Mila Miroslavova Marcheva

Neural language modelling for differentiating between texts of imagined versus recalled stories

Computer Science Tripos – Part II

Gonville & Caius College

14 May 2021

Declaration

Acknowledgements

I would like to thank my supervisors, Prof. Paula Buttery and Dr. Andrew Caines, for their continuous support through all stages of the project. Without their help I would not have been able to find a topic that I so thoroughly enjoyed and is simultaneously a viable project idea. Also, many thanks are in order for my DoS, Dr. Tim Jones, for his understanding, responsiveness and guidance during the last 3 years. Finally, I want to thank my partner, JMN, who stoically listened about the highs and lows of the project for months, reviewed many dissertation drafts and just overall was my rock in this challenging year.

Proforma

Candidate number:	**2423B**
Project title:	**Neural language modelling for differentiating between texts of imagined versus recalled stories**
Examination:	**Computer Science Tripos – Part II 2021**
Word count:	**11,991**[1]
Lines of code:	**1,392**[2]
Project originator:	The dissertation author
Supervisors:	Prof. Paula Buttery and Dr. Andrew Caines

Original Aims of the Project

I set out to reimplement aspects of the following paper presented at ACL 2020: "Recollection versus Imagination: Exploring Human Memory and Cognition via Neural Language Models" [1]. I investigated the differences between imagined and recalled stories in the "Hippocorpus" dataset. My main goals were to use lexicon-based measures and the newly established measurement of narrative flow to distinguish between the two categories. Possible extensions were relating the measurements to episodic and semantic memory and/or analysing a news corpus to see if the explored measurements can differentiate between fake and reliable news.

Work Completed

Throughout the project best software engineering practices were employed to build a state-of-the-art NLP system. The selected aspects from the ACL 2020 paper by Sap et al. [1] were implemented and the Hippocorpus dataset was analysed as planned using lexicon-based measurements and narrative flow. Optimisations of the code utilising the neural language model GPT2 were implemented to improve efficiency. All analysis results were statistically evaluated using paired t-tests and compared with the results of Sap et al. [1]. The fake news extension was completed.

Special Difficulties

None

[1]Overleaf word count
[2]Lines of code counted using `https://plugins.jetbrains.com/plugin/4509-statistic`

Contents

Chapter 1

Introduction

This project investigates the differences between imagined and recalled stories. The stories are taken from the Hippocorpus [1] dataset.

1.1 Motivation

Storytelling has been part of human culture throughout history [3]. Defining what a story is and especially distinguishing it from other types of discourse is a challenging task without a unified solution [4]. However, one might be able to gain some insight through the analysis of the function of storytelling [4]. Stories started as a mean of sharing and preserving cultural heritage, but this also evolved into giving instructions and explaining the world, and has most recently come to include making sense of personal experiences [4]. More specifically, this last type of story can be composed of events obtained in juxtaposing ways: by recalling or by imagining them. Recalling and imagining an event both involve accessing knowledge about it from memory, but experimental work has shown that different neural structures are involved depending on the origin of the event, whether it is a personal experience or generic knowledge [5]. Since language is a manifestation of human cognition, could it be that the differing cognitive processes of recollection and imagination of an event leave distinctive artefacts in language? Investigating this makes for a challenging natural language processing task in which traces of cognitive processes are sought using classical and neural language modelling methods.

1.2 Original and related work

This project is inspired by "Recollection versus Imagination: Exploring Human Memory and Cognition via Neural Language Models" by Sap et al. [1], which was presented at the Annual Conference of the Association for Computational Linguistics, 2020 (further referred to as ACL 2020). The high-level ideas and methodology in the project are based on the original work, but their implementation is the author's interpretation. The goal of the project is to replicate the results from the paper by Sap et al. [1] and to explain any differences in outcomes in terms of specific implementation choices.

1.2.1 Cognition

Tulving's explanation of long-term memory states that recalling and imagining a story are based on different cognitive process. He establishes a distinction between two types of declarative long-term memory: episodic and semantic [6]. In accordance with Tulving's theory, semantic memory, which is responsible for storing common-sense ideas and facts (e.g. "the birth of a child is a joyous event"), is used for imagining, whereas episodic memory, which is responsible for storing personal facts (e.g. "I was having a chocolate-chip muffin alongside my coffee when I saw the smoke and heard the sirens"), is used to recall a personally experienced situation. Since imagination and recollection are also proven experimentally to be based on different cognitive processes [5], it is reasonable to expect that the manifestation of those processes, in this case within storytelling, will carry traces of those differences.
Using natural language processing techniques, the paper by Sap et. al [1] explores the following hypotheses:

- imagined stories are based on general knowledge, whereas recalled stories are going to be more concrete and contain personal knowledge;
- imagined stories flow more linearly, since their content is built on commonsense relations of events.

Lexicon-based measurements are investigated, including word-count analysis using the Linguistic Inquiry and Word Count (LIWC) tool [7]; and concreteness analysis is carried out using a lexicon by Brysbaert et al. [2]. The paper introduces a narrative flow measurement which is calculated using single-word probabilities from a neural language model, more specifically the Generative Pre-trained Transformer (GPT) [8]. Neural language models are also used for realis (concrete) events tagging, for the purpose of which the Bidirectional Encoder Representations from Transformers (BERT) [9] is trained on the literary event detection corpus by Sims et al. [10]. Finally, knowledge graphs are used to measure the amount of commonsense knowledge stored in semantic memory, using the Atlas of Machine Commonsense for If-Then Reasoning (ATOMIC) knowledge graph [11].
The above hypotheses are confirmed by Sap et al. [1] and the goal of this project is a confirmatory replication of parts of the original study.

Chapter 2

Preparation

The preparation for the project involved researching the data, resources and tools to be used and designing a high level overview for each of the software modules.

2.1 Data, resources and tools

2.1.1 Hippocorpus

A dataset called the "Hippocorpus"[1] was released alongside the paper by Sap et al. [1]. It contains 6,854 stories of length between 15 and 25 sentences in English that are labelled as either imagined, recalled, or retold. The dataset was compiled using U.S. based workers on Amazon Mechanical Turk. The process of data collection involved several stages: initially a worker would write a recalled story and provide a short summary for it, then another worker would be provided that summary as a prompt to write an imagined story, and finally some of the workers from the first cohort of recalled stories would be called back to write a retold story of their first entry. The recalled and imagined stories from the Hippocorpus are analysed throughout this project. The retold stories are not investigated. The original dataset's 'README' file, which contains information about the content structure can be found in the Appendix (see A.1).

2.1.2 Concreteness lexicon

In this work, concreteness refers to Brysbaert, Warriner, and Kuperman's concreteness lexicon for English [2], which is used to calculate the average concreteness analysis for each story. The lexicon contains roughly 40,000 single-word and two-word expressions associated with a mean concreteness score and some more metadata (see section A.2 of the Appendix). Lexicon entries were created from averaged human judgements; 20-30 participants rated each word on a 1 (most abstract) to 5 (most concrete) scale; participants were recruited using Amazon Mechanical Turk. In this project the average concreteness of words or bigrams (two-word phrases) was used to calculate the average concreteness score of the stories in the Hippocorpus.

[1] `https://msropendata.com/datasets/0a83fb6f-a759-4a17-aaa2-fbac84577318`

Limitations

Although Brysbaert et al.'s lexicon provides an easy approach to concreteness analysis, it is not necessarily the most accurate measurement of this psycholinguistic variable. The primary drawback can be spotted when looking at the words with mean concreteness between 2 and 4 and a high standard deviation. For those words, the participants' agreement was low: there were ratings on either ends of the spectrum, but the mean without the standard deviation fails to show that disagreement and instead leaves the impression of agreement around some middling score [12].
Another issue with Brysbaert et al.'s lexicon concerns inflected forms. It is supposed to be a "reference list of generally known English lemmas", and is not meant to include inflected forms or proper names [2]. To take the verb 'be' as an example: 'am' and 'are' are inflected forms of 'be', so they should not be contained in the lexicon, but they are. Similarly, this is the case for other irregular verbs such as 'go' and its inflected forms. When choosing how to preprocess the data, one should bear this peculiarity in mind: a common lemmatiser will reduce those inflected forms to their canonical form, but if the canonical form and the irregular inflected form have different scores, this will affect the overall score of the story. However, lemmatisation of the story is a useful and necessary step for this task, as most cases of inflection involve regular inflected forms, which have to be reduced to their canonical form because only the canonical form can be found in the lexicon.
For the reasons stated above the concreteness lexicon analysis score should be approached with some skepticism.

2.1.3 LIWC

The Linguistic Inquiry and Word Count (LIWC) is a language analysis tool that aggregates the psychological types of words in a text. To do this it uses pre-set dictionaries of words associated with certain psychological states or processes and investigates the words in a text one by one to find whether they belong to some of the categories [7]. A word can belong to multiple categories simultaneously and, if that is the case, the count for each category is increased. LIWC offers insight on various psychological dimensions, but only a few of those are relevant to this project, as discussed in the paper by Sap et al. [1]. Those dimensions are:

- cogproc: cognitive process words, such as 'cause', 'know', 'ought'
- posemo: positive emotion words, such as 'love', 'nice', 'sweet'
- negemo: negative emotion words, such as 'ugly', 'nasty', 'hurt'
- i: first person singular pronouns, such as 'I', 'me', 'mine'
- analytic: analytical thinking
- tone: emotional tone

The analytic and tone variables are 'summary' variables, which means they are an aggregation of multiple parameters, rather than a count of one specific category of words.

2.2 Overview of the preprocessing module

The original Hippocoprus dataset needs to be reshaped, which means that only the stories that will be used in the analyses need to remain. The reshaping also needs to remove any recalled stories without an imagined pairing, due to the evaluation with paired t-tests. The LIWC module requires no preprocessing. The concreteness analysis module requires that the stories are tokenised, lowercased and lemmatised and the punctuation and numerical characters are cleaned, which is performed using the `spaCy` Python library for NLP [13]. The preprocessing for the narrative flow analysis is performed using the GPT2Tokenizer from the `transformers` library.

2.3 Overview of the different analysis modules

In order to carry out the comparison of story types three analysis modules were created. Two of them are lexicon-base, LIWC and concreteness, and the last is neural-language-model-based, narrative flow.

2.3.1 LIWC analysis

The LIWC module incorporates the scores for the set of outlined psychological dimensions produced by the LIWC tool [7] into the main data file.

2.3.2 Concreteness lexicon analysis

The concreteness lexicon analysis module produces an average concreteness score of each story using Brysbaert et al.'s concreteness lexicon [2].
After the stories have been preprocessed, the average concreteness of each story is calculated. Since the paper by Sap et al. [1] does not provide detail on how to calculate the score, it is assumed the average concreteness of a story is equal to the mean of the concreteness scores of all tokens in the story that can be found in the lexicon [2]. Special care was taken to also allow for bigrams to be found in the story and scored accordingly, as the lexicon contains a number of two-word expressions. Please note the limitations of the concreteness analysis approach described in section 2.1.2.

Note on spaCy and pronouns

The built-in `spaCy` lemmatiser reduces all pronouns to a synthetic token '-pron-'. However, pronouns still can have a concreteness related to them, so they should be kept in the final tokens list and a workaround that allows using the `spaCy` lemmatiser whilst keeping the pronouns was implemented.

Note on spelling and normalisation

The concreteness lexicon is compiled using U.S.-based raters of the words and only the American English spelling of words is included in the lexicon [2]. This was verified through manual inspection as well. The Hippocorpus is also reported to have been compiled using U.S.-based workers [1]. Therefore, the American variety of English is used in both cases and there is no need for normalisation.

2.3.3 Narrative flow analysis

The narrative flow analysis module evaluates the average narrative flow score (also known as 'linearity') of each story.
The narrative flow score was introduced by Sap et al. [1]. It is related to the likelihood of a sentence appearing in its context. The score calculation involves two generative models: the bag model and the chain model. In the bag model the sentences are drawn independently of each other based on a context that includes only the predefined topic, which can be the short summary of the story, the main event, or an empty sequence. In the chain model, the context for each sentence is the predefined topic prepended to the sequence of all previous sentences.

Theoretical background

The narrative flow measurement is based on the probabilities of sentences appearing in their context based on two models, the bag and chain model, illustrated in figures 2.1 and 2.2.

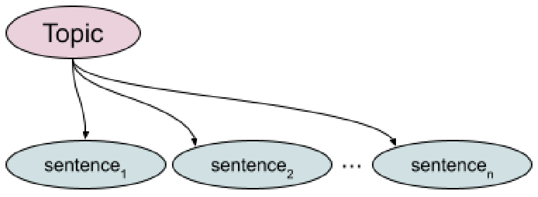


Figure 2.1: Bag model of sentence generation.

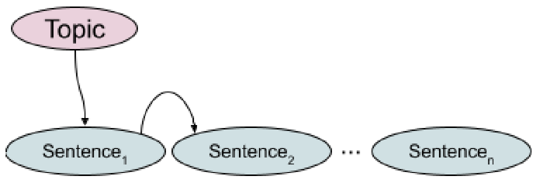


Figure 2.2: Chain model of sentence generation.

The mathematics underlying this computation is presented in the paper by Sap et al. [1]. Starting from the outermost layer, the narrative flow (also known as linearity) for the i$^{\text{th}}$ sentence in a story is calculated as follows:

$$\Delta_l(s_i) = -\frac{1}{|s_i|}[\log p(s_i|\mathcal{E}) - \log p(s_i|\mathcal{E}, s_{1:i-1})] \tag{2.1}$$

where $|s_i|$ denotes the length of sentence s_i, the first logarithmic probability is the logarithm of the probability of s_i being generated given a context consisting of the topic $\mathcal{E}$ according to the bag model, and the second logarithmic probability is the logarithm of the probability of s_i being generated given a context consisting of the topic $\mathcal{E}$ as well as the previous sentences in the story (the history) according to the chain model. To further break this down, the probability of a sentence given some context $\mathcal{C}$ and history $w_{1,j-1}$ is:

$$\log p(s|\mathcal{C}) = \sum_t \log p(w_j|\mathcal{C}, w_{1,j-1}) \tag{2.2}$$

The logarithmic probability of a sentence s given a context $\mathcal{C}$ is equal to the sum over all t words in a sentence of the logarithm of the probability of every word w_j given the context $\mathcal{C}$ and the previous words in the sentence $w_{1,j-1}$. The probability of a single word (or token) can be found using a neural language model, such as GPT2. For the goal of finding the probability of a token, the GPT2LMHeadModel is used because it has a language-modelling head and the outputs of the model are logits that can be turned into logarithmic probabilities by simply applying a softmax on them. The GPT2Tokenizer does not always tokenise text into natural-language words (e.g. concerts is tokenized into 'concert' and 's' and each of those tokens is mapped to an integer ID in the vocabulary of the model), but this is not a problem as the summation does not necessarily need to be over all natural-language words in a sentence but rather over all tokens comprising the sentence.

