Warsaw University of Technology

FACULTY OF
MATHEMATICS AND INFORMATION SCIENCE

Master's diploma thesis

in the field of study Computer Science and Information Systems
and specialisation Artificial Intelligence Methods

Preparation of Sets and Machine Learning Models for Recognizing Names of Literary Characters

Weronika Łajewska

student record book number 276952

thesis supervisor
Anna Wróblewska, PhD Engineer

WARSAW 2021

....

supervisor's signature

....

author's signature

Abstract

Preparation of Sets and Machine Learning Models for Recognizing Names of Literary Characters

Natural Language Processing (NLP) is an area of research that can be extremely useful in analyzing long texts, such as novels. One of the essential components of novels are their protagonists. Recognizing and identifying literary characters in texts is a first step for detecting relationships between them, analysing individual characters, creating timelines of events, etc.

This research investigates the problem of recognizing people in novels and attempts to annotate them to allow more detailed analysis of the novels. The prepared tool is used to create a corpus of annotated novels.

The process of identification of literary characters in a text was divided into two main stages. The first one focused on finding named entities (NE) of category *person*. It took advantage of Named Entity Recognition (NER) models. Since standard NER models are not prepared to be used for novels, they must have been fine-tuned for this specific purpose. The second stage was responsible for matching each recognized NE of category *person* with a full name of the literary character associated with it. The matching process was based on *approximate text matching*. The result of the matching process was stored in the form of the tag being a full name of the protagonist put on the NE. These two stages were encapsulated in the functionality of a prepared tool called *protagonistTagger*.

The analysis of the performance of *protagonistTagger* on thirteen novels shows that the tool achieves both the precision and the recall above 80% in case of almost all analyzed novels. The performance of the tool is strongly dependent on the type of analyzed novel. Both metrics vary even by twenty percentage points in case of some novels. It is shown that the complexity, literary genre, number of literary characters and the complexity of the names of literary characters influence the tool's performance, especially during the second stage.

Keywords: Natural Language Processing (NLP), Named Entity (NE), Named Entity Recognition (NER), Approximate Text Matching

Streszczenie

Przygotowanie Zbiorów oraz Modeli Uczenia Maszynowego Rozpoznających Bohaterów w Tekstach Literackich

Przetwarzanie języków naturalnych jest dziedziną badań, która może zostać wykorzystana podczas analizowania złożonych tekstów literackich, takich jak powieści. Jednym z najistotniejszych elementów każdej powieści są jej bohaterowie.

Umiejętność automatycznego rozpoznawania i identyfikacji bohaterów w tekście pozwoli na wykrywanie relacji między bohaterami, tworzenie planu wydarzeń, przeprowadzenie analizy poszczególnych bohaterów, itd.

Podczas moich badań przeanalizowałam problem rozpoznawania jednostek nazewniczych (eng. named entities, NE) z kategorii *osoba* oraz stworzyłam narzędzie do ich anotacji w sposób umożliwiający dalszą analizę tekstów. Narzędzie to zostało użyte do stworzenia korpusu zaanotowanych powieści.

Proces rozpoznawania bohaterów w tekście został podzielony na dwa główne etapy. Pierwszy z nich dotyczy znajdowania NE z kategorii *osoba* bazując na gotowych modelach Named Entity Recognition (NER). Modele te musiały zostać dostrojone, ponieważ ich standardowe wersje osiągają na powieściach stosunkowo słabe wyniki. Drugi etap dotyczył przyporządkowywania każdej rozpoznanej NE z kategorii *osoba* pełnego imienia bohatera, którego dotyczyła. Proces przyporządkowywania bazował na *approximate text matching*. Wynik procesu przyporządkowywania przechowywany był w formie taga przypisanego NE i będącego pełnym imieniem bohatera. Dwa powyższe etapy zostały zaimplementowane tworząc narzędzie nazwane *protagonistTagger*'em.

Analiza wyników *protagonistTagger*'a przeprowadzona dla trzynastu powieści pokazuje, że narzędzie osiąga wyniki przekraczające 80% w przypadku obu metryk: precision i recall. Wyniki narzędzia są silnie uzależnione od rodzaju analizowanej powieści. Wartości obu metryk mogą się różnić nawet o dwadzieścia punktów procentowych w przypadku niektórych powieści. Udało mi się wywnioskować, że skuteczność, szczególnie podczas drugiego etapu, jest ściśle związana ze złożonością powieści, jej gatunkiem, liczbą bohaterów oraz złożonością nadanych bohaterom imion.

Słowa kluczowe: przetwarzanie jzyków naturalnych, jednostki nazewnicze (ang. named entity), rozpoznawanie jednostek nazewniczych (ang. named entity recognition), approximate text matching

Warsaw, ...

Declaration

I hereby declare that the thesis entitled „Preparation of Sets and Machine Learning Models for Recognizing Names of Literary Characters", submitted for the Master degree, supervised by dr inż. Anna Wróblewska, is entirely my original work apart from the recognized reference.

.....

Contents

Introduction

Novels are a fascinating field of study, not only for philologists or literary scholars but also for scientists involved in Natural Language Processing (NLP). They are an excellent repository of knowledge about the language, people and their relations, historical events, places, expected behaviours, etc. Analysis of novels can pertain to the detection of the relations between protagonists, creating summaries, locations detection, creating timelines of events, and many more (see Section 1.2). A corpus of novels is the first step in completing these tasks. High-quality data in the corpus with proper annotations make the further work much more manageable.

The first type of annotations which is crucial when discussing novels is this related to protagonists. Marking all appearances and all references to the novel's main literary characters seems to be a handy feature to have in the corpus. Due to many ambiguities appearing in novels, their complex structure, and often a broad spectrum of protagonists, this task is not as easy as it seems. All these problems make the annotation of the literary characters much more complicated and sophisticated.

Adequate protagonists' annotations are useful in further and more precise analysis of novels. It is not a straightforward task to decide how these annotations should be specified. Unfortunately, in some cases, it may be not enough to annotate each literary character with a general tag *person*. In order to be able to analyse the novel on deeper levels, we need contradistinction between protagonists. The most desired way is to have a separate tag for each protagonist and assign it to this literary character's appearance in a text. Each tag ideally should contain a full name of the protagonist along with a personal title. Tracking down full names of literary characters sometimes may not be easy. In the novels, it is often the case that only a first name occurs or only a surname preceded with a personal title. However, high-quality data should always be a priority. The possibilities that protagonists' annotations offer are described in details in Chapter 1.

The desired outcome of the project is a reasonably big corpus with novels. Every protagonist in each novel should be annotated with his/her proper name. Ideally, the label is a full name of the literary character preceded with the proper personal title if needed. An example of such annotated text is given in Table 0.1.

Generally, the prepared tool is supposed to work automatically with a list of protagonists'

"Her disappointment in ***Charlotte***$_{\text{«Charlotte Lucas»}}$ *made her turn with fonder regard to her sister, of whose rectitude and delicacy she was sure her opinion could never be shaken, and for whose happiness she grew daily more anxious, as* ***Bingley***$_{\text{«Charles Bingley»}}$ *had now been gone a week and nothing more was heard of his return.* ***Jane***$_{\text{«Jane Bennet»}}$ *had sent* ***Caroline***$_{\text{«Caroline Bingley»}}$ *an early answer to her letter and was counting the days till she might reasonably hope to hear again. The promised letter of thanks from* ***Mr. Collins***$_{\text{«Mr William Collins»}}$ *arrived on Tuesday, addressed to their father, and written with all the solemnity of gratitude which a twelvemonths abode in the family might have prompted."*

Table 0.1: An exemplary text extracted from novel *Pride and Prejudice* by Jane Austen. Correct tags are written in the subscript of each recognized named entity of category *person*.

names and a text of the novel given as an input. Let us call this tool a *protagonistTagger* just for reference. An indirect outcome of the project is a model for recognizing appearances of protagonists in a novel. This model is based on Named Entity Recognition (NER).

General Project Workflow

The process of creating the corpus of annotated novels and the *protagonistTagger* tool comprises several stages (see Figure 0.1). Each stage is, generally speaking, a separate task with different assumptions. The main stages are as follows:

1. Gathering an initial corpus with plain novels' texts without annotations.

2. Creating a list of full names of all protagonists for each novel in the initial corpus. These names are the predefined tags that will be used in further steps for annotations.

3. Recognizing named entity of category *person* in the novels' texts in the initial corpus. Training NER model from scratch for this specific problem is not reasonable due to the amount of time and computing power required. It is possible to use some pre-trained NER model and fine-tune it using a sample of manually annotated data. The NER mechanism evaluation is done on a testing set extracted from the full texts of novels. The task of fine-tuning NER model is quite complex and may include several iterations.

4. Each named entity of category *person*, recognized by NER in the previous step, is a potential candidate to be annotated with one of our tags predefined in step 2. At this point, an algorithm (let us call it *matching algorithm* for reference), based on *approximate string matching*, is used to choose from the predefined tags the one that matches most accurately the recognized named entity.

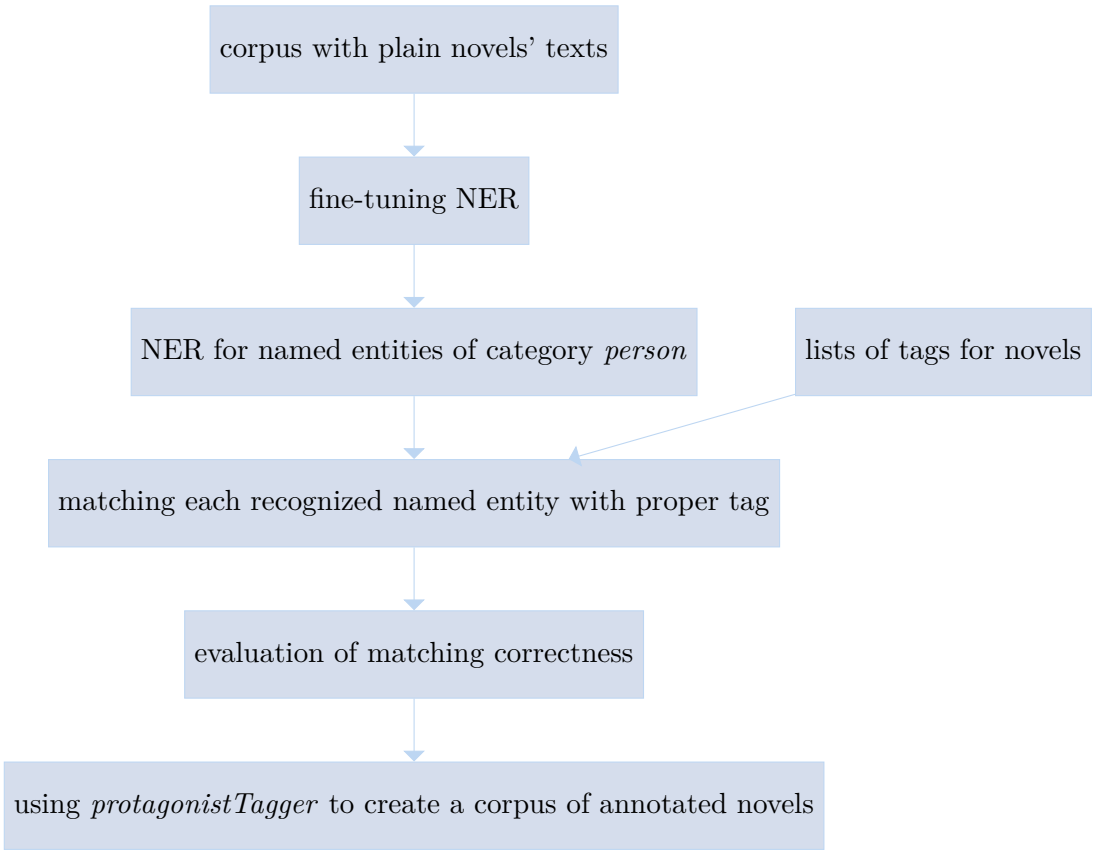


Figure 0.1: Simplified process of creating the corpus of annotated novels and the *protagonistTagger* tool.

5. The annotations done by the *matching algorithm* are evaluated according to their accuracy and correctness.

6. The *protagonistTagger* (fine-tuned NER + *matching algorithm*) is used to annotate more novels in order to create the corpus of annotated novels.

The two most interesting parts of the procedure are fine-tuning NER from step 3 and the *matching algorithm* from step 4.

The analysis of NER in novels is a fascinating field of study because there are plenty of surprising literary characters in the novels. For example, *The creature* in *Frankenstein* by Mary Shelley is one of the main protagonists in the novel. However, it is difficult to blame the NER model for not recognizing it as a *person*. Many nuances need to be considered in order to achieve the satisfying performance of NER in novels.

The process of creating *matching algorithm* offers no fewer surprises. The method is about to mark all protagonists in a given text with a proper tag, having given the list of tags predefined for each novel. Creating this algorithm required an in-depth analysis of the problem of identifying protagonists in novels. Of course, it is possible to read all the novels in the corpus and tag all the

protagonists manually. However, it requires a tremendous amount of time and many annotators' engagement. Novels, being rather long and complex texts, are not suitable for such arduous work.

In a real-world situation, while reading a novel from the very beginning, we do not have any problems figuring out which protagonists are mentioned in the following sentences. However, when we take a random sentence from the novel, even having read the whole text before, it may be challenging to say which protagonist is being considered.

In novels, there are many ambiguities which are not apparent at the outset. For example, there may be a whole family with the same surname, let us say, Smith. In a novel, a protagonist named *Mr Smith* is mentioned. How can we say if he is John Smith - a son or Edward Smith - a father. It is only one of the most common problems.

While analyzing some exemplary novels, one can encounter many more ambiguities, and some of them are handled in the *matching algorithm*. The whole research process, along with the most interesting problems encountered on the way, is described in Chapter 3.

1. State-of-the-art Research on Literary Text Analysis

Literary texts analysis has been performed for centuries by humanists and linguists in a manual, arduous way. Now, this field of study is offered a brand new perspective thanks to digital literary studies. It is still a new and not fully discovered area. Indeed, it can positively influence the study of literature. Computational linguistics makes tasks such as in-depth statistical analysis of literary texts much more comfortable and more precise. There already exist many projects dedicated to literary studies.

The first part of this chapter presents a brief overview of the most impressive projects connected with the analysis of novels and the problem of annotating literary characters. Whereas the second part of the chapter focuses on the possibilities in novels analysis connected with literary characters' annotations.

1.1. Annotating Literary Characters

1.1.1. Project Gutenberg – Literary Texts Corpus

Creating a corpus of digital texts is the first step in computational linguistics. The most popular library of free eBooks is *Project Gutenberg (PG)*. It offers a vast collection of texts of various kinds, such as fiction novels, collections of poetry, dramas, cookbooks, bibliographies, dictionaries, etc. The primary purpose of the project is to grant easy access to cultural works to as many people as possible for free. The majority of classical works of English literature can be found in this archive [21].

1.1.2. GutenTag – Tool for Analysis of Texts in PG

GutenTag [5], [20] is a software tool offering NLP techniques for the analysis of literary texts from PG. GutenTag has three main functionalities:

- **Corpus reader** which handles the inconsistent formatting and structure of texts in the corpus. This component makes it possible to work on plain texts, taking care of extra spacing, headings, references to illustrations, prefaces, quotations, etc.

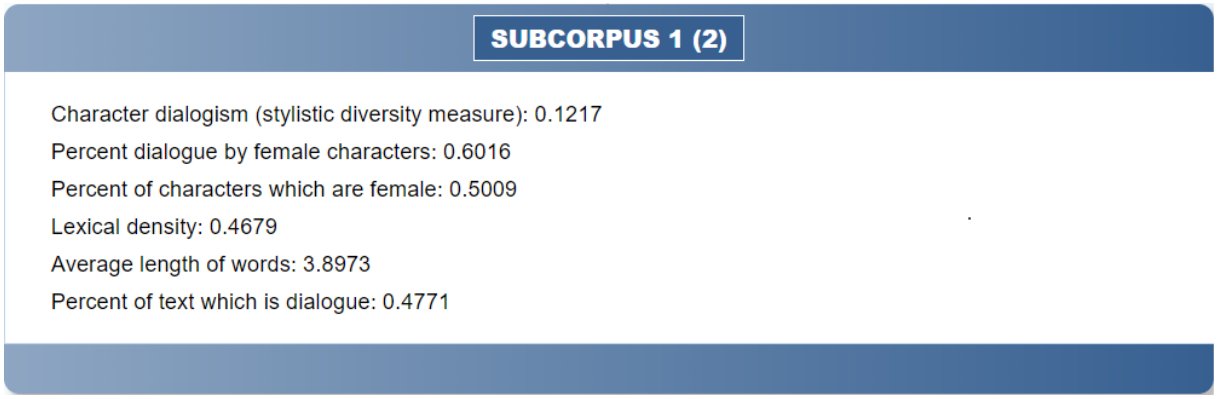


Figure 1.1: Exemplary statistical analysis of the text of *Pride and Prejudice* by Jane Austen [19]

- **Subcorpus filters** which enables choosing a subset of interest of all texts available in PG. The information contained within the Project Gutenberg or obtained from some additional resources helps to identify a relevant subcorpus of interest.
- **Tagging** which, based on different tags specified by a user, performs a statistical analysis of the chosen subcorpus. Tags offered by the tool include single tokens, part-of-speech (POS), structural elements such as chapters, quoted speech.

GutenTag is an advanced tool offering tagging feature. Nevertheless, it does not support annotation of literary characters in the desired way. Even though the NER is used in order to identify the main literary characters, the information gained this way is used only for statistical measures. There is no advanced mechanism of recognizing specific literary characters and matching them with their full names. GutenTag offers a wide range of standard textual metrics (exemplary output in Figure 1.1). Metrics connected with dialogues are computed based on the identification of structural elements such as quoted speech. GutenTag tool enables annotations with tag *said* and identification of the characters to whom the quotation can be assigned. Another example of a provided metric is lexical density. It is based on build-in lemmatization and POS tagging components of GutenTag.

1.1.3. Detecting Characters in Literary Texts

As the problem of identification of literary characters in a text is a first major task in text analysis, many approaches have already been proposed and tested.

Clustering noun phrases

One of the methods is proposed as a part of the project devoted to extracting social network from literary texts [15]. It assumes that the noun phrases recognized in the text by NER can

be clustered into groups referring to the same person. They are sets of co-referents for the same named entity. Coreferent is, for example, the token *Elizabeth* somewhere in the text of the novel *Pride and Prejudice* that corresponds to the literary character entity *Elizabeth Bennet*.

A similar approach to the problem of identifying characters is presented in work devoted to inferring latent character types in English novels [3]. The main goal of the project is to recognize a set of character types in an unsupervised way from a large corpus of novels. The method used there to detect characters is also based on character clustering. Unfortunately, this method, in the form proposed in the paper, may not fully cover cases of protagonists with the same name. It may also have problems distinguishing between males and females with the same surname. For complex texts, the basic method of characters clustering can be not enough.

