University *of Ljubljana*
Faculty *of Computer and Information Science*

Literacy situation models knowledge base creation

Jan Bajt, Anže Habjan, and Tadej Stanonik

Abstract

In this project the main goal was to find the family relations between the characters in Game of Thrones books series and visualize the relations in a form of a graph. This was a challenging task because of the high number of different characters and the complexity of the relationships between them. To achieve our main goal we used different natural language processing techniques like named entity recognition, coreference resolution and relation extraction. We build a pipeline which we used to process each book separately and at the end we gathered the results together and evaluate them. For named entity recognition we used three different models: NLTK, Stanza and Spacy. For the coreference resolution we used transformer based Neuralcoref model and for the family relations extraction step we used 2 different models, CoreNLP and LUKE. We compared the results of each family extraction method and named entity recognition. The results exposed some weaknesses of our approach and showed some room for later improvements.

Keywords

named entity recognition, coreference resolution, family relationship extraction, natural language processing, LUKE, CoreNLP, neuralcoref

Advisors: Slavko Žitnik

Introduction

The creation of a comprehensive knowledge base plays a crucial role in understanding and improving literacy situations. In this project, we employ Natural Language Processing (NLP) techniques to automatically extract and analyze the family relations within the Game of Thrones books. To tackle this challenge, we rely on a combination of NLP techniques, including named entity recognition (NER) to identify character names and co-reference resolution to handle pronoun references within conversations.

The literary work commonly recognized as a television show, *Game of Thrones*, originates from a collection of five books authored by George R. R. Martin. Each book features a copious number of characters who are all affiliated with a particular noble house (although this affiliation is not always precise). The Game of Thrones book series is renowned for its intricate and convoluted family relationships, which have captivated readers around the globe. Unraveling these complex ties is a challenging task, given the sheer volume of characters and the interplay of alliances, rivalries, and hidden lineages. Thus, this compilation of books offers an ideal resource for evaluating various techniques of named entity recognition and the extraction of family relationships from the identified entities.

Related work

In article [1] the authors tested and explained different NER libraries including Python's SpaCy, Apache OpenNLP, and TensorFlow. The comparison of these libraries is done based on training accuracy, F-score, prediction time, model size, and ease of training. They discovered that SpaCy gives better and accurate results as compared to Apache OpenNLP and TensorFlow when it comes to identifying entities in the input text. The prediction time of the Spacy model is also better, which indicates that Spacy is the best NER algorithm that was tested in this article.

In article [2] the authors compared the performances of different NER algorithms on old and modern novels. They discovered that although many studies on information extraction from literature typically focus on 19th and early 20th century source material there are no significant differences between old and modern novels but both are subject to a large amount of variance. They also found out that novels written in 3rd person perspective perform significantly better than those written in 1st person. From found entities they created social networks and performed network analysis by observing multiple network features.

There are several NLP techniques used to extract family relationships from text, including rule-based methods

and machine learning algorithms. Rule-based methods, like CustRE[3], involve the creation of hand-crafted rules that specify patterns or keywords that indicate family relationships. These rules can be applied to text data to identify relationships between individuals. CustRE is a rule based system, that uses regular expressions for pattern matching to extract family relations explicitly mentioned in text, and uses co-reference and propagation rules to extract family relations implicitly implied in the text. However, these methods can be limited by the complexity and variability of natural language, making it difficult to capture all possible variations of family relationships. Machine learning algorithms, on the other hand, use statistical models to learn patterns in the data and automatically identify family relationships. These algorithms are trained on large datasets of annotated text, where each relationship is labeled with a specific category (e.g., parent-child, sibling, grandparent-grandchild). The model then uses these labeled examples to identify patterns in the data and predict relationships in unseen text.

Methods

In this section we represent our pipeline and methods used for extracting the family relations between characters in Game of Thrones books series. The main steps of our pipeline are:

1. data collection (0.1)
2. basic preprocessing (0.2),
3. alias replacement (0.3)
4. coreference resolution (0.4),
5. named entity recognition (0.5)
6. family relation extraction (0.6).

The Game of thrones series consists of 5 different books. We used this pipeline to process each book separately. The result of this pipeline were the triplets (person;relation;person) obtained at the last step of the pipeline. At the end we grouped the results from all books together and evaluated the results.