Neural language model

The narrative flow analysis relies on a language model to get word probabilities. In the original paper the GPT [8] model is used, but in this project the language model is GPT2 [14]. GPT2 is the newest freely available model by OpenAI and it has slight improvements over GPT [14]. Also, because it is newer, there is much more support and discussion online about it, which can be a helpful crib when facing challenges specific to the model. More specifically, the GPT2LMHeadModel is used because it has a language-modelling head. A language-modelling head is an additional linear layer with weights tied to the input embeddings, whereas the GPT2Model does not have a specific head and its outputs are raw hidden states. The outputs of the LMHead model are logits, so the log-probability of a token can be obtained by applying a log-softmax on the logits. The implementation of GPT2 used in this project is provided in the HuggingFace `transformers` library [15]. The preprocessing for the narrative flow analysis module is performed using the GPT2Tokenizer, also from the `transformers` library.
The GPT2 model can be used both with PyTorch and Tensorflow. However, some research on which framework is preferable to use reveals that there are several GitHub issues[2] claiming that the PyTorch implementation is considerably faster. Those issues have been confirmed but not resolved by team members of HuggingFace. Thus, the PyTorch implementation of the model was chosen.

2.4 Overview of the evaluation module

In this project the statistical evaluation of the results produced during the analysis stage is performed using paired t-tests. The original paper by Sap et al. [1] also uses paired t-tests for the evaluation of the metrics discussed in this dissertation (LIWC metrics, average concreteness and average narrative flow). The stories are evaluated in pairs of recalled and imagined due to the process of generating the stories (see more in section 3.2.1).

[2]e.g. `https://github.com/huggingface/transformers/issues/6264`

Background on paired t-tests

The original paper uses paired t-tests for evaluation, however, some background from "Discovering statistics using R" [16] was helpful in understanding this choice of evaluation and what the produced numbers mean. A paired t-test evaluates two groups and matches each instance from one group with an instance in the other group. Paired t-tests work under the assumption that the samples are normally distributed, so the means of the samples of the two groups should have a normal distribution (rather than the actual scores). The paired t-tests are performed using the implementation in the `sciPy`[3] library: `Scipy.stats.ttest_rel`.

Background on effect size

The original paper also reports an effect size, but does not specify how it is calculated. The formula used in this project is the difference of means of the two groups divided by the pooled standard deviation, also known as Cohen's d and illustrated mathematically below:

$$d_s = \frac{\bar{X}_{imagined} - \bar{X}_{recalled}}{\sqrt{\frac{(n-1)(SD^2_{imagined}+SD^2_{recalled})}{2n-2}}} \tag{2.3}$$

Following Cohen's own formulation [17] the effect sizes should be interpreted as follows: d=0.2 is considered small, d=0.5 is medium and d=0.8 is large. Furthermore, since Cohen's d is a magnitude, the modulus of the effect size should be reported without the sign, as the sign conveys the direction of the effect, which is included in a separate column. The sign in front of an effect size depends on which mean is subtracted from which, so the sign will be reversed if the order of subtraction is reversed. However, the direction remains the same. This idea is equivalently applied for the t-statistic.

2.5 Overview of the fake news extension

The goal of the fake news extension is to establish whether the analysis methods developed so far (LIWC measures, average concreteness and narrative flow) can be potent in distinguishing between reliable and fake news. The rationale behind the extension is that reliable news similarly to recalled stories contain events that have actually happened, whereas fake news similarly to imagined stories are based on events that were made up by the author based on their commonsense knowledge about the world. However, before embarking on this exploration it is important to note that the stories from the Hippocorpus and news articles differ greatly in style: the Hippocorpus stories are told from first person point of view and are autobiographical, i.e. the events in the story either have been personally experienced or imagined by the author, whereas news articles are usually written in the third person and the events discussed within the article are not usually autobiographical. The corpus that is used for this extension can be accessed via Github[4].

[3]`https://www.scipy.org/`

[4]`https://github.com/several27/FakeNewsCorpus`

The original database of fake news contains almost 9 million news stories of varying credibility and type, so it needed to be truncated in order to allow preprocessing and analysis in a timely manner. The code already written for the main body of the project was readily reusable in most cases. The strategy for executing the code for preprocessing and analysing the news without making major modifications to the already existing code was to introduce a new `mode` argument, which takes a value of either `hc` to run the Hippocorpus version of the code, `news` to run the news version of the code, or `test` to run the unit tests of the utility functions used within that module. Places where extra work is needed for the extension are discussed at length in section 3.7 and briefly listed below:

- Reshaping: the fake news original dataset is considerably larger (27GB) than the Hippocorpus so appropriate tools need to be used for reading it in. A large number of stories need to be truncated based on the following criteria:
  - only fake and credible news stories should remain and all other types of stories should be dropped (such as satire, bias, conspiracy...);
  - only stories with a non-empty meta-description should stay as this field will be used equivalently to the summary field in the Hippocorpus;
  - each fake news story should be matched with a reliable news story based on common named entities and any story that could not be matched in this process should be dropped.
  - this reshaping yields 3,176 news stories, 1,588 fake and 1,588 reliable.
- Preprocessing: unlike the Hippocorpus stories, the news stories are scraped from the Internet so they may not be as nicely formatted and an extra preprocessing step of cleaning characters, such as new lines, is required before the already established preprocessing for the Hippocorpus stories.
- Neural language model analysis: the difference in narrative flow was most strongly observed when using the summary of a story as its topic, so instead of calculating narrative flow with three different topics (summary, event or empty sequence), it should only be calculated using the equivalent of a summary for the news, which is the meta-description.
- Evaluation: unlike with the Hippocorpus, here the pairings between fake and reliable stories are 1:1, so the evaluation is more straightforward.

2.6 Requirements analysis

2.6.1 Programming requirements

1. Preprocessing module
   - Dataset reshaping: dropping unnecessary metadata, retold stories and recalled stories without an imagined pairing

    - `spaCy` preprocessing for the concreteness module: tokenisation, lemmatisation and lower-casing
    - GPT2Tokenizer preprocessing for the narrative flow module

2. Analysis modules
    - LIWC analysis
    - Concreteness analysis
    - Narrative flow analysis
3. Evaluation module, using paired t-tests
4. (Extension) Fake news analysis
5. (Extension) Episodic vs. semantic memory analysis

2.6.2 Results analysis requirements

1. Produce conclusions based on the numerical results from the analysis and evaluation.
2. Compare the results with the ones from the original paper by Sap et al. [1].

2.7 Success criteria

According to the success criteria defined in the project proposal (see Appendix B) the project would be a success if:

- I have reimplemented the aspects of the paper by Sap et al. [1], more specifically the analysis of the Hippocorpus dataset using lexicon-based and neural-language-model-based techniques and statistically evaluated the results;
- I have accounted for any differences between my results and the results from Sap et al. [1].

2.8 Software development

2.8.1 Languages and libraries

Python is used for the implementation of the project. Some notable libraries are `pandas`[5], `numPy`[6], `sciPy`[7], `spaCy`[8], HuggingFace's `transformers` [15], `PyTorch`[9], `argparse`[10] and the `literal_eval` fucntion from `ast`[11].

[5] `https://pandas.pydata.org/`

[6] `https://numpy.org/`

[7] `https://www.scipy.org/`

[8] `https://spacy.io/`

[9] `https://pytorch.org/`

[10] url://docs.python.org/3/library/argparse.html

[11] `https://docs.python.org/3/library/ast.html`

2.8.2 Software engineering best practices

The code was incrementally optimised to achieve faster run times. Unit tests for utility functions were implemented to ensure they work as expected. The Python style guide[12] was followed and referenced where necessary. Functions were named according to the Python style guide: function and variable names should be lowercase with underscores and private functions are marked by starting the name with an underscore.

2.8.3 Version control and back-up

Git was used for version control and ensuring that the work resided locally and remotely. GitKraken[13] was used to facilitate the process of working with Git. The dissertation itself was written in Overleaf[14]. The code and dissertation drafts were automatically updated and hosted online and once weekly a backup on a local hard-drive was performed. The source code is publicly available on GitHub under the MIT license[15], which allows commercial and private use of the code, but does not offer a warranty and limits my liability.

2.8.4 Organisational techniques

A physical Kanban board was used to organise priorities for the project and track their progress. The software development methodology used throughout this project was the waterfall model.

2.9 Systems that had to be learned

Firstly, I needed to expand my knowledge of neural networks by learning more about OpenAI's GPT2 model and using the model with PyTorch. I had to learn about statistical testing, more specifically paired t-testing in order to understand how to properly evaluate my data and look for statistically significant results. Finally, I learned how to work with the university's High Performance Computing (HPC) unit because my personal laptop has fairly deprecated hardware and no GPU.

2.10 Starting point

Having paid special attention to them, I believe the following Computer Science Tripos courses were useful: Scientific Computing, Machine Learning and Real World Data, Foundations of Data Science, Formal Models of Language, Natural Language Processing and Artificial Intelligence. I have had experience working with Python over the last two summers: in 2019 I worked in an enormous codebase and learned the importance of testing,

[12]url://www.python.org/dev/peps/pep-0008/
[13]`https://www.gitkraken.com/`
[14]https://www.overleaf.com/
[15]`https://choosealicense.com/licenses/mit/`

whereas in 2020 I implemented an NLP classification project in Python and gained specific knowledge of some helpful libraries, such as spaCy. I had very limited knowledge of working with neural networks. I had taken a course on Coursera called "Natural Language Processing in TensorFlow"[16]. I was aware I had much more to learn in this area and was prepared to take more online courses that seemed appropriate, as well as to go back to my notes on Artificial Intelligence for a more fundamental understanding. I had read the paper "Recollection versus Imagination: Exploring Human Memory and Cognition via Neural Language Model" [1] on which my project is based and had taken note of the structure of the Hippocorpus. As for my environment, I had PyCharm[17] installed, but needed to do some research on the libraries to be installed.

2.11 Summary of preparation

During the preparation stage data, resources, tools and underlying theories were researched. The high-level overview of the software modules was outlined. A software development strategy was produced. The work done in this initial stage served as a solid foundation during the later stages of the project.

[16]`https://www.coursera.org/learn/natural-language-processing-tensorflow`
[17]`https://www.jetbrains.com/pycharm/`

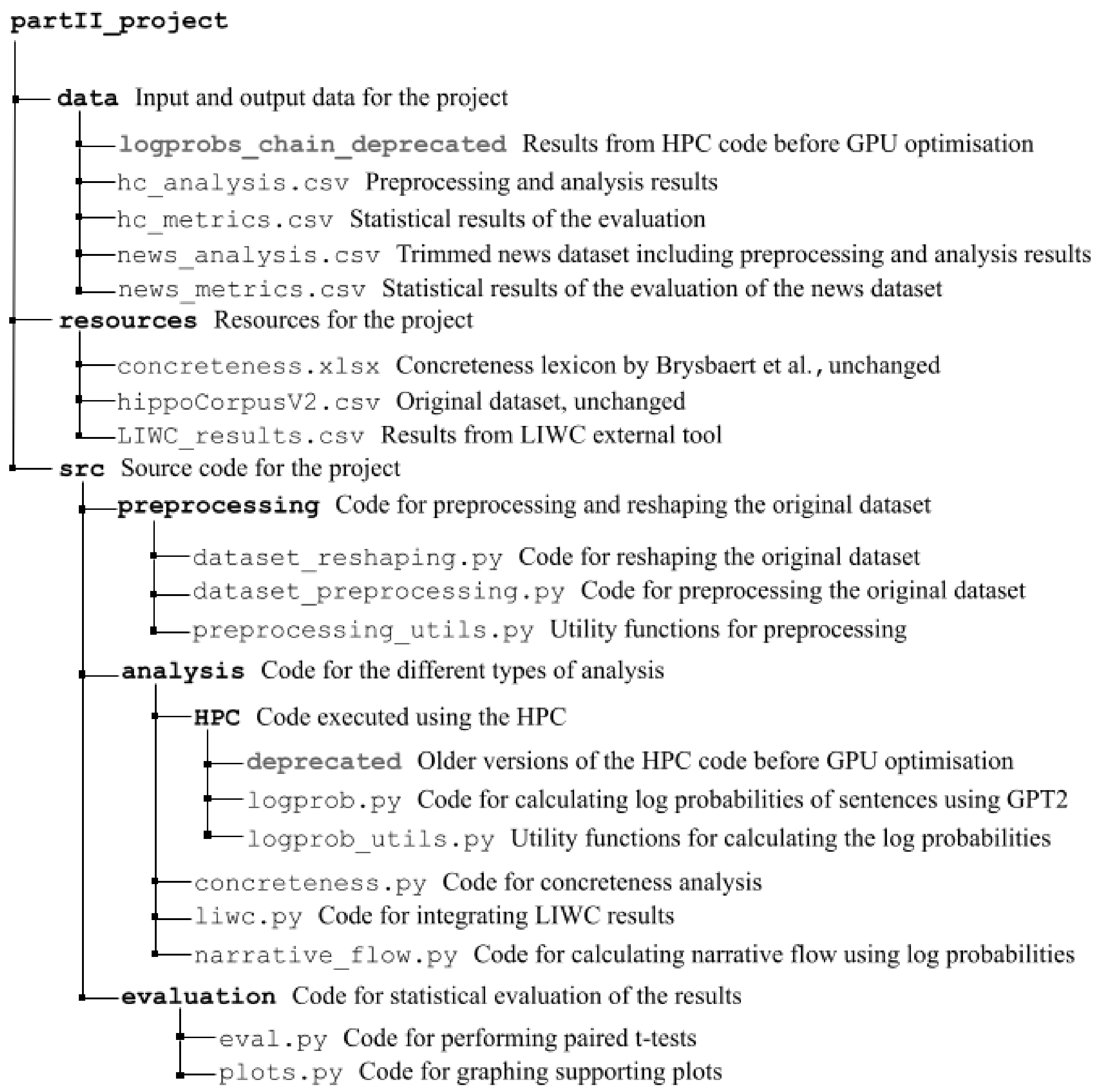
Chapter 3

Implementation

The project had three defined stages: preprocessing, analysis and evaluation and this structure is clearly seen in the organisation of the repository (see figure 3.1).