Graph Representation

Another method [36] is based on representing characters using a graph, where each node corresponds to a name found using NER and edges connects nodes referring to the same character. In the first step, the method links all nodes by edges using i.a. coreference resolution, names variations and lists of hypocorisms. Then a set of heuristics is used in order to identify pairs prohibited from being connected and removing corresponding edges. This transformation is visible in Figure 1.2. The next step attempts to identify entities that have not been recognized by the NER. It is done using a technique aiming at uncovering prototypical characters behaviours from the novels themselves [36]. The technique is inspired by semantic predicates based on creating a set of verbs-and-dependency pairs. This set of predicates is then used to verify if a given noun appears with the verb in a defined dependency relation. If it is the case, it is categorized as a literary character in the novel.

This method of detecting characters is much more advanced than the previous one and has better performance. However, it aims at recognizing all literary characters, even minor or episodic characters, which are not introduced with a name. In many application, such precision is unnecessary. In the analysis of literary texts, it is usually desired to focus on a group of main characters. Therefore other annotations are redundant.

1.2. Further Possibilities

The analysis of literary texts is very promising and offers many possibilities. Being able to identify characters in unstructured texts is the first step in numerous tasks connected with representing and interpreting narratives. This section presents an overview of possible applications

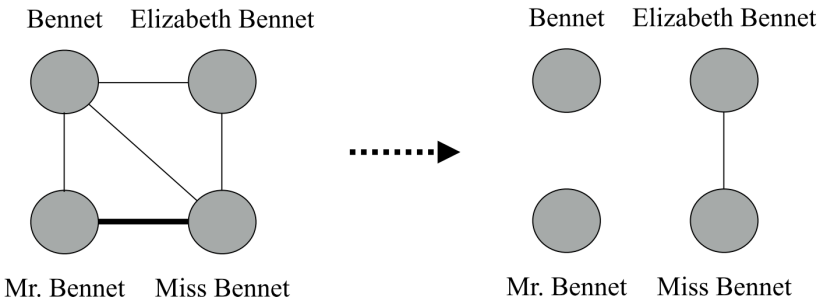


Figure 1.2: Illustration of a process of detecting characters in texts using graph representation [36].

of characters recognition and annotation mechanism along with state-of-the-art research done in this field.

1.2.1. Detecting Relationships Between Literary Characters

The narrative can be modelled from two perspectives in the novel, either the events or the characters appearing in the text. Usually understanding relations between literary characters is the first choices when the analysis of literary texts is considered. It is caused by the fact that characters can be detected more easily than events. Understanding characters relationships provide in-depth knowledge about the novel and ultimately contains information not stated explicitly in it. The ability to model relationships between characters can also be beneficial while analysing political texts, news, articles. Many different approaches to this problem have already been proposed.

Relationships Based on Dialogue Interactions

A famous work devoted to extracting social networks from novels attempts to model social conversations that occur between characters in a form of a network [15]. The network is derived from dialogue interactions, and the method focuses on determining whether pairs of characters are involved in conversations and how often it happens. A visualisation of such exemplary social network is presented in Figure 1.3. The research presented in this work is focused on the relation between the interactions of the characters in a given novel and its setting (urban or rural). It may be worth checking whether higher precision in identifying characters improves the results and leads to some other conclusions.

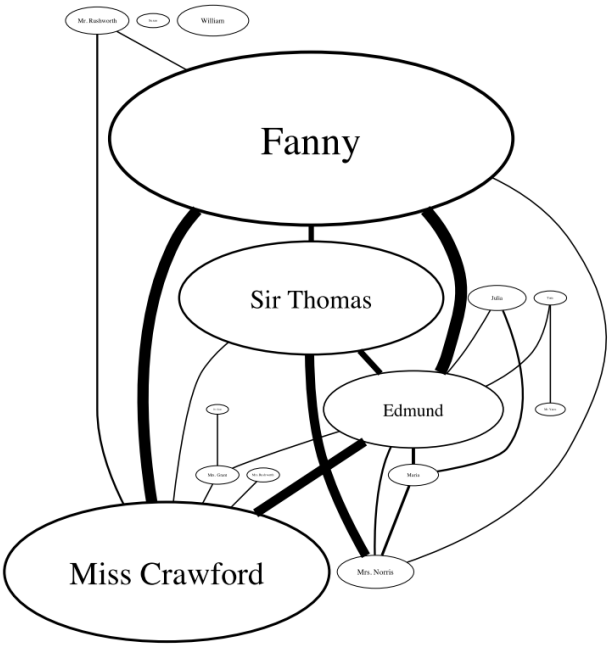


Figure 1.3: Visualisation of social network generated for the text of the novel *Mansfield Park* by Jane Austen [15].

Detecting Evolving Relationships

While reading any novel, it is easy to notice that literary characters and their relationships evolve with the progress of the novel. Analysis of fixed relations between them may not suffice or sometimes may be even impossible. Therefore some research concerning the evolution of these relations has been performed in projects [7], [6]. Both projects aim to model dynamic relationships between pairs of characters by detecting relationships sequences in data (an example is given in Figure 1.4). The main drawback of this approach is that the analysed data has to be pre-processed. The frameworks are tested only on narrative summaries with a pair of characters appearing in it. Additionally, the data needs to be labelled. Having the whole text of the novel annotated with all characters names it may be possible to broaden the idea presented in the project and extend it to analysing whole parts of text connected with a pair of characters rather than only summaries of the novel.

Social Events

A fascinating approach is presented in a paper devoted to social network analysis of *Alice in Wonderland* [1]. The idea is to create a network analysis of the novel in which links between characters are so-called *social events*. Such analysis unveils the roles of characters in the story. Different types of network help to analyze different aspects of the text. The main drawback of

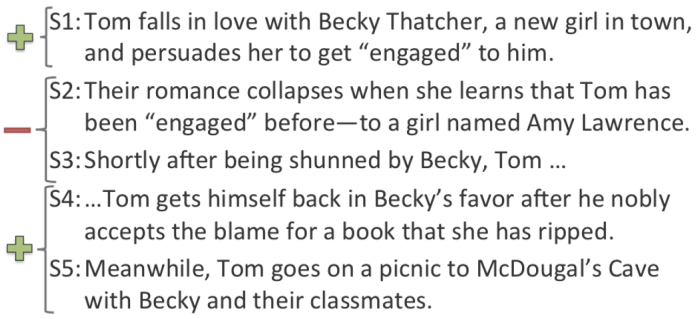


Figure 1.4: The overview of dynamic changes in the relationship between two characters: Tom and Becky. (+) stands for cooperative character of the relationship, whether (-) stands for non-cooperative [7].

the proposed solution is that it is using hand-annotated text with well-defined social events. The possible and very promising extension of the method may be creating an automatic mechanism annotating social events. It seems to be much easier when we have a text with annotated characters.

1.2.2. Sentiment Analysis

Another general area of research while considering computational literary study is sentiment analysis [24]. It includes among the others classification of literary texts based on emotions they convey, emotion-based character analysis and character network construction. Some of the most interesting examples of sentiment analysis connected with characters detection are presented in this section.

Sentiment-based Literary Texts Classification

Sentiment-based literary texts classification attempts to discover a literary genre through sentiment analysis. An exciting example of such analysis tries to represent a plot of a story in the form of an emotional arc. The x-axis of this graph represents a time point in a story, whereas the y-axis represents the measure of emotional content connected with events of the story. Events can be either favourable or unfavourable, represented respectively in the form of peaks and valley on the graph [28] (see Figure 1.5). In this research, it is proven that there are six types of emotional arc representing a story. The authors of the analysis are proposing an extension of the method which could analyze separate characters through time in a story. It would require the ability to recognize which parts of the story are associated with this character. Therefore characters detection may be beneficial here.

Another research connected with literary texts classification takes into consideration a type

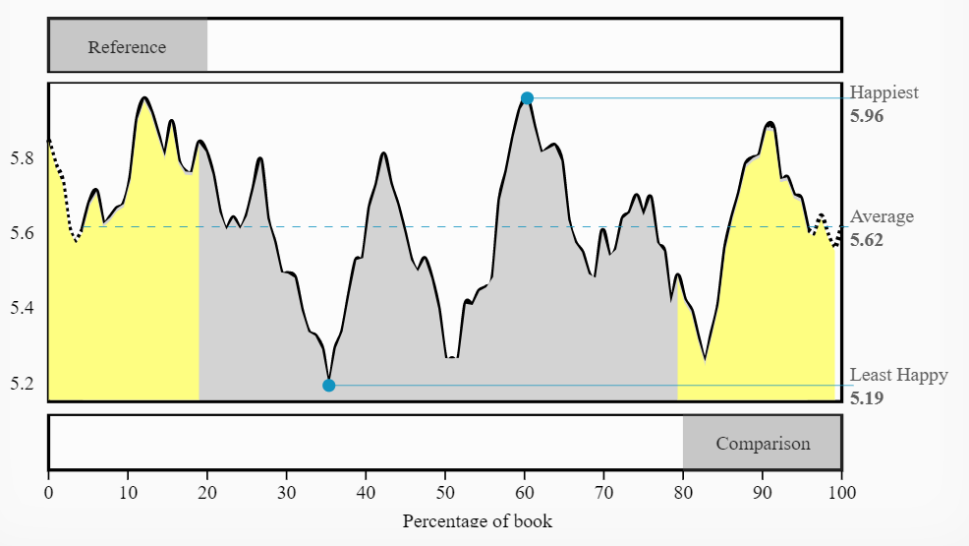


Figure 1.5: The visualization of an emotional content of *Wuthering Heights* done using a method of emotional arcs [28].

of an ending of a story and tries to classify it as happy or non-happy [41]. The situation of the main character is being monitored through the story. If it improves or remains favourable for the main protagonist, the story is classified as the one with a happy ending. It may be interesting to investigate the ending of the story for other characters appearing in the texts. It can be achieved by analyzing parts of the text connected with this character. Here again, a good characters detection mechanism can be useful.

Characters' Analysis

Understanding the literary characters and being able to describe them is one of the biggest challenges of literary analysis. Characters can provide the reader with the majority of the most crucial information about the storyline itself. Nevertheless, literary characters' analysis is a very demanding and complex task. The most basic approach to describing characters is by their role in the story, for example, hero, princess, villain. This approach has been investigated among others in the domain of folktales [18]. The authors of the work aimed at determining the relationships of a character in a tale, as well as finding its role in the story. The characters were divided into 9 different intersecting types. It was assumed that it is possible for a character to be at the same time a positive and negative protagonist (have two roles assigned). The methods presented in the project are based mainly on NLP and reasoning on domain ontologies.

What is worth mentioning is that a role in a story provides relatively few information about the personality of a character and his nature. Especially in cases when a character has multiple different roles assigned in a given story. Determining the personality of a character is therefore a

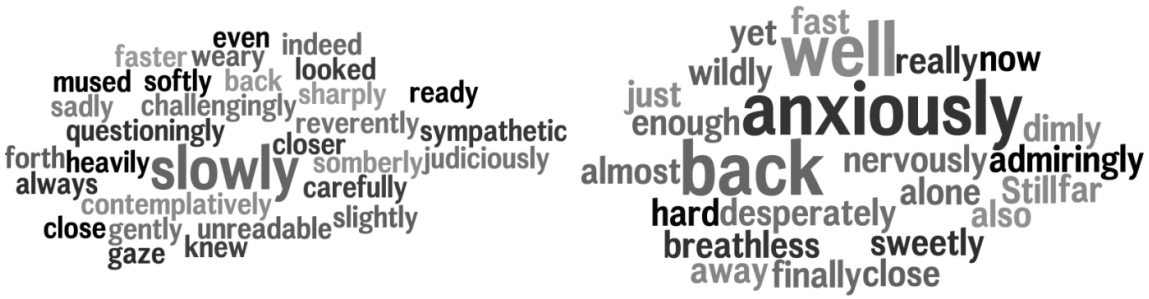


Figure 1.6: Frequency word clouds for two characters descriptions (predicatives and adverbs) extracted from novels [16]. The left cloud corresponds to Master Yoda from the *Star Wars*, whereas the right cloud corresponds to Sansa Stark from *Game of Thrones*.

very challenging task. It was attempted in a project [16] that is based on the psychological Five-Factor Model of personality [14]. The authors tried to perform personality profiling using the automated classification of personality traits. They wanted to answer a question about literary character's extraversion (whether he is primarily an introvert or an extrovert). They prepared and evaluated three different machine learning models:

- model based on direct speech in texts attempting to verify whether style and content of character's utterances influence his extraversion
- model based on actions/behaviours of a character that tries to examine their impact on the character's extraversion
- model based on descriptions of a character using predicatives and adverbs (for example *Elizabeth cried* ***mournfully***)

The system was able to classify a character as an introvert or an extrovert. An example of frequency word clouds for model based on characters' descriptions is presented in Figure 1.6. In order to be able to investigate the personality of a character, we need to know which parts of the analysed text are dedicated to this character. Consequently, proper characters' annotations can have many applications in a task of personality profiling in novels.

2. Theoretical Background

2.1. Metrics used for evaluating NLP models

In order to evaluate the results of the model, several standard metrics used in NLP are used. This section contains a brief description of each of them.

2.1.1. Precision and Recall

Two most common measures for model effectiveness are precision and recall [10]. Notion used for defining them is based on a contingency table [8] (see Table 2.1). A model is about to forecast a proper label for a named entity. Every such prediction falls into one of the four classes defined in the table. This table completely describes the outcome of the model. However, analysing each entry of the table separately is very time-consuming, precision and recall are used in order to describe the quality of the result of the model.

Precision can be intuitively understood as the ability of the model not to label as X a named entity that is in fact Y. In other words precision is the percentage of results of the model that are relevant. It is defined as:

$$precision = \frac{tp}{tp + fp}$$

Recall can be intuitively understood as the ability of the model to find all named entities which correspond to some label X. In other words, recall is the percentage of all relevant results correctly recognized by the model. It is defined as:

$$recall = \frac{tp}{tp + fn}$$

	predicted positive	**predicted negative**
actual positive	true positives (tp)	false positives (fp)
actual negative	false negatives (fn)	true negatives (tn)

Table 2.1: Standard confusion matrix describing all possible outcomes of classification.

2.1.2. F measure

F measure is a single metric which trades off precision vs. recall [10]. It can be described as a weighted harmonic mean of the precision and the recall. It ranges in [0, 1] where value equal to 1 is the maximum score. F measure is defined as:

$$F = \frac{(\beta^2 + 1) \cdot precision \cdot recall}{\beta^2 \cdot precision + recall}$$

β in this formula weights the importance of recall and precision. $\beta = 1$ means that recall and precision are equally important, $\beta < 1$ emphasizes precision and $\beta > 1$ emphasises recall.

2.2. Named Entity Recognition

Named entity recognition (NER) is a subtask of information extraction that locates named entities in unstructured texts. In this context 'named entity' is a general name for all real-world objects which have a name assigned, for example, people, countries, cities, organizations. NER is assigning a proper category to every located named entity. The result of NER is an annotated block of text in which all named entities have their category assigned. An exemplary output of the NER system is presented in an example 2.1.

$$\begin{gathered} Bill\ was\ hired\ by\ Apple\ in\ 2008. \\ \downarrow \\ [Bill]_{Person}\ was\ hired\ by\ [Apple]_{Organization}\ in\ [2008]_{Time}. \end{gathered} \tag{2.1}$$

The most common categories are *person, location, organization*. In more domain-specific versions of the task, the considered named entities may include *drug names, names of pharmacological substances, works of art, brands*, etc.

Due to many ambiguities present in natural languages, the NER task is relatively difficult. Considering a sentence *Bush was found in a park.*, it is almost impossible to say, what is meant by word *Bush*. Is it a plant growing in the park or maybe the president George Bush that was having a walk on a sunny day in a park near his residence? Having read the whole text from which the sentence was extracted, a human would not have any problems with assigning meaning to this word. However, for a NER system, it is not straightforward. Especially due to the fact that the capitalization of the word does not provide any information, as the word stands at the beginning of the sentence. This is an extreme example of the ambiguities that are connected with NER, nevertheless, it indicates the difficulty of the task in general.

Special cases that need to be handled by NER models differ depending on an analyzed language, domain, source of texts. A NER system having good performance on New York Times

news very often may fail on informal tweets. Therefore, the perfect NER model should be unified and should not require fine-tuning for different languages and domains. What makes NER task so difficult is the fact, that even for us recognizing and classifying named entities in sentences out of context is not trivial, or even impossible without domain-specific knowledge.

The amount of special cases that need to be handled requires special guidelines for this task. These guidelines assure the consistency with the gold standard created by human annotators. In order to verify the performance of NER systems a set of standard Natural Language Processing (NLP) metrics is used including precision, recall and F1.

Approaches to NER can be divided into four general groups (they are presented in a survey [38]):

- knowledge-based,
- unsupervised and bootstrapped,
- feature-engineered supervised,
- feature-inferring neural network.

The early knowledge-based NER systems were mostly based o handcrafted rules, ontologies, lexicons or orthographic features. They were therefore strongly domain-dependent. These systems were followed by feature-engineered systems based on, for example, a hidden Markov model. The most recent approaches are designed using mainly neural networks. They are becoming very popular, as they do not require domain-specific resources or feature engineering [38]. Due to the number of possibilities of using neural network for the NER task, this group of systems requires significant attention. The choice of discussed systems is inspired by the survey [38]. Clear division of NER systems' categories described there, helps to understand the differences between presented systems and each approaches motivation.

2.2.1. Knowledge-based NER Systems

The main characteristic of knowledge-based NER systems is relying on domain-specific resources, such as lexicons or dictionaries. This approach is presented in a paper devoted to the recognition of drugs' name, and extraction of drug-drug interactions appearing in biomedical texts [31]. This research's primary motivation is the automatic discovery of the change in one drug's effects by another drug's presence. The discussed task is divided into two parts: recognition and classification of drug names in texts, and the extraction and classification of the interactions between recognized drugs. From the perspective of the NER task, the first part of the research is

crucial. The presented NER methods are applied to the pharmacological domain. Therefore, the named entities that need to be recognized in the text are pharmacological substances, among others, *brand* (branded drug name) and *drug-n* (an active substance not approved for human use).

The most successful method presented in the paper makes use of *ChemSpot* tool [29]. *ChemSpot* combines conditional random field (CRF) with a dictionary. Both of them are independently used to annotate the text. Subsequently, the annotations of both components of the system are merged. *ChemSpot* employs a second-order CRF and offset conjunction of two, which adds all features of the two preceding and the two succeeding tokens to the token's features. It provides CRF with more contextual information. The dictionary used by *ChemSpot* is post-processed with many rules including expanding partial matches (boundaries should lie next to whitespace, tab or line-break character), correcting the boundaries of these matches and truncating common suffixes (such as *-induced*, *-related*).

The second top best performing approach is a combination of biomedical resources, such as pharmaceutical knowledge base *DrugBank*, *ATC* classification system (active substances classification tool) or MeSH (National Library of Medicine controlled vocabulary thesaurus). However, the authors of the paper are not providing any detailed information about it.

When it comes to performance, it is observed that *brand* drugs are recognized more easily than other categories. The authors of the paper suspect that it can be caused by the fact that brand names of drugs vended by pharmaceutical companies are carefully selected to be short and unique. It makes the recognition of these types of drugs less complicated. On the other hand, drugs of type *drug-n* are much more challenging to recognize. It is probably due to the more significant variation and complexity in their naming.

