Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Masterarbeit

Characterizing Literature Using Machine Learning Methods

vorgelegt von

Jan Bílek

Fakultät für Mathematik, Informatik und Naturwissenschaften
Fachbereich Informatik
Arbeitsbereich Wissenschaftliches Rechnen

Studiengang:	Informatik
Matrikelnummer:	6644825
Erstgutachter:	Dr. Julian Kunkel
Zweitgutachter:	Prof. Thomas Ludwig
Betreuer:	Dr. Julian Kunkel

Hamburg, 2016-10-14

Abstract

In this thesis, we explore the classical works by famous authors available in Project Gutenberg – a free online ebook library. The contemporary computational power enables us to analyze thousands of books and find similarities between them. We explore the differences between books and genres with respect to features such as proportion of stop words, the distribution of part of speech classes or frequencies of individual words. Using this knowledge, we create a model which predicts book metadata, including author or genre, and compare the performance of different approaches. With multinomial naive Bayes model, we reached 74.1 % accuracy on the author prediction task out of more than 1 400 authors. For other metadata, the random forest classifier achieved the best results. Through most predictive features, we try to capture what is typical for individual genres or epochs. As a part of the analysis, we create Character Interactions model that enables us to visualize the interactions between characters in the book and define the main or central character of the book.

Contents

1 Introduction

This chapter explains the motivation for this work, introduces its goals and presents an outline of the thesis.

1.1 Motivation

There are multiple applications of text analysis in the real world. It enable us to use search engines or get information interesting to us using recommendation systems. There is also high demand for automatic summarization and adding tags and keywords, as it makes the orientation easier for a human.

In this thesis, we explore the books using machine learning. We want to know which books are similar, and when they are similar, what is the reasoning behind that. Can we create a mathematical model which describes a given genre like a human would?

People are quite good at some tasks using their intuition. They can easily describe the genre of the book after they read it. The interesting question is how did they know. Is it the specific word choice in the text or more the style of writing? Such information would be valuable to the automatic processes, as they not only don't have any intuition, but also don't understand the underlying data.[1] The classifications are made using statistical methods. Neverthless, there are tasks where algorithms outperform humans, as there already exist algorithms on capturing pseudonymes or plagiarism. To do it manually, lot of expert knowledge is usually required.

1.2 Goals

The main goal of this thesis is to explore the English books from the large corpus available in Project Gutenberg and gain some insight on their similarities and differences. We extract multiple types of features from the texts and evaluate their contribution to explaining the variance in the data. The further goals include the following ones:

1. Inspect what kind of books are available in Project Gutenberg and what metadata are provided. Explore how do some interesting statistics, such as propotion of stopwords or lengths of words and sentences, vary among different books, genres and authors.

[1] The algorithms don't understand individual words in general, but some can already capture relationships between them as in case of word2vec.

2. Create a model which automatically classifies unknown books, identifies its characters and displays interactions between them.

1.3 Outline

In the next chapter, we introduce Project Gutenberg as well as some key principles of Natural Language Processing. In Chapter 3, we describe the process and design of the analysis. Chapter 4 shows some details and examples of the implementation. The core of this thesis is Chapter 5, where we explore the books, discuss obtained results and create a prediction model for the book metadata. Finally, Chapter 6 summarizes everything up and discuss the overall approach and results as well as possible future work and improvement.

2 Background and Related Work

In this chapter, we first introduce Project Gutenberg and its history along with classification systems they use, introduced in Section 2.2. Next, Section 2.3 gives an overview of the methods and concepts in the Text Analysis. In Section 2.4, we mention the models we use in the analysis. Finally, Section 2.5 refers on some related work in this field.

2.1 Project Gutenberg (PG)

Project Gutenberg [8] is an online library of free books. It was founded in 1971 by Michael S. Hart and the main idea behind the project was to collect works with expired or free copyright[1]. For the first 20 years, Michael S. Hart was the main contributor when he got access to a computer at the University of Illinois being one of the first people creating an ebook in the world. The amount of the books in Project Gutenberg started to grow rapidly in the 90's[2] with the spread of the Internet. The project rose rapidly thanks to many volunteers who transcribed, digitized, formatted and proofread the texts as well as looked for potential books with confirmed copyright clearance.[12]

Project Gutenberg's philosophy is to make its books available to the wide audience. Therefore it focuses on two objectives,[9]:

- Provide the books for as little costs as possible that everyone can afford them.
- Keep the format of the etexts as simple as possible that they can be read and searched in on all platforms.

The basic format of the etexts is a 7-bit ASCII text which is still kept although there are more modern formats, such as UTF-8 today. For languages with accents and special characters, such as German or French, the 8-bit extended ASCII text is used, however the etext version in ASCII with accents stripped is also added to the book repository. Apart from that, the community is encouraged to add other formats, such as UTF-8, AsciiDoc, EPUB or HTML to improve the user experience of the readers by adding, e.g., illustrations.[9]

Project Gutenberg collects, apart from books, also other materials which might be interesting to the general public or future generations – there are over 1000 audiobooks.

[1] According to the U.S. copyright law, all books created before 1923 are in the public domain, books written between 1923 and 1977 are protected by copyright for 95 years after the author's death (if the copyright was prolonged during the 28th year) and for 70 years for books written after 1978.