3.1 Repository overview

The repository is divided into three folders: data, which holds the data that is analysed as well as the results from the analysis, resources, which hold resources necessary for the project, and src, which contains the source code and is further divided into preprocessing, analysis and evaluation. More detail can be seen in figure 3.1.
Each non-utilities file utilises the boilerplate code line `if __name__ == "__main__"` as the marker for the beginning of the main code body. The `argparse` library is used to facilitate setting a parameter to determine the `mode` in which the program should run: `hc` for executing the version of the code used with the Hippocorpus, `news` for executing the version of the code used with the news dataset, and `test` for executing the unit tests on the utilities functions (if any) used within that program.



```
partII_project
├── data  Input and output data for the project
│   ├── logprobs_chain_deprecated  Results from HPC code before GPU optimisation
│   ├── hc_analysis.csv  Preprocessing and analysis results
│   ├── hc_metrics.csv  Statistical results of the evaluation
│   ├── news_analysis.csv  Trimmed news dataset including preprocessing and analysis results
│   └── news_metrics.csv  Statistical results of the evaluation of the news dataset
├── resources  Resources for the project
│   ├── concreteness.xlsx  Concreteness lexicon by Brysbaert et al., unchanged
│   ├── hippoCorpusV2.csv  Original dataset, unchanged
│   └── LIWC_results.csv  Results from LIWC external tool
└── src  Source code for the project
    ├── preprocessing  Code for preprocessing and reshaping the original dataset
    │   ├── dataset_reshaping.py  Code for reshaping the original dataset
    │   ├── dataset_preprocessing.py  Code for preprocessing the original dataset
    │   └── preprocessing_utils.py  Utility functions for preprocessing
    ├── analysis  Code for the different types of analysis
    │   ├── HPC  Code executed using the HPC
    │   │   ├── deprecated  Older versions of the HPC code before GPU optimisation
    │   │   ├── logprob.py  Code for calculating log probabilities of sentences using GPT2
    │   │   └── logprob_utils.py  Utility functions for calculating the log probabilities
    │   ├── concreteness.py  Code for concreteness analysis
    │   ├── liwc.py  Code for integrating LIWC results
    │   └── narrative_flow.py  Code for calculating narrative flow using log probabilities
    └── evaluation  Code for statistical evaluation of the results
        ├── eval.py  Code for performing paired t-tests
        └── plots.py  Code for graphing supporting plots
```

Figure 3.1: Repository overview.

3.1.1 Overview of the main data file

The main data file where all of the preprocessing and analysis data is stored is `hc_analysis.csv`. Please refer to figure 3.2 to see the organisation of the modules and the data-flow between them and `hc_analysis.csv`. For a more detailed summary of the specific data-field dependencies of each module, please refer to figure 3.3

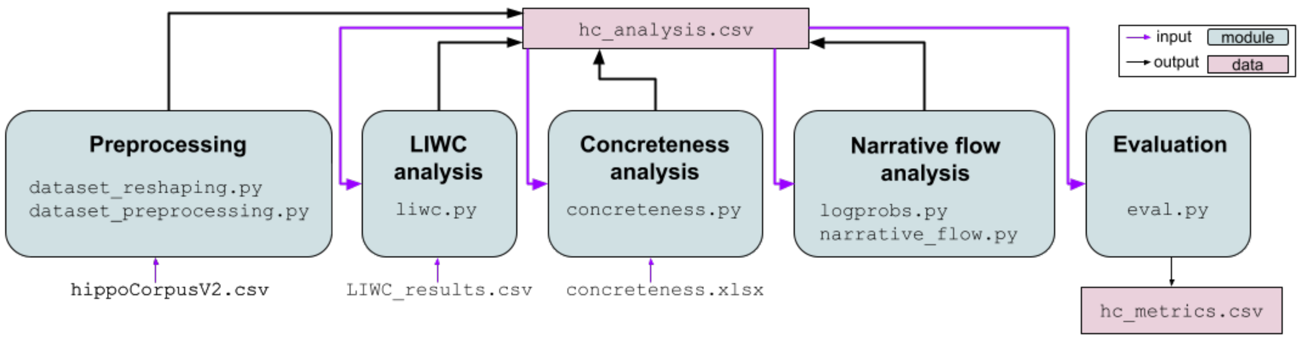
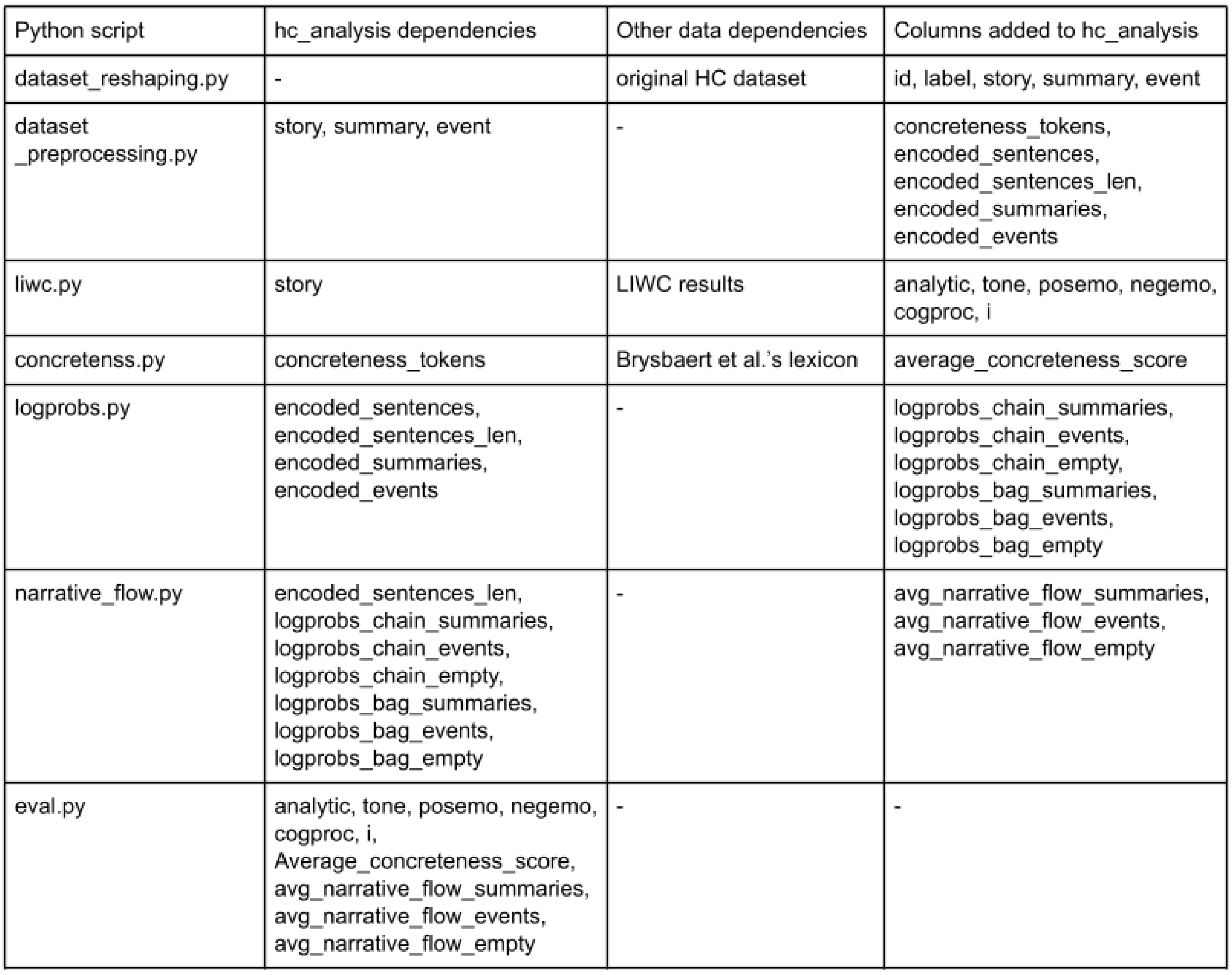


Figure 3.2: Data-flow diagram showing the resource dependencies of the modules as well as their dependencies and interactions with the hc_analysis.csv file.



Python script	hc_analysis dependencies	Other data dependencies	Columns added to hc_analysis
dataset_reshaping.py	-	original HC dataset	id, label, story, summary, event
dataset _preprocessing.py	story, summary, event	-	concreteness_tokens, encoded_sentences, encoded_sentences_len, encoded_summaries, encoded_events
liwc.py	story	LIWC results	analytic, tone, posemo, negemo, cogproc, i
concretenss.py	concreteness_tokens	Brysbaert et al.'s lexicon	average_concreteness_score
logprobs.py	encoded_sentences, encoded_sentences_len, encoded_summaries, encoded_events	-	logprobs_chain_summaries, logprobs_chain_events, logprobs_chain_empty, logprobs_bag_summaries, logprobs_bag_events, logprobs_bag_empty
narrative_flow.py	encoded_sentences_len, logprobs_chain_summaries, logprobs_chain_events, logprobs_chain_empty, logprobs_bag_summaries, logprobs_bag_events, logprobs_bag_empty	-	avg_narrative_flow_summaries, avg_narrative_flow_events, avg_narrative_flow_empty
eval.py	analytic, tone, posemo, negemo, cogproc, i, Average_concreteness_score, avg_narrative_flow_summaries, avg_narrative_flow_events, avg_narrative_flow_empty	-	-

Figure 3.3: The data-field dependencies of each script in relation to hc_analysis.csv.

3.2 Preprocessing

The preprocessing stage prepares the data for the analysis stage. The dataset is reshaped and then the data entries are preprocessed with `spaCy` and the `GPT2Tokenizer`.

3.2.1 Dataset reshaping

Dataset reshaping serves to remove all data not needed for the analysis and to produce an easier-to-use dataset from the original. This stage heavily relies on the functionalities of the `pandas` library.

Subset of the Hippocorpus used in the project

The original dataset was truncated to include only the entries needed for this project. Firstly, the dataset was truncated by removing all rows containing a retold story. There are some recalled stories in the dataset that do not have a pairing with an imagined story, and they also were removed as the evaluation method requires that each recalled story has a corresponding imagined story. After these truncations, 2572 recalled and 2756 imagined stories remain for analysis and evaluation (more information can be found in section 3.5). Furthermore, a large part of the metadata collected alongside the stories is unnecessary for the analysis, so those columns also were removed. The categories of data that remain in the dataset for analysis are the following:

- ‘AssignmentId’: Unique ID of this story, renamed to ‘id’. This entity is retained in order to be able to access the rest of the information that applies to a story from the original dataset during the evaluation and conclusion stages.
- ‘mainEvent’: Short phrase describing the main event, renamed to ‘event’. This entity will be used in the narrative flow analysis as one possible topic for the story.
- ‘memType’: Type of story (recalled, imagined, retold), renamed to ‘label’. The label will be used to identify the type of story during the evaluation.
- ‘story’: Story about the imagined or recalled event (15-25 sentences). This entity contains the story that will be run through the various types of analyses.
- ‘summary’: Summary of the events in the story (1-3 sentences). This entity will be used in the narrative flow analysis as another possible topic for the story.

Initial reshaping

The first step of the reshaping is to read in the original dataset from the `hippoCorpusV2.csv` file as a `pandas` DataFrame and make a copy of it so that the original dataset remains unchanged for future reference. The DataFrame that is used for storing the reshaped data is `hc_analysis`, which is later saved as `hc_analaysis.csv`. Only the necessary columns are copied and renamed for ease of use. A description of the five columns that remain as well as the name changes is available in section 3.2.1. The next step is to remove all retold stories, which is achieved using Boolean indexing in `pandas`: only the stories whose `label` is not equal to 'retold' remain. After dropping the retold stories, the index of the DataFrame needs to be reset, which can be done in place using the `reset_index` `pandas` function.

Reshaping necessary for evaluation stage

The next step of reshaping is performed in preparation for the evaluation stage. The evaluation is performed using paired t-tests, which require that the data from the two compared groups, imagined and recalled, can be paired based on some shared characteristic. In this case the basis on which stories are paired is a shared summary. However the dataset has some recalled stories that do not have an imagined pairing. Due to the limitations of the evaluation method, recalled stories without an imagined pairing need to be dropped. For that purpose, the routine in algorithm 1 is performed. Python `sets` and `pandas` Boolean indexing are used in the process to maximise space and time efficiency. The remaining entries in `hc_analysis` are explored to see if there are recalled-imagined pairings sharing the same summary consisting of more than two stories. It turns out that the following count of recalled-imagined pairings is present:

- 2395 pairs (1 recalled, 1 imagined)
- 176 triples (1 recalled, 2 imagined)
- 1 decuple (1 recalled, 9 imagined)

This information is important for the paired t-test evaluation stage.

Algorithm 1 Reshaping in preparation for the evaluation stage

```
$hc\_analysis\ DataFrame$
$summaries \leftarrow all\ summaries\ in\ hc\_analysis$
$intersection \leftarrow summaries\ of\ recalled\ stories\ \cap\ summaries\ of\ imagined\ stories$
for $summary \in summaries$ do
    if $summary \notin intersection$ then
        $hc\_analysis \leftarrow hc\_analysis \setminus entry\ with\ that\ summary$
```

Summary of reshaping

To sum up, the `hc_analysis` DataFrame is created by selecting and renaming only the necessary columns of a copy of the original Hippocorpus [1]. Then it is cleaned from retold stories and recalled stories that do not have a matching imagined story. A brief exploration of the remaining entries is performed to find out the types and number of recalled-imagined pairings in the DataFrame. Finally, the `hc_analysis` DataFrame is saved as a `.csv` file of the same name and the dataset reshaping is complete. This file, `hc_analysis.csv` is going to be populated with preprocessing and analysis data in the next stages.