The performance was tested on two different datasets. The first one contains texts focusing on the description of drugs and their interactions. The performance on this dataset is much better than on the other one containing abstracts on the subject of drugs interactions but not necessarily the drugs interactions themselves.

The results presented in the paper show that it is not always trivial to recognize pharmacological substances in texts. It is caused by a large number of ambiguities of pharmacological terms. It can also conclude that knowledge-based systems may have several problems with entities not seen before (like here with drugs of category *n-drug*). New words are not included in the dictionaries. Therefore their recognition and correct annotation are far more complicated.

2.2.2. Unsupervised and Bootstrapped NER Systems

Some of the earliest NER systems were unsupervised, using minimal training data [38]. In approaches based on labelled examples, many rules are needed to cover the whole domain. Therefore, in order to train a classifier, a large number of labelled examples is required. The approach to NER presented in [12] is based on unlabelled data, showing that it can significantly reduce the requirements for supervision to only seven simple *seed* rules.

These seven rules proposed by the authors in [12] are the only supervision included in the model. They make use of the natural redundancy in the unlabelled data. Namely, for many named entities, both the spelling and their context, are sufficient for their type classification. The used rules can be divided into *spelling* (a look-up for the string or looking at words such as **Mr.** in a string) and *contextual* (considering words surrounding the string in the sentence). The example below taken from [12] illustrates perfectly the redundancy in data that is crucial in this approach:

..., says **Mr. Cooper**, a vice president of ...

Here, we can observe both: a spelling rule (the bold string contains word **Mr.** and a contextual rule (in the context of the bold string there appears a word **president**). Both rules imply the named entity of type **person**. It can be concluded then that both these features should predict the same label. It is a very functional dependency in building a classifier. The data can be represented by a *spelling, context* pair associated with a set of features. In the case of our example, the pair is *Mr. Cooper, president*.

The first unsupervised algorithm proposed in [12] is a heuristic based on a decision list learning. The algorithm starts with these seven initial *seed* rules, and at each iteration, it increases the number of rules. It is done by separating the spelling and contextual features, alternating between labelling and learning with the two types of features. Here again, the assumption about data redundancy is visible. It is assumed that either spelling or contextual rule alone is enough to build a classifier.

The second approach presented in a paper [12] is based on a boosting algorithm. It is based on *AdaBoost* algorithm developed for supervised learning. The algorithm attempts to find a weighted combination of basic classifiers, that minimizes the bound on the number of unlabelled examples on which these two classifiers disagree. The classifiers are built in parallel from labelled and unlabeled data. It works in rounds in such a way that each round is composed of two stages. Each stage then updates one of the classifiers while keeping the other classifier fixed. This method uses an objective function a measure of agreement between the two classifiers (the number of examples on which the classifiers do agree). The algorithm attempts to optimize this function.

The approaches presented in paper [12] show the applications of unlabelled data in unsupervised methods. What is crucial in this type of systems, is benefiting from the features and characteristics of natural languages. In the case of paper [12], it is the language redundancy. Thanks to this observation, the authors managed to reduce the need for supervision significantly. However, it is worth pointing out that the presented system requires in-depth knowledge about the language and its detailed analysis.

2.2.3. Feature-engineered Supervised NER Systems

Supervised models are characterized by being trained to predict a set of exemplary inputs and their expected outputs. This way, there is no need for using human-made rules. Such systems for NER can be based, for example, on hidden Markov model (HMM) that learns to recognize and classify names, dates, times, numerical quantities and other named entities. One of the earliest NER system based on HMM is presented in [4].

The authors define NER task in the form of classification problem, where every word is either part of some named entity category (it belongs to one of the types of named entities) or not (*not-a-name* class). Therefore, it is possible to develop a variant of HMM for the named entity recognition task. All the HMM states are organized into regions in such a way that each region is assigned to one desired class (a type of named entity or *not-a-name*). A statistical bigram language model is used within each region for computing the likelihood of words occurring within a given region. In this language model, every word's likelihood is based only on the previous word. The task can be then formulated as finding the most likely sequence of classes given a sequence of words.

In the proposed approach, a generative model is assumed, i.e., HMM generates the sequence of words and labels (name-classes). It is done in three steps:

- generating a name-class, basing on the previous name-class and the previous word;
- generating the first word in this name-class, basing on the current and previous name-classes;
- generating all subsequent words inside this name-class (as a bigram language model is considered here, each subsequent word is based only on its immediate predecessor).

These steps are repeated as long as the entire word sequence is not generated.

The proposed model assumes the usage of word features that take into consideration the language-dependent characteristics of words. The argument for including word features in the system simplifies some cases in the recognition process. For example, in Roman languages,

capitalization of the first character in the word is good evidence of all kind of names. This rule is easy when it comes to implementation, and at the same time, it can significantly improve the model's performance. Therefore, throughout the model, each word is accompanied by one of fourteen proposed word-features. They help indicate whether the given word is a number, the first word of the sentence, all uppercase, all lowercase, etc. The implemented word features are generally simple and deterministic computations that can be performed on every word.

One of the interesting conclusions about the proposed NER model's performance concerns the effect of word features. Namely, it turned out, that the systems perform better with the full set of word features than with only some part of it. It means that word features, being the only language-dependent component of the whole system, really influence its performance. What is surprising, is the fact that a simple probabilistic model, connected with a small set of word features can quite well handle the task of NER. Additionally, the authors performed some tests on texts given in the mixed case (original capitalization), all upper case (all letters capitalized) and all upper case without punctuation (dots and commas removed). It turned out that the proposed model outperformed all previous approaches on texts in upper case (when capitalization could not be used for named entity recognition).

2.2.4. Feature-inferring Neural Network Systems

Initial attempts to use neural network architectures for NER were using feature vectors constructed from orthographic features, dictionaries and lexicons. Soon this manual way of creating feature vectors was replaced with using word embeddings [2]. Neural network-based NER systems can be classified into several groups depending upon their representation of the words in a sentence. Such representations can be based on words, characters, sub-word units different than characters or any combination of these [38].

Word Level Architectures

In the case of word-level neural network-based NER systems, the words of a sentence, represented by their word embeddings, are given as an input to a neural network. The first such model is presented in [13]. The architecture and the learning algorithm presented there are unified and applied to various natural language processing tasks, including NER. The basic idea behind this architecture is to avoid task-specific manual engineering in creating input feature vectors. It can be achieved automatically, as the system itself learns the internal representation adequate for the given task on a large amount of mostly unlabelled data. Instead of constructing word feature vectors manually, the paper presents a method that uses word embeddings.

The proposed architecture uses lookup tables that transform discrete features (words or characters) into continuous vector representations. It takes sentences as input and learns several layers of feature extraction that process it. The features computed by layers of the neural network are automatically trained by backpropagation so that they are relevant for the given task. The first layer of the neural network extracts feature vectors for each word in a sentence (by a lookup table operation). Generally speaking, a word representation can be considered as K discrete features, and with each feature, there is an associated lookup table. A feature vector is then obtained by concatenating all lookup table outputs.

The next step is extracting higher-level features from the word feature vectors produced by the lookup table layer. These feature vectors need to be combined in the next layers of the neural network to provide a tag decision for each word in a sentence. It is possible to tag one word simultaneously using a window approach, which assumes that the tag of a word depends mainly on the surrounding words in a fixed size window. Each word in such window is passed through the lookup table layer to produce a word feature window (a matrix of word features of words in a window). Further neural network layers include *linear* ones performing affine transformations and *HardTanh* introducing non-linearity to the model by applying hyperbolic tangent. The output of the network can be interpreted as scores for all possible tags.

An alternative for the window approach is a sentence approach, which considers the whole sentence while tagging a word. The main difference here is using an additional convolutional layer. This neural network takes the complete sentence and passes it through the lookup table layer. Then, thanks to convolutional layers, local features around each word of the sentence are produced, and they are combined into a global feature vector. What is worth mentioning is that the size of the output depends on the number of words in the sentence. However, the global feature vector is supposed to have a fixed size, independent of the sentence length, to apply subsequent standard affine transformations (in *linear* layer just like in window approach). Therefore, an additional *Max layer* needs to be used. It forces the network to capture the most useful local features produced by the convolutional layers. It results in a fixed size global feature vector just as it is desired.

The neural networks discussed in [13] are trained using stochastic gradient descent, by maximizing a likelihood over a training data. The presented design of the system avoids task-specific engineering. The system can discover internal representation adequate for NER, having an extensive, unlabelled data set and the designed learning algorithm. This approach is much more general and unified compared to the ones discussed earlier in this paper.

Character + Word Level Architectures

Another subgroup of neural network-based NER systems is combining word context and the characters of this word [40]. It was proven that these systems work quite well in case of NER task. The main advantage of them is that they do not require much domain-specific knowledge or resources.

Word as a combination of its embedding and a convolution over its characters

One way of applying word and character level approach is combining a word embedding with a convolution over word's characters. Such a model is discussed in [9]. The authors again try to avoid hand-crafting rules and applying a lot of feature engineering. They attempt to improve the approach presented in section 2.2.4, which limits the context taken into consideration to a fixed size window around each word. The approach presented in [9] attempts to analyze long-distance relations between words as well. Additionally, the authors include in the word representation, also the character level features. They enrich the final word representation and make it possible to exploit, for example, suffixes or prefixes of words. It can be handy with rare words, as in their case, the embeddings may not be sufficiently trained.

Even though the model presented in [9] is inspired by the one described in section 2.2.4, there are some significant differences. For each word, a *convolution* and a *max* layer are employed to extract a new feature vector from character embedding and optionally character type. A character embedding is taken from a lookup table randomly initialized with values drawn from a uniform distribution. The character type is one of the following options: *upper case, lower case, punctuation, other*. Both, the character embedding and the character type feature, are computed through lookup tables, concatenated and then passed into the convolutional neural network. The entire word embedding (lowercased beforehand) is based on some pre-trained, available model.

This approach makes use of external knowledge in the form of lexicons. For every considered named entity category, there is a compiled list of known, tokenized named entities. For each category, every n-gram is matched against entries in the lexicon. A match is considered successful when the n-gram matches the prefix or suffix of an entry and is at least half the entry length. Partial matches of length smaller than two are discarded (the only exception are named entities of category *person*). When, for some named entity category, there are multiple overlapping matches, exact matches have priority over partial matches. Additionally, the longer matches are preferred over shorter ones and the earlier matches over later ones. What is also essential, all matches are case insensitive. The feature for each token in a match is encoded to indicate the position of the token in the matched entry (for example begin, inside, end).

Instead of the feed-forward neural network, like in approach presented in section 2.2.4, the

authors propose a bi-directional long short-term memory (LSTM). The extracted features (word embedding concatenated with character-level features) are fed into a forward LSTM network and a backward LSTM network. The output of each of these networks is decoded using a *linear* layer and a *log-softmax* layer into log-probabilities for each tag category. The two output vectors are then added together, producing the final output with scores for each possible tag category. The usage of LSTM unit with the forget gate allows learning long-distance dependencies between words. This way, it is possible to take into account an infinite context on both sides of a word. As mentioned in the introduction to the NER task, the context of the word may be crucial for choosing the correct named entity category. Therefore this approach, making use of LSTMs, is auspicious.

Word as a combination of its embedding and a LSTM over its characters

Another way of applying word and character level approach is combining a word embedding with LSTM over the word's characters. This kind of approach is discussed in [25]. The authors propose two neural architectures for NER that use no language-specific features. They rely on character-based word representations, learned from the supervised corpus, and unsupervised word representations, learned from unannotated corpora.

The solution is based on two facts. The first one states that names may consist of multiple tokens over which we should reason jointly. It is handled in the model, thanks to LSTM. The second fact states that to say whether a token is a name, we need to consider both: orthography of the word and the word's distribution in a corpus. The word representations combine both: character-based word representation and distributional representation (the used embeddings are learned from a large corpus sensitive to word order). The dropout training is used to prevent the model from being dependant on one representation too strongly.

The first proposed neural network architecture is a combination of LSTM and conditional random fields (CRF). LSTM was proven to deal with long dependencies by incorporating a memory-cell. It uses several gates that control the proportion of the input provided to the memory cell, and the proportion of the information from the previous state that can be forgotten at this point. The representation of a word in a context, using bidirectional LSTM, is a concatenation of its left and right context representations. The tagging decisions are modelled jointly using a CRF. This way, neighbouring tags are taken into consideration.

The second presented architecture is inspired by transition-based dependency parsing with states represented by stack LSTMs. It incrementally constructs chunks of inputs maintaining a *summary embedding* of the stack's content. It means that stack LSTMs permit embedding of a stack of objects that are both added and removed, while sequential LSTMs model sequences from

left to right. The chunking algorithm, presented in the paper, is based on transition inventory using several stacks of words (for example *output* with completed chunks) and containing several transitions (for example *reduce(y)* which pops all items from the top of the stack creating a *chunk*, labels it with label y and pushes the representation of this chunk onto the *output* stack). To compute an embedding of a sequence of tokens of a chunk from *output* chunk, a bidirectional LSTM is run over the embeddings of its constituent tokens, as well as over the token representing the type of the chunk (a label y assigned to it with *reduce(y)*). The output contains a single vector representation for each labelled chunk that is generated. This approach is beneficial with names that consist of multiple tokens.

The main characteristic of the proposed architectures for sequence labelling is the fact that the model output label dependencies in a straightforward way. It is done either via simple CRF layer or using a transition based chunking algorithm. The crucial part in both architectures is using both, pre-trained word representation and the character-based one, that capture orthography and morphology of words. It is precious as it attempts to implement our intuition about NER task, namely, that the named entity category is strongly related to its orthography and morphology (at least in English).

Character + Word + Affix Level Architectures

The last subgroup of neural network-based NER systems worth covering here is an architecture that augments the character+word level one with affixes. This kind of systems learns affix embeddings, alongside word and character representations. The origin of this idea lies in feature engineering, where affixes play a significant role.

This type of architecture is described in [39]. The paper's authors propose a specific learned representation of prefixes and suffixes of the word, avoiding reliance on any dictionary features. This way, they make use of the semantics of specific sub-word units (morphemes). Using affixes is especially useful with words, that was not seen before, as the model can create a better approximation of their meanings.

To avoid language-specific affix lexicons or morphological tools, the authors use a simple approximation of morphemes. Affixes at the beginnings and ends of words are considered as words features, complementary to character-based features. All n-gram prefixes and suffixes in the training corpus are considered to select only the ones with a frequency above some fixed threshold. Only the most frequent ones are chosen, as it is highly probable that these affixes behave like true morphemes of an analyzed language. It is a very promising and at the same time inexpensive way to enrich the representation of the words. This way it contains separate

embeddings for characters, prefixes, suffixes and words.

The neural network architecture is similar to the one described in section devoted to systems with the word as a combination of its embedding and an LSTM over its characters. The learned representation for approximated affixes is concatenated with bidirectional characters' LSTM encoding and the word's learned representation. Then the final representation of each word is fed to another bidirectional LSTM, followed by CRF layer to produce the named entity tags for each token in a sentence. The output is encoded to indicate the position of the token in the recognized named entry (beginning, inside, out).

The presented approach offers an exhaustive way of creating word representations making use of semantics of morphemes. The idea is relatively straightforward and not complex from an implementation point of view. However, it injects to the model the linguistic intuition about words not seen before. It is built on previously proposed and well-tested solutions, emphasizing the semantics of the words.

2.2.5. Summary of Approaches to NER

The analysis of the group of most common approaches to the NER task confirms the initial impression about the difficulty of the task. Recognizing named entities and their correct classification requires addressing many ambiguities and exceptions present in natural languages [32]. Systems designed to solve this task are a starting point for many other NLP tasks, such as questions answering, topic modelling, relation extraction, information retrieval. Therefore, recently, a high emphasis was put on creating well-performing systems for NER.

Presenting the approaches in the order, in which they are given in [38], exposes that these systems were built upon each other or, more precisely, they were inspired by each other. Every next system tries to address some problems encountered in the previously proposed solution. This way, more functionalities are added or replaced with those with better performance on this specific task. Every next system is also more faithful to our linguistic intuition about recognizing named entities.

The first group of systems, attempting to rely mainly on domain-specific knowledge and lexicons, is reliable only possessing exhaustive resources. However, in the real world tasks, we rarely have access to such bases of domain-specific knowledge. Therefore, such systems, from the point of view of the performance, are not overly successful. The discussed unsupervised NER system, on the other hand, attempts to address linguistic observation about language redundancy. Using features and characteristics of natural language reduces the need for supervision to a handful of handcrafted rules. It is a step in the right direction, as the systems begin to reflect our intuition

about NER task. Unsupervised learning is beneficial, from the perspective of collecting data, as it can be based on unlabelled one.

As for supervised systems, it is crucial to bear in mind the need of collecting a big amount of labelled data for training. In the case of the NER task, manual annotations are very time-consuming. Nonetheless, this kind of systems is most successful so far. NN based NER systems generally outperform all the other groups of systems discussed in this paper. Thanks to word embeddings, there is no need for vector feature engineering, which should be done by the system itself. Moreover, architectures taking advantage of LSTMs can analyze sentences and words on multiple levels, considering the broad context of words. The word's representation can be composed of affix embedding, characters embeddings, and the word embedding itself. This way, the entire word semantics is taken into consideration. Such an approach is especially beneficial in case of data containing words not seen before. The way of handling these new words (using affix embeddings) is intuitive from a linguistic perspective.

Creating systems that attempt to reflex our intuition about the solved task makes it easier to predict, understand and improve the system's behaviour. Therefore, from all the systems presented in this paper [38], the last one is the best, in my opinion. Given all these studied, I would hypothetically pay significant attention to creating exhaustive representations of words, as they are the main building blocks of the whole solution.

Even though, the results of state-of-the-art NER systems are getting better and better, there is still room for improvement. As mentioned before, most of the systems perform well on the type of data on which they were trained. Models trained for NER on the news most probably have lower performance on a tweet or an extract from a nineteenth-century novel than on a piece of news. Creating a model indifferent to the type of the text and its source is still an open challenge in Named Entity Recognition.

2.3. Approximate Text Matching

Definition 2.1 (Approximate text matching). The problem of *approximate text matching* can be formally stated as follows: given a long text $T_{1,\ldots,n}$ of length n and a comparatively short pattern $P_{1,\ldots,m}$ of length m, both sequences over an alphabet $\sum$ of size ρ, find the text positions that match the pattern with at most k "errors". [27]

Approximate text matching is generally a problem of finding the positions of a text where a given pattern occurs with some limited number of "errors" in this match. The term *error* may

be interpreted in various ways in this context. The error model, which defines how different two sequences are, usually depends on the applications of the approximate text matching. The idea is to make the distance between two strings small when there is a high likelihood of one string being a modified, erroneous version of the second one, under the error model used. [26]

There two categories classifying approximate string matching:

- on-line - the pattern is preprocessed, but the text is not
- off-line - a data structure (index) is built on the text before searching

The second category of algorithms is much more sufficient for large text because text indexing makes searching dramatically faster.