[2] The 10th book was completed in August 1989. By the end of the year 2000, they were already 3000 books.

Project Gutenberg contains also files totally unrelated to books, such as datasets (e.g., first 1 000 000 decimals of π[10]) or several videos capturing, e.g., Apollo 11 landing on the Moon[6].

At the moment (January 2017), Project Gutenberg has over 54 000 repositories, almost 53 000 of those being etexts of books. Project Gutenberg contains books in 54 languages. The most common language is English (82% of all books), followed by French (5%), Finnish and German (both 3%).

Project Gutenberg provides a Complete Project Gutenberg Catalog, which can be downloaded from their website[7].[3]

The metadata in the catalog provide lots of administrative information on the repository itself, e.g., format description, created and modified timestamps, number of downloads or copyright licenses. Apart from that, it also contains some descriptive metadata of the book. These contain title, author information, as well as some book classification tags.

2.2 Literature Classification and Tags

Project Gutenberg uses two systems of book classifications. Both are maintained by the Library of Congress (LOC), the research library of the United States Congress and the largest library in the world[16]. The first system, *Library of Congress Classification*[17], describes the category of the book, such as History, Science or Law. The second system, the *Library of Congress Subject Headings*[18], provides more information on genres and topics of the book.

2.2.1 Library of Congress Classification (LCC)

The Library of Congress Classification system was introduced by the American librarian Herbert Puntnam in 1897. As the classification system comes from the United States Congress' library, it focuses a bit more on the American literature, which is finer granulated than other literature[4]. The LCC category usually consists of 2 letters. The first letter states the parent class, the second letter defines its subclass. The LCC system has 21 parent classes, which are shown in Listing 2.1[5]. Several selected subclasses of LCC parent class P are shown in Listing 2.2. Each book can have multiple LCC categories, although most of them have just one.

2.2.2 Library of Congress Subject Headings (LCSH)

The LCSH tags are more specific than the LCC categories and provide more information on subjects, topics and genres of the book. Every tag can be hierarchically organized

[3]There is a rdf file for each document containing the metadata information in a structured form. Every night, all the rdf files are compressed and added to a single archive in ZIP and BZIP2 format.

[4]An example might be the History category – classes E and F both cover the history of America, whereas history of the rest of the World is all in one category – D.

[5]We adopt the LOC's notation, which writes the parent classes names in capitals. It is then easier to differentiate the parent classes from subclasses, which are written in first letter capitals only.

```
A -- GENERAL WORKS
B -- PHILOSOPHY, PSYCHOLOGY, RELIGION
C -- AUXILIARY SCIENCES OF HISTORY
D -- WORLD HISTORY
E -- HISTORY OF THE AMERICAS
F -- HISTORY OF THE AMERICAS
G -- GEOGRAPHY, ANTHROPOLOGY, RECREATION
H -- SOCIAL SCIENCES
J -- POLITICAL SCIENCE
K -- LAW
L -- EDUCATION
M -- MUSIC AND BOOKS ON MUSIC
N -- FINE ARTS
P -- LANGUAGE AND LITERATURE
Q -- SCIENCE
R -- MEDICINE
S -- AGRICULTURE
T -- TECHNOLOGY
U -- MILITARY SCIENCE
V -- NAVAL SCIENCE
Z -- BIBLIOGRAPHY, LIBRARY SCIENCE
```

Listing 2.1: LCC parent classes

```
PA -- Greek & Latin language and literature
PG -- Slavic, Baltic & Albanian language
PQ -- French, Italian, Spanish & Portuguese literature
PR -- English literature
PS -- American literature
PT -- German, Dutch & Scandinavian literature
PZ -- Fiction and juvenile belles lettres
```

Listing 2.2: Several subclasses of the LCC class P

```
Great Britain -- History -- Anglo-Saxon period, 449-1066
United States -- Foreign relations -- 1945-1989
Private investigators -- England -- Fiction
Mississippi River Valley -- Fiction
Runaway children -- Fiction
```

Listing 2.3: Examples of LCSH tags

based on genre, region, time period or others by double dash. Some examples of LCSH tags are listed in Listing 2.3. A book has usually several subject tags.

2.3 Text Analysis

In text analysis, the goal is to process text data in order to derive some useful knowledge. The aim may be to classify the text into different categories, extract the sentiment[6] or add useful tags. Another goal could be to find similarities between the given texts.

The texts in the text analysis are usually called *documents* and the set comprising of all documents is called *corpus*. Individual words in the document are created by the *tokenization* process and thus called *tokens* or *terms*.

2.3.1 Vector Space Model

Vector space model is a model often used in information retrieval. Each document is represented as a vector of identifiers. Every vector dimension corresponds to one term. The value in the vector is non-zero if the term occurs in the document. The model enables an easy computation of the document similarity. The cosine similarity of two vectors u and v can be computed as:

$$\frac{u \cdot v}{||u||\ ||v||}$$

For normalized vectors, the similarity equals to their dot product. The Vector space model was first used in the *System for the Mechanical Analysis and Retrieval of Text (SMART)* developed at Cornell University in the 1960s.