3.2.2 Dataset preprocessing

Code organisation and set-up

The preprocessing is split between two Python files: `dataset_preprocessing.py` and `preprocessing_utils.py`. The file `dataset_preprocessing.py` serves as an outer layer for the preprocessing where the utility functions defined in `preprocessing_utils.py` are called. Its only library import is `pandas`. It loads the file `hc_analysis.csv` locally as the DataFrame `hc`. Firstly, it gets the summaries and events from `hc` as they are needed for preparing the input for the GPT2 model. The `preprocessing_utils.py` file uses two imports: `spaCy` and GPT2Tokenizer from `transformers`. The `spaCy` `en_core_web_lg` language model is loaded into an object named `nlp` to be used for the preprocessing with `spaCy`. The GPT2Tokenizer is constructed using the pretrained tokenisers of the `transformers` library.
The first call from `dataset_preprocessing.py` into `preprocessing_utils.py` is `get_docs(hc)`, which turns all of the stories from `hc_analysis` into `spaCy` docs. A `spaCy` doc contains all of the necessary linguistic information for further stages of preprocessing. It is obtained by calling `nlp(story)` for each story. The `get_docs` function takes as an input a DataFrame `hc` which needs to have the column labels used in `hc_analysis` and returns a list of `spaCy` docs. The docs are stored locally at `dataset_preprocessing.py` in a list called `docs`. The `spaCy` docs contain helpful information used both for the concreteness and the narrative flow analysis. Throughout this project list comprehensions (implicit for loops) are utilised to make the code neater.

Preprocessing for LIWC

LIWC does not require preprocessing as the data that would normally be cleaned during preprocessing (e.g. verb tense is removed during lemmatisation) is useful for the LIWC tool.

Preprocessing for concreteness analysis

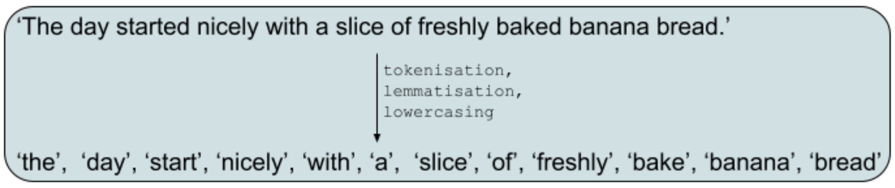

'The day started nicely with a slice of freshly baked banana bread.'
tokenisation, lemmatisation, lowercasing
'the', 'day', 'start', 'nicely', 'with', 'a', 'slice', 'of', 'freshly', 'bake', 'banana', 'bread'

Figure 3.4: Preprocessing, which prepares the input for the concreteness analysis.

The concreteness preprocessing is entirely reliant on the `spaCy` docs. The goal is to produce lowercased, lemmatised, entirely alphabetical token lists for each story. Those tokens are obtained using the `get_tokens_concreteness(docs)` function and are stored in a column of the dataset labelled as `tokens_concreteness`. The function input is the list of `spaCy` docs and its output is a list of lists of tokens that are going to be used in the concreteness analysis. This function is a wrapper for the private function _preprocess_doc(doc) which takes as input a single spaCy doc and returns only the alphabetic tokens from it after they have been run through a specific preprocessing for the concreteness analysis. That specific preprocessing is obtained through another private function _preprocess_token(t) which takes as input a spaCy doc token *t* and returns its prepossessed version which is lemmatised and lowercased. A simplified version of the process is visualised in figure 3.4. A special case is when the token is a pronoun token. As discussed before spaCy lemmatises all pronouns as the artificial token '-pron-', however, this is not beneficial in this project (as pronouns have different concreteness ratings), so for any token whose lemma is '-pron-' it is only lowercased and not lemmatised as to not lose the concreteness information contained in different types of pronouns.

Preprocessing for narrative flow analysis

The narrative flow preprocessing relies both on the spaCy docs as well as on the GPT2Tokenizer. The goal is to end up with tokens encoded specifically for GPT2 and grouped by sentence for each story. The narrative flow preprocessing step relying on the spaCy docs is sentence separation, which is implemented using a call to get_sentences. The function get_sentences defined in preprocessing_utils.py takes as input a list of spaCy docs, in this case the docs variable, and returns a list of lists of sentences. For each doc in docs the text encoding (as opposed to the spaCy doc encoding) of a sentences is put in a list for each sentence in doc.sents which iterates through all of the sentences in a doc. The spaCy built in sentence separator is used. There is no need for a more sophisticated separator because the data of the Hippocorpus is not incredibly difficult (e.g. not highly technical and/or with many abbreviations).
Sentences are encoded and stored in hc as encoded_sentences. They are encoded using a call to get_encoded_sentences(sentences) which takes as input a list of lists of textual sentences and returns sentences encoded to be used with GPT2. For every list of sentences this function calls the private function _encode_list_of_inputs(inputs) which takes as input a sentence list (there is one per each story) and calls tokenizer.encode(sentence) for every sentence in the sentence list and returns a list of lists of the encoded tokens in each sentence. The next call is to get_encoded_sentences_len(hc['encoded_sentences']) to get the sentence lengths of each sentence and store them in hc['encoded_sentences_len']. The get_encoded_sentences_len function takes as input the encoded sentences and returns a list of lists of integers where every sublist corresponds to a story and the integers indicate the lengths of each sentence in that story. The next calls are two calls to get_encoded_topics(topics), once with summaries and once with events and the encoded summaries and events are stored in hc in columns encoded_summaries and encoded_events. The function get_encoded_topics(topics) takes as input a list of strings and returns a list of lists of encodings for each string element of the input list. It also calls the private function _encode_list_of_inputs(inputs) to obtain the encodings.
The final call in the dataset_preprocessing.py file saves the hc DataFrame with all of the newly added information from the preprocessing as hc_analysis.csv in the data

folder. The `hc_analysis.csv` file is updated to include the new information.

Summary of preprocessing

To conclude, the preprocessing code is held in two files, a higher-level file `dataset_preprocessing` where the `hc_analysis` dataset is interfaced and a file that holds all of the definitions of functions called in the higher-level file as well as some private functions only called within the `preprocessing_util.py` file. LIWC needs no preprocessing and the necessary preprocessing results for concreteness and narrative flow analysis are all stored and saved in the `hc_analysis.csv` at the end of the preprocessing. Preprocessing was performed using the HPC because the functions that involve `GPT2Tokenizer` take a long time to be executed locally due to the use of the large model and the lack of a GPU or high performance processor locally.

3.3 Lexicon-based analyses

3.3.1 LIWC analysis

This stage of the project involves feeding the original stories into the LIWC program, which works as a local application. The output of the program is a spreadsheet with the metrics for each story. Afterwards, this spreadsheet output is reshaped to keep only the six metrics that will be further evaluated: cogproc, posemo, negemo, i, analytic and tone (see section 2.1.3). The goal of my LIWC analysis module is to reshape the data and integrate it with the rest of the analysis data.
The Linguistic Inquiry and Word Count (LIWC) is an external tool that provides psychological state inquiry based on linguistic cues. The external LIWC tool is run on the stories in the `hc_analysis.csv` file and all of the results are saved in a new file called `LIWC_results.csv` stored in the resources folder.
The `liwc.py` file uses the `pandas` library. The data files `hc_analysis.csv` and `LIWC_results.csv` are loaded into DataFrames locally as `hc` and `liwc_results` correspondingly. The necessary parameter scores from LIWC are accessed from `liwc_results` and saved in `hc` as the following DataFrame columns: 'analytic', 'tone', 'i', 'posemo', 'negemo' and 'cogproc'. The updated `hc` is saved as `hc_analysis.csv`.

3.3.2 Concreteness analysis

In the original paper by Sap et al. the concreteness measurement is described as follows: "measure the average concreteness level of words in stories" [1]. In this project this is interpreted as the following steps:

- Look for story token in the concreteness lexicon
- Calculate the mean of all tokens from a story that are found in the lexicon
- Average concreteness score = sum of concreteness score of all words in the story found in lexicon / number of words in the story found in the lexicon

The concreteness analysis is implemented in the file `concreteness.py`. This file uses `pandas`. It first loads the `hc_analysis.csv` file as a local DataFrame named `hc` as well as the Brysbaert et al. concreteness lexicon as a DataFrame named `concretness_lexicon` from an `Excel` file in the resources folder `concreteness.xlsx`. The data categories from the lexicon used for the average concreteness analysis in this project are:

- Word: the word (or bigram).
- Conc_M: the mean concreteness rating of the expression.

The next step is to create a dictionary named `lex` that is keyed by the words in the concreteness lexicon and the values are the mean concreteness score of the word. The dictionary is created using the `dict()` constructor as well as the `zip()` function which 'zips' together the keys and values.
Next the function `get_concreteness_score(story_tokens)` used for calculating the concreteness score of each word is defined. In the context of the Brysbaert lexicon a word can be composed of one or two tokens (the latter is called a bigram) and the function also enables detection of bigrams as well as of single token words from the Brysbaert lexicon. The function takes as input the concreteness tokens of a story and returns a float which represents the concreteness score for that story.
The function in algorithm 2 is called for each story in `hc` and the resulting scores are saved in a new column of the DataFrame called `concretensss`. Finally the updated version of `hc_analysis` is saved in the data folder.

Algorithm 2 Calculating the average concreteness score of a story

```
function GET_CONCRETENESS_SCORE(story)
    $n \leftarrow length(story)$                         ▷ lexicographically-based length
    $i \leftarrow 0$                                     ▷ lexicographically-based counter
    $count \leftarrow 0$                                 ▷ semantically-based counter
    $sum \leftarrow 0$
    while $i < n$ do
        $t_1 \leftarrow i^{th}$ token of story
        if $i \neq n - 1$ then                            ▷ if i is not pointing to the last token
            $t_1t_2 \leftarrow i^{th}$ and $(i+1)^{th}$ tokens of story
            if $t_1t_2 \in lex$ then
                $i \leftarrow i + 2$
                $count \leftarrow count + 1$
                $sum \leftarrow sum +$ concreteness value of bigram $t_1t_2$ in lex,
                Continue onto the next iteration of the while loop
            if $t_1 \in lex$ then
                $count \leftarrow count + 1$
                $sum \leftarrow sum +$ concreteness value of token $t_1$ in lex,
        $i \leftarrow i + 1$
    return $sum/count$                                ▷ average concreteness score of the story
```

Note on bigrams

The drawback of the above-described implementation acknowledging bigrams is that it first checks for a bigram and only if a bigram is not present it proceeds to check if the single token is present. This means that some bigrams might occur by chance, i.e. that two tokens that form a bigram found in Brysbaert's lexicon occur one after another in a story without having the expected linguistic dependency relation. An improvement could be implemented by using a parser.
Let's take the sentence 'He was sober up until his friends left.' as an example to explain the point made above. This sentence will be preprocessed to result in the following token list: 'he', 'be', 'sober', 'up', 'until', 'his', 'friend', 'leave'. The function described above will score 'sober up' as a bigram, because it can be found in the Brysbaert lexicon, when instead it should have been scored as two independent words, 'sober' and 'up' based on the meaning of this sentence. A dependency parser establishes the grammatical relations between the words in a sentence and the dependency relations obtained from such a parser can be used to prevent the above mistake [18].

3.4 Neural-language-model-based analysis: narrative flow

The narrative flow analysis, which makes use of the neural language model GPT2, is performed in part locally and in part using the HPC.

3.4.1 Overview of the GPT2 model usage

The GPT2LMhead model is used to calculate the logarithmic probability of a sentence given its context (the text preceding the sentence, more specifically determined using the models from figures 2.1 and 2.2). This is further broken down into summing the logarithmic probability of all words (or tokens) in the sentence given their context, following equation 2.2. The GPT2LMHead model is used in the private function `_get_sentence_log_probability(encoded_input, context_len)` defined in `logprob_utils.py` in the HPC directory which takes as input an encoded input (list of integers) and the length of the context (an integer) and returns the logarithmic probability of the sentence. The function is modelled after the equation for calculating the logarithmic probability of a sentence (see equation 2.2). Please refer to the function definition below:

```python
def _get_sentence_logprobability(encoded_input, context_len):
    sentence_logprobability = 0
    offset = context_len + 1

    input_ids = torch.tensor([encoded_input])
    input_ids = input_ids.to(device)

    with torch.no_grad():
        outputs = model(input_ids)
```

```
10          logits = outputs[0][0]
11          logprobs = torch.nn.functional.log_softmax(logits, 1)
12
13      for i in range(offset, len(input_ids[0]) - 1):
14          token_id = input_ids[0][i]
15          logprob = logprobs[i - 1][token_id].item()
16          sentence_logprobability += logprob
17
18      return sentence_logprobability
```

In lines 2 and 3 two useful variables are declared: the `sentence_logprobability`, which will hold the ongoing sum of logarithmic probabilities and eventually will have the logarithmic probability of the entire sentence and the `offset`, which determines where the context ends and the sentence whose probability is sought begins. In lines 5 and 6 the torch tensor `input_ids` is constructed from the data (the `encoded_input`) and added to the device, which allows for running this code on GPU and significantly decreasing the run time (see section 3.4.3). Line 8 disables the gradient calculation (as it is not necessary for the purposes of this function because no backward operations are performed on the tensor), which reduces the memory consumption for the computation. In lines 9 and 10 we get the model's outputs given the input tensor `input_ids` and separate the logits from them. In line 11 the logarithmic probabilities for every input id are acquired by applying a logarithmic softmax to the logits. The for loop in lines 13-17 traverses over the range of the ids encoding the sentence (and excludes the ids encoding the context) and for every token finds its `token_id` from the encodings vocabulary of GPT2, finds its logarithmic probability and adds it to the ongoing sum of the sentence logarithmic probability. Once all of the logarithmic probabilities of the tokens comprising the sentences are added, the function returns the sentence logarithmic probability.