On-line Approximate String Matching

Levenshtein distance or edit distance is the most popular error model used in case of on-line approximate text matching [26], [27]. An error is measured in terms of single-character operations required to change one sequence of characters to the other. The considered operations are insertion, deletion and substitution. Formally speaking the distance $d(x, y)$ between two strings x and y is the minimum number of such operations needed to convert one into the other.

Off-line Approximate String Matching

Off-line methods are defined in two dimensions: data structure and search method [27]. The most commonly used data structures are presented below. All of them are used for text indexing, but they differ on space and time usage.

- suffix trees - a compressed tree containing all the suffixes of the given string as their keys and positions in the string as their values;
- suffix arrays - a sorted array of all suffixes of a string;
- q-grams - substrings of the text of length q; in a q-gram index, every different text q-gram is stored, and for each q-gram, all its positions in the text are stored in increasing text order;
- q-samples - similar to q-gram; the only difference is that only some q-grams (called q-samples) are stored, so not any text q-gram can be found;

When it comes to search approaches, there are also several options used in the context of this problem. They are listed below:

- **Neighbourhood generation** is based on the idea of *k-neighbourhood*, which is a set of all strings that match some pattern with at most k errors. In the beginning, all the strings in *k-neighbourhood* are generated. The next step is to find all their text occurrences without errors in the text.

- **Partitioning into exact searching** selects patterns substrings that match without errors. The text areas that surround their occurrences are verified for an approximate occurrence of the whole pattern. This method is called *filtration*.

- **Intermediate partitioning** is the combination of the two approaches listed above. The search is limited to some pattern pieces, which still may be large and may appear with errors. However, searching for pattern pieces is still easier and less erroneous than searching for the whole pattern. The neighbourhood generation is used to search for these pieces of the pattern.

2.4. Programming Libraries and Tools

This section is devoted to the programming libraries used in the implementation of the project. It contains a brief description of each library along with typical applications:

- **SpaCy** [34], [37] is an open-source library for advanced natural language processing written in python. It provides tools for tasks commonly used in NLP projects such as named entity recognition, tokenization, part-of-speech tagging, dependency parsing and many more.

- **FuzzyWuzzy** [11] is a python library for "fuzzy" (approximate) string matching. The method it uses for calculating the distance between two sequences of characters is based on Levenshtein distance (see Section 2.3). The library was designed for one of the most popular Web's event ticket search engine *SeatGeek*. It basically searches all tickets sites in order to find the best price for a ticket for a specific event.

- **SpaCy NER Annotator** is a web service offering an easy way to annotate text with self-defined labels/tags manually [33].

3. My Research Process and Encountered Problems

This chapter describes in details the whole research process that accompanied creating *protagonistTagger*, along with problems encountered on the way. The work on the project can be divided into several separate steps described in details in Chapter 4. Each of these steps includes an analysis of the problem, testing several tools and preparing a workshop for more advanced topics further in the project. Almost during every phase, some exciting ambiguities appeared or surprising difficulties which are worth mentioning. They are described in details in this chapter. The research process was performed on a fixed set of ten novels, for which all the results and findings are presented. A testing set used also further in the projects, was extracted from these novels. Details on constructing the testing sets are given in Chapter 4.

3.1. Imperfections of NER in Novels

NER (described in details in Section 2.2) aims at fining all named entities of category *person* which may be matched in the further step with the proper label. Therefore, we want the NER model to find as many entities of this kind as possible. Ideally, it should be able to find all named entities that correspond to any protagonist in a discussed novel. In case when named entity of the other category, for example, *location*, is recognized as *person* it is simply not matched with any protagonist's label in the matching phase. The *matching algorithm* (described in details in Section 4.4) ignores named entities that don't resemble any of the given tags. Therefore, we want the NER model to have the highest possible recall. Precision at this step is not crucial. It is taken into account to a greater extent in the *matching algorithm*. At this point, I want to find all relevant entities in the analyzed text.

3.1.1. NER Model Performance

The novel is a particular type of text. Therefore, I needed to verify the performance of standard NER model on exemplary sentences extracted from novels. The used model is a general-purpose, pre-trained one, that can be used to predict named entities, part-of-speech tags and

syntactic dependencies. The model, in a form provided by the spaCy library (described in Section 2.4), can be used straight away or it can be fine-tuned on more specific data. It was trained on web data such as blogs, news and comments. The type of training data differs significantly from novels. Therefore, it may be necessary to fine-tune the standard model with some novel-specific data [30], [23].

The testing set for standard NER model includes 100 sentences containing named entities extracted randomly from 10 novels. All together the testing set for NER model includes 1000 sentences, and it has been annotated manually with the general tag *person*. The metrics describing the performance of the NER model on this testing set are presented in Table 3.1. They are given only for named entities of category *person*. The overall recall on the whole testing set is 0.8. The result is not bad at first sight. However, we need to bear in mind that this is the first part of the annotation process. When we add the potential error of the *matching algorithm*, the performance of the whole method may turn out to be very low.

3.1.2. Not Recognized Named Entities

It is worth to investigate which named entities are troublesome for the standard NER model. The analysis of the NER model performance is based mainly on the three novels with the lowest recall according to Table 3.1. The examples of named entities of category *person* not recognized by the NER model are given in Table 3.2. These lists of named entities may not be complete. Not recognized or incorrectly classified named entities are discovered by manually analysing the annotations done by NER model on a testing set. It is possible that some names of tangential (not playing a crucial role in a plot) literary characters are not present in the testing set. Therefore, it is not verified whether the names of these characters are correctly recognized and classified as proper named entity categories. However, as it has been mentioned before, the main goal is to achieve a good performance on novels in general. Therefore, the main protagonists are crucial here and their names always appear in the testing sets.

The most alarming thing is the fact that in some of the tested novels, the main protagonist's name is not recognized by the NER model as an entity of category *person*. It is the case in *The Picture of Dorian Gray* where the entity *Dorian* is recognized as *norp*, which stands for Nationalities or religious or political groups. A similar situation is with the novel *Emma*. In this case, the entity *Emma* is given a label *org* that should be assigned to companies, agencies or institutions. It is easy to imagine what happens with the overall performance of the *protagonist-Tagger* on a novel when the name of the main protagonist is not recognized correctly. It drops drastically, even with good *matching algorithm*. Therefore, it is crucial to fine-tune the NER

Novel title	precision	recall	F-measure	support
The Picture of Dorian Gray	0.69	0.41	0.51	90
Frankenstein	0.91	0.62	0.74	93
Treasure Island	0.75	0.66	0.7	97
Emma	0.84	0.77	0.81	115
Jane Eyre	0.86	0.78	0.82	97
Wuthering Heights	0.95	0.87	0.91	108
Pride and Prejudice	0.85	0.87	0.86	124
Dracula	0.86	0.94	0.9	97
Anne of Green Gables	0.91	0.96	0.94	114
Adventures of Huckleberry Finn	0.71	0.99	0.83	86
***** Overall results *****	0.84	0.8	0.82	1021

Table 3.1: Metrics computed for **standard, pretrained, not fine-tuned NER model** for general label *person* for each part of the testing set devoted for different novels, as well as for the whole testing set in general. The *support* is the number of occurrences of class *person* in the correct target values. In red, there are marked the most alarming results regarding the recall.

model, so that it can handle these entities. The process of fine-tuning NER model is described in details in section 4.3.

3.2. Problems in Assigning Recognized Named Entities to Specific Literary Characters

At this point, it is assumed that we are given the list of full names of protagonists in the novel. The task is to recognize the named entities in the text and match them properly with labels from this list. The task of assigning named entities to the literary characters from the list may seem trivial. However, we need to keep in mind that the named entities found in the novel rarely take the form the same as in the list. Sometimes only the first name is used, in other cases surname preceded with a personal title (as described in Section 3.2.3). In extreme cases, a diminutive or a nickname may be used (for example *Lizzy* instead of *Elizabeth* or *Nelly* instead of *Ellen*).

Taking it all into consideration, it becomes visible that the task is not trivial anymore. Somehow it needs to be verified how similar is a named entity to each label from the list. Only then it will be possible to assign a proper, most similar label to the entity. A technique called *approximate text matching* can be used to calculate this similarity. The theoretical background of this technique is presented in Section 2.3. The method is available via an open-source python

Novel title	Named entities of category *person* not recognized by NER
The Picture of Dorian Gray	Dorian/Dorian Gray, Sibyl Vane, Hallward/Basil/Basil Hallward
Frankenstein	Safie, Victor, Felix, Walton, Justine, creature/monster, Clerval, De Lacey
Treasue Island	Flint/Cap'n Flint, Silver (however John Silver is recognized), Black Dog, Gray, Trelawney, Billy Bones, Hawkins, Arrow (however Mr. Arrow is recognized), Pew
Emma	Emma/Miss Woodhouse, Harriet
Jane Eyre	Blanche/Blanche Ingram/Miss Ingram, Bessie, Leah, Miss Eyre, Helen, Georgiana, Rosamond, Fairfax Rochester, Rivers, Madam Reed, Miss Temple, Grace
Wuthering Heights	Nelly (however Ellen is recognized), Linton, Hindley, Hareton, Isabella, Heathcliff
Pride and Prejudice	Charlotte, Bingley, Wickham, Lydia, Gardiners, Georgiana, Kitty
Dracula	Arthur, Art, Count Dracula, Harker
Anne of Green Gables	Anne, Gilbert, Diana (in some cases)
Adventures of Huckleberry Finn	the duke, Aunty (Aunt Sally Phelps)

Table 3.2: Examples of entities not recognized in the testing set. These errors were discovered while manually checking the correctness of annotations with a general tag *person* on the testing set.

library called *FuzzyWuzzy* (see Section 2.4 for detailed technical information).

3.2.1. Regular and Partial String Matching

String similarity can be counted in multiple different ways. The general method uses Levenshtein distance to calculate differences between two sequences of characters (see Section 2.3 for more theoretical details). In the context of our problem, the basic measurement of edit distance between a named entity found in a text and a character name from a list may not solve the problem. Using this method for entity *Elizabeth* and full name *Elizabeth Bennet* gives the similarity of only 72%. Such a result is not satisfying. Fortunately, there is also a modification of basic method which calculates so-called *partial string similarity*. It uses a heuristic called *best partial* which, given one sequence of length n and a noticeably shorter string of length m, calculates the score of the best matching substring of length m of the sequence. Using this method in the previous example, we get a similarity of 100%, which is what we expected (see Table 3.3).

3.2.2. Handling Diminutives of Literary Characters

Not all assignments are as easy and straightforward as in the previous example. The most difficult cases are diminutives and nicknames. Let us analyse all occurrences of the character

Named entity	Character's full name	Regular string similarity	Partial string similarity
Elizabeth	Elizabeth Bennet	72%	100%
Lizzy	Elizabeth Bennet	19%	40%
Lizzy	Mr Fitzgerald Darcy	24%	40%

Table 3.3: Example of calculated string similarities for some of the named entities recognized in *Pride and Prejudice*.

Elizabeth Bennet in the novel presented in Table 3.4.

Having read the book it is possible to notice that *Elizabeth* is sometimes called *Lizzy* by her family. Diminutive *Lizzy* is used only approximately 12% of the cases so it is not very common in this specific novel. However, in some novels using diminutives may be more popular and they need to be handled somehow. Information about diminutive *Lizzy* is not given on Wikipedia so it is not included in our list of labels. *Lizzy* is recognized as a named entity of category *people*, but it is not similar enough to any of the protagonists from the list. The partial string similarity between *Lizzy* and *Elizabeth Bennet* equals 40% and is the same as between *Lizzy* and *Mr Fitzwilliam Darcy* (see Table 3.3). As we can see, the straightforward approximate string matching technique is not enough in this case.

Entity	Appearances
Elizabeth	635
Lizzy	96
Miss Bennet	72
Miss Elizabeth	12
Elizabeth Bennet	8

Table 3.4: Appearances of the references to *Elizabeth Bennet* in the novel in different configurations

Fortunately, the list of diminutive for each common English name is finite and usually not very long. Example of the most common diminutives for several names is presented in Figure 3.1. The complete list of diminutives is used only when the recognized named entity is not similar enough to any of the protagonist listed in a list of labels for a given novel. The detailed description of the procedure is presented in Section 4.4.

3.2.3. Named Entities Preceded with Personal Title

The NER is responsible for finding names of protagonists in texts. In order to simplify the verification of the method, initially, only short pieces of texts are analyzed. NER analyses descriptions of literary characters taken from Wikipedia. Table 3.5 presents an exemplary description

```
abigail,nabby,abby,gail
abijah,ab,bige
abner,ab
abraham,ab,abe
abram,ab
absalom,app,ab,abbie
ada,addy
adaline,delia,lena,dell,addy,ada
adam,edie,ade
addy,adele
adela,della
adelaide,heidi,adele,dell,addy,della
```

Figure 3.1: Example of diminutives for some common English names.

of Elizabeth Bennet, the main character of *Pride and Prejudice* by Jane Austen.

> *"the second eldest of the **BENNET** daughters, she is twenty years old and intelligent, lively, playful, attractive, and witty but with a tendency to form tenacious and prejudicial first impressions. As the story progresses, so does her relationship with Mr **DARCY**. The course of **ELIZABETH** and **DARCY**'s relationship is ultimately decided when **DARCY** overcomes his pride, and **ELIZABETH** overcomes her prejudice, leading them both to surrender to their love for each other."*

Table 3.5: An exemplary description of Elizabeth Bennet, the main character of *Pride and Prejudice* by Jane Austen

All named entities of the category *people* recognized by NER in the description of Elizabeth Bennet presented in Table 3.5 are in bold. Names of *Elizabeth* and *Mr Darcy* can be easily matched with protagonists of the novel. However, there is a problem with the entity *Bennet*. In this context, *Bennet* is not the name of any specific character, but rather the name of the whole family. Somehow we need to distinguish between *Bennet* meaning the whole family and *Bennet* being the surname of a single character. There is a regularity that can be observed in this case. Namely, *Bennet* preceded with a personal title such as Mr., Mrs., Ms. or Miss should be identified as a single person, whose surname is *Bennet*. In all other cases, *Bennet* is treated as the whole family and not a single person that can be identified in a text.

To illustrate the scale of the problem, let us investigate the situation of the entity *Bennet* in the whole novel *Pride and Prejudice*. The novel has 121,533 words. The entity *Bennet* appears 323 times according to the Table 3.6. However, in these appearances entities like *Mrs. Bennet, Mr Bennet, Miss Bennet* are also included which are not recognized properly by the NER,

even though the personal title totally changes the meaning here. Therefore, in 314 out of 323 appearances of the entity *Bennet*, it is misunderstood, and some meaning is lost. This problem can be easily handled by verifying whether there is a personal title before the surname in cases when the named entity includes only a surname. In case when a surname is preceded with the name, for example, *Elizabeth Bennet* there is no problem because NER recognizes it simply as a one entity containing two tokens.

Entity	Appearances
Bennet	323
Mrs. Bennet	153
Mr. Bennet	89
Miss Bennet	72

Table 3.6: Appearances of the entity *Bennet* in the novel in different configurations. Each configuration is simply searched in the whole text of the novel and the number of its appearances is given in the table.

3.3. Summary of the Most Important Conclusions and Findings

This section contains a summary of all findings made during the research problem. It concludes the aspects of the problem that need to be handled in the final solution. The table 3.7 contains the most important findings along with some simple examples. This section aims to assemble all the information in one place and organize them to make the understanding of the whole method easier.

Conclusion	**Detailed information about the finding**	**Example**
Standard NER model performance		
Standard NER models do not perform very well on novels	Due to the fact that most standard NER models are trained on data available on the web, such as news, comments, blogs, they may not be well-adjusted to literary texts from novels	The recall of standard NER model on sentences extracted from novels is below 70% in some cases (see Table 3.1).
Some typical English names are not given a correct named entity category	In some novels, the names of the main protagonist are recognized as named entities, however, they are assigned with an incorrect category (such as *norp*, or *org*)	Name *Emma* is given a label *org*, that should be assigned to companies
Assigning recognized named entities to specific literary characters		
Partial string similarity is necessary for matching a recognized named entity with a proper protagonist	Due to the fact that the protagonists in novels are sometimes called only by their first name or only surname regular string similarity is not enough to make a correct match. Therefore a *best partial* strategy is used.	Regular string similarity of *(Elizabeth Bennet, Elizabeth)* equals 72%, whereas partial-string similarity equals 100%
Personal title (eg. Mr, Miss) can be used as an indicator of the protagonist	In the novels, it may happen that there are several protagonists with the same surname. The personal title preceding the surname can be used as a hint for matching it with a proper protagonist	*Miss Bingley* is definitely *Caroline Bingley*, not *Charles Bingley*
Diminutives can be handled with an external source of data	Diminutives may decrease the performance of the *protagnistTagger* as they can not be easily matched with protagonists using string similarity. Therefore, an external source of data is used to find a base form of a name for a diminutive appearing in a text.	A name *Abby* can be connected with its base form *Abigail* and then easily matched with a protagonist named *Abigail Smith* (using string similarity).

Table 3.7: The summary of the information gathered in the research process. It contains a list of all main conclusions along with several simple examples. The first part of the table concerns the first stage of *protagonistTagger* - recognizing named entities of category *person*. And the second part of the table concerns the second stage of the tool - matching each recognized named entity with a tag of a proper protagonist.

4. My Approach

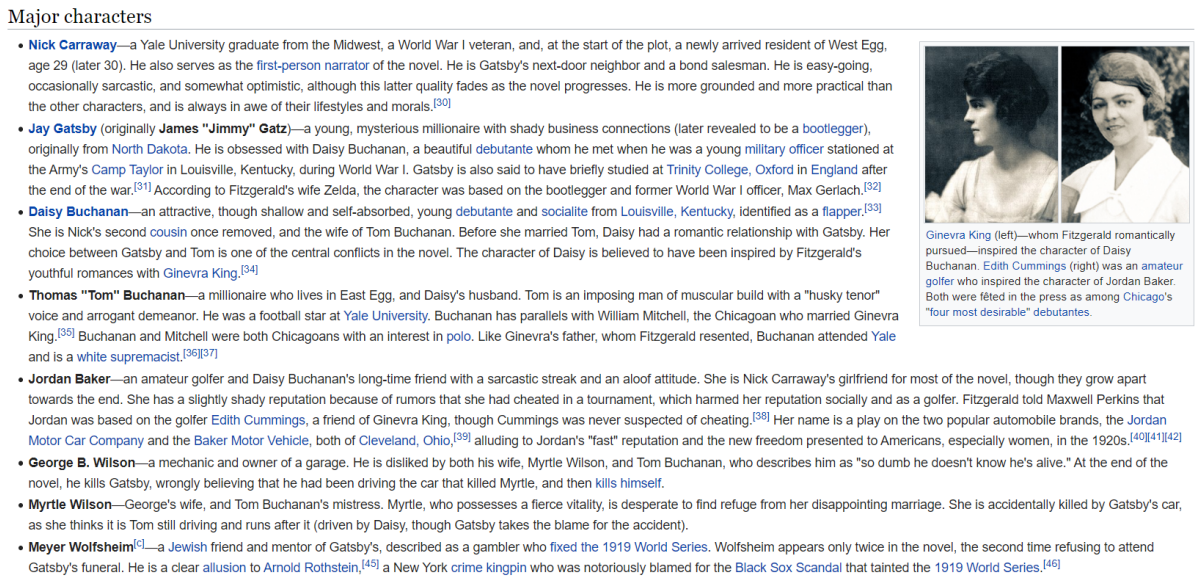
This chapter describes in details all steps of my project workflow. Theoretical aspects of methods and algorithms used on the way are discussed in Chapter 2 and are omitted here. This chapter offers only an overview of the desired outcomes of each step, and it does not cover the analysis of the encountered problems and the ways of solving them. The in-depth analysis of these topics is given in Chapter 3.