2.3.2 Bag of Words Model (BOW)

The Bag of words model is an application of Vector space model. Each dimension of the document vector corresponds to one word. The value in the vector is then number of occurrences of the word in the document. Instead of absolute number of occurrences, a relative frequency is used, i.e., number of word occurrences divided by the total number

[6]Goal of the sentiment analysis is to determine the attitude of the writer. We want to know, e.g., if the review of a product is overall positive or negative.

measure	doc id	a	black	dog	has	he	his	is	the
BOW	doc1	0	0.25	0.25	0	0	0	0.25	0.25
	doc2	0.125	0.125	0.25	0.125	0.125	0.125	0.125	0
BOW (l2 norm)	doc1	0	0.5	0.5	0	0	0	0.5	0.5
	doc2	0.316	0.316	0.632	0.316	0.316	0.316	0.316	0

Table 2.1: BOW example

of all words in the document. Table 2.1 shows the BOW representation of a corpus with two documents, each containing one sentence as follows:

```
doc1: The dog is black.
doc2: He has a dog. His dog is black.
```

The document vectors can be also binary. The vector values are equal to 1 if the corresponding word occurred at least once in the document. Even though it is a simplified version of the BOW with relative frequencies, it might be useful for some tasks. A popular one is the usage in filtering out spams[21].

2.3.3 Term Frequency–Document Inverse Frequency (tf–idf)

The vectors in the BOW model get easily dominated by the high frequency words. Many similarity and distance metrics are then determined by the frequencies of these words ignoring the low frequency ones. Tf–idf term weighting introduces a measure to boost the importance of words in the model that don't occur very often and, conversely, reduce the significance of frequent words. The tf–idf is a product of two statistics – term frequency and inverse document frequency. The inverse document frequency was first formulated in 1972 by K. S. Jones[11].

Term Frequency (tf)

The term frequency is usually computed as a simple number of occurrences in the document, i.e., the same as in the basic BOW model without vector normalization. We denote the term frequency of term t in document d as $tf(t, d)$.

Inverse Document Frequency (idf)

Inverse document frequency can be seen as a *uniqueness* measure of the term t. The fewer documents containing t, the higher the idf value of t. There exist various definitions of the idf computation. Let D be the set of all documents. The classical definition is then:

$$idf(t, D) = log(\frac{|D|}{|\{d \in D : t \in d\}|})$$

The idf score is computed as the logarithm of the total number of documents divided by the number of documents containing t. In this thesis, we use the smoothed version of idf

measure	doc id	a	black	dog	has	he	his	is	the
tf	doc1	0	1	1	0	0	0	1	1
	doc2	1	1	2	1	1	1	1	0
idf	-	1.405	1.0	1.0	1.405	1.405	1.405	1.0	1.405
tf–idf	doc1	0	1	1	0	0	0	1	1.405
	doc2	1.405	1	2	1.405	1.405	1.405	1	0
tf–idf (l2 norm)	doc1	0	0.448	0.448	0	0	0	0.448	0.630
	doc2	0.377	0.268	0.536	0.377	0.377	0.377	0.268	0

Table 2.2: Tfidf example

defined as:

$$idf(t, D) = log(\frac{|D| + 1}{|\{d \in D : t \in d\}| + 1}) + 1$$

The idf value is computed as if we added one extra document that contains the whole vocabulary (all terms). Moreover, 1 is added to the idf value.

Tf–idf Score

The tf–idf value of a term t and document d in the context of documents D is computed as:

$$tfidf(t, d, D) = tf(t, d) \cdot idf(t, D)$$

After that, each document vector is usually normalized to uniform length – l2 norm. Table 2.2 shows the values of all tf–idf related measures[7] on the two sample documents:

```
doc1: The dog is black.
doc2: He has a dog. His dog is black.
```

2.3.4 Part of Speech (POS)

Each word in the English language is assigned one part of speech category, such as *noun* or *verb*. This category describes word's grammatical properties. The current part of speech taggers achieve an accuracy of more than 97 %, which is comparable to the human performance on that task[13]. In the implementation, we use the Penn Treebank tag set[19] containing 36 POS tags. The python module spacy aggregates those tags to classes, such as verb and noun, and adds a class for punctuation. It enables us, e.g., to inquire into the usage of question marks or exclamation marks in the document. Table 2.3 shows POS classes and tags for the sentence *I saw two flies.*

2.3.5 Stemming & Lemmatization

One word can occur in the document in different forms. These forms differ due to grammatical reasons, but still carry the same meaning. An example might be a verb *see*.

[7] The smoothed version of idf is used.

token	POS class		POS tag	
I	PRON	pronoun	PRP	personal pronoun
saw	VERB	verb	VBD	verb, past tense
two	NUM	numeral	CD	cardinal number
flies	NOUN	noun	NNS	noun, plural
.	PUNCT	punctuation	.	full stop

Table 2.3: Example of POS tagging

The words *sees*, *saw* and *seeing* have all different syntax but the same semantics. Both stemming and lemmatization process focus on removing word inflections and map the word to its common base form. Although they both have the same goal, the approach differs a bit. We will show output[8] of both algorithms for the following sentences:

```
1 I saw two flies.
2 He sees three mice.
3 A saw is a useful tool.
```

Stemming

A stemmer uses a fixed set of rules to chop off ends of words, therefore it may run into problems for words with multiple meanings, such as *saw*, which can be both a noun and a verb. The Porter Stemmer[20] modifies the words in the sentences as follows:

```
1 i saw two fli
2 he see three mice
3 a saw is a use tool
```

The stemmer stripped correctly the third person *s* from the verb *sees*. The word *flies* was converted to *fli*[9]. It didn't recognize that *mice* is a plural of *mouse* and *saw* in the second sentence an irregular form of *see*. The word *useful* was changed to *use*.