3.4.2 Code for the HPC

The code run on the HPC is the code that uses the GPT2 model. The code is divided between two files `logprob.py` and `logprob_utils.py`. As in the preprocessing stage of the project the utilities file contains the function definitions of utilities functions, which are in turn called in the outer file, which interfaces into the data file(s).

Setting up for work with the HPC

WinSCP[1] was installed to be used for transferring files between the local directory and the HPC. The guide by Dr. Caines[2] also was a very helpful introduction on how to login into the HPC, create batch scripts, enqueue new jobs and monitor their progress and use the log and error files to debug where necessary.

[1] `https://winscp.net/eng/index.php`

[2] `https://www.cl.cam.ac.uk/~apc38/usingTheHPC.html`

Setting up the utilities file

The file requires use of the `torch` library as well as the GPT2LMHeadModel from the `transformers` library. The `eot` (end of text) token value, which is specific to the model, is hard coded, but it could also be acquired through importing and loading the GPT2Tokenizer and accessing it from there (but this is unnecessary time and space consumption). The `device` is set to CUDA (if available) in order to allow for a GPU optimization (if unavailable, the CPU is used). The GPT2LMHeadModel is loaded with the pretrained weights for GPT2 and the model is put on the device.

Generating the input (context) for the models

The first public function defined in the utilities file is:

`get_inputs(model_name, encoded_contexts, encoded_sentences)`,

where the model name is the type of model (bag or chain), the encoded contexts are the list of encoded topics (summaries, events, or empty string) over all stories, encoded using the GPT2Tokenizer in the preprocessing stage, and the encoded sentences are the sentences of each story, encoded using the GPT2Tokenizer as well. There are private helper functions called by the public function to create the inputs. For the bag model the inputs are concatenations of the topic and each sentence from the story, as the bag model theorises each sentences in the story is drawn independently solely based on the topic. For the chain model the inputs are the topic and all sentences up to and including the current one.

Calculating logarithmic probability

There are two public functions defined for calculating the logarithmic probability of each sentence, one for the bag model and one for the chain model:

```
get_stories_logprobs_bag(encoded_contexts, inputs),
get_stories_logprobs_chain(encoded_contexts, encoded_sentences, inputs).
```

The encoded sentences are also passed for the chain model function (unlike for the bag model function) because the context for the chain model is of varying length as it includes the topic and all sentences up to the current one. Each of those public functions calls a private function to get the sentence probability of each sentence in a story and both of the private functions that get sentence probabilities rely on the `_get_sentence_log_probability(encoded_input,context_len)` function where the GPT2 model is directly used (see section 3.4.1 for the function definition).

3.4.3 Optimisation

The original plan was to calculate all of the logarithmic probabilities using one script and one job. However, on attempting this whilst running the job on CPU it turned out that the maximum run time of the HPC (12 hours, including any queuing time) was reached without the job completing. The next step in obtaining the computations was to divide the work into two scripts: `logprob_bag.py` and `logprob_chain.py`. The 12 hour limit was enough to obtain the logarithmic probabilities based on the bag model, but did not allow enough time for the chain model computations to complete, due to their far longer context. It became apparent that GPU optimisation is necessary in order to complete the fundamental goals of the project.

Attempt 1: unsuccessful

The first attempt to optimise the computation was to utilise Parallel from joblib[3]. However, this did not lead to a significant speed up and the job once again did not complete. The next action undertaken was to split up the computation into 6 separate scripts and run them as independent jobs. This allowed me to obtain the data but was not an elegant solution.

Attempt 2: successful

The next optimisation attempt was to perform the computation on GPU (rather than on CPU as before). This optimisation turned out to be surprisingly quick to implement and only took several lines of non-challenging code (adding the model and the input tensor to `device`, i.e. the GPU) and in hindsight should have been the first attempt of optimisation. The `logprob_chain.py` script completed in 35 minutes, whereas on the CPU with 4 parallel processes it would not complete in 12 hours (and the inelegant solution of 6 separate scripts took a total of 23 hours). The `logprob_bag.py` which previously took 6 hours to complete on the CPU now was also brought down to 30 minutes. Please note that those times are for calculating the logarithmic probability of all sentences in all stories but using only one topic (only the summary), whereas the goal was to also get the results with all three types of topics suggested in the original paper (summary, event, empty sequence). Computing on the GPU allowed fulfilling the initial plan of calculating all logarithmic probabilities in one script, `logrpob.py`, and the computation took 3 hours total for the three different types of topics.

3.4.4 Code run locally

The file `narrative_flow.py` utilises the `pandas` and `numPy` libraries as well as `literal_eval` imported from the `ast` library. Firstly, the `hc_analysis.csv` file is imported as a DataFrame and locally referenced as `hc`. The encoded sentences' length, and the logarithmic probabilities for the bag and chain model are read from `hc` into local variables and the narrative flow calculation is performed for each pair of bag and chain log probabilities (there are three pairs, as there are 3 different topics used: the summary of the story, the event of the story, or an empty sequence). Narrative flow is calculated by calling `get_narrative_flow(logprobs_bag,logprobs_chain,encoded_sentences_len)`. Since the narrative flow is initially a metric at sentence level, the average narrative flow

[3]`https://joblib.readthedocs.io/en/latest/parallel.html`

over all sentences in a story is also calculated using the `numpy.average` function and recorded in `hc`. Finally `hc_analysis.csv` is updated with the new data.

3.4.5 Summary of neural-language-model-based analysis

The narrative flow analysis, which utilises the neural language model GPT2, was a challenging part of the project as there were unexpected challenges related to working with the HPC. However, those were overcome through optimisation and the originally envisioned software design was implemented in the end.

3.5 Implementation of the evaluation module

The evaluation of the data for statistical significance is performed using paired (dependent) t-tests.

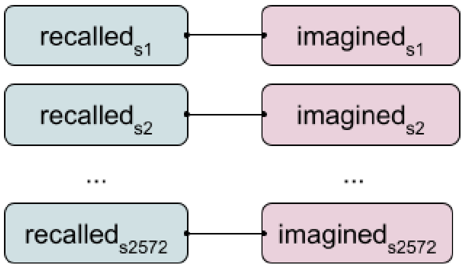


Figure 3.5: Each recalled story is paired with an imagined story based on them having the same summary.

3.5.1 Paired t-tests within the context of the Hippocorpus

In this case the two groups are of recalled stories and imagined stories and the pairings occur based on a shared summary. The recalled stories from the Hippocorpus which do not have a paired imagined story were dropped from the analysed dataset, as discussed in section 3.2.1. The remaining recalled-imagined stories pairings are: 2,395 pairs (1 recalled, 1 imagined), 176 triplets (1 recalled, 2 imagined), and 1 decuplet (1 recalled, 9 imagined). Two sets of paired t-tests were performed due to the latter specificity of the data: testing with duplicates, where the recalled story for each imagined story with the same summary is repeated and this results in 2,756 pairs, and testing without duplicates where some imagined stories were dropped at random to leave only one story per recalled story resulting in 2,572 pairs, and this simplified no duplicates case is illustrated in figure 3.5.

3.5.2 Implementation of paired t-tests

The evaluation is implemented in the `eval.py` file which makes use of the `pandas`, `Scipy.stats`, `numPy` and `math` libraries. Firstly, the `hc_analysis.csv` file is read in as it contains all of the recorded metrics from the analysis. A new DataFrame `hc_eval` is defined which holds the values of the columns `id`, `label`, `summary`, `concreteness`, `analytic`, `tone`, `i`, `posemo`, `negemo`, `cogproc`, `avg_narrative_flow_summaries`, `avg_narrative_flow_events`, `avg_narrative_flow_empty` from `hc_analysis`. Two additional DataFrames are created from `hc_eval`: `hc_recalled`, with all of the metrics of stories where the label is recalled, and `hc_imagined`, with all of the metrics of stories that are imagined. Two dictionaries are declared and populated: `imagined_recalled_ids` and `recalled_imagined_ids`, which correspondingly map from an imagined to a recalled id and from a recalled id to (possibly multiple) imagined id(s).

Once the dictionaries have been populated and the pairs of stories linked by the same summary are known, it is time to define the function responsible for performing the paired t-test. The function provided in Scipy.stats would normally suffice but it only returns a t-statistic and a p-value. In this project the effect size and direction of the effect are also necessary. Furthermore, the t-tests need to be ran twice: once for a population without duplicates of recalled stories and once for a population with duplicates of the recalled stories to match every imagined story. A high-level view of the function `paired_t_test(duplicates,metric)` implemented for that purpose is displayed in algorithm 3. The function takes as inputs a Boolean value `duplicates` and a string value `metric` which identifies the metric (from all of the possible analysis metrics) for which the t-test needs to be performed.

Afterwards a for loop over all of the analysis metrics calls the function `paired_t_test` for each metric once with and once without duplicates and based on the t-statistic signs determines the direction of the effect (towards the imagined stories if the t-statistic is positive and the recalled stories if it is negative). The statistical scores of the metrics are recorded and saved into a new DataFrame called `hc_metrics` and the `hc_eval` DataFrame is also saved in a csv.

Algorithm 3 Paired t-test evaluation function

```
  $imagined\_recalled\_ids,\ recalled\_imagined\_ids,\ hc\_analysis$
  function PAIRED_T_TEST($duplicates,\ metric$)
      $recalled \leftarrow [],\ imagined \leftarrow []$
      if $\neg dupliactes$ then
          for $id\_r \in recalled\_imagined\_ids$ do
              $id\_i = recalled\_imagined\_ids[id\_r][0]$
              $recalled \leftarrow recalled + metric\ value\ for\ hc\_analaysis\ entry\ keyed\ by\ id\_r$
              $imagined \leftarrow imagined + metric\ value\ for\ hc\_analaysis\ entry\ keyed\ by\ id\_i$
      else
          for $id\_i \in imagined\_recalled\_ids$ do
              $id\_r = imagined\_recalled\_ids[id\_i]$
              $recalled \leftarrow recalled + metric\ value\ for\ hc\_analaysis\ entry\ keyed\ by\ id\_r$
              $imagined \leftarrow imagined + metric\ value\ for\ hc\_analaysis\ entry\ keyed\ by\ id\_i$
      $effect\_size \leftarrow$ effect size calculated using equation 2.3
      $t\_stat,\ p\_val \leftarrow scipy.stats.ttest\_rel(imagined, recalled)$
       return $t\_stat,\ p\_val,\ effect\_size$
```

3.6 Testing

Unit tests were implemented for utility functions and can be ran through setting the argument mode to test (adding parameter `--mode test`) when executing a module. When in test mode, only the tests are run and the regular behaviour of the module is not.
The tests use the utility functions defined within the scope of the specific module or imported from one of the utility modules. Function test inputs were carefully curated to be representative of the actual data and to include edge cases and the expected returns of the utility functions were manually calculated.

3.7 Fake news extension

The fake news extension utilises the code already written for the Hippocorpus analysis with minor alterations to analyse a fake news dataset[4]. As the code for the Hippocorpus analysis is already discussed at length above, only the alterations necessary for this extension will be discussed in this section.

3.7.1 Reshaping the fake news dataset

The reshaping of the news dataset has the goal of trimming the very large (27GB) original dataset, which due to its size is not stored in the repository, into a more compact dataset that could seamlessly be plugged into the already built preprocessing, analysis and evaluation infrastructure. The code which performs the reshaping can be run when executing the `dataset_reshaping.py` program using the parameter `--mode news`.

Chunking the dataset

Selected columns (`type, content, title, meta_description, scraped_at`) from the large `news.csv` file are read from outside of the repository using `pandas`' `read_csv` method with parameter `chunksize` set to 100,000. Without chunking the original dataset, my personal computer crashes. The chunking allows to process each chunk individually, as to not get an out-of-memory error, and perform the first trim of the dataset. For that purpose the `chunk_filtering(chunk)` function was implemented: it takes, as input, one of the `pandas` chunks, drops any rows with an empty cell using the method `dropna()`, sorts the chunk by the `scraped_at` date and returns only the top 2.5% most recently scraped chunks. This is necessary so that the remaining DataFrame can be processed without crashing the computer. During the filtering of the chunks, only the news with a type of 'fake' or 'reliable' remain and the original proportion of each type of story in the full dataset is maintained.
After this initial filtering, what remains from each chunks is put back together in one DataFrame named `news_df`. The `scraped_at` column is no longer necessary, so it is dropped, and the `type` column is renamed to `label` which is the name used in further

[4]https://github.com/several27/FakeNewsCorpus

steps of the preprocessing and analysis. An `id` column is introduced which is used for pairing the stories, a step necessary for the paired t-test evaluation.

Exploring the encodings and cleaning erroneous encodings

Firstly, the encoding of the complete `news.csv` file was examined using `chardet`[5] and this manual examination revealed that the encodings of the news stories are mixed with the majority being `utf-8`. However, there were also leftover `windows1252` characters, which would need to be cleaned in order to present the further processing module, utilising `spaCy` and the GPT2Tokenizer, with recognisable symbols.
To address that some of the characters are encoded according to `windows1252`, a `clean_news_stories(story)` function is defined in the `preprocess_utils.py` file and is imported to be used in the reshaping file. This function relies on a privately defined also within `preprocess_utils.py` function called `_debug_utf8_encodings(story)` which makes use of a dictionary of erroneous to correct encodings, based on a table of such common errors [6]. This dictionary need not contain all of the possible erroneous characters but just ones that are necessary for grammatical relations (such as apostrophes, commas, quotations). One example of a wrong to correct mapping is `â€™` to `'` and after the function to debug the encodings is applied the following text, 'otherâ€™s' will be rewritten as 'other's'. After cleaning the common erroneous characters, `string.printable` is used within `clean_news_stories(story)` so that any leftover non-grammatical characters, such as new lines `\n` are also cleaned.