0.1 Data collection

At first step we needed to get data about characters from Game of thrones book series. We found such data on *A Wiki of Ice and Fire* [4]. There we found information about all the characters from Game of thrones books which we scraped using *Selenium* [5] and *undetected-chromedriver* [6]. For each character on a character list from wiki page we scraped the data from *info-box* which contained key-value pairs summarizing important data about the character like the character's aliases, titles, full name, relatives (mother, father) and many other properties. We scraped the whole info-box and save it to *json* format file which we later used in our pipeline. We ended up with 3757 different characters and each of them had a dictionary of key-value pairs of scraped properties from character's info-box.

0.2 Basic preprocessing

As already mentioned, each book was processed separately. First we removed all parts from the book that are not part of the story, which means that we removed the text before the *Prologue* and the text after the last chapter. Then we performed some basic preprocessing. We noticed that the remaining text contained some characters that were not in the right format and thus suitable for further processing. So we decided to replace the characters like curly double and single quotes (“”, ‘’) with normal double and single quotes(””, ”) etc. We also removed all empty lines between the paragraphs and chapters. We also removed some of the contractions (’m, ’re, ’ll, etc.) with their full form (am, are, will etc.), because during development we noticed that these kind of contraction impacted the named entity recognition and our evaluation process.

0.3 Alias replacement

After this basic preprocessing was done we replaced all the aliases for each character. This was done by looping through each character's aliases in scraped data and replace each one of them with their real name. During the development of our pipeline we noticed some problems with our alias replacement process which was caused by the complexity of the data that we scraped. We found out that on the first sight easy task of replacing the aliases is not as simple as we thought. When we were replacing aliases with character's real names we encountered some character aliases that were the same as some of the character's real name. Such aliases were for example *Aemon Targaryen* and *Arya Stark*. The first one was not a big problem since Aemon Targaryen is not really a main character, but in case of Arya Stark this was a big problem since Arya Stark is one of the main characters in the series. This meant that all the occurances of real Arya Stark were substituted with the *Jeyne Poole* which has Arya Stark as alias since she pretended to be Arya Stark in a story. Because of that we had to ignore replacing such aliases that were the same as the real name of the character. Another problem were same names which belonged to different persons. Such an example is the name *Aegon Targaryen* which is the name of several different characters (most of them are only mentioned in the books). Because of such case we substituted the aliases with their full name and short distinctive description that we also scraped from wiki page (e.g. *Aegon Targaryen (son of Jaehaerys I)*. We also encountered examples where alias was used as a part of the character full name which led to wrong replacement. For example full name of the character is *Eddard Stark* and one of the aliases is *Ned*. A few times the character was mentioned as *Ned Stark*, which resulted in *Eddard Stark Stark* after alias replacement. Such cases are inevitable and we didn't try to correct them.

After that was done we also split the text of each book to chapters which we later used in coreference resolution step of the pipeline.

0.4 Coreference resolution

Coreference resolution is the task of finding all expressions that refer to the same entity in a text. It is an important step for a lot of higher level NLP tasks that involve natural language understanding such as document summarization, question answering, and information extraction [7]. This process can be challenging due to the complexity of language and the many ways that entities can be referred to in text. For example, an entity may be referred to using different names, pronouns, or descriptions throughout a text. Additionally, some expressions may be ambiguous and could refer to multiple entities.

Because characters can be represented with multiple different aliases, we decided to first replace these aliases with an actual name of the character. That way, each character is always referred to with only one name, which makes coreference resolution algorithms perform better. Removing aliases also helps NER algorithms find characters, because otherwise the algorithm would find a new entity for each alias of the same character.

The next step was to replace pronouns (such as 'he' or 'she') with the names of entities that these pronouns represent.

0.4.1 NeuralCoref

For Coreference Resolution we used NeuralCoref [8], a pipeline extension for spaCy 2.1+. NeuralCoref is a state-of-the-art model for coreference resolution, which is the task of determining the relationships between words or phrases that refer to the same entity in a text. Developed by the researchers at Hugging Face, NeuralCoref is based on neural network architectures. By resolving coreference, NeuralCoref enhances the coherence and understanding of text, improving the performance of NLP applications.