Lemmatization

The goal of lemmatization is to turn each word to its lemma – the base form that can be found in dictionaries. Lemmatization needs to know the POS of the lemmatized word and does a vocabulary lookup to create the most accurate lemma. Therefore, it is more computationally intensive than the stemming algorithm. The lemmas of the words in the sentence are:

```
1 i see two fly
2 he see three mouse
3 a saw is a useful tool
```

[8] The punctuation is omitted and each word is turned to lowercase.

[9] Porter Stemmer changes the y at the end of nouns to i to capture the plurals more easily.

The lemmatization managed to convert all verbs and nouns correctly to its lemma. It returned different lemmas for the same word *saw* based on its context (POS tag). Unlike the stemmer, it didn't change the adverb *useful* to the verb *use*, as the lemmatization process maintains the POS tag of the word.

All in all, both processes try to turn verbs to their infinitive forms and nouns to singular number. We need to keep in mind that we lose the information about the tense of the verb. The used tense in the document might also be a good indicator if one author writes, e.g., in past tense and the other in present tense.

2.3.6 Stop Words

The most common words in the document, such as *the* or *it*, usually don't provide much insight, as they occur in almost all documents. These words are called stop words. The POS classes of stop words are usually articles, conjunctions, particles, auxiliary verbs or pronouns. The length of the stop words list varies based on the text analysis task[10]. The stop words list most be also compatible with the tokenization process. If the tokenization splits *don't* as *do* and *n't*, the token *n't* has to be on the stop words list. Other tokenizers might split it to *don* and *t* without including the apostrophe – then the stop words list should include the token *t*.

2.3.7 N-gram

The BOW model carries just the information of occurrences for each token. It doesn't capture the order of the tokens in the original document. Therefore, two following documents has the same vector representation:

```
1 Cats eat mice.
2 Mice eat cats.
```

The information who likes whom gets lost. One way to deal with this is to save not only single tokens, but also pairs or triples of words as they occur in the document. Pairs of consecutive words are called bigrams and triples trigrams. The bigrams of the first document are[11]:

```
1 (cats, eat)
2 (eat, mice)
```

A good practice is to include in the model terms consisting of both single words (unigrams) and bigrams. The drawback of this approach is that the model can get much bigger as the number of bigrams grows quadratically with the number of words. The model needs to be then limited to n most frequent terms.

[10]Search engines or related questions queries usually filter out the most used words including words with a meaning - the word *want*, for example, to find the semantically most similar content. These might, however, cause problems when the query consists only of stop words, such as the band name *The Who*.

[11]In this example, tokens are just turned to lowercase, stemming or lemmatization is not used.

Skip-gram

Skip-gram is a generalization of n-gram where the words doesn't have to occur in a sequence. Instead, the k-skip-n-gram is defined as a subsequence of the document where all n words occur in the document order, but, unlike n-grams, can skip up to k words for each skip-gram component. The n-gram can be then seen as a 0-skip-n-gram as no skips are allowed. For the sentence *I saw two flies*, all 1-skip-2-grams would be:

```
I saw, I two, saw two, saw flies, two flies
```

This sentence and a sentence *He saw three flies* has no n-grams in common. However, both sentences contain an 1-skip-2-gram *saw flies* which might provide valuable information.

2.3.8 Named Entity

Named entities are one or more words in text that can be classified into categories such as person names (Sherlock Holmes), locations (London) or organizations (Scotland Yard). Apart from that, there is a general agreement in the Named Entity Recognition Community on the inclusion of temporal expressions (e.g. Monday, two years ago, at 3 p.m.) and some numerical expressions such as amounts of money and other types of units[15]. The process which extracts the named entities from the text is called Named entity recognition (NER).

2.4 Machine Learning Models & Algorithms

We expect reader is familiar with the basic Machine Learning methods and concepts, so we just briefly introduce each method and provide references for further reading.

2.4.1 Naive Bayes Classifier

Naive Bayes Classifiers are quite popular in text processing. It is called naive as the classifier works with features as if they were independent. When the classifier makes a prediction it chooses the hypothesis (i.e., the sample comes from the given class) with maximum a posteriori (MAP) decision rule. Let n be number of features, K number of classes, then the prediction is made as:

$$y = argmax_{k \in \{1,\dots,K\}}\ p(C_k) \prod_{i=1}^{n} p(x_i|C_k)$$

. In other words, compute the probability that x comes from the class C_i when we know the distribution of feature values for each class. The predictions are made for every feature independently and then multiplied together. The result is then multiplied by the a priori probability of the given class. The prediction is then the class with the maximum value.