Filtering using named entities

At this stage there are more than 50,000 stories remaining and further filtering is necessary to allow for preprocessing and analysis within a reasonable time frame. The goal is to have any stage take less than 12 hours, so it can easily be ran on the HPC. The next filtering step is to use the named entities in the stories to pair them and to remove any stories that have not been paired.
The named entities are obtained form the `spaCy` docs using the `get_named_entities(news_df)` function. The entities returned by `spaCy` include both numerical and named entities. However, only named entities are needed for this filtering stage, so the `get_named_entities(news_df)` function only returns entities whose type is not `'DATE', 'TIME', 'PERCENT'` or any other numerical entity type. The entities are returned as a Python `set` rather than as a list to slightly decrease memory and space complexity in the named entity filtering stage and also to ensure story pairings are based on a significant intersection of named entities.
The named entity filtering is performed using the function

`named_entity_filtering(news_df)`,

defined in the reshaping file. A high-level overview of it is presented in algorithm 4.

[5] `https://pypi.org/project/chardet/`
[6] `https://www.i18nqa.com/debug/utf8-debug.html`

The function iterates over all of the fake ids in `news_df`, as the fake stories are less than the reliable ones, so this saves some time. The threshold of more than 4 entities was determined after empirical observation of pairings at different thresholds. A lower threshold does not eliminate enough stories and the pairings do not have much in common (e.g. for a threshold of 1 the pairing may be based on only one mention of a location).
Finally, the stories that have not been found a match are dropped which results in only 3,176 stories (half of each type) remaining in `news_df` and it is saved as a csv file in the `data` folder as `news_analysis.csv`, analogically to `hc_analysis.csv`.

Algorithm 4 Pairing fake and reliable news stories based on shared named entities

```
$all\_news\_df$, $fake\_news\_df$, $reliable\_news\_df$
function NAMED_ENTITY_FILTERING($news\_df$)
    $already\_matched = \emptyset$
    for $id\_fake \in fake\_news\_df$ do
        $ne\_set\_f \leftarrow$ named entity set keyed by id_fake from fake_news_df
        if $ne\_set\_f \neq \emptyset$ then
            for $id\_reliable \in reliable\_news\_df$ do
                $ne\_set\_r \leftarrow$ named entity set keyed by id_reliable from reliable_news_df
                if $|ne\_set\_r \cap ne\_set\_f| > 4 \ \wedge \ id\_reliable \notin already\_matched$ then
                    $already\_matched \leftarrow already\_matched + id\_reliable$
                    $all\_news\_df[id\_fake, paired\_id] \leftarrow id\_reliable$
                    Break out of the inner for loop, as a match has been found
```

3.7.2 Preprocessing the fake news dataset

The preprocessing of the news is equivalent to the prepossessing of the Hippocorpus with the only exception that instead of the topic being `summary` it is now `meta-description`.

3.7.3 Neural language model analysis of the fake news dataset

The narrative flow analysis of the news is equivalent to that of the Hippocorpus, however, instead of evaluating with three different topics (summary of the story, main event in the story and an empty sequence) only the topic equivalent to the summary, which is meta-description, is used. Furthermore, whilst working with the Hippocorpus each story and its context (the topic as well as any preceding sentences, for the chain model, see 3.4) could be encoded as a sequence whose length did not exceed 1,024 token ids, which is the maximum length of a sequence that the GPT2 model can take as an input[7]. However, some of the news stories alongside the context exceeded this limit and had to be truncated to include only the first 1,024 token ids. A more refined solution would be sectioning the overflowing input into several lists of encodings where each previous list is used as the context for the next one. However, the truncation is the better solution for the scope of this project and for the purpose of reusing the code written originally for the Hippocorpus. Furthermore,

[7] `https://huggingface.co/transformers/model_doc/gpt2.html`

the truncation is applied to all stories, fake and reliable, so any effect it might have on the measurement of narrative flow should be present in both cases.

3.7.4 Implementation of evaluation module for the fake news

The evaluation of the metrics produced from the analysis of the news stories is performed via paired t-tests, just like with the stories from the Hippocorpus. However, due to the different reshaping process (using named entities for pairing) the news stories have a one-to-one mapping between fake and reliable news which makes the paired t-test routine a bit simpler, as there is no need to choose between evaluating with or without duplicates.

3.8 Summary of implementation

Throughout the implementation stage a modular system to analyse imagined vs. recalled stories was implemented. The stages of the implementation are preprocessing, analysis and statistical evaluation. After the main implementation meant for analysis of the Hippocorpus, the system was extended to also analyse a fake news corpus.

Chapter 4

Evaluation

The results from the analysis of the Hippocorpus are evaluated for statistical significance and discussed independently as well as in the context of Sap et al.'s results [1]. The results from the fake news analysis are presented and discussed in a similar manner.

4.1 Results

The results of the evaluation can be seen in table 4.1. The metrics which were produced and evaluated through paired t-tests are:

- Lexicon-based metrics:
  - LIWC metrics: analytic, tone, i, cogproc, negemo, posemo
  - average concreteness
- Neural-language-model-based metrics:
  - average narrative flow, using summary as the topic
  - average narrative flow, using the main event as the topic
  - average narrative flow, using an empty string as the topic

The statistical tools used to evaluate the metrics are t-statistic, which is used to decide whether or not to reject the null hypothesis, effect size (Cohen's d), which represents the magnitude of the effect of the treatment, p-value, which determines whether the results are statistically significant, and the direction of the effect.

4.1.1 Independent discussion of results

All of the results are statistically significant according to a predetermined threshold of p <0.1. The effect sizes are overall small (∼0.2), with the exception of the average narrative flow using summaries as the topic (which has a nearly medium effect size, closer to 0.5). It can be observed that using different types of topic (events and an empty sequence instead of summary) leads to smaller effect sizes, but remains statistically significant. Imagined stories flow more linearly (have higher narrative flow) than recalled

metric	**t-stat**	**t-stat**†	d	$d^\dagger$	**p-value**	**p-value**†	**direction**
avrg Δ_l (summary)	17	18	0.43	0.44	1.7×10^{-63}	1.5×10^{-71}	imagined
avrg Δ_l (event)	5.1	5.5	0.14	0.15	4.0×10^{-7}	4.3×10^{-8}	imagined
avrg Δ_l (empty)	4.5	4.5	0.12	0.12	8.6×10^{-6}	6.1×10^{-6}	imagined
concreteness	2.6	2.4	0.06	0.06	8.3×10^{-3}	1.5×10^{-2}	recalled
analytic	10	11	0.25	0.24	5.2×10^{-25}	1.8×10^{-25}	recalled
tone	5.8	5.9	0.12	0.12	1.0×10^{-8}	3.2×10^{-9}	imagined
i	8.2	8.2	0.16	0.16	2.7×10^{-16}	3.0×10^{-16}	imagined
posemo	9.9	9.8	0.23	0.21	9.4×10^{-23}	3.4×10^{-22}	imagined
negemo	2.9	2.9	0.06	0.06	3.4×10^{-3}	3.9×10^{-3}	imagined
cogproc	12	12	0.29	0.29	3.0×10^{-30}	4.0×10^{-33}	imagined

Table 4.1: T-statistic, effect size, p-value and direction of effect for the neural-language model-based (average narrative flow) and lexicon-based (concreteness, analytic, tone, i, posemo, negemo, cogproc) metrics obtained through paired t-tests († superscript denotes results from dataset when using duplicates).

stories, as they are based on semantic memory which stores logical information [6]. Imagined stories are also more self-centred (higher use of i-words) and more emotional (higher use of words related to positive and negative emotions and also higher score on the tone variable). Imagined stories also have a higher cognitive process score showing that they involve more challenging mental processing to compose. Recalled stories on the other hand are more concrete and score higher on the analytic parameter of LIWC.

4.1.2 Comparing results with Sap et al.'s results

The direction of the effect in all cases is as expected from the original paper and the results are in line with the cognitive theories and results of the original paper. However, the effect sizes in this work are smaller than the ones reported by Sap et al. [1]. In the case of the average narrative flow where the difference in the effect sizes is 0.08 (when using duplicates) and 0.09 (without duplicates) this could be explained by the use of GPT2 in this project instead of GPT, which was used in the original project. For the concreteness measurement Sap et al. report an effect size of 0.13 whereas the effect size in this project is 0.06, which is very small. This difference can be accounted for by the different implementations of the average concreteness measurement, as there was not much detail on it in the original paper so there was a significant amount of freedom for this calculation. For all of the LIWC metrics, there is either no difference between the effect sizes reported by Sap et al. or a difference in the order of 0.01, which could possibly be explained by the use of different hardware. Overall, the results in this project confirm the effects observed and reported by Sap et al. [1].

4.1.3 Discussion of concreteness

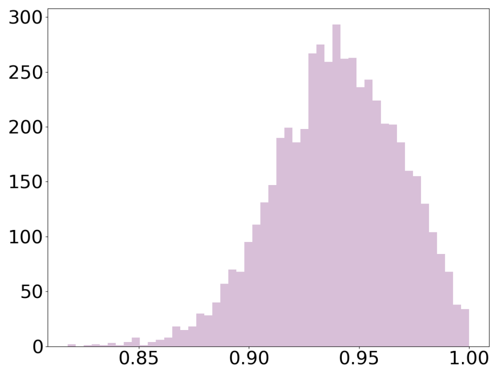


Figure 4.1: Percentage of total number of (semantic) words in a story found in the lexicon. The x-axis represents the percentage and the y-axis the number of stories corresponding to that percentage range.

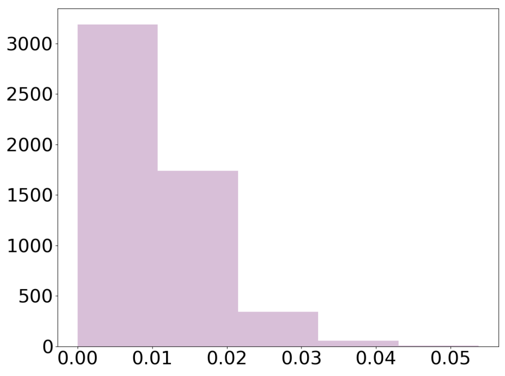


Figure 4.2: Percentage of bigrams out of total number of (semantic) words in a story. The x-axis represents the percentage and the y-axis the number of stories corresponding to that percentage range.

The concreteness score is lexicon-based so it is worth looking at how the vocabulary of the stories is represented in the lexicon. The ratio of the number of words from a story found in the concreteness lexicon to the total number of words in the story is illustrated in figure 4.1, where it can be seen that for the majority of stories at least 90% of the words in the story are found in the lexicon, which is a reasonable number. In this above investigation a word is taken to mean a preprocessed (lowercased, lemmatised, alphabetical) token. A notable peculiarity of the concreteness lexicon and measurement is that the lexicon contains bigrams (two word expressions) which are given a single score. It is challenging to decide whether to count bigrams as one or two words. If a bigram is counted as one word then the above described ratio would not be accurate as the overall word count in a story is a lexicographic word count rather than a semantic word count. If a bigram is counted as two words this would skew the average concreteness score as only one number between 1 and 5 is added to the ongoing sum but the overall count of words in the lexicon is increased by two. The approach chosen here is to count bigrams as one word and calculate a 'semantic' length of the story which is the lexicographic length of the story minus the number of bigrams detected in that story. Having mentioned the bigrams, the vast majority of stories contain between 0% and 4% bigrams out of all semantic words in the story, as can be seen in figure 4.2. However, 4,578 out of 5,329 examined stories do contain at least one bigram and it is important to note that for the purpose of this project a lexicographic (rather than semantic) parsing was used, so it is possible that some of the detected bigrams have occurred by chance. Thus, a future improvement could be using a semantic parser to detect those bigrams. Finally, it is worth mentioning that the average concreteness score has one of the lowest effect size values of 0.06 (tied with negative emotions) and also a relatively high p-value. This can also visually be perceived in figure 4.3 of all of the concreteness scores below: there is no clear distinction between the groups of imagined and recalled stories.

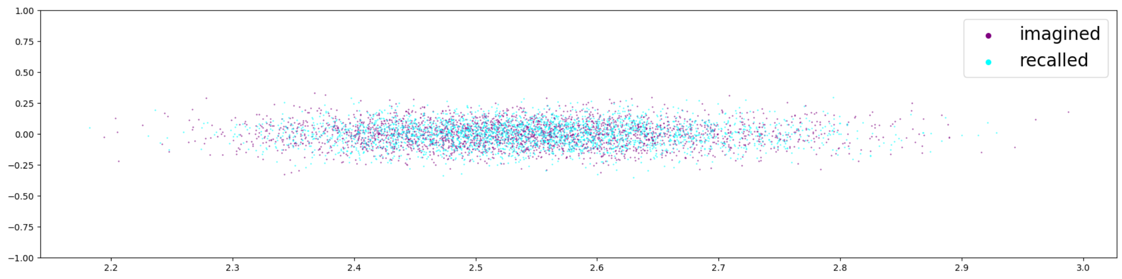
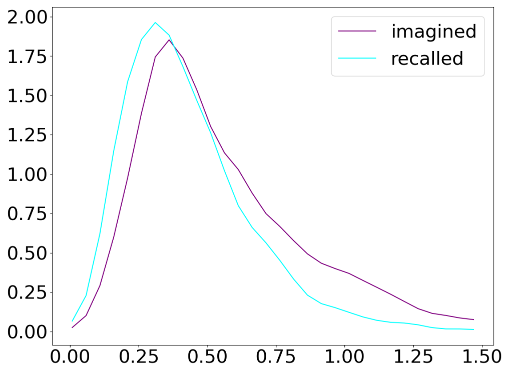


Figure 4.3: Average concreteness of the stories, recalled in cyan and imagined in purple, revealing there is not a drastic separation between the two categories.

4.1.4 Discussion of narrative flow

The average narrative flow of stories is higher for imagined stories regardless of the specific type of topic. However, the effect is most clearly observable when the topic is set to summary. As can be seen in figure 4.4 the recalled stories' average narrative flow is shifted more towards zero than the imagined stories: the imagined stories have higher narrative flow. The variation in topic results in a different effect size: the highest effect size is measured when the summary of the story is used as a topic, whereas when the main event of the story or an empty sequence are used the topic the effect size decreases, but remains significant. This is reflected in figure 4.5.



