
> crouch_harry           CREATE VIEW crouch_harry AS SELECT * from all_parts WHERE character = "Bartemius Crouch" AND (dialog like '%Harry%' OR dialog l
> dobby_harry            CREATE VIEW dobby_harry AS SELECT * from all_parts WHERE character = "Dobby" AND (dialog like '%Harry%' OR dialog like '%Potte
> draco_harry            CREATE VIEW draco_harry AS SELECT * from all_parts WHERE character = "Draco Malfoy" AND (dialog like '%Harry%' OR dialog like '%
> fred_harry             CREATE VIEW fred_harry AS SELECT * from all_parts WHERE character = "Fred Weasley" AND (dialog like '%Harry%' OR dialog like '%I
> fudge_harry            CREATE VIEW fudge_harry AS SELECT * from all_parts WHERE character = "Cornelius Fudge" AND (dialog like '%Harry%' OR dialog like
> george_harry           CREATE VIEW george_harry AS SELECT * from all_parts WHERE character = "George Weasley" AND (dialog like '%Harry%' OR dialog lik
> ginny_harry            CREATE VIEW ginny_harry AS SELECT * from all_parts WHERE character = "Ginny Weasley" AND (dialog like '%Harry%' OR dialog like '
> harry_albus            CREATE VIEW harry_albus AS SELECT * from all_parts WHERE character = "Harry Potter" AND (dialog like '%Dumbledore%' OR dialog l
> harry_arthurweasley    CREATE VIEW harry_arthurweasley AS SELECT * from all_parts WHERE character = "Harry Potter" AND (dialog like '%Arthur%' OR dialo
> harry_bellatrix        CREATE VIEW harry_bellatrix AS SELECT * from all_parts WHERE character = "Harry Potter" AND (dialog like '%Bellatrix%' OR dialog lik
> harry_crouch           CREATE VIEW harry_crouch AS SELECT * from all_parts WHERE character = "Harry Potter" AND (dialog like '%Bartemius%' OR dialog li
> harry_dobby            CREATE VIEW harry_dobby AS SELECT * from all_parts WHERE character = "Harry Potter" AND (dialog like '%Dobby%' OR dialog like '%
> harry_draco            CREATE VIEW harry_draco AS SELECT * from all_parts WHERE character = "Harry Potter" AND (dialog like '%Draco%' OR dialog like '%
> harry_fred             CREATE VIEW harry_fred AS SELECT * from all_parts WHERE character = "Harry Potter" AND dialog like '%Fred%'
> harry_fudge            CREATE VIEW harry_fudge AS SELECT * from all_parts WHERE character = "Harry Potter" AND (dialog like '%Cornelius%' OR dialog like
```

EXTRA: What is Named Entity Recognition (NER) & Basic Python NER with SpaCy (unused documentation)

Named Entity Recognition (NER) is the task of automatically finding and tagging entities in a text and of identifying and catigorizing them. Possible entities are, for example: persons, locations, dates.

When working with NER, it is important to tell the model how to recognise the words that are to be marked as named entities. For this purpose, a pattern for labelling the data is defined, with which the model is trained to enable the most accurate recognition possible.

Bibliography

Antonio M. Chiesi: *Network Analysis* in: *International Encyclopedia of the Social & Behavioral Sciences* Edited by James D. Wright (2015: 518-523)

Jan Horstmann: *Topic Modeling* on: https://fortext.net/routinen/methoden/topic-modeling (January 15, 2018)

S. Oberbichler & E. Pfanzelter: *Topic-specific corpus building: A step towards a representative newspaper corpus on the topic of return migration using text mining methods.* in: Journal of Digital History, jdh001 https://journalofdigitalhistory.org/en/article/4yxHGiqXYRbX (2021)

Mareike Schumacher: *Netzwerkanalyse* on: https://fortext.net/routinen/methoden/netzwerkanalyse (November 12, 2018)

Mareike Schumacher: *Named Entity Recognition (NER)* on: https://fortext.net/routinen/methoden/named-entity-recognition-ner (May 17, 2018)

James, William: *The Principles of Psychology*. Cosimo (1 April 2007) Inc. ISBN 9781602063136. Retrieved January 28, 2022 – via Google Books.

Weblinks

https://towardsdatascience.com/my-absolute-go-to-for-sentiment-analysis-textblob-3ac3a11d524

https://www.sciencedirect.com/science/article/pii/B9780128044124000073

https://towardsdatascience.com/building-a-topic-modeling-pipeline-with-spacy-and-gensim-c5dc03ffc619

https://medium.com/analytics-vidhya/sentiment-analysis-using-textblob-ecaaf0373dff

https://towardsdatascience.com/my-absolute-go-to-for-sentiment-analysis-textblob-3ac3a11d524

https://stackabuse.com/sentiment-analysis-in-python-with-textblob/

https://prodi.gy/docs/text-classification