Even though the independence criteria is rarely fulfilled, the algorithm is usually a good base classifier. It can choose the most probable class, but can't deliver a good confidence guess on how good the prediction is.

Gaussian Naive Bayes Classifier

Gaussian Naive Bayes classifier can be used also for non-integer values. If it can be expected that the features are normally distributed, we can compute the probability of feature coming from a given normal distribution. For each class, means μ_c and standard deviations σ_c^2 of all features are computed. The probability that x has value v when the observation comes from the class c is then:

$$p(x = v|c) = \frac{1}{\sqrt{2\pi\sigma_c^2}} e^{\frac{(v-\mu_c)^2}{2\sigma_c^2}}$$

This formula normalizes the value v based on the underlying distribution. Than it computes the probability of v coming from this distribution. For more detailed information see [22].

2.4.2 Decision Trees

Decision trees are a popular class of algorithms for classification and regression. Their main advantage is that they provide complete reasoning for the given prediction, which is desirable for some applications. Apart from that, a decision tree can use a heterogeneous set of features, including, e.g., categorical or ordinal types, without an explicit need for normalization. The reason is that the decision (split) is made independently on other features. This is also the main disadvantage of decision trees that they can't learn some concepts which require decisions based on a combination of multiple features – e.g. XOR[**?**].

Random Forest

Random forests are an ensemble learning method which creates multiple decision trees and predicts the mode of the predictions of individual trees[3]. For each decision tree, when the feature for making the split is being chosen, it doesn't choose the best feature out of all features which reduces the entropy the most. Instead, it chooses the best feature of a randomly chosen subset of features available. Usual value is $\sqrt{n}$ with n bring total number of features. There are two reasons for this approach:

1. The model creates a different decision tree every time and is more robust and less prone to overfit on the training set.[12]

2. The time needed to train the random tree classifier is reduced greatly, as not all features are tried every time.

[12]The extreme situation would be if the algorithm created all decision trees the same. Then the advantage of the variability is gone.

2.4.3 K-mean Clustering

K-mean clustering is a unsupervised algorithm, which splits the data points into k clusters with cluster centers called *means*, as they are the geometric centre of all data points assigned to the cluster.

The training process is done iteratively. In the beginning, the centers are chosen randomly[13]. Next, in each iteration, every data point from the training set is assigned to its nearest center. After that, centers are moved to be in the middle of the data points assigned to this center. These two steps are repeated until no data point changes its cluster.

K-means clustering algorithm is popular due to its simplicity. Its drawbacks are that different ks have to be tried out, or expert knowledge incorporated, to find the optimal k for the number of clusters.

2.5 Related Work

The Utility of Information Extraction in the Classification of Books

In The Utility of Information Extraction in the Classification of Books[2], Betts also analyzed the books in PG. However, the analysis more specialized. The focus was on the LCC categories B (Philosophy, Psychology, Religion) and D (History). The goal was to predict correctly both the parent class as well as the specific LCC tag. In the extended version, classes H (Social Sciences) and Q (Science) were included as well. The paper compares multiclass and one-vs-all approaches to multivariate predictions. Betts combines different approaches and concludes with the best classifier using methods of both Text Categorizing and Named Entity Recognition models. The F_1 score of 0.81 is reported.

Word2vec

In 2013, Tomas Mikolov with his Google research group introduced a new model for text analysis – word2vec[14]. This approach assigns each word in the document a vector in a high dimensional space. The coordinates of the vector are computed looking at the words, which are often in its neighbourhood. In other words, it looks for various skip-grams. The model is trained on the continuous bag of words (CBOW) – in this task, the word is to be guessed based on its context, i.e., its surrounding words. It has been shown that words with similar meanings are also close to each other in word2vec representation, which raised the popularity of the model.

Summary:

In this chapter, we provided an introduction on PG along with some basic knowledge regarding Text analysis, which is used in the next chapter where we create a processing

[13] Or using some heuristics which spreads the initial centers more evenly with respect to the underlying data.

pipeline. At the end of the pipeline are the feature vectors we can use for exploration and prediction.

3 Design

In this chapter, we first describe the data processing pipeline – from downloading data to extracting document features. Section 3.2 provides information on types of features that are used for exploration and prediction. Section 3.3 discusses the specifics of the prediction task and overall methodology and approach of the analysis. Finally, Section 3.4 introduces the algorithm for capturing the interactions between characters.

3.1 Data Processing Pipeline

First part of the processing part is to collect metadata. Based on them, we can choose documents we want to download. After that, we process the documents and create feature vectors. All steps in the pipeline can be seen in Figure 3.1.

3.1.1 Preparing Metadata

There are two reasons why we need metadata. First, it helps us to select the documents we are interested in. Second, the metadata provides us with labels, such as author names, which we need to train the corresponding classifier. The metadata fields of interest are:

- title
- author information – name, year of birth, year of death
- subjects – some content information in form of LCC and LCSH tags (see below)
- language
- repository type – text, audiobook ...

In the analysis, we are interested only in the English text documents. Metadata *language* and *repository type* can be used to select the desired documents. However, as we see later during the data exploration (Section 5.1), such filtering is not sufficient. The *text* label for the *repository type* means only that it is a text document – these can be also foreign language dictionaries or lists of people or books etc., which are not in our area of interest.