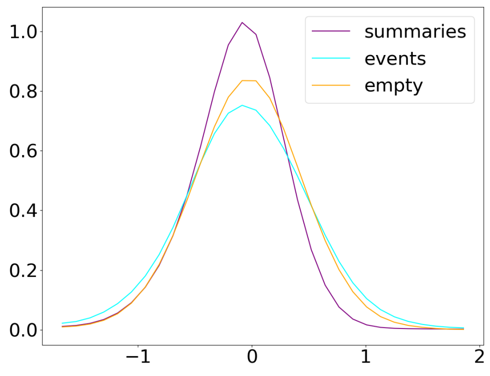


Figure 4.4: Density plot of the average narrative flow with topic set to summary, which reveals that imagined stories have a higher average narrative flow than recalled stories as their curve is more shifted to the right.

Figure 4.5: The difference between the average narrative flow of imagined and recalled stories with varying topics: summary, event or an empty string. The difference is most noticeable when using the summary as topic.

4.2 Fake news extension results

The results from the fake news extension were evaluated with paired t-tests just like the stories from the Hippocorpus and the effect size, p-value and direction of the effect can be seen in table 4.2. During the evaluation, the fake label is equated to imagined and the reliable label is equated to recalled for simplicity of notation.

metric	**t-stat**	d	**p-value**	**direction**	**expected**
avrg Δ_l (summary)	15	0.54	9.8×10^{-48}	recalled	imagined
concreteness	5.2	0.16	2.5×10^{-7}	imagined	recalled
analytic	13	0.42	1.1×10^{-34}	recalled	recalled
tone	7.0	0.23	3.9×10^{-12}	recalled	imagined
i	0.17	0.01	0.86	recalled	imagined
posemo	4.2	0.14	3.4×10^{-5}	recalled	imagined
negemo	7.4	0.25	2.2×10^{-13}	imagined	imagined
cogproc	0.42	0.01	0.67	imagined	imagined

Table 4.2: T-statistic, effect size, p-value and direction of effect for each metric of analysis for the fake news dataset obtained through paired t-tests.

The majority of metrics have an effect which differs from the expected (i.e. not equivalent to the effect directions from the evaluation of the Hippocoprus), with the exception of three of the LIWC metrics: analytic, negative emotions and cognitive processes. Two of the metrics, the I word count and the cognitive processes word count, have very high p-values which majorly exceed the predetermined threshold of significance of $p<0.01$, so at least for those categories it can be concluded that there is no significant difference between the fake and reliable news. The effect sizes and t-statistic for these metrics are also correspondingly low. The analytic metric has the lowest p-value and highest effect size of 0.42 (nearly medium according to Cohen's original classification) showing that there is a very significant difference in the number of words related to analytical thinking between fake and reliable news with a higher value for reliable news. This observed effect also has a direction of the effect as expected. The most significant result in the fake news analysis is of the narrative flow: it has a very low p-value and a medium effect size (0.54). However, the direction of the effect is towards the recalled (i.e. reliable) stories rather than towards the imagined (i.e. fake) stories. Thus, this measurement is not helpful in distinguishing between fake and reliable news in a way that is consistent with the cognitive theory.

4.2.1 Discussion of fake news extension results

The lexicon-based and neural language model-based analyses method developed for this project show little potential for distinguishing between fake and reliable news. The fake and reliable news used in this project are sourced from a web scraping dataset so many of the news stories contain unnecessary words or phrases such as 'Advertisement' or 'Click here to read the full article'. Cleaning the stories from such phrases would definitely lead to an improvement in the scientific validity of any findings, however developing a preprocessing step to perform this cleaning would be a challenging step and is beyond the time scope of this project. The current results, which have been obtained without this additional cleaning step, might still look promising at a first glance due to the statistically significant difference between the two groups, but should be approached with scepticism. For example, the negative emotion word count has a very low p-value and a considerable effect size with a direction equal to the expected. Thus, a naive conclusion is that a higher negative emotion word count can be used as an identifier for fake news stories. However, a deeper dive into the topic reveals that fake news are often written and shared with the goal of ruining someone's reputation, so it is logical that they would be quite negative and would rely on emotional affect rather than rational conclusions to convey their claim. Although the difference between the average narrative flow of fake and reliable stories is statistically significant, the observed effect is in a different direction from expected. More theoretical research on the topic and experimentation is necessary to be able to conclude the potency of narrative flow in identifying fake news. Some disadvantages of the method in this project that future work could improve on are: preprocessing the news stories to remove catchphrases such as 'Click here to read the full article', using human evaluators to pair the stories instead of the named entity algorithm, ensuring that the probabilities of every word in the story is used in the measurement rather than truncating to only the first 1,024 encoded tokens (see section 3.7.3). If a similar significant difference is observed after those points of improvement have been addressed and the result is reproducible on other datasets, narrative flow could turn out to have the unintentional power to detect fake news, albeit by detecting an effect in the wrong group than expected.

4.3 Summary of evaluation

The evaluation was performed using paired t-tests. The results from the analysis of the Hippocorpus were analysed independently and for all metrics there was a significant difference between the group of recalled and the group of imagined stories. The results in this project were compared with the results by Sap et al. [1] and their original findings were confirmed through this reimplementation. The results from the fake news analysis were also statistically evaluated, but not all metrics were significantly different between the group of fake and reliable news and the effect of the ones who were significant was rarely in the expected direction.

Chapter 5

Conclusion

The project was a success. The two success criteria (see section 2.7) were met and the fake news extension was implemented. Throughout the project a state-of-the-art NLP approach was used to address an intriguing issue in cognitive science. The results were in line with the cognitive theories discussed in the introduction and confirmed the original findings by Sap et al. [1].

5.1 Work performed

The selected aspects from the ACL 2020 paper by Sap et al. [1] were implemented and the Hippocorpus dataset was analysed as planned. Iterative improvements of the code, including a GPU optimisation of the code utilising the neural language model GPT2, were implemented to increase the efficiency. The data results were analysed independently and compared with the published results. The project execution was rooted in best software engineering practices: writing legible self-standing code, implementing unit tests, frequently backing up the project, actively communicating with the supervisors of the project, adhering to the predetermined schedule and making adjustments where necessary.

5.2 Lessons learned

Through the project I learned the importance of making notes about decisions made. For the duration of the project I kept an informal log describing everything I did for the project, from research to actual implementation, and although I write code to be self-explanatory, using descriptive names and writing comments where necessary, this log reminded me why I have made the choices that I have for the code, especially in instances when I was writing the dissertation and reviewing the code and some decisions seemed wrong at first glance. I did not enjoy keeping the log, but it was incredibly helpful during the write up as it reminded me what my decision process was. I will continue following this approach in my future work and would recommend it to anyone embarking on a long-term project. Another lesson learned is to be braver when approaching unfamiliar technology. I learned this through delaying the GPU optimisation of the code: I was worried it would be very challenging and time-consuming, and it would be easier to implement a non-optimised solution first. In hindsight it should have been part of my basic implementation as the ‘simpler’ solutions ate away considerably more time than the time necessary to understand and implement the computation for GPU. In the future I would attempt what seems to be the optimal implementation first.

5.3 Future work

A possible line of future work that was not explored in this project due to the lack of time is the proposed extension of analysing the Hippocorpus stories for realis and commonsense events and comparing the produced results with the ones in the original paper by Sap et al. [1]. Minor improvements were also suggested throughout the text: using a parser in the calculation of the average concreteness score to avoid scoring accidental bigrams (see section 3.3.2; analysing the entire context and length of the news stories instead of truncating to meet the limit of the GPT2 model (see section 3.7.3). Although the fake news analysis extension did not yield promising results, a reimplementation with the goal to confirm or refute my results could be a good idea, but the dataset should be manually or more carefully curated and the truncation problem should be solved. Finally, an attempt to measure the potency of the above measurements in a lie detector program could be implemented.

Bibliography

[1] Maarten Sap, Eric Horvitz, Yejin Choi, Noah A. Smith, and James Pennebaker. Recollection versus imagination: Exploring human memory and cognition via neural language models. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 1970–1978, Online, July 2020. Association for Computational Linguistics.

[2] Marc Brysbaert, Amy Beth Warriner, and Victor Kuperman. Concreteness ratings for 40 thousand generally known english word lemmas. *Behavior Research Methods*, 46(3):904–911, October 2013.

[3] Jerome Bruner. Telling stories: Language, narrative, and social life. chapter 3. Narrative, Culture, and Mind, pages 45–49. Georgetown University Press, 2010.

[4] Nancy L. Stein. The definition of a story. *Journal of Pragmatics*, 6(5-6):487–507, December 1982.

[5] Martin A Conway, Christopher W Pleydell-Pearce, Sharron E Whitecross, and Helen Sharpe. Neurophysiological correlates of memory for experienced and imagined events. *Neuropsychologia*, 41(3):334–340, 2003. Functional Neuroimaging of Memory.

[6] Endel Tulving. Episodic and semantic memory. In *Organization of memory*, chapter 10, pages 381–403. Academic Press, 1972.

[7] James Pennebaker, Ryan Boyd, Kayla Jordan, and Kate Blackburn. The development and psychometric properties of liwc 2015, 09 2015.

[8] Alec Radford and Karthik Narasimhan. Improving language understanding by generative pre-training. 2018.

[9] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.

[10] Matthew Sims, Jong Ho Park, and David Bamman. Literary event detection. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3623–3634, Florence, Italy, July 2019. Association for Computational Linguistics.

[11] Maarten Sap, Ronan Le Bras, Emily Allaway, Chandra Bhagavatula, Nicholas Lourie, Hannah Rashkin, Brendan Roof, Noah A. Smith, and Yejin Choi. ATOMIC: An atlas of machine commonsense for if-then reasoning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33:3027–3035, July 2019.

[12] Lewis Pollock. Statistical and methodological problems with concreteness and other semantic variables: A list memory experiment case study. *Behavior Research Methods*, 50(3):1198–1216, July 2017.

[13] Matthew Honnibal, Ines Montani, Sofie Van Landeghem, and Adriane Boyd. spaCy: Industrial-strength Natural Language Processing in Python, 2020.

[14] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2018.

[15] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online, October 2020. Association for Computational Linguistics.

[16] *Discovering statistics using R.* SAGE Publications, 2012.

[17] Jacob Cohen. *Statistical Power Analysis for the Behavioral Sciences.* 2nd edition, 1988.

[18] Daniel Jurafsky James H. Martin. Speech and language processing. 3rd edition, draft, 2020.

Appendix A

Dataset contents

A.1 Hippocorpus 'README' file [1]

This CSV file contains all the stories in Hippcorpus v2 (6854 stories)
Breakdown by type of story:
‘‘‘
recalled 2779
imagined 2756
retold 1319
Name: memType, dtype: int64
‘‘‘

These are the columns in the file:

- 'AssignmentId': Unique ID of this story
- 'WorkTimeInSeconds': Time in seconds that it took the worker to do the entire HIT (reading instructions, storywriting, questions)
- 'WorkerId': Unique ID of the worker (random string, not MTurk worker ID)
- 'annotatorAge': Lower limit of the age bucket of the worker. Buckets are: 18-24, 25-29, 30-34, 35-39, 40-44, 45-49, 50-54, 55+
- 'annotatorGender': Gender of the worker
- 'annotatorRace': Race/ethnicity of the worker
- 'distracted': How distracted were you while writing your story? (5-point Likert)
- 'draining': How taxing/draining was writing for you emotionally? (5-point Likert)
- 'frequency': How often do you think about or talk about this event? (5-point Likert)
- 'importance': How impactful, important, or personal is this story/this event to you? (5-point Likert)

- ‘logTimeSinceEvent’: Log of time (days) since the recalled event happened
- ‘mainEvent’: Short phrase describing the main event described
- ‘memType’: Type of story (recalled, imagined, retold)
- ‘mostSurprising’: Short phrase describing what the most surpring aspect of the story was
- ‘openness’: Continuous variable representing the openness to experience of the worker
- ‘recAgnPairId’: ID of the recalled story that corresponds to this retold story (null for imagined stories). Group on this variable to get the recalled-retold pairs.
- ‘recImgPairId’: ID of the recalled story that corresponds to this imagined story (null for retold stories). Group on this variable to get the recalled-imagined pairs.
- ‘similarity’: How similar to your life does this event/story feel to you? (5-point Likert)
- ‘similarityReason’: Free text annotation of similarity
- ‘story’: Story about the imagined or recalled event (15-25 sentences)
- ‘stressful’: How stressful was this writing task? (5-point Likert)
- ‘summary’: Summary of the events in the story (1-3 sentences)
- ‘timeSinceEvent’: Time (num. days) since the recalled event happened

A.2 Brysbaert’s concreteness lexicon [2]

The 8 columns:

- Word: The word
- Bigram: Whether it is a single word or a two-word expression (bigram)
- Conc_M: The mean concreteness rating
- Conc.SD: The standard deviation of the concreteness ratings
- Unknown: The number of persons indicating they did not know the word
- Total: The total number of persons who rated the word
- Percent_known: Percentage participants who knew the word
- SUBTLEX: The SUBTLEX-US frequency count (on a total of 51million; Brysbaert & New, 2009) “average concreteness level of words in stories” for

Appendix B

Project Proposal

Computer Science Tripos Part II Project Proposal

Neural language modelling for differentiating between texts of imagined versus recalled stories

2423B
Originator: 2423B
22$^{\text{nd}}$ October, 2020

Project overseers: Prof. Alan Mycroft and Dr. Cengiz Oztireli
Project Supervisors: Prof. Paula Buttery and Dr. Andrew Caines

Introduction

I am going to reimplement aspects of the following paper presented at ACL 2020: "Recollection versus Imagination: Exploring Human Memory and Cognition via Neural Language Models" by Maarten Sap, Eric Horvitz, Yejin Choi, Noah A. Smith and James W. Pennebaker.
I will investigate the differences between imagined and recalled stories in the Hippocorpus dataset. The underlying psychological theory is that imagination and recollection are based on different cognitive processes and structures: to imagine a story we use semantic memory, which is responsible for recalling common-sense ideas and facts (e.g. "terrorism is a sad phenomenon"), whereas to tell a story that happened to us, we use episodic memory, which is responsible for recalling personal facts (e.g. "I was having a chocolate chip muffin alongside my coffee when I saw the smoke and heard the sirens"). Since imagination and recollection are based on different cognitive processes, we expect the manifestation of those processes, in this case storytelling, to carry traces of those differences.
Thus, the first hypothesis is that imagined stories are going to be based on general knowledge, whereas recalled stories are going to contain more concrete, personal knowledge.