4.1. Initial Corpus with Plain Novels' Texts

A corpus of novels' texts is the pillar of the whole projects. The desired format of each novel is a plain text starting with a first chapter and ending with the last one. All text connected with publications, editorship, footnotes, prefaces, etc. should be removed. What one wants to analyse is only the plain novel text. Because novels themselves are usually very long and complex texts, there is no need to create a corpus with a large number of titles in it. Thirteen novels should suffice to begin with.

4.2. Lists of Full Names of the Protagonists

This step uses *Wikipedia* parser, which goes through an article about given novel looking for a section devoted to its literary characters. In the case of well-known novels, such section is usually a part of *Wikipedia* article. The standard structure of this section is a list of full names of all crucial protagonists along with tangential ones. Sometimes the names of protagonists are followed by a short description of the character. In the majority of cases, it is possible to easily extract the list of all the names of the protagonists from the given article. An example of such a protagonists' section taken from Wikipedia is presented in Figure 4.1. The extracted list of names of literary characters is used as the predefined tags for a given novel in the annotation process in the next steps.



Major characters

- **Nick Carraway**—a Yale University graduate from the Midwest, a World War I veteran, and, at the start of the plot, a newly arrived resident of West Egg, age 29 (later 30). He also serves as the first-person narrator of the novel. He is Gatsby's next-door neighbor and a bond salesman. He is easy-going, occasionally sarcastic, and somewhat optimistic, although this latter quality fades as the novel progresses. He is more grounded and more practical than the other characters, and is always in awe of their lifestyles and morals.[30]
- **Jay Gatsby** (originally **James "Jimmy" Gatz**)—a young, mysterious millionaire with shady business connections (later revealed to be a bootlegger), originally from North Dakota. He is obsessed with Daisy Buchanan, a beautiful debutante whom he met when he was a young military officer stationed at the Army's Camp Taylor in Louisville, Kentucky, during World War I. Gatsby is also said to have briefly studied at Trinity College, Oxford in England after the end of the war.[31] According to Fitzgerald's wife Zelda, the character was based on the bootlegger and former World War I officer, Max Gerlach.[32]
- **Daisy Buchanan**—an attractive, though shallow and self-absorbed, young debutante and socialite from Louisville, Kentucky, identified as a flapper.[33] She is Nick's second cousin once removed, and the wife of Tom Buchanan. Before she married Tom, Daisy had a romantic relationship with Gatsby. Her choice between Gatsby and Tom is one of the central conflicts in the novel. The character of Daisy is believed to have been inspired by Fitzgerald's youthful romances with Ginevra King.[34]
- **Thomas "Tom" Buchanan**—a millionaire who lives in East Egg, and Daisy's husband. Tom is an imposing man of muscular build with a "husky tenor" voice and arrogant demeanor. He was a football star at Yale University. Buchanan has parallels with William Mitchell, the Chicagoan who married Ginevra King.[35] Buchanan and Mitchell were both Chicagoans with an interest in polo. Like Ginevra's father, whom Fitzgerald resented, Buchanan attended Yale and is a white supremacist.[36][37]

Ginevra King (left)—whom Fitzgerald romantically pursued—inspired the character of Daisy Buchanan. Edith Cummings (right) was an amateur golfer who inspired the character of Jordan Baker. Both were fêted in the press as among Chicago's "four most desirable" debutantes.

- **Jordan Baker**—an amateur golfer and Daisy Buchanan's long-time friend with a sarcastic streak and an aloof attitude. She is Nick Carraway's girlfriend for most of the novel, though they grow apart towards the end. She has a slightly shady reputation because of rumors that she had cheated in a tournament, which harmed her reputation socially and as a golfer. Fitzgerald told Maxwell Perkins that Jordan was based on the golfer Edith Cummings, a friend of Ginevra King, though Cummings was never suspected of cheating.[38] Her name is a play on the two popular automobile brands, the Jordan Motor Car Company and the Baker Motor Vehicle, both of Cleveland, Ohio,[39] alluding to Jordan's "fast" reputation and the new freedom presented to Americans, especially women, in the 1920s.[40][41][42]
- **George B. Wilson**—a mechanic and owner of a garage. He is disliked by both his wife, Myrtle Wilson, and Tom Buchanan, who describes him as "so dumb he doesn't know he's alive." At the end of the novel, he kills Gatsby, wrongly believing that he had been driving the car that killed Myrtle, and then kills himself.
- **Myrtle Wilson**—George's wife, and Tom Buchanan's mistress. Myrtle, who possesses a fierce vitality, is desperate to find refuge from her disappointing marriage. She is accidentally killed by Gatsby's car, as she thinks it is Tom still driving and runs after it (driven by Daisy, though Gatsby takes the blame for the accident).
- **Meyer Wolfsheim**[c]—a Jewish friend and mentor of Gatsby's, described as a gambler who fixed the 1919 World Series. Wolfsheim appears only twice in the novel, the second time refusing to attend Gatsby's funeral. He is a clear allusion to Arnold Rothstein,[45] a New York crime kingpin who was notoriously blamed for the Black Sox Scandal that tainted the 1919 World Series.[46]

Figure 4.1: Example of a section from the article from Wikipedia devoted for characters of a novel.

4.3. Recognizing Protagonists Appearances Using NER

The next step is to recognize tokens in the text of the novel, which pertain to named entities of type *person*. These named entities are potential candidates to be annotated with a proper tag with the protagonist's full name. Even though the names of people are basic examples of named entities, standard models used for NER may not be enough to handle novels. The majority of names, surnames or combinations of both is usually found without any problems. However, there are plenty of cases which a standard NER model may not be able to handle. It may result in a drop in performance. Examples of entities problematic for standard NER model are given in Table 3.2 in Section 3.1.2.

4.3.1. Fine-tuning NER Model – Outline

An iterative procedure of fine-tuning NER model is used in order to improve its performance on novels. It assumes a few iterations of fine-tuning NER model and verifying its performance. It can be divided into the following steps:

1. NER is applied to the testing set. The results (the model output) are compared with testing set manually annotated with the tag *person*.

2. If the results of NER are not satisfying (many entities that are of category *person* are not recognized or they are assigned a different category) then NER needs to be fine-tuned. In

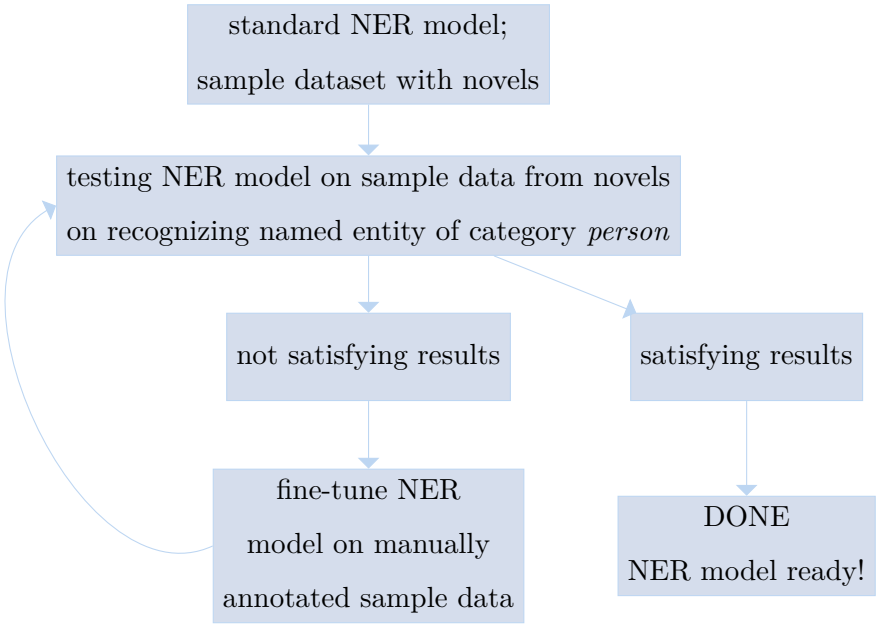


Figure 4.2: Simplified process of fine-tuning NER model.

order to do it, a set with manually annotated entities of category *person* not recognized by NER is created. This set is used to fine-tune the NER model. And we go back to step 1.

3. If the results of NER are satisfying, we can move further.

The testing set for evaluating the performance of NER is extracted from the texts of the novels. When the performance is not satisfying, it needs to be verified which entities of category *person* are not recognized in the testing set. The next step is to create a training set containing examples of such entities annotated manually. This set is used to fine-tune the NER model. Furthermore, the whole procedure repeats until the performance of the NER model is satisfying. The general idea of this procedure is presented in Figure 4.2.

4.3.2. Training Sets for NER Fine-tuning

I considered two approaches to creating a training set for a NER model. The imperfections of the standard NER model visible on novels are meaningful enough to pay significant attention to fine-tuning it.

Training_set_1 – Sentences with not Recognized Named Entities of Category *person*

The first approach to creating a training set assumes using the named entities of type *person* that were not recognized or that were not assigned a proper type. The novels used for creating the testing set are scanned in search of sentences containing these entities. Each entity is represented

by a similar number of sentences. Then the chosen sentences are annotated in a semi-automatic way with a general tag *person* creating a training set for fine-tuning the standard NER model. Exemplary training set created this way contains approximately five sentences with each not correctly recognized named entity of type *person*. It gives all together 485 sentences.

Training_set_2 – Sentences from Novels with Injected Common English Names

Many common English names, such as *Emma, Charlotte, Arthur or Grace*, are not recognized at all by a standard NER model, or they are classified as entities of a type different than *person*. Thus the second approach to creating a training set is based on this observation. In order to improve the performance in this area, the training set needs to contain sentences typical for novels, when it comes to style, vocabulary, syntax. These sentences should additionally contain as many common English names as possible. The easiest way to create such a set of syntactically and semantically correct sentences is to extract from novels sentences containing, for example, names of the main protagonist. Then each such name can be replaced by some other common English name in order to enrich our training set [35]. An example of such replacement in given in Table 4.1. The considered list of most common English names contains almost 300 female and almost 300 male names. The exemplary training set created this way contains more than 1600 sentences with each common English name appearing there approximately three times.

*"**Jane**'s delicate sense of honour would not allow her to speak to **Elizabeth** privately of what **Lydia** had let fall; **Elizabeth** was glad of it; till it appeared whether her inquiries would receive any satisfaction, she had rather be without a confidante."*
*"**Deborah**'s delicate sense of honour would not allow her to speak to **Harvey** privately of what **Lydia** had let fall; **Harvey** was glad of it; till it appeared whether her inquiries would receive any satisfaction, she had rather be without a confidante."*

Table 4.1: An example of replacing a name of the main protagonist with some other common English name. *Jane* is replaced by *Deborah* and *Elizabeth* is replaced by *Harvey*. Sentence is extracted from *Pride and Prejudice* by Jane Austen.

4.3.3. Fine-tuning NER Model

As a base for fine-tuning, I use an existing, pre-trained language model. It is using embeddings with subwords features, convolutional layers with residual connections, layer normalization and maxout non-linearity (theoretical background is given in Section 2.2). The training data is shuffled and batched. For each batch, the model is updated with the training sentences from

a batch. In order to make it a little bit harder for the model to memorize data and to reduce overfitting, the dropout is used.

Name of the testing set	# sentences per novel	# novels included	Size (in sentences)	Tags in gold standard
Testing_set_large_Tag-person	100	10	1000	*person*
Testing_set_large_Tag-full-names	100	10	1000	full names of literary characters
Testing_set_small_Tag-person	100	3	300	*person*
Testing_set_small_Tag-full-names	100	3	300	full names of literary characters

Table 4.2: Summary of all testing sets used. For every set, there is a gold standard created manually.

4.3.4. Testing Sets for NER Fine-tuning

Two testing sets are used for verifying the performance of the NER model. The first one is the same as the one used in the research process described in Section 3.1. It contains 100 sentences from 10 novels - altogether 1000 sentences. These sentences contain various named entities of category *person*. The gold standard of this dataset was annotated manually with a tag *person*. The novels included in this testing set are also used in the training data, even though the training set and the testing set are disjoint. Additionally, this testing set has been already used for verifying which named entities are not recognized and classified properly. Taking this into consideration, the results of the model on this dataset may not be fully trustworthy. Therefore an additional testing set has been created. It contains 100 sentences from 3 different novels - altogether 300 sentences containing different named entities of category *person*. Again, this testing set has been annotated manually with a tag *person* in order to create a gold standard. The results of the NER model on this testing set are undoubtedly authoritative. A summary of all testing sets used so far, as well as the ones that are created for testing the performance of the *ProtagonistsTagger* are given in Table 4.2.

4.3.5. Fine-tuned NER Model Performance

The performance has been analyzed for three different versions of the NER model [17]:

- standard NER model - existing, pre-trained, standard model provided by the spaCy library (the same model as in the research process described in Section 3.1)
- fine-tuned model 1 - standard NER model fine-tuned with a training_set_1

Novel title / NER model		precision	recall	F-measure	support
The Picture of Dorian Gray	standard	0.69	0.41	0.51	90
	fine-tuned 1	0.71	1	0.83	90
	fine-tuned 2	0.74	1	0.85	90
Frankenstein	standard	0.91	0.62	0.74	93
	fine-tuned 1	0.76	0.99	0.86	93
	fine-tuned 2	0.78	0.98	0.87	93
Treasure Island	standard	0.75	0.66	0.7	97
	fine-tuned 1	0.75	0.95	0.84	97
	fine-tuned 2	0.78	1	0.87	97
Emma	standard	0.84	0.77	0.81	115
	fine-tuned 1	0.83	1	0.91	115
	fine-tuned 2	0.85	1	0.92	115
Jane Eyre	standard	0.86	0.78	0.82	97
	fine-tuned 1	0.77	0.96	0.85	97
	fine-tuned 2	0.74	0.95	0.83	97
Wuthering Heights	standard	0.95	0.87	0.91	108
	fine-tuned 1	0.88	0.98	0.93	108
	fine-tuned 2	0.88	0.99	0.93	108
Pride and Prejudice	standard	0.85	0.87	0.86	124
	fine-tuned 1	0.75	0.98	0.85	124
	fine-tuned 2	0.8	0.98	0.88	124
Dracula	standard	0.86	0.94	0.9	97
	fine-tuned 1	0.72	0.93	0.81	97
	fine-tuned 2	0.72	0.99	0.83	97
Anne of Green Gables	standard	0.91	0.96	0.94	114
	fine-tuned 1	0.82	1	0.9	113
	fine-tuned 2	0.85	0.99	0.92	113
Adventures of Huckleberry Finn	standard	0.71	0.99	0.83	86
	fine-tuned 1	0.6	0.99	0.75	86
	fine-tuned 2	0.61	1	0.75	86
***** Overall results *****	standard	0.84	0.8	0.82	1021
	fine-tuned 1	0.76	0.98	0.85	1020
	fine-tuned 2	0.77	0.99	0.87	1020

Table 4.3: Metrics computed for: standard, pretrained NER model and both fine-tuned NER models for general label *person*. They are computed separately for each part of testing set *Testing_set_large_Tag-person* devoted for different novels as well as for the whole testing set in general. The *support* is the number of occurrences of class *person* in the correct target values. In red, there are marked the most alarming results regarding the recall.

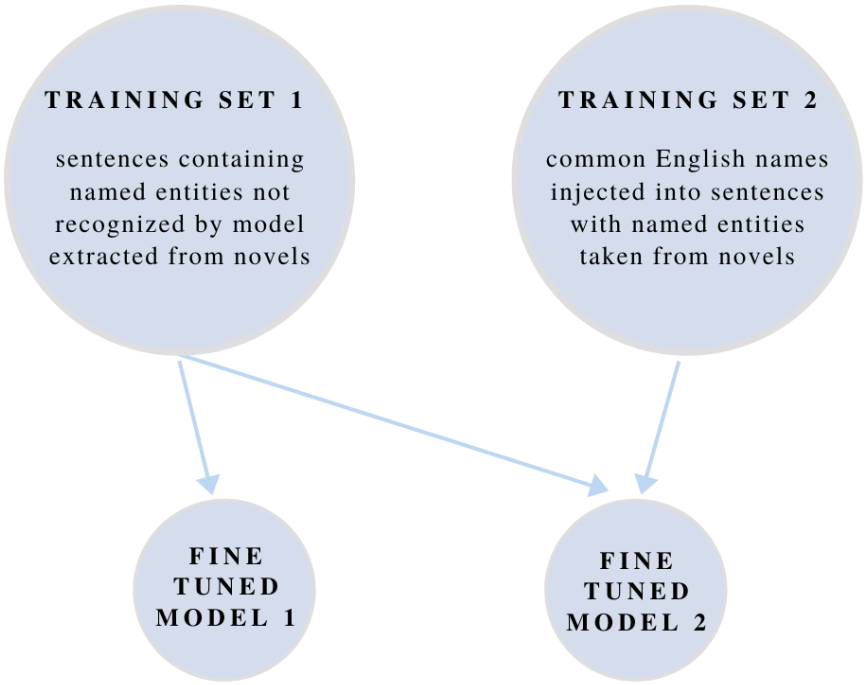


Figure 4.3: Visualization of fine-tuned NER models.

- fine-tuned model 2 - standard NER model fine-tuned using both training sets: training_set_1 and training_set_2

The fine-tuned models are presented in Figure 4.3 just for reference.

Performance of all NER models is presented in Table 4.3. The testing set that is used at this point is *Testing_set_large_Tag-person* (see Table 4.2 for more details about the testing sets). At this point, we are interested mostly in the recall on the testing set. We are fine-tuning NER in order to be able to recognize all named entities of category *person*. The obvious conclusion from the results on this testing set is that the recall for standard NER model, in almost all cases, is significantly lower than for fine-tuned models. It means that indeed the standard NER models are not prepared for novels. Now the question is, which training set used for fine-tuning the model is the best. Taking into consideration the whole testing set, NER model fine-tuned with both training sets (**fine-tuned model 2**) reaches slightly better performance. Considering every novel in the testing set separately, **fine-tuned model 2** has a higher recall in case of 4 out of 10 novels and **fine-tuned model 1** in case of 3 out of 10 novels. Therefore we can say that the model fine-tuned with two training sets (**fine-tuned model 2**) turns out to be a little bit better on this testing set. It means that including in the training set sentences with names of the main protagonists not recognized by standard NER model is not enough. It turns out that there is also a non-negligible set of names of secondary or episodic characters that are not

Novel title / NER model		precision	recall	F-measure	support
The Catcher in the Rye	standard	0.68	0.68	0.68	74
	fine-tuned 1	0.59	0.78	0.67	74
	fine-tuned 2	0.58	0.91	0.71	74
The Great Gatsby	standard	0.75	0.84	0.79	102
	fine-tuned 1	0.66	0.95	0.78	102
	fine-tuned 2	0.66	0.98	0.79	102
The Secret Garden	standard	0.9	0.82	0.86	97
	fine-tuned 1	0.81	0.96	0.88	97
	fine-tuned 2	0.83	0.95	0.88	97
***** Overall results *****	standard	0.78	0.79	0.78	273
	fine-tuned 1	0.69	0.91	0.78	273
	fine-tuned 2	0.69	0.95	0.8	273

Table 4.4: Metrics computed for: standard, pretrained NER model and both fine-tuned NER models for general label *person*. They are computed for new testing set *Testing_set_small_Tag-person*. They are computed separately for each part of the testing set devoted for different novels as well as for the whole testing set in general. The *support* is the number of occurrences of class *person* in the correct target values.

properly recognized and that are influencing the performance.