In the following, we discuss the usage of the metadata fields mentioned above.

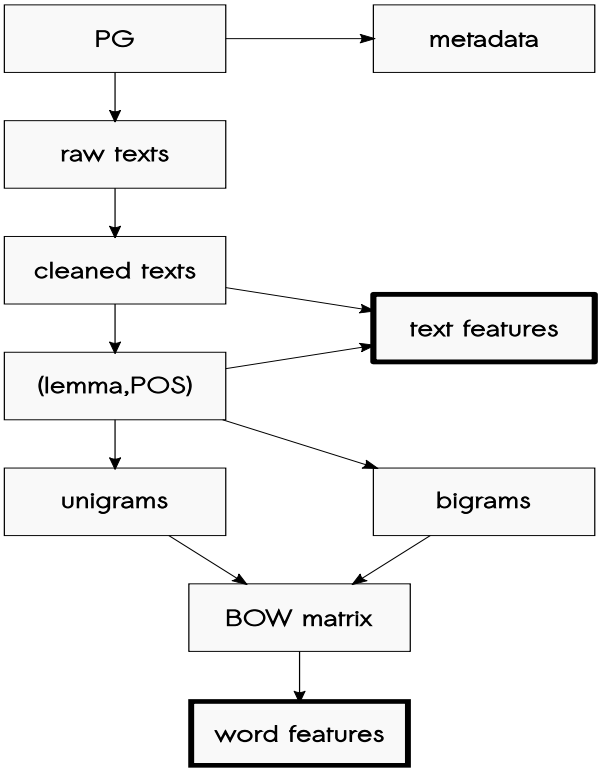

Figure 3.1: Overview of all steps in the data processing pipeline

Title

The title label is used for exploring and understanding the data. In a few cases, it can be used for data filtering, e.g., to exclude all repositories relating to *Copyright Extensions*[1].

Author

Each author in PG is assigned a unique id. This id relates to the person, not the pen name. It means that authors who published under multiple names are aggregated together and assigned one name. Later, we train an author classifier on the popular authors. An author's name is stored in the format *surname, name*, such as:

```
Shakespeare, William
```

Year

The metadata doesn't contain any information on document's publish date[2]. Therefore, for year predictions, we use the year of the author's birth.

[1] These lists aggregate for every year the book names whose author applied for the copyright extension. This enables PG to see which books are (or will be soon) in the public domain and can be legally included in the project.

[2] The catalog actually contains the field *publish date*, but it relates to the date the document was published to PG.

Epoch

As we want to predict an epoch for the given document, we partition author's birth years into 9 classes trying to approximately capture the individual literary movements. Such an attempt is ambicious, however, as the literary movements were not in all places at the same time. Moreover, during the periods of a literary movement change, there had to exist multiple literary movements at the same time. We define the birth year borders of the eras so that we capture important and influential authors in their most significant epochs:

- Ancient literature (750 BC - 499 AD)
- Medieval literature (500 - 1449)
- Renaissance (1450 - 1549)
- Baroque (1550 - 1684)
- Enlightenment (1685 - 1749)
- Romanticism (1750 - 1799)
- Realism (1800 - 1849)
- Late 19th century (1850 - 1899)
- Modern literature (authors born 1900 and later)

Language

Project Gutenberg contains books in more than 50 languages. One book can be written in one or more languages. We are interested in books that are written in English only. Based on this label, we filter out all books that are not in English or are written in multiple languages.

Type

As not all repositories in Project Gutenberg are book texts, this field states if the repository is a dataset, audiobook, video or another format. We are interested only in repositories with the label *Text*.

Library of Congress Classification (LCC) Tags

Each document is assigned a set of LCC tags (see Section 2.2.1). This set can be also empty if no tag was assigned. For example, *Leviathan* by *Thomas Hobbes* has the following tag:

```
{JC}.
```

The book is classified as *Political Science* (the main category J) and subclass *Political theory* (JC)

An example of a book with multiple tags is *The Adventures of Tom Sawyer* by *Mark Twain*, which has following tags:

```
{PS, PZ}
```

That means the book belongs to the main category *Language and Literature* (as both tags start with P). Within this category, it can be further classified as *American Literature* (PS) and *Fiction and juvenile belles lettres* (PZ).

Categories

As there are 21 LCC parent classes and many more LCC tags, we need to aggregate them somehow, so that it enables us to explore the data more convieniently. Therefore, we create 6 super categories[3]:

- Language and Literature
- History and Geography
- Science and Technology
- Philosophy, Psychology, Religion
- General works
- Social science and Arts

Subjects

Subjects are described by the LCSH tags as defined in Section 2.2.2. Each book has a set of subjects. For example *The Adventures of Tom Sawyer* by *Mark Twain* has following subjects:

```
Adventure stories,
Bildungsromans,
Boys -- Fiction,
Child witnesses -- Fiction,
Humorous stories,
Male friendship -- Fiction,
Mississippi River Valley -- Fiction,
Missouri -- Fiction,
Runaway children -- Fiction,
Sawyer, Tom (Fictitious character) -- Fiction
```

We extract those subjects which we are going to predict a classifier for:

[3]We refer to these artificially created categories as *super categories* if the term *categories* could arise confusion with the LCC's main class or other metadata.