0.5 Named Entity Recognition (NER)

Named entity recognition (NER) is the task of identifying and classifying named entities in text, such as people, organizations, locations, and dates. NER is an important component of many natural language processing applications, such as information extraction, machine translation, and question answering. However, NER is a challenging task for machines because named entities can appear in many different forms and contexts, and there can be ambiguity in the interpretation of entity mentions. For example, a person's name may be misspelled, abbreviated, or referred to using a nickname or title. In our project we used three differnet NER models which are described in the next subsections.

0.5.1 NLTK

The Natural Language Toolkit (NLTK) [9] is a Python library designed for natural language processing. It provides a wide range of tools and resources for tasks such as tokenization, stemming, and part-of-speech tagging. NLTK also offers pretrained models and algorithms for sentiment analysis, topic modeling, and text classification. It supports multiple languages and integrates with other Python libraries such as NumPy and SciPy.

0.5.2 Spacy

Spacy [10] is an open-source Python library used for natural language processing. It offers tools for tokenization, part-of-speech tagging, named entity recognition, and dependency parsing. Spacy has pre-trained models for multiple languages and also allows for custom model training. It is fast and memory efficient, making it ideal for large-scale natural language processing tasks. Spacy is widely used in academia, industry, and government for various natural language processing applications.

0.5.3 Stanza

Stanza [11] is a Python natural language analysis package. It contains tools, which can be used in a pipeline, to convert a string containing human language text into lists of sentences and words, to generate base forms of those words, their parts of speech and morphological features, to give a syntactic structure dependency parse, and to recognize named entities. The toolkit is designed to be parallel among more than 70 languages. It is built with highly accurate neural network components that also enable efficient training and evaluation with your own annotated data. The modules are built on top of the PyTorch library.

0.6 Family relationship extraction

Relationship extraction is the task of identifying and extracting relationships between entities in text. These relationships can take many different forms, such as family relationships, social relationships, organizational relationships, and spatial relationships. Relationship extraction is important for many applications, such as social network analysis, event extraction, and knowledge graph construction. However, relationship extraction is a difficult task for machines because it requires understanding the complex semantics and context of natural language, as well as dealing with the variability and ambiguity in the expression of relationships.

For the characters that we found with NER, we wanted to retrieve the family relationships between them. We used two different methods to do that: CoreNLP and LUKE. The result for each one of them was a list of triplets of form *character;relation;character* where the relation between the two characters was one of four possibilities:

- per:children,
- per:parents,
- per:siblings and
- per:spouse

0.6.1 Stanford CoreNLP

For relation extraction in our project we used the Stanford CoreNLP [12] which enables users to derive linguistic annotations for text, including token and sentence boundaries, parts of speech, named entities, numeric and time values, dependency and constituency parses, coreference, sentiment, quote attributions, and relations. It is implemented in Java, so we

first needed to download CoreNLP from their download page [13]. After that we started a CoreNLP server to which we were sending requests with *knowledge base population* annotator parameter. This annotator is suited for entity linking which we need. The response gave us annotations for each relation that CoreNLP detected between two entities. There are many different relations possible between entities. On a higher level we can group them by the type of the entity (organisation, person etc.). We were interested only in family relations between persons, so we filtered out the non-family relations. We ended up with all the relations that were marked as *per:children, per:parents, per:siblings* and *per:spouse*. With that process we obtained the final results which we then used in the last evaluation step.

0.6.2 LUKE

LUKE [14] is a pretrained contextualized representation of words and entities based on a bidirectional transformer. The model proposed by Yamada et al. [15] treats words and entities in a given text as independent tokens, and outputs contextualized representations of them. Model is trained using a new pretraining task based on the masked language model of BERT and achieves impressive empirical performance on a wide range of entity-related tasks such as entity typing, relation classification, named entity recognition and question answering.

Results

0.7 Coreference resolution

When trying to define family relationships, coreference resolution is an important step. This procedure finds all pronouns that refer to the same character and replaces them with the name of the character. If the coreference resolution algorithm is good, it can make family relationship extraction much easier.

As we can see in example below, the NeuralCoref does replace some of the pronouns correctly, but also produces some mistakes. The algorithm replaced "him" by "someone" instead of "Bran".