The novels used for creating testing set are also used for creating training sets for fine-tuning NER. It may make the results of this testing set not fully reliable. Therefore it is also worth checking how the analyzed NER models perform on a brand new data extracted from different novels. The second testing set used is *Testing_set_small_Tag-person* (see Table 4.2 for more details about the testing sets). The performance of all NER models on this new testing set is presented in Table 4.4. The recall of this new testing set makes us draw similar conclusions as for the first testing set. Again **fine-tuned model 2** has a higher recall on the whole testing set in general, as well as in case of almost every novel analyzed individually. However, what is interesting, the differences in recall between these two fine-tuned models on this new testing set are more visible. All in all, we can draw a conclusion that NER model fine-tuned using both testing sets (**fine-tuned model 2**) achieves better results and is more desired for this specific task. What is important, its performance on a new testing set is also very high.

It is also worth analyzing other metrics on both testing sets. Observations are almost the same for both testing sets (with results presented in Table 4.3 and in Table 4.4). Low precision in our case means that there is a lot of named entities recognized as *person* that are in reality named entities of some other category. Standard NER model has, in general, the highest precision but it is achieved at the expense of much lower recall. Taking into consideration each novel

separately, **fine-tuned model 2** presents slightly higher precision than **fine-tuned model 1**. It is another argument in favour of **fine-tuned model 2**. When it comes to F-measure, taking into consideration results for both testing sets, again **fine-tuned model 2** overachieves the other two NER models.

4.4. Using *Matching Algorithm*

The *matching algorithm* is supposed to assign a proper tag with the protagonist's full name to the named entity recognized in the previous step. The algorithm evaluates the match between the recognized named entity and each tag from the list predefined for a given novel. The tag with the highest resemblance is chosen as an answer. The method is mostly based on *approximate string matching* (see Section 2.3 for more theoretical details).

Matching Algorithm Outline

This section presents the general idea for the matching algorithm, that is finding the best match for a recognized named entity of category *person* in the list of protagonists appearing in the novel. The algorithm tries to solve the problem of personal titles presented in Section 3.2.3. Furthermore, diminutives appearing as a part of recognized named entities are handled. Due to many ambiguities and the number of cases that need to be handled, the method is not 100% correct and precise. However, the problems that are appearing most frequently in the analyzed novels are taken into consideration. The pseudo-code of the whole procedure is presented in Algorithm 1.

The algorithm takes as input:

- **named_entity** - a named entity of category *person* found by NER model,
- **protagonists** - a list of all literary characters/protagonists in the novel,
- **prefix** - a prefix that is a token appearing before the recognized named entity; it can be a personal title or an article *the*, or empty string otherwise,
- **partial_similarity_precision** - value indicating how similar two strings need to be in order to be considered as potential matches; it is used as lower bound for *partial string similarity* described in Section 3.2.1.

Algorithm 1: Finding best match for the recognized named entity in the list of literary characters

```
1  potential_matches = [];
2  for protagonist in protagonists do
3      ratio = regular_string_similarity(protagonist, named_entity);
4      if ratio == 100 then
5          return match = protagonist;
6      partial_ratio = partial_string_similarity(protagonist, named_entity);
7      if partial_ratio >= partial_similarity_precision then
8          potential_matches.add(protagonist);
9  end
10 potential_matches = sorted(potential_matches) # wrt partial_ratio ;
11 match = None ;
12 if len(potential_matches) > 1 then
13     match = potential_matches[0];
14     if prefix is not None then
15         if prefix == the then
16             return match = the + named_entity
17         title_gender = get_title_gender(prefix) # either female or male ;
18         for potential_match in potential_matches do
19             if get_name_gender(potential_match) == title_gender then
20                 return match = potential_match
21         end
22     return match
23 else if len(potential_matches) == 0 then
24     original_name = get_name_from_diminutive(named_entity);
25     if original_name is not None then
26         return match = protagonist from protagonists that contains original_name
27     else
28         return "person"
29 return potential_match[0]
```

The Algorithm 1 is the core of a proper annotation process. It is responsible for the correct assignment of a protagonist from a list to a named entity found in the text of the novel. It addresses all the important problems recognized and described in the Chapter 3. Due to many

exceptions that need to be handled, each named entity is analyzed very carefully. The detailed description of the matching algorithm is given below in order to clarify the process of analysis of each named entity of type *person* found in the text.

Algorithm Analysis

The main idea of the *matching algorithm* is to collect potential candidates from the list of protagonists that may correspond to the given named entity. Then the algorithm chooses the best match from this list of potential candidates. In order to verify if a literary character from the *protagonists* list is a potential match, the *approximate string matching* method is used. First of all it is checked (using *regular_string_similarity*) if the *named_entity* is identical to any of the literary characters from the *protagonists* list (lines 3-5). If so, the work of the algorithm is done, and no further analysis is needed. However, if it is not a case, the *partial_ratio* is computed for *named_entity* and each literary character form the *protagonists* list using *partial_string_similarity*. If it is above the given threshold (*partial_similarity_precision*) a given *protagonist* from the *protagonists* list is considered as a potential match (lines 6-8). Having completed these steps, we dispose of a list of potential matches from which the best one needs to be chosen. Therefore, the list of potential matches is sorted decreasingly with respect to the computed *partial_ratio* (line 10).

At this stage, the only thing that is left to do is to check whether the considered named entity is one of the exceptions that we are handling. First of all, we check whether the prefix (token preceding the recognized named entity) can give us any clue (lines 14-21). If the prefix is the article *the*, it is assumed that the whole family with the surname given in the named entity is considered. Therefore, all the literary characters from the *protagonists* list with the same surname as in the given named entity should be returned (lines 15-16). If the prefix is one of the personal titles, the gender indicated by it is recognized. Then the first literary character from *potential_matches* list that has the same gender as the personal title in the prefix is returned (lines 17-21). Here we need to use an assumption that the name which is higher in the list is more probable due to the higher similarity score. It may not always be the case, but some simplifications are necessary. The last considered variant appears when not even a single literary character was qualified as a potential match. The reason for such a situation may be the fact that the named entity includes not the common name of the literary character but the diminutive. In such a case, the additional search is performed in an external dictionary of diminutives (lines 23-26). If the named entity is not found in the dictionary of diminutive a general tag *person* is returned. It means that any of the predefined tags match the named entity.

The matching algorithm, of course, is not handling all of the possible cases. It would be highly ineffective when it comes to its complexity. The role of the matching algorithm is to assign a literary character from a list to a recognized named entity in the text of the novel as precisely as possible. Some assignments may turn out to be incorrect. However, we need to keep in mind that the purpose of the project is not creating a corpus of entirely correctly annotated novels but a corpus annotated in a way enabling further analysis of the novels. It means that some margin for error can be accepted.

4.5. *ProtagonistTagger* Workflow

The steps did up to this point compose into the functionalities of the *protagonistTagger*. This tool combines finding named entities of category *person* in a text (using fine-tuned NER) and matching them with a proper tag (using *matching algorithm*). It means that we have now all the components needed to annotate the text of the novel with our predefined tags. This process is presented in the block diagram (see Figure 4.4). A title of a novel is given as an input to the *Wikipedia articles parser*. As a result, a list of protagonists listed in the Wikipedia article is returned. The plain text of the novel is taken from the corpus and analyzed using the fine-tuned NER. The NER finds named entities of category *person* which need to be labelled. Having the list of protagonists and the named entities recognized in the text, the *matching algorithm* is used. It returns, as a result, a text of the novel labelled with proper protagonists' names.

4.6. Evaluating Annotations Done by the *protagonistTagger*

In order to verify the correctness of the *matching algorithm,* a testing data set is created. It is annotated using the *matching algorithm* and compared with the gold standard created manually. The results are given in the form of standard metrics such as precision, recall and F measure. The *matching algorithm* is being improved until the results can be considered as satisfying. A detailed description of the evaluation process is given in Chapter 5.

4.7. Creating a Corpus of Annotated Novels

The final stage is creating a corpus of novels using the *protagonistTagger*. For each novel that is desired to be in a corpus, we need to prepare a full text of it and a list of protagonists' names

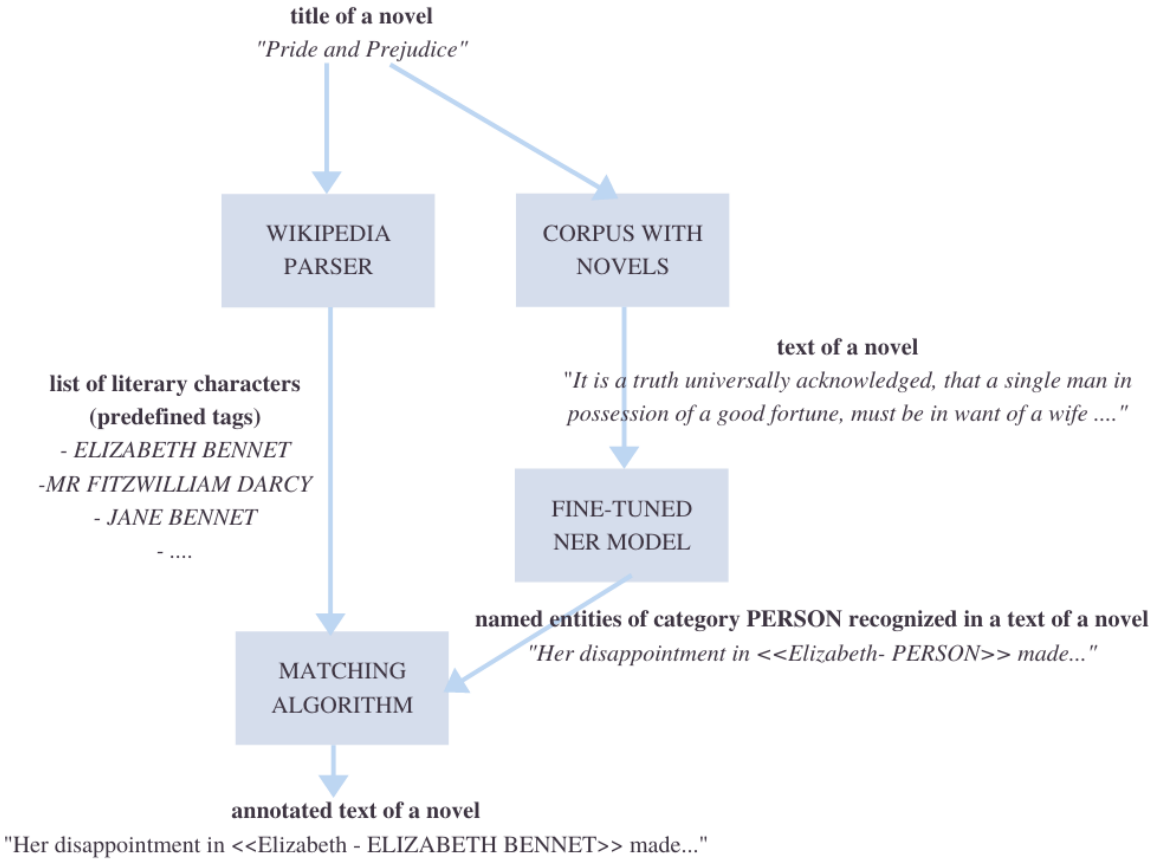


Figure 4.4: Simplified process of annotating named entities of category 'person' with proper literary characters' names in a novel.

that is used as a list of tags. This data are given as an input to the *protagonistTagger*. The final corpus contains thirteen annotated texts of novels. However, thanks to the created tool (*protagonistTagger*) it can be easily extended as needed.

5. Evaluation of the *ProtagonistTagger*

The *protagonistTagger* (described in details in Chapter 4) is evaluated on the prepared testing sets (see Table 4.2 for details about testing sets):

- *Testing_set_large_Tag-full-names*,
- *Testing_set_small_Tag-full-names*.

Testing sets are sampled from the novels' texts. They include the same sentences as the sets used for testing the NER model (see Section 5.2 for more details about testing sets). The sentences are manually annotated with full names of literary characters, creating the gold standard. The annotations done by the *protagonistTagger* are compared with the gold standard. Performance is presented using standard metrics described in details in Section 2.1.

5.1. Requirements for the *protagonistTagger*

It is not expected that the *protagonistTagger* will correctly annotate all literary characters appearing in the 500-pages book. However, it is desired that it will be able to annotate the majority of the main characters' appearances correctly.

Since the algorithm processes large text files, it needs to work relatively fast. We need to bear in mind that to create a plausible corpus, many novels need to be processed by the *protagonistTagger* and annotated.

Another aspect, worth mentioning, is the variety of difficulties connected with annotating literary characters appearing in the novels (they ere recognized and defined in Chapter 3). Indeed, they need to be considered in the process of creating the corpus. The corpus should contain as many different examples representing these problems as possible, and the *protagonistTagger* should be able to deal with them. Therefore the following requirements can be pointed out for the created tool:

- high, but not perfect accuracy,

- high performance, namely annotating a novel should be done in a reasonable time (in most cases no more than an hour),
- resistance to difficulties connected with annotating literary characters in different novels.

5.2. Testing Dataset

For the testing dataset to be valid, it needs to have two main features. The testing dataset needs to be:

- representative – format, genre and domain vocabulary of the data (in our case different novels in text format) is defined and preserved; furthermore, the testing set resembles closely the intricacies of data;
- balanced – dataset contains examples of novels representing all the difficulties recognized in the analysis of the problem.

The testing datasets are created bearing these conditions in mind.

They contain sentences chosen randomly from multiple novels differing in style and genre. The testing sets contain all together 1300 sentences (100 sentences from each novel). Each sentence in the testing set contains at least one named entity of category *person* recognized with SpaCy (it does not have to be a protagonist of the novel). Therefore, it is guaranteed that each testing set contains sufficient examples on which the performance of the *protagonistTagger* can be verified and evaluated. We need to bear in mind that not all literary characters appearing in a novel are given on Wikipedia. Therefore, not all literary characters appearing in the novel are included in predefined tags. Nevertheless, sentences containing some names of minor literary characters, not included in the list of tags, are considered in the testing set. They should be simply given a general tag *Person*. It helps to verify if some additional, undesired annotations are not put.

5.3. Gold Standard Annotations

A gold standard is about to describe what a *correctly performing model* would output. It defines the requirements for the *protagonistTagger*. Gold standard annotations are created manually. Each sentence in the testing dataset is manually annotated with the tags associated with corresponding literary characters. The annotations are different for novels, and of course, they correspond to the novel from which the sentence is extracted. This way, the dataset with gold

standard annotations containing 1300 sentences is created.

5.3.1. Ambiguities

The annotation may seem trivial at first sight. However, when one starts considering specific examples, it usually turns out that there are many cases when the decision whether to put a tag or which tag to choose is not obvious [22]. Also, in the problem of annotating novels with proper literary characters there occur many such situations. The most common ambiguities are as follows:

1. *Should a personal title be included in an entity or should it be omitted?* - In this case, a decision is caused by the specificity of SpaCy NER model. SpaCy NER model, while recognizing a person, includes in a personal entity titles such as *Miss* or *Madam* but omits *Mr* and *Mrs*. While creating the gold standard, the same convention as in SpaCy standard NER model is preserved.

2. *Should nouns in the genitive, expressing for example possession, be annotated?*. In this case, a named entity can be treated only as a marker of the possession, not the character's actual appearance. However, it is assumed that named entity, regardless of the grammatical case, is treated the same way for simplicity.

3. *Should nouns denoting a function, such as Captain or Uncle, be annotated as a part of named entity?* - In many novels, literary characters are called with names including some function, for example, *Uncle Silas* or *Captain Flint*. Also in the list of literary characters, such collocations appear. We can assume, that a function of the literary character is treated as a part of the named entity only when a noun denoting this function is written in upper case. Again the same convention as in SpaCy standard NER model is preserved.

4. *When should an article 'the' be included in an entity?* - As discussed in Section 3.2.3 the article 'the' is considered meaningful only when it precedes a surname. The named entity should then be understood as a group of people with the same surname, e.g. a whole family, brothers. Thus, the goal is to include the article 'the' in a named entity only in this exact case.

The whole testing set is manually annotated, creating the gold standard, following the above requirements and guidelines. The ambiguities listed above were discovered while manually annotating the gold standard. As the testing set is hopefully representative and large enough for this specific task, the provided list should cover the most critical and common ambiguities in novels.

Novel title	Precision	Recall	F-measure
Pride and Prejudice	0.84	0.85	0.83
The Picture of Dorian Gray	0.96	0.97	0.96
Anne of Green Gables	0.94	0.96	0.95
Wuthering Heights	0.79	0.77	0.77
Jane Eyre	0.8	0.75	0.76
Frankenstein	0.91	0.88	0.89
Treasure Island	0.92	0.91	0.91
Adventures of Huckleberry Finn	0.89	0.93	0.9
Emma	0.93	0.86	0.88
Dracula	0.9	0.89	0.89
***** Overall results *****	0.88	0.87	0.87

Table 5.1: Performance of the *protagonistTagger* on the first testing set *Testing_set_large_Tag-full-names*.

Novel title	Precision	Recall	F-measure
The Catcher in the Rye	0.8	0.77	0.78
The Great Gatsby	0.88	0.9	0.89
The Secret Garden	0.8	0.79	0.79
***** Overall results *****	0.83	0.83	0.83

Table 5.2: Performance of the *protagonistTagger* on the second testing set *Testing_set_small_Tag-full-names*.

5.4. *ProtagonistTagger*'s Results

This section presents the performance of the prepared method in numbers. Statistics are presented for each tested novel separately and each testing set as a whole. Statistics are based on the metrics defined as in Section 2.1. Tables 5.1 and 5.2 present standard NLP metrics computed for each part of the testing set devoted to a different novel. In the last row of each table, an overall result for the whole dataset is given.

These statistics are specifically helpful in analyzing the performance of the method depending on the genre and style of the novel, and the type of difficulties appearing in it. The analysis of these dependencies was performed on the available testing sets and it pertains only to the sentences extracted from novels included in these testing sets. Unfortunately, the set of considered novels is not big enough to draw general conclusions. The performed analysis confirms the efficiency of the tool on novels. However, it can not answer the question about the general dependency of factors, such as the literary genre of novel or amount of literary characters, on the performance of the tool. This analysis is only considering the novels included in the testing

set and is used for drawing conclusions about them, not about novels in general.