- science fiction
- adventure stories
- love stories
- short stories
- historical fiction
- poetry
- drama
- detective and mystery stories

3.1.2 Preparing Documents

Next, we collect texts of all English documents – i.e., documents with metadata labels being *en* for language and *Text* for type. The files contain headers and footers added by PG which need to be stripped. We also do some minor changes regarding the formatting of the document.

Stripping Headers and Footers

The documents contain headers and footers that are not related to the content of the original book. These might be, e.g., copyright notes or comments relating to the transcription of the book. As the headers and footers developed during the time, we maintain a list with header and footer beginnings. When the document is read, the last occurrence of a header and first occurence of a footer is marked. The cleaned text of the document is the text between the markers.

Listing 3.1 shows the header and beginning of the footer for the most popular[4] text in Project Gutenberg - *Pride and Prejudice* by *Jane Austen*. The last header in this case would be *Produced by Anonymous Volunteers*. Documents usually end with line beginning with *End of the Project Gutenberg Ebook*.

Minor Formatting Changes

The documents in ASCII format are usually written using only 80 characters per line. The line is then ended by the new line symbol. This might arise problems for the POS tagger, as it considers newlines as a separator, which might reduce the tagging accuracy. Therefore, we replace all newlines having an alphanumeric symbol to the left and right[5] with single space.

[4]Based on the number of downloads in the last 30 days in January 2017.
[5]That means that the new line character was used instead of space.

```
The Project Gutenberg EBook of Pride and Prejudice, by Jane Austen

This eBook is for the use of anyone anywhere at no cost and with
almost no restrictions whatsoever.  You may copy it, give it away or
re-use it under the terms of the Project Gutenberg License included
with this eBook or online at www.gutenberg.org

Title: Pride and Prejudice
Author: Jane Austen
Posting Date: August 26, 2008 [EBook #1342]
Release Date: June, 1998
Last updated: February 15, 2015]
Language: English
Character set encoding: ASCII

*** START OF THIS PROJECT GUTENBERG EBOOK PRIDE AND PREJUDICE ***
Produced by Anonymous Volunteers

PRIDE AND PREJUDICE
By Jane Austen

Chapter 1

It is a truth universally acknowledged, that a single man in possession
of a good fortune, must be in want of a wife.

However little known the feelings or views of such a man may be on his
first entering a neighbourhood, this truth is so well fixed in the minds
of the surrounding families, that he is considered the rightful property
of some one or other of their daughters.

"My dear Mr. Bennet," said his lady to him one day, "have you heard that
Netherfield Park is let at last?"

(...)
End of the Project Gutenberg EBook of Pride and Prejudice, by Jane Austen
*** END OF THIS PROJECT GUTENBERG EBOOK PRIDE AND PREJUDICE ***
***** This file should be named 1342.txt or 1342.zip *****
This and all associated files of various formats will be found in:
        http://www.gutenberg.org/1/3/4/1342/

(...)
```

Listing 3.1: Raw text of Pride and Prejudice by Jane Austen

Apart from that, we replace locale-specific quotation marks, which occur in ASCII-8bit documents, by the default ones from the base set of ASCII.

After the preprocessing, the document would look like this:

```
1  PRIDE AND PREJUDICE
2  By Jane Austen
3
4  Chapter 1
5
6  It is a truth universally acknowledged, that a single man in
       possession of a good fortune, must be in want of a wife.
7
8  However little known the feelings or views of such a man may be on
       his first entering a neighbourhood, this truth is so well fixed in
       the minds of the surrounding families, that he is considered the
       rightful property of some one or other of their daughters.
9
10 "My dear Mr. Bennet," said his lady to him one day, "have you heard
       that Netherfield Park is let at last?"
11 (...)
```

Listing 3.2: Stripping headers and footers

3.1.3 Document Processing

In the next steps, we process the text in order to be able to compute the feature vector. The text is tokenized, lemmatized and POS are computed. After that, we transform the lemmatized tokens to unigrams and bigrams and compute their occurences.

Tokenization, Lemmatization, POS Tagging

The python module *spacy* does all three steps in one go. It takes a text and outputs list of tokens with their linguistic properties, including token's lemma, POS class and POS tag. At this stage, punctuation, such as commas or quotation marks, are also tokens. The tuples are shown in the following order:

```
(original word, lemma, POS class, POS tag)
```

For sentence *Violets are red, roses are blue.*, tokenization gives following output:

```
Violets  violet  NOUN   NNS
are      be      VERB   VBP
red      red     ADJ    JJ
,        ,       PUNCT  ,
roses    rose    NOUN   NNS
are      be      VERB   VBP
blue     blue    ADJ    JJ
.        .       PUNCT  .
```

In the previous list, JJ denotes adjactive in the basis form, NNS a plural noun and VBP a verb in the present tense not in the 3rd person singular form.