Input text: *"Bran closed his third eye and opened the other two, the old two, the blind two. In the dark place all men were blind. But someone was holding him. He could feel arms around him, the warmth of a body snuggled close."*

NeuralCoref: *"Bran closed Bran third eye and opened the other two, the old two, the blind two. In the dark place all men were blind. But someone was holding someone. someone could feel arms around someone, the warmth of a body snuggled close."*

This kind of mistakes can lead to extraction of false family relationships, because if a pronoun is replaced by wrong character name, that could mean, that family relationship extraction algorithm will assume relationship between incorrect characters. Mistakes like that, can then lead to large number of false positive cases in the process of evaluation.

0.8 Family relations extraction

Family relationship extraction was evaluated in terms of standard evaluation metrics, *i. e.* precision, recall, and F1 score based on the ground truth character relationships. Initially, all possible characters and their relations were included in the ground truth, but at later stages of development we stumbled up on some inconsistencies of character relationships in our data. We therefore decided to handpick 22 most important characters by a subjective measure. With this slight modification we not only ensured that our grand truth is exact and manually verified but also removed relations otherwise barely (if even) mentioned in the books which would therefore be difficult for our model to recognize. Note that although this restriction might seem like a shortcut, in reality our model only has the capacity to recognize relationships derived directly from the text, not the relationships that can be derived based on logical reasoning. Hence our character downsizing is not just justifiable, it also substantially improves the relationship extraction scores.

0.8.1 CoreNLP

One of the methods we used for the family relations extraction is CoreNLP which is described in 0.6. Model is performing entity linking for each sentence and only family relations are extracted from all the links obtained. As we can see in the results in 1 the results are not very good. We can see very low precision and much higher but still quite low recall values. This is because CoreNLP found many relations with person pronouns (he, she, it etc.) which we were not able to match with any of the characters. This is because of bad performance and results of the coreference resolution. Many of the pronouns were not substituted with a person's name as expected and thus this affected the results of the CoreNLP relation annotator as well.

Table 1. Evaluation results of family relationship extraction using CoreNLP.

TP	FP	FN	Precision	Recall	F1
51	1041	84	0.05	0.38	0.08

0.8.2 LUKE

LUKE is implemented as a transformer, and in our case a pretrained model is imported by the authors of [15], same model is also used as a tokenizer. In order to predict relationships between entities (not restricted to only family relationships) in a given text, two query entities must be provided, as opposed to coreNLP. The entity extraction was done using 3 different entity extraction approaches mentioned in Section 0.5. Model is however not predicting relationships from each and every sentence in the book that includes two entities, but rather from sentences that include at least one of the relationship keywords provided by the authors of [3]. In case of more than two entities occurring in such sentence, the ones closest to the position of relationship keywords are chosen.

Results of LUKE are presented in table 2. As we need to provide entities, we implemented three different named entity

recognition methods under the hood of LUKE to observe the difference and found the favourable one.

Table 2. Evaluation results of family relationship extraction using LUKE and three different named entity recognition methods.

NER	TP	FP	FN	Precision	Recall	F1
NLTK	65	279	70	0.19	0.48	0.27
Spacy	**83**	**341**	**52**	**0.20**	**0.61**	**0.30**
Stanza	38	174	97	0.18	0.28	0.22

As opposed to CoreNLP, in LUKE relation extraction is controlled by carefully choosing sentences that would prove to be trivial for LUKE *i. e.* sentences with relationship keyword and entities included. This way the number of extracted entities is reduced and limited to only the ones with the highest probability of being correct. It is easy to see that due to this approach, the evaluation results of LUKE are vastly superior when compared to CoreNLP. Note that relations were extracted also for entities which were not part of our reduced character ground truth relations. Such relations were ignored and not included in the evaluation calculation.

When comparing *Spacy*, *Stanza* and *NLTK* as entity relation extraction, Stanza was found to be more suitable for LUKE and therefore produced better results than LUKE with Spacy or NLTK. This work is not focused on ways of named entity recognition, however as it turns out for the combination of our specific text dataset and LUKE for entity relationship extraction, Stanza is the better choice.

0.8.3 Visualization

After we evaluated found family relationships, we visualized the true positive cases using network representation, where the nodes represent characters and the edges are family relationships with each color representing one relationship type (parent, children, spouse and sibling). As we can see in Figure 1, we can confirm that we connected the characters according to the right type of family relationship. For visualization of the network we used python library igraph [16]. The network on Figure 1 represents TP family relationships found by LUKE algorithm using Stanza for named entity recognition. That combination of used methods produced the best F1 results.