5.4.1. Discussion

The general performance of the *protagonistTagger* on both testing sets is quite good (results are presented in Tables 5.1 and 5.2). In the performance, all the named entities of category *person* are included. Therefore, the performance addresses recognizing and annotating not only most important protagonists in the novel appearing in the predefined tags, but also the minor, tangential ones.

In the case of most novels, the precision oscillates between 79% and 96%. Relevant results are in our case all correctly recognized named entities of category *person* that are annotated with their full names out of all named entities of category *person*. The tool must not annotate with some protagonist's name a named entity of different category, for example, *org* associated with organizations or companies. It is also essential that named entities of category *person* are annotated with a proper tag, being indeed the full name of the character associated with it (or a general tag *person* in the case when the name of the literary character is not included in a list of predefined tags).

On the other hand, it is equally vital for the *protagonistTagger* to find all the named entities assigned to each tag. According to the statistics computed for the method, the recall fluctuates between 75% and 97%. It means that the *protagonistTagger* can accurately identify the relevant data and annotate them properly.

Considering the main goals of the *protagonistTagger*, it is difficult to say whether precision is more crucial than recall, or the other way round. On the one hand, it is desired to find all the named entities associated with the novel's literary characters. On the other hand, it is necessary to annotate them accordingly in the next step, as the named entities with incorrect tags are not very useful. The created tool can be, of course, better, being able to recognize more named entities. However, the ability of the tool to assign correct tags seems to be more critical. While using *protagonistTagger* for any of the tasks described in Section 1.2 it is crucial to trust the tags we have in our data. Otherwise, the results of any further analysis of the prepared annotated data, for example, relationship detection or character analysis, can be erroneous.

From what can be observed from the *protagonistTagger*'s performance, precision and recall for the tool are comparable. It means that it can quite accurately identify the named entities corresponding to each tag and provide them with correct tags. Even though the results are relatively high, it is worth to work on the precision of the tool in the future. Further analysis of the incorrect annotations may result in some new problems and ambiguities not discovered so

far. Testing the tool on new novels may provide additional cases to be handled as well.

5.4.2. Performance Dependency on the Testing Set Used

While comparing the tool's performance on the testing sets, it is visible that better results are achieved on the larger one.

It may be because some names of literary characters appearing in the larger set were explicitly used for recognizing cases that need to be handled by the *matching algorithm*. Even though the rules on which the *matching algorithm* is based are novel-independent, they still were inspired by analyzing novels from the testing set *Testing_set_large_Tag-full-names*.

Maybe some special cases and dependencies which should have been handled are not present in the novels from the other set – *Testing_set_large_Tag-full-names*. However, they appear in the novels from *Testing_set_small_Tag-full-names*. Of course, it is impossible to analyze all novels in search of exceptional cases and rules that need to be implemented to match perfectly each recognized named entity of category *person* with a full name of a proper literary character. The novels for the *Testing_set_large_Tag-full-names* were chosen to be as much diversified as possible.

Taking it all into consideration, the fact that the performance on *Testing_set_small_Tag-full-names* is lower is foreseeable and can be easily explained. The relatively small discrepancy between the two testing sets results proves that the set of novels chosen for the research process is representative enough for the analyzed task.

5.4.3. *ProtagonistTagger* Performace vs. NER Performance

There is an obvious dependency between the *protagonistTager* and the underlying NER model. A drop in NER model performance on some novel causes a drop in the whole tool performance. It is presented in Figure 5.1. In this case, again, the recall of the NER model is crucial. The named entities of category *person* not identified by NER model are not considered by the *matching algorithm* in the annotation phase. Rejecting named entities of category different than *person* or words that are not named entities is a task of a *matching algorithm*. The dependency between *protagonistTagger*'s performance and the NER model recall is visible, especially in novels from the *Testing_set_small_Tag-full-names*. The lowest recall for both: NER model and *protagonistTagger* is reported for *The Catcher in the Rye*. In this specific example, this dependency is evident.

What is also worth mentioning, is the lower NER model performance on novels from the *Testing_set_small_Tag-full-names*, what is visible in the Figure 5.1. Especially in case of *The*

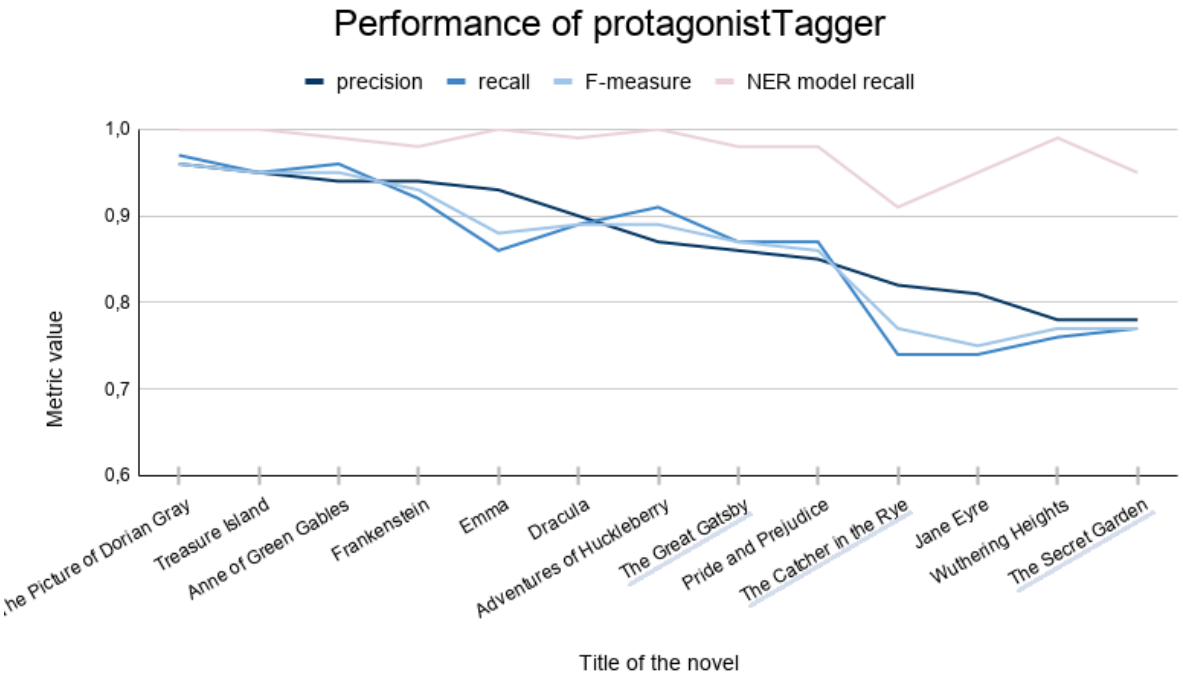


Figure 5.1: Performance of the *protagonistTagger* on both testing sets presented for each novel separately. Additionally, the recall of the used NER model is given on the graph (the pink line), in order to verify the dependency between NER and the performance of the *protagonistTagger*. Novels used in *Testing_set_small_Tag-full-names* are marked with gray underlining.

Catcher in the Rye and *The Secret Garden* from the *Testing_set_small_Tag-full-names* NER model may be the main reason of the drop in the performance of the *protagonistTagger*. Additionally, as mentioned before, it is more probable for the tool to have some problems with matching named entities with predefined tags on brand new novels. It is caused by the fact, that the rules included in the *matching algorithm* may not cover all possible special cases and ambiguities appearing in novels not included in the analysis.

5.4.4. *ProtagonistTagger* Performace vs. Number of Literary Characters in a Novel

One of the factors that intuitively should influence the performance of the *protagonistTagger* is the number of literary characters in a novel that is analysed. The number of protagonists in a novel determines the number of tags that are used by the *protagonistTagger*. The more tags the tool has to choose from, the more difficult is the task of matching them correctly to each recognized named entity. The relation between the precision of the *protagonistTagger* and the number of tags per each novel is presented in Figure 5.2. It can not be said unambiguously that these two values are in inverse proportion in the testing sets. The novels on which the

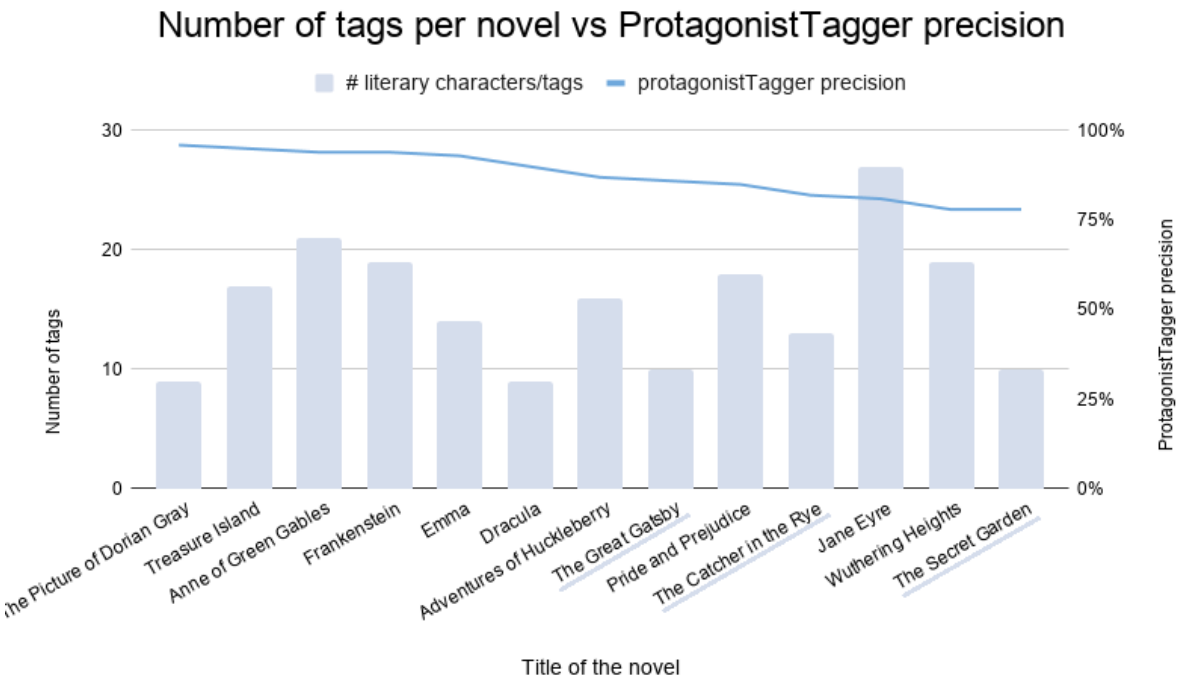


Figure 5.2: The left vertical axis describes the number of literary characters (tags used by the *protagonistTagger*) in each novel, whereas the right vertical axis describes the precision of the *protagonistTagger* (given in percents) for each novel. Novels used in *Testing_set_small_Tag-full-names* are marked with gray underlining.

tool achieved both the lowest and the highest precision – (*The Picture of Dorian Gray* and *The Secret Garden*) – have a relatively small number of literary characters.

Another factor that was suspected to negatively influence the performance of the *protagonist-Tagger* is the number of tags sharing some common part. The case of *Bennet* family in *Pride and Prejudice* by Jane Austen is discussed in details in Section 3.2.3. It is stated there that protagonists with the same surname are problematic even for human annotators. Sometimes a personal title preceding the named entity can be helpful. However, matching correctly tags that share the same surname or even name may be nontrivial. For that reason, I created statistics of tags that share some common part. These statistics are given for each novel included in both testing sets in Table 5.3. Additionally, they are presented on graph in Figure 5.3 along with the performance of the *protagonistTagger*.

Unfortunately, again no obvious relation between these two vales is visible. It may be caused by the fact that common parts in tags may not be related to the main protagonists (the ones that appear most often in the novel and the testing sets). Therefore, the testing sets are not representative enough in this case. For example, in the case of *Anne of Green Gables* that has

relatively many literary characters, half of which share the same name or surname, the tool's performance is very high. Nonetheless, in the case of *Wuthering Heights*, with a similar number of protagonists, half of which again share a common name or surname, performance is much lower. It is caused by the fact that in *Wuthering Heights* the tags that share common parts correspond to the main protagonists. Whereas, in case of *Anne of Green Gables* such common elements appear rather in tags corresponding to tangential characters.

In general, it can be concluded that the performance of the *protagonistTagger* depends on many factors, not only the number of tags and the percentage of tags with the common part in a novel. These two factors, in some cases, can negatively influence the performance of the tool. However, this impact is not certain in the case of all novels.

Title of the novel	# literary characters/tags	# tags that share a common part	% tags that share a common part
Pride and Prejudice	18	13	72%
The Picture of Dorian Gray	9	4	44%
Anne of Green Gables	21	11	52%
Wuthering Heights	19	12	63%
Jane Eyre	27	13	48%
Frankenstein	19	4	21%
Treasure Island	17	4	23%
Adventures of Huckleberry Finn	16	7	43%
Emma	14	9	64%
Dracula	9	2	22%
The Catcher in the Rye	13	4	31%
The Great Gatsby	10	4	40%
The Secret Garden	10	7	70%

Table 5.3: The number of literary characters (tags used by *protagonistTagger*) appearing in each novel and the number of tags that share a common part. A common part can be the same name or surname. The same personal title in two tags is not considered as a common part.

5.5. Linguistic Analysis of Tested Novels

The analysis done in the previous sections of this chapter is not providing a clear answer to the question about the characteristics of the novel that have the most substantial impact on the tool's performance. Therefore, in this section, I attempt to answer this question by analyzing the performance of the *protagonistTagger* on novels of different literary genre. Intuitively the type of the analyzed novel, the style in which it is written should influence the tool's performance. To

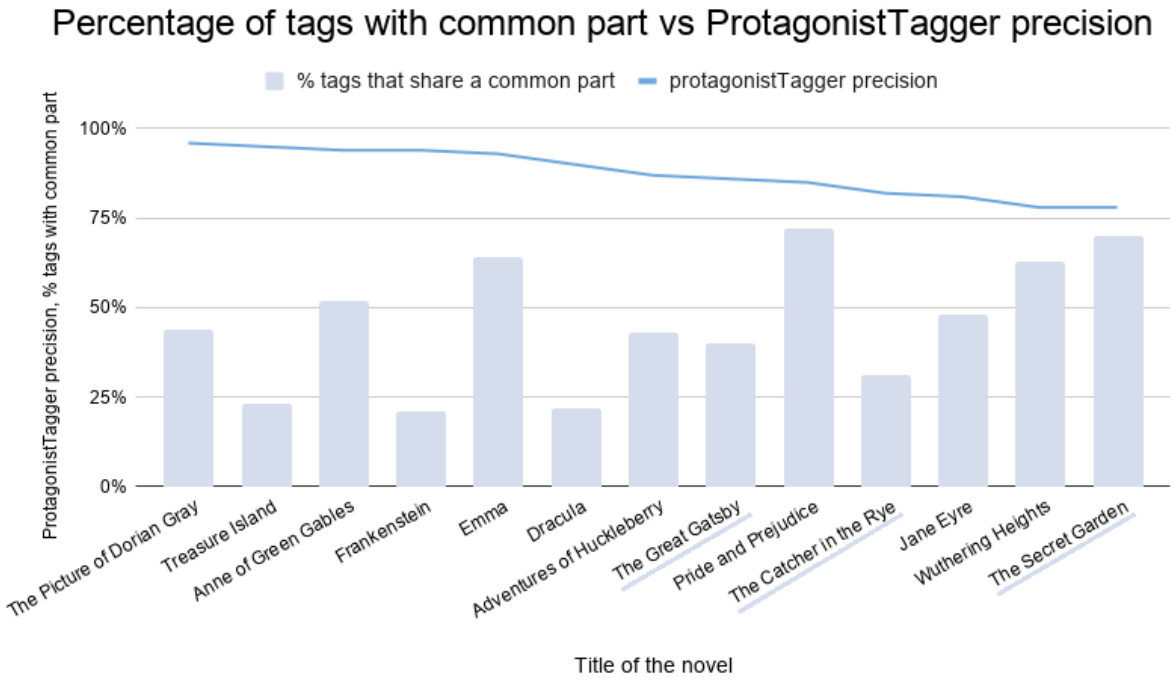


Figure 5.3: The right vertical axis describes the percentage of tags sharing a common part (grey bars), as well as the precision of the *protagonistTagger* (given in percents) for each novel in the testing sets. Novels used in *Testing_set_small_Tag-full-names* are marked with gray underlining.

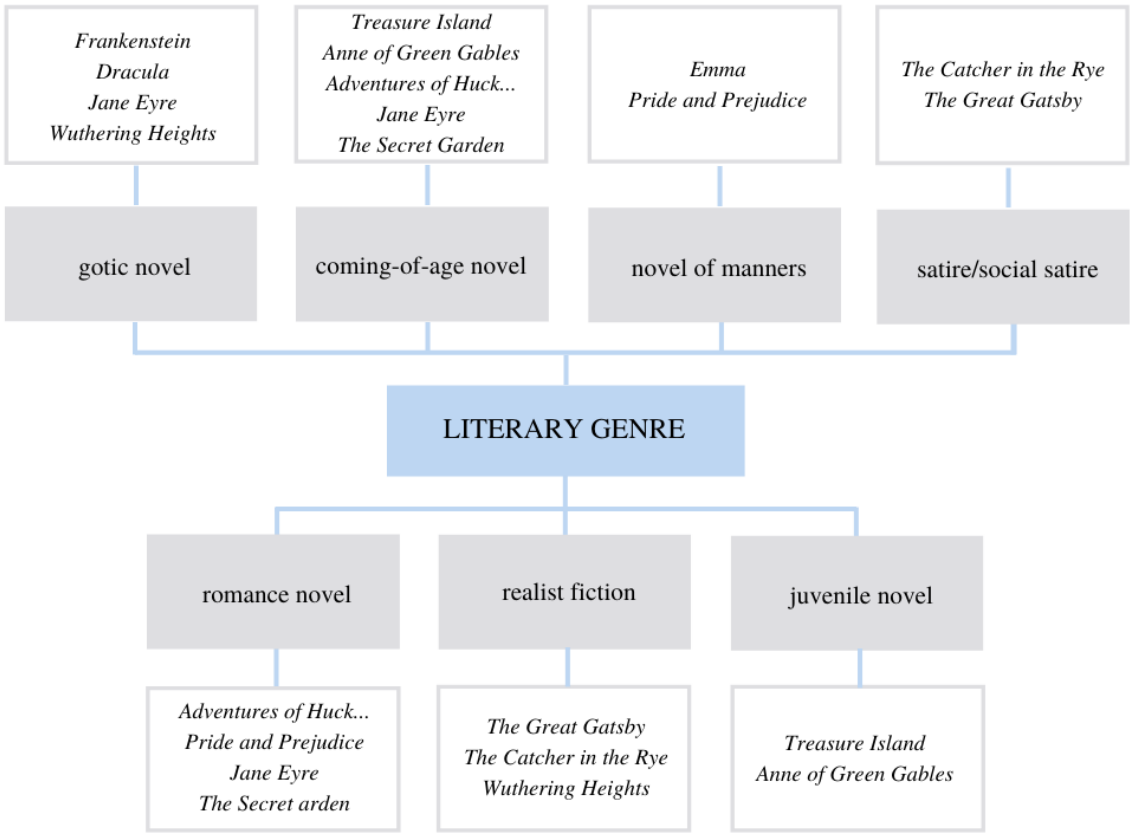


Figure 5.4: Literary genres of novels used in both testing sets.

verify this hypothesis, at least on the available testing sets, it is necessary to divide the novels used in testing sets by their literary genre.