The second hypothesis is that imagined stories are going to flow more linearly, since their content is built on common-sense relations of events, whereas recalled stories will be less linear.
In order to investigate those questions, different methods of analyses can be used. The first line of work is doing lexical and stylistic analysis using tools such as the Linguistic Inquiry Word Count and Brysbaert's concreteness lexicon. The second line of work is analysing the narrative flow of the stories: this step involves using OpenAI's GPT model to find the probability of a sentence occurring in its context, where the context can be defined using two models, the bag model and the chain model about which I explain more below in "Work to be done. Analysis" .

Starting point

Having paid special attention to them, I believe the following University courses will be useful: Scientific Computing, Machine Learning and Real World Data, Foundations of Data Science, Formal Models of Language, Natural Language Processing and Artificial Intelligence. I have had experience working with Python over the last two summers: in 2019 I worked in an enormous codebase and learned the importance of testing, whereas in 2020 I implemented an NLP classification project in Python and gained specific knowledge of some helpful libraries, like spaCy for lemmatization, nltk and so on.
As for my linguistic background, over the summer I read the textbook *Linguistics: An Introduction* by Andrew Radford et al. and took a Linguistics course called *Miracles of Human Language* on Coursera.
I have limited knowledge of working with neural networks. I have taken a course on Coursera called *Natural Language Processing with TensorFlow.* I am aware I have much more to learn in this area and I am prepared to take more online courses that seem appropriate, as well as going back to my notes on Artificial Intelligence for a more fundamental understanding.
I have read the paper *Recollection versus Imagination: Exploring Human Memory and Cognition via Neural Language Model* on which my project is based and have taken note of the structure of the Hippocorpus.
As for my environment, I have PyCharm installed, but need to do some research on the libraries I am going to use and install them.

Work to be done

Research

I am going to explore the Hippocorpus and the tools that I am going to be using in the next steps. These include but may not be limited to: Linguistic Inquiry Word Count and Brysbaert's concreteness lexicon for the lexical and stylistic analysis and OpenAI's GPT-2 (or 3 if my request to use it is approved) for language modeling and estimating the probabilities of a sentence appearing in its context.

Environment

I am planning to use PyCharm, which I have installed. I need to install spaCy, numpy, scipy, matplotlib and other useful libraries. I need to choose a testing framework for Python and install it. For the more costly processing I am going to use the HPC by pushing the Python code and running it there.

Preprocessing

I need to preprocess the data in the Hippocorpus. This includes lowercasing, lemmatizing, removing stopwords and punctuation and normalizing and tokenizing the data. Alongside supporting data, such as the labels of the stories [‘imagined’, ‘recalled’, ‘retold’][1], the Hippocorpus contains 6,854 stories that are non-processed: they are in the form in which they were written by the participants in the creation of the corpus. Preprocessing is a standard NLP step which aims to clean the data from features that do not usually carry a lot of information. There are various libraries available to do preprocessing in Python: I am going to use spaCy because it is currently the fastest library. My backup would be nltk which is slower than spaCy but allows more freedom in how to do the preprocessing. With spaCy I would be able to do the lowercasing, lemmatizing, removal of stopwords and punctuation, normalizing and tokenizing. The paper I am basing my project off of does not explicitly talk about preprocessing so this is an area where I would have to make decisions myself. For example, when normalizing the word spellings (e.g. I want “color” and “colour” to be recognized as the same word) I would have to choose which spelling to stay and replace the instances of the other. Once the preprocessing is done, the output from it (i.e. the tokenized and “cleaned” data) becomes the input for all future modules.

Analysis

I am going to analyse the Hippocorpus dataset using two main approaches:
Firstly, I am going to use lexical and stylistic measures to analyse the dataset. This step would involve using pre-existing resources to gather statistics on the cognitive and linguistic information present at the lexical level within the stories. The two resources I am going to be using at this stage are:

- Linguistic Inquiry Word Count (LIWC)
  - Background: LIWC is a tool which provides a psychological analysis of a text, basing its findings on percentages of words reflecting a number of specific dimensions (e.g. negative emotion).
  - Task: Analyse the stories with respect to words related to cognitive processes, positive or negative emotions and I-word categories. Pay attention to the ANALYTIC and TONE variables, too. I am going to run the stories through the LIWC processing unit and analyse the numerical results.

[1]I am only going to use the ‘recalled’ and ‘imagined’ stories. There is a whole section of the paper dedicated to the ‘retold’ stories, namely “5 Narrativization of Recalled Stories” , but I am not going to reimplement it in this project.

- Brysbaert's concreteness lexicon:
  - Background: Brysbaert's lexicon contains approximately 40 000 well known English words or expressions in dictionary form (i.e. lemmas) rated for their concreteness by 4000 participants, where concreteness is defined in the original paper as "the degree to which the concept denoted by a word refers to a perceptible entity" .
  - Task: Estimate the concreteness level of words in the stories. After preprocessing the data, I am going to implement a routine that finds the concreteness score of each word in a story, given that it is present in Brysbaert's lexicon, and sums those scores up to calculate the overall concreteness score of the story. I need to decide a strategy that accounts for words not present in Brysbaert's lexicon as this is not discussed in the paper.

Secondly, I am going to use the measurement of narrative flow, as defined in the ACL 2020 paper, to distinguish between the two categories:

- Background: Narrative flow / narrative linearity is based on the likelihood of a sentence appearing in a given context. It is measured by comparing the likelihood of a given sentence appearing in its context in two generative models: a bag model and a chain model. In the bag model, the sentences are drawn independently based solely on the context, which consists of the short summary attached to every story. In the chain model, only the first sentence is generated based on the context and each following sentence is based on the previous one.
- Task: I need to calculate the linearity for each story by using Open AI's GPT model to estimate the likelihood of sentences with each of the two models. More specifically, the linearity calculation is defined as Δ_1, the difference in the negative log-likelihoods between the bag and chain models. Within those, the log-probability of each sentence in context is the sum of the log-probabilities of its tokens in context. For the context I am going to use the short summaries included alongside every story in the Hippocorpus.

Evaluation

There are two stages to the evaluation: the first is the comparison and analysis of the texts using the metrics described above; the second is the comparison of results with the original paper.

Evaluation of the lexical and stylistic analysis

I am going to investigate the differences between the results of the LIWC and concreteness analysis found in recalled vs. imagined stories. I am going to look for differences in the count of different word categories (cognitive processes, positive emotions, negative emotions, I-words ANALYTIC variable, TONE variable) as well as for differences in the concreteness scores.

Evaluation of narrative flow (linearity) analysis

I am going to compare the Δ_l (defined in the analysis section) averaged over the entire story and look for statistically significant differences in the scores of the recalled and the imagined stories. For both of the analyses, I am going to perform paired t-tests for each

numerical variable. For example, let the variable under examination be the count of I-words produced by LIWC: I am going to average this metric over all recalled stories and then over all imagined stories and then calculate the t-value to establish if differences in the results are statistically significant.

Comparison of results with the original paper

After I've completed the evaluation independently, I am going to compare my results to the ones described in the paper. I am aiming to follow the same methodology, at least where it has been explicitly described in the paper, so I will see whether I have replicated the original findings or if I have not, what might be the reason for the differences.

Testing

I am going to write unit tests for my functions to ensure they do what they are supposed to. I am planning to use a testing framework where I can mock the data. For the preprocessing functions, I am going to prepare mock sentences or texts for which I know what the preprocessed output would be. For the calculations with Brysbaert's lexicon, I am going to prepare mock examples for which I have manually calculated and verified the scores using the lexicon to ensure the tokenized stories are correctly evaluated for concreteness. For the likelihood estimates from the Open AI GPT model, I am again going to test the overall function correctness by preparing an example for which I will manually calculate the narrative flow.

Success criteria

The project would be a success if:

- I have reimplemented the aspects of the paper described above using NLP techniques, performed the analysis of the dataset through the two main approaches and statistically evaluated the results.
- Differences between my work and the ACL 2020 paper have been accounted for.

Possible extensions

Episodic vs. semantic memory analysis

- Background: Episodic memory allows us to recall personal facts, whereas semantic memory allows us to recall common-sense facts. Realis events are associated with episodic memory because when the author of a story talks about a realis event, they claim it has happened, e.g. "I washed the dishes" , whereas irrealis events are more hypothetical, e.g. "I should have washed the dishes" . Common-sense knowledge is associated with semantic memory when discussing it in the context of social events.
- Task: Count realis events using BERT and the Literary event detection tagger by Sims et al. Count common-sense knowledge using the social common-sense knowledge graph ATOMIC.

Analysing a fake news corpus

- Background: Episodic memory allows us to recall personal facts, whereas semantic memory allows us to recall common-sense facts. Fake news is quite a big challenge in the social media age and if the above evaluations prove to be useful in distinguishing between fake and accurate news, this would be yet another tool to fight against misinformation.
- Task: Using the LIWC, Brysbaert's lexicon, the measurement of narrative flow and the episodic vs. semantic memory investigation, analyse the news in the following fake news corpus:`https://github.com/several27/FakeNewsCorpus`. Evaluate the results and conclude if there are statistically significant differences between fake and correct news.

Timetable, milestones, deliverables

26th October - 8th November *Michaelmas. NLP 1 due.*

- Task: Data exploration. Research the tools, install where necessary and verify that it works. Set up Git repository. Log tool and design choices in the Preparation chapter. Start a bibliography in Overleaf.
- Milestone: Git repository set-up. Environment ready for work.
- Deliverable: Git repository.

9th November - 22nd November *Michaelmas. NLP 2 due.*

- Task: Preprocessing. Log the details of the preprocessing procedure in the Preparation chapter.
- Milestone: Preprocessing is complete and tokenized data is ready to be used in the following modules.

- Deliverable: Preprocessed data.

23rd November - 6th December *NLP 3 due. End of Michaelmas. 3-5 days completely off.*

- Task: Lexical and stylistic analysis. Log the details of the lexical and stylistic analysis implementation in the Implementation chapter.

7th December - 20th December *Writing revision notes for Michaelmas courses.*

- Task: Narrative flow analysis. Log the details of the narrative flow analysis implementation in the Implementation chapter.

21st December - 3rd January *Probably bach home, some days completely off.*

- Break from dissertation or catch-up if necessary.

4th January - 17th January

- Task: Evaluation of the results of the analysis. Log the evaluation process in the Evaluation chapter.
- Milestone: Implementation of regular (non-extension goals) completed.

18th January - 31st January *Lent.*

- Task: Evaluation of the results of the analysis (continued). Log the evaluation process in the Evaluation chapter.
- Milestone: Evaluation of the main implementation completed. Success criteria is met.

1st February - 14th February *Lent.*

- Task: Complete Progress Report. Begin the implementation of extension goal 1: episodic vs. semantic memory analysis.
- Milestone: Progress Report (5th February)
- Deliverable: Progress Report

15th February - 28th February *Lent.*

- Task: Episodic vs. semantic memory analysis (continued). Log the details of the episodic vs. semantic memory analysis implementation in the Implementation chapter.

1st March - 14th March *Lent.*

- Task: Evaluation of the results of the extension. Log the evaluation process in the Evaluation chapter.

15th March - 28th March *End of Lent. 3-5 days completely off.*

- Task: Wrap up any task in progress. Ensure code is in presentable state.
- Milestone: Implementation completed.
- Deliverable: Git repository containing all the code for the project.

29th March- 4th April

- Task: Writing the dissertation.

5th April - 18th April

- Task: Writing the dissertation.

19th April - 2nd May *Easter.*

- Task: Writing the dissertation. Address comments by supervisors and DoS.
- Milestone (by 23rd April): Dissertation completed and passed on to supervisors, DoS and any other volunteers.
- Deliverable: Final draft of dissertation.

3rd May - 14th May *Easter.* *Final deadline: 14th May*

- Task: Address comments by supervisors and DoS (continued).
- Milestone (by 7th May): Dissertation submitted.
- Deliverable: Dissertation.

Resource declaration

I will use my personal laptop (ASUS UX310U: 7th generation Intel core i7, 2.7 GHz CPU, 8 GB RAM, Windows 10 Pro) for researching, coding and writing up the dissertation. It is equipped with the McAfee antivirus provided by the university. I am going to use online services to store the project: Overleaf for the dissertation, Google Docs for a logbook, Google Drive for other textual files and GitHub for the code. I will perform weekly backups of all files onto a Seagate 1TB hard-drive so that I have my files locally in the unlikely event of my online documents being corrupted. In the case of a problem with my laptop, I will have everything online and on my hard drive and I will purchase a new laptop to continue my work. I accept full responsibility for this machine and I have made contingency plans to protect myself against hardware and/or software failure.

Software and corpus resources:

- For the code I am going to use Python and some of its relevant NLP Packages.
- I will also need the Hippocorpus dataset, which is available online at Microsoft's datasets. The dataset is only 12MB, so storage locally and online is not an issue.

- The Linguistic Inquiry Word Count needs to be purchased. It costs £85.
- Brysbaert's lexicon is available online and is only 2MB, so not a problem for storage.
- I have applied for access to OpenAI's GPT-3, but in the likely event that I am not approved for access to it, I will use GPT-2 which is freely available online.
- I have received access to the university's high performance computing facilities, which I plan to use for some more costly computations.

Helpful links

- ACL 2020 paper: `http://erichorvitz.com/cognitive_studies_narrative.pdf`
- ATOMIC: `https://aaai.org/ojs/index.php/AAAI/article/view/4160`
- BERT: `https://www.aclweb.org/anthology/N19-1423/`
- Brysbaert's lexicon:
  `https://link.springer.com/article/10.3758/s13428-013-0403-5#MOESM1`
- Hippocorpus:
  `https://msropendata.com/datasets/0a83fb6f-a759-4a17-aaa2-fbac84577318`
- Literary event detection: `https://www.aclweb.org/anthology/P19-1353/`
- LIWC: `https://liwc.wpengine.com/`
- OpenAI GPT-2:
  `https://huggingface.co/transformers/model_doc/gpt2.html`