Discussion

In this project we used natural language techniques extract the family relations within the Game of Thrones books. We built a pipeline which we used to process each book separately and gather the results of the found relations together. We used natural language techniques such as basic preprocessing, named entity recognition and coreference resolution to obtain the results and then we tried to visualize those relations in a graph. We used two different relation extraction techniques and compared the results. We observed that the LUKE method which uses transformers performed much than the CoreNLP. The results we obtained were not as good as we first expected since

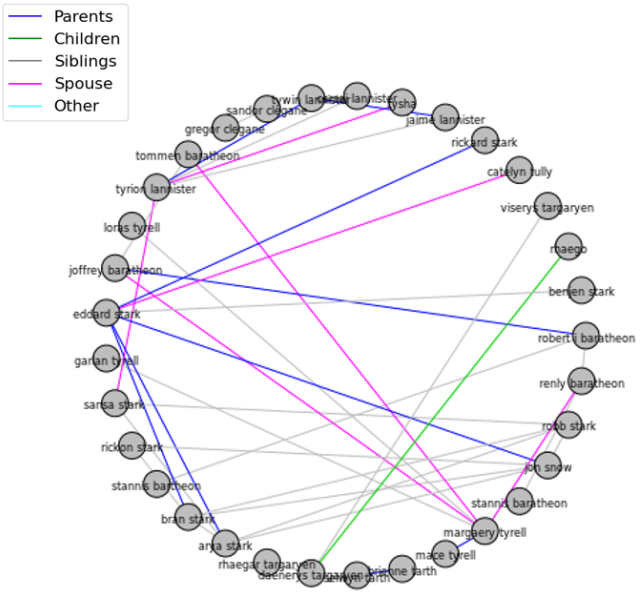

Figure 1. Visualization of TP family relationships found by LUKE algorithm using Stanza.

the task of family relation extraction is quite a challenging one. In addition the family relations in the Game of Thrones books series are very complex and hard to comprehend. The very high number of the characters also adds up to the complexity of those family relations. The pipeline we constructed had flaws on each step. With better performance of coreference resolution and also named entity recognition steps we would probably obtain much better results.

References

[1] Hemlata Shelar, Gagandeep Kaur, Neha Heda, and Poorva Agrawal. Named entity recognition approaches and their comparison for custom ner model. *Science & Technology Libraries*, 39(3):324–337, 2020.

[2] Niels Dekker, Tobias Kuhn, and Marieke van Erp. Evaluating named entity recognition tools for extracting social networks from novels. *PeerJ Computer Science*, 5:e189, April 2019.

[3] Raabia Mumtaz and Muhammad Abdul Qadir. CustRE: a rule based system for family relations extraction from english text. *Knowledge and Information Systems*, 64(7):1817–1844, June 2022.

[4] A Wiki of Ice and Fire. https://awoiaf.westeros.org/index.php/Main_Page. Accessed 22 May 2023.

[5] Selenium Contributors. Selenium: Web Browser Automation. https://www.selenium.dev/, 2023. Accessed 22 May 2023.

[6] Ultrafunk Amsterdam. undetected-chromedriver. https://github.com/ultrafunkamsterdam/undetected-chromedriver, 2023. Accessed 22 May 2023.

[7] The stanford nlp group. https://nlp.stanford.edu/projects/coref.shtml.

[8] neuralcoref. https://github.com/huggingface/neuralcoref.

[9] Nltk. https://www.nltk.org/.

[10] Spacy. https://spacy.io/.

[11] Stanza. https://stanfordnlp.github.io/stanza/.

[12] Stanford corenlp. https://stanfordnlp.github.io/CoreNLP/.

[13] Stanford NLP Group. Stanford CoreNLP - Download. https://stanfordnlp.github.io/CoreNLP/download.html, 2023. Accessed 22 May 2023.

[14] Luke. https://github.com/studio-ousia/luke.

[15] Ikuya Yamada, Akari Asai, Hiroyuki Shindo, Hideaki Takeda, and Yuji Matsumoto. Luke: Deep contextualized entity representations with entity-aware self-attention, 2020.

[16] igraph. https://python.igraph.org/en/stable/.