In most cases, it is not easy to choose only one literary genre associated with a novel. Usually, novels combine the characteristics of many different literary genres. Novels appearing in both testing sets are categorized into seven groups presented in Figure 5.4. This division is a simplification, as there are many more literary genres which characteristics can be found in the novels. Nonetheless, the seven chosen genres are a good starting point for the analysis of the *protagonistTagger*'s performance. A more detailed analysis of each novel in the testing sets is given in Table 5.4.

5.5.1. Performance Dependency on the Genre and Type of Text

The performance of the *protagonistTagger* is likely dependent on the style and the genre of the novel. Both the precision and the recall for short and simple texts are relatively higher. The best performance is achieved for novels such as *Treasure Island* or *Anne of Green Gables* which are juvenile novels dedicated to children. The tool copes well with rather short novels with

Title of the novel	Literary genre	Style of the novel
The Picture of Dorian Gray	gothic novel, novel of manners, comedy	kept in dark mood; supernatural motifs; sardonic but comedic
Treasure Island	adventure story, coming-of-age novel, juvenile novel	revealing little emotions; pirate style of speech; focused on reporting events
Anne of Green Gables	coming-of-age novel, juvenile novel	poetic, descriptive, focused on emotions and inner life of the protagonists, as well as on the world around them
Frankenstein	gothic novel, science fiction	formal, elevated; using complex vocabulary
Emma	novel of manners, comedy	subtle; simple but direct; witty, sharp, epigrammatic, abstract; focused on dialogues
Dracula	gothic fiction, horror	epistolary (novel is composed of diary entries, letters, etc.); straightforward
Adventures of Huckleberry Finn	picaresque novel, coming-of-age novel, romance novel	written in the vernacular of the characters; casual, sometimes even incorrect way of speaking in dialogues
Pride and Prejudice	romance novel, novel of manners, comedy	exaggerated, ironic and witty; focused on dialogues
Jane Eyre	romance novel, gothic novel, coming-of-age novel	descriptive and formal; long sentences; verbiage and lengthy syntax
Wuthering Heights	tragedy, gothic novel, realist fiction	designed to horrify and fascinate; incorporating supernatural elements; novel kept in dark, foreboding atmosphere
The Secret Garden	coming-of-age novel, romance, novel of ideas	flowery and rich; descriptive; using multiple adjectives
The Catcher in the Rye	coming-of-age novel, realist fiction, satire	vernacular style with slang and curse words; hyperbolic; using generalizations
The Great Gatsby	tragedy, realism, modernism, social satire	sophisticated, elegiac, wry; including extended metaphors and poetic language; incorporating sharp and sardonic humor

Table 5.4: Literary genre and a short description of a writing style of each novel from testing sets.

relatively few literary characters, such as *The Picture of Dorian Gray* or *Frankenstein*. Similarly, *The Great Gatsby*, being relatively short text with only a few protagonists, is relatively high in the tool performance ranking.

On the other hand, there are a few novels for which the performance of *protagonistTagger* is relatively low. They are the two gothic novels: *Jane Eyre* and *Wuthering Heights* characterized with a wide range of protagonists appearing in them. It may be the reason for a drop in performance.

Another novel that seems problematic for a tool is a coming-of-age novel (describing the

protagonist's growth from youth to adulthood), *The Secret Garden*. Even though the number of literary characters appearing in it is not high, the tool had some problems with two protagonists with the same surname. It caused a significant decline in the correctness of the annotations.

Interestingly, novels of manners featuring many characters (sometimes whole families with the same surname) did not cause many problems for the tool. For example, the annotations in *Pride and Prejudice* or *Emma* are relatively accurate (both recall and precision above 85%). Therefore, it is difficult to draw one precise conclusion about the dependency of the tool performance on the novel's genre.

5.6. Summary of the Performance Analysis

Generally, the tool achieves really good results on all tested novels (precision and recall above 80%). The performance of the tool on brand new novels from *Testing_set_small_Tag-full-names* shows that it can be successfully used for creating a larger corpus of annotated novels.

The analysis done in order to find a factor that has the strongest influence on the performance of the tool provides some clues about the possible direction of improvements. However, it also shows that the analysed testing sets are not big and representative enough to draw general conclusions. In order to be able to draw such conclusions, it is necessary to gather a fully representative testing set. For example, in case of investigating the impact of literary genre on tool's performance, we would need relatively big subsets of novels of each considered literary genre included in this set.

Nevertheless, taking into consideration only the novels included in the testing sets, it can be concluded that there are numerous factors negatively influencing the performance of the *protagonistTagger*. It depends preferably on the number of literary characters appearing in the novel, its complexity, literary genre, uniqueness of predefined tags and probably many more. The main challenge is to assign tags to named entities in a form common for the novel (for example, only first names used or only surnames preceded with the personal title or many diminutives). What is crucial, this last dependency is novel-specific and author-specific.

6. Conclusions

In this thesis, I have performed an in-depth analysis of recognizing named entities of category *person* and identifying them as corresponding literary characters of the novel.

The research process resulted in the tool *protagonistTagger* and the corpus of novels annotated with literary characters' full names (both can be found on GitHub *https://github.com/WerLaj/protagonist_tagger*). The main focus of the performed research process evolved, along with new problems and ambiguities encountered on the way. Two main areas of research include:

- NER applied to novels - analysis focused mainly on literary characters whose names were not recognized or were recognized incorrectly by the NER model, as well as methods of creating training sets for fine-tuning NER model;
- Matching recognized named entities with proper literary characters - analysis focused mainly on variations in naming conventions in novels, as well as on methods for computing similarity between two strings.

Named Entity Recognition in Novels

The most important conclusion from the performed research process refers to the relatively low performance of the standard NER models on novels. It is caused by the fact that novels differ significantly from texts on which standard NER models are trained (such as web articles, blogs, tweets). It turned out that NER models provided by NLP libraries do not recognize many common English names, or they are classified as a category different than *a person* (see Section 3.1.2).

These imperfections of the standard NER model were the main obstacle in the project. Low performance of NER implies the low performance of the entire tool *protagonistTagger*. Significant attention on creating a training set for fine-tuning the standard NER model resulted in an almost perfect performance in recognizing named entities of category *person* in novels (see Section 4.3).

The training set includes two types of data:

- sentences extracted from novels with semi-automatically injected common English names,

- sentences from novels with names of the protagonist that standard NER model failed to recognize and classify correctly.

Combining these two types of training data resulted in the NER model that achieves satisfying performance on the testing sets. The recall of the fine-tuned NER model for almost every tested novel is above 95%.

Matching Named Entities with Literary Characters

Another research problem presented in the thesis concerns matching recognized named entity of category *person* with one of the literary characters of a novel. The analysis of the problem unveiled the complexity of the names appearing in the novels. It is reflected, among others, in commonly using only part of the full name (first name or surname), diminutives or nicknames, and personal titles followed only by the surname. Additionally, many novels feature literary characters with the same name or surname.

These characteristics required creating a set of rules and using external resources (such as the dictionary of diminutives) to create a mechanism able to solve the problem. The *matching algorithm* presented in the thesis (see Section 4.4) is based mostly on the technique called *approximate string matching* that computes the similarity between two strings.

ProtagonistTagger's Performance

The *protagonistTagger* achieved both the precision and the recall above 80% on the testing set containing thirteen novels, in the case of almost all analyzed novels. Considering that *novel* is a term defining a wide range of various texts, these statistics are more than satisfying. The best proof of novels' diversity is the tool's performance, whose precision varies from 79% to even 96%.

In analyzing the created tool's performance, I attempted to answer the question about the characteristics of the novels that have the most substantial impact on the correct annotations. The tool's performance depends on the type of the novel, which is visible in specific novels' results. Furthermore, it depends on the NER model's performance because identifying named entities of category *person* is the initial step in the process of annotation.

However, it is not clear which characteristics of the novels cause most problems for the tool. The analysis showed that the number of literary characters, the percentage of the literary characters with the same name or surname, and the novels' literary genre influence the tool's performance. However, there is no apparent dependency of these characteristics on the performance of the *protagonistTagger*. The testing sets on which the analysis was performed is not

big and representative enough to draw such general conclusions. Instead, it should be said that novels are such a complicated type of text that many factors impact the correctness of the annotations. Indeed, the performance depends mainly on the ability of the *matching algorithm* to differentiate the extracted named entities of category *person*, for example, characters with the same name or surname.

7. Future Work

Improvements of the *ProtagonistTagger*

According to the fine-tuned NER model's performance, there is still room for improvement in the case of some novels. The apparent dependency of the performance of *protagonistTagger* on the recall of the NER model means that it is worth working further on fine-tuning the NER model. Increasing the training set and including more names common in English may improve the performance. Nevertheless, it needs to be borne in mind that in case of some novels the NER model performance is already perfect (recall of 100%). Manipulating the model with new examples may harm such novels.

Another component of the tool that can be improved is the *matching algorithm*. Further analysis of the names appearing in novels and characteristics of the novels that influence the performance of the tool may result in new rules that should be handled. It is impossible to define all ambiguities and exceptional cases that may appear in novels. However, more detailed analysis can only improve the rules implemented in the *matching algorithm* and allow to handle different cases more precisely.

Applying the Created Corpus for More Detailed Analysis of the Novels

The variety of possibilities presented in Section 1.2 makes further analysis of novels based on the created corpus very tempting. One of the most exciting applications is the detection of the relationships between literary characters. Such analysis can be based on dialogues interactions and creating social networks of protagonists. Our annotated corpus can be extremely useful in this task.

Another attractive application of the created corpus is a sentiment-based analysis of literary characters. Extracting parts of the novels associated with specific characters make such analysis of individual protagonists much easier. All these applications are still to be verified. However, the created corpus and the tool for annotating new novels seems to be a good starting point in many NLP tasks performed on novels. The analysis of the state-of-the-art solutions to these tasks with the annotated novels can be fascinating.

Tool *use case* in non-literary domain

The *protagonistTagger* was created having in mind mainly literary domain, especially novels. Nevertheless, it may be applied to all kinds of texts in which we want to annotate named entities of category *person*. The only condition is the access to the predefined tags defining the full names to be matched to named entities identified in a text. A very interesting field of application are social media. Texts extracted from social media very often feature many names in different forms. Being able to recognize and annotate them properly can be very beneficial from the point of view of investigating human opinions and analysing the sentiments.

Bibliography

[1] Apoorv Agarwal et al. “Social network analysis of Alice in Wonderland”. In: *Proceedings of the NAACL-HLT 2012 Workshop on computational linguistics for literature*. 2012, pp. 88–96.

[2] Alan Akbik, Tanja Bergmann, and Roland Vollgraf. “Pooled contextualized embeddings for named entity recognition”. 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)*. 2019, pp. 724–728.

[3] David Bamman, Ted Underwood, and Noah A Smith. “A bayesian mixed effects model of literary character”. In: *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*. 2014, pp. 370–379.

[4] Daniel M Bikel, Richard Schwartz, and Ralph M Weischedel. “An algorithm that learns what’s in a name”. In: *Machine learning*. 1999, pp. 211–231.

[5] Julian Brooke, Adam Hammond, and Graeme Hirst. “GutenTag: an NLP-driven tool for digital humanities research in the Project Gutenberg corpus”. In: *Proceedings of the Fourth Workshop on Computational Linguistics for Literature*. 2015, pp. 42–47.

[6] Snigdha Chaturvedi, Mohit Iyyer, and Hal Daumé III. “Unsupervised Learning of Evolving Relationships Between Literary Characters.” In: *AAAI*. 2017, pp. 3159–3165.

[7] Snigdha Chaturvedi et al. “Modeling evolving relationships between characters in literary novels”. In: *Thirtieth AAAI Conference on Artificial Intelligence*. 2016.

[8] Davide Chicco and Giuseppe Jurman. “The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation”. In: *BMC genomics*. Vol. 21. 1. Springer, 2020, p. 6.

[9] Jason PC Chiu and Eric Nichols. “Named entity recognition with bidirectional LSTM-CNNs”. In: *Transactions of the Association for Computational Linguistics*. 2016, pp. 357–370.

[10] Prabhakar Raghavan Christopher D. Manning and Hinrich Schütze. *Introduction to information retrieval*. 2010.

[11] Adam Cohen. "FuzzyWuzzy: Fuzzy string matching in python". In: *ChairNerd Blog*. Vol. 22. 2011.

[12] Michael Collins and Yoram Singer. "Unsupervised models for named entity classification". In: *Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora*. 1999.

[13] Ronan Collobert et al. "Natural language processing (almost) from scratch". In: *Journal of machine learning research*. 2011, pp. 2493–2537.

[14] Paul T Costa Jr and Robert R McCrae. *The Revised NEO Personality Inventory (NEO-PI-R)*. Sage Publications, Inc, 2008.

[15] David Elson, Nicholas Dames, and Kathleen McKeown. "Extracting social networks from literary fiction". In: *Proceedings of the 48th annual meeting of the association for computational linguistics*. 2010, pp. 138–147.

[16] Lucie Flekova and Iryna Gurevych. "Personality profiling of fictional characters using sense-level links between lexical resources". In: *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. 2015, pp. 1805–1816.

[17] Filip Graliski et al. "GEval: Tool for Debugging NLP Datasets and Models". In: *Proceedings of the 2019 ACL Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*. Florence, Italy: Association for Computational Linguistics, 2019, pp. 254–262.

[18] Adrian Groza and Lidia Corde. "Information retrieval in falktales using natural language processing". In: *2015 IEEE International Conference on Intelligent Computer Communication and Processing (ICCP)*. IEEE. 2015, pp. 59–66.

[19] *GutenTag - web application*. `https://gutentag.sdsu.edu/`.

[20] Adam Hammond and Julian Brooke. *GutenTag: A User-Friendly, Open-Access, Open-Source System for Reproducible Large-Scale Computational Literary*. 2017.

[21] Michael Hart. "The history and philosophy of Project Gutenberg". In: *Project Gutenberg*. 1992.

[22] Nancy Ide and James Pustejovsky. *Handbook of linguistic annotation*. Springer, 2017.

[23] Ridong Jiang, Rafael E Banchs, and Haizhou Li. "Evaluating and combining name entity recognition systems". In: *Proceedings of the Sixth Named Entity Workshop*. 2016, pp. 21–27.

[24] Evgeny Kim and Roman Klinger. "A survey on sentiment and emotion analysis for computational literary studies". In: *arXiv preprint arXiv:1808.03137*. 2018.

[25] Guillaume Lample et al. "Neural Architectures for Named Entity Recognition". In: *Proceedings of the the 15th Annual Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. 2016, pp. 260–270.

[26] Gonzalo Navarro. "A guided tour to approximate string matching". In: *ACM computing surveys (CSUR)*. Vol. 33. 1. ACM New York, NY, USA, 2001, pp. 31–88.

[27] Gonzalo Navarro et al. "Indexing methods for approximate string matching". In: *IEEE Data Eng. Bull.* Vol. 24. 4. Citeseer, 2001, pp. 19–27.

[28] Andrew J Reagan et al. "The emotional arcs of stories are dominated by six basic shapes". In: *EPJ Data Science*. Vol. 5. 1. SpringerOpen, 2016, pp. 1–12.

[29] Tim Rocktäschel, Michael Weidlich, and Ulf Leser. "ChemSpot: a hybrid system for chemical named entity recognition". In: *Bioinformatics*. Oxford University Press, 2012, pp. 1633–1640.

[30] Xavier Schmitt et al. "A Replicable Comparison Study of NER Software: StanfordNLP, NLTK, OpenNLP, SpaCy, Gate". In: *2019 Sixth International Conference on Social Networks Analysis, Management and Security (SNAMS)*. IEEE. 2019, pp. 338–343.

[31] Isabel Segura-Bedmar, Paloma Martnez, and Mara Herrero-Zazo. "SemEval-2013 Task 9: Extraction of Drug-Drug Interactions from Biomedical Texts (DDIExtraction 2013)". In: *2nd Joint Conference on Lexical and Computational Semantics, Volume 2: Proceedings of the 7th International Workshop on Semantic Evaluation (SemEval 2013)*. 2013, pp. 341–350.

[32] Mohammad Golam Sohrab and Makoto Miwa. "Deep Exhaustive Model for Nested Named Entity Recognition". In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. Brussels, Belgium: Association for Computational Linguistics, 2018.

[33] *SpaCy NER Annotator*. `https://manivannanmurugavel.github.io/annotating-tool/spacy-ner-annotator/`.

[34] Bhargav Srinivasa-Desikan. *Natural Language Processing and Computational Linguistics: A practical guide to text analysis with Python, Gensim, spaCy, and Keras*. Packt Publishing Ltd, 2018.

[35] Tomasz Stanislawek et al. “Named Entity Recognition - Is There a Glass Ceiling?” In: *Proceedings of the 23rd Conference on Computational Natural Language Learning (CoNLL)*. Association for Computational Linguistics, 2019, pp. 624–633.

[36] Hardik Vala et al. “Mr. bennet, his coachman, and the archbishop walk into a bar but only one of them gets recognized: On the difficulty of detecting characters in literary texts”. In: *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. 2015, pp. 769–774.

[37] Ralph Weischedel, Martha Palmer, and Mitchell Marcus. *OntoNotes Release 5.0 - Linguistic Data Consortium*. 2013.

[38] Vikas Yadav and Steven Bethard. “A Survey on Recent Advances in Named Entity Recognition from Deep Learning models”. In: *Proceedings of the 27th International Conference on Computational Linguistics*. 2018, pp. 2145–2158.

[39] Vikas Yadav, Rebecca Sharp, and Steven Bethard. “Deep affix features improve neural named entity recognizers”. In: *Proceedings of the 7th Joint Conference on Lexical and Computational Semantics*. 2018, pp. 167–172.

[40] Xiaodong Yu et al. “On the Strength of Character Language Models for Multilingual Named Entity Recognition”. In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. Brussels, Belgium: Association for Computational Linguistics, 2018.

[41] Albin Zehe et al. “Prediction of happy endings in German novels based on sentiment information”. In: *3rd Workshop on Interactions between Data Mining and Natural Language Processing, Riva del Garda, Italy*. 2016.

List of symbols and abbreviations

NLP	Natural Language Processing
NE	Named Entity
NER	Named Entity Recognition
PG	Project Gutenberg
POS	part-of-speech
TP	True Positive (outcome of classification)
TN	True Negative (outcome of classification)
FP	False Positive (outcome of classification)
FN	False Negative (outcome of classification)

List of Figures

List of tables