Unigrams, Bigrams

All the tokens from the tokenization process are not necessarily real words, as it can also be a punctuation symbol. Tokens not consisting of alphanumeric characters are discarded. We are left with following unigrams (we write only lemmas):

```
violet, be, red, rose, be, blue
```

To create list of bigrams, we take every consecutive couple from the list of tokens where both tokens are words (i.e., they are both on the unigram list):

```
(violet, be), (be, red), (rose, be), (be, blue)
```

Now we merge both unigrams and bigrams to a single list of terms:

```
'violet', 'be', 'red', 'rose', 'be', 'blue', 'violet, be', 'be,
   red', 'rose, be', 'be, blue'
```

Term Counts

In the next step, we count all the terms in the whole document:

```
'be'          2
'be, blue'    1
'be, red'     1
'blue'        1
'red'         1
'rose'        1
'rose, be'    1
'violet'      1
'violet, be' 1
```

The term counts are computed for every document. The terms are then sorted based on the total occurrence in the whole corpus. Only n most common terms are stored, others are discarded. We used $n = 50000$. In that way, we keep all possibly important terms and can choose a smaller subet of features later along with customizable stop words filtering.

3.2 Creating feature vectors

To apply machine learning methods on the documents, we need to create some features which can thoroughly describe a document. Therefore, we extract two types of features from the text. First, we compute some overall statistics relating to the document text as a whole. These might be the average word length in the document, proportion of stop words or frequency of verbs. Second type of features is based on the frequencies of the individual words.

3.2.1 Text features

In total, we create 51 text features. Two of them can be extracted direct from the cleaned text without any need of further processing:

1. **Average word length**. A word is a sequence of alphanumerical symbols.
2. Proportion of **stop words**[6].

Before computing the next batch, we need to tokenize the cleaned document into sentences. As the sentence tokenization is not a trivial task, we let *nltk*'s function *sent_tokenize* do the job. It can distinguish, full stops from dots in the abbreviations such as *Mrs.* and split the document into sentences more reliably than naive split on a dot or question marks. Then we can compute following features:

3. **Average number of words per sentence**.
4. Proportion of **sentences starting with quotation marks**.
5. Proportion of **sentences ending with quotation marks**.
6. Proportion of **sentences ending with a dot** (i.e. full stop).
7. Proportion of **sentences ending with a question mark**.
8. Proportion of **sentences ending with an exclamation mark**.

For the previous three features, quotation marks are ignored. It means, in case the sentence ends with quotation marks, we look at the symbol in front of them.

For the rest of the features, the POS tagging is needed. Following 11 features describe the proportion of tokens in the document being in this class. Even though punctuation is not counted as a POS, we include it as well to see the usage of commas and other punctation marks in the document. As each token belongs to exactly one class, the values of these 11 features sum up to 1. These are the POS class features:

9. Proportion of **adjectives**.
10. Proportion of **adpositions**. This class contains prepositions and postpositions[7].
11. Proportion of **adverbs**.
12. Proportion of **conjunctions**.
13. Proportion of **determiners**.
14. Proportion of **nouns**.
15. Proportion of **particles**.

[6]The stop words list is taken from *nltk.corpus.stopwords.words('english')*. This list was chosen as it seems to contain all important stop words and is also compatible with the way we create the word tokens (*he's* is tokenized to *he* and *s* both beeing on the stop words list).

[7]Postpositions are the same as prepositions except that they come after the word. The examples are *ago* or *aside*.

16. Proportion of **pronouns**.
17. Proportion of **proper nouns**.
18. Proportion of **verbs**.
19. Proportion of **punctuation**.

The remaining 32 features relate to the distributions of specific POS tags with their POS class. The complete list of POS tags can be found in [19]. Here is an example of POS tags which belong to the POS class VERB:

20. MD – modal verb
21. VB – verb in base form
22. VBD – verb in past tense
23. VBG – verb in gerund or present participle
24. VBN – verb in past participle
25. VBP – verb in present tense, non-3rd person singular
26. VBZ – verb in present tense, 3rd person singular (inf. + s)

It means that based on different verb classes, we should be able to determine the prevailing tense of the document. The used tense and the frequency of using the modal verbs can be a good style identifier. Moreover, the frequency of modals might imply a particular style such as law literature or educational literature for children.

The non zero features for the sentence *Roses are red, violets are blue* are:

```
avg word length             4.17
stop words proportion       0.33
avg sentence length         6.00
ends with quot. marks       1.00
adjective proportion        0.25
noun proportion             0.25
punctuation proportion      0.25
verb proportion             0.25
// following 5 features are those regarding the POS tags
adjectives, in base form    1.00
verbs, in present tense     1.00
nouns, plural out of nouns  1.00
commas, out of punctuation  0.50
full stops, out of punct.   0.50
```

3.2.2 Word Features

In Section 3.1.3, we already computed the document-term occurence matrix – rows correspond to individual documents and columns to terms. A word vector is specified by:

1. n most common terms
2. type – either tfidf term weighting or binary
3. list of stop words to be excluded[8]

For example, *tfidf-2000* model contains 2 000 most common terms from the corpus which are not on the stop words list[9]. The vector values are computed using the tfidf term weighting (example was shown in Section 2.3.3). Therefore, the word feature vector can be adapted based on the classification model used for the given prediction task.

Some algorithms profit from training on many features (such as Naive Bayes Classifier), some algorithms have to cope with the Curse of Dimensionality (e.g. K-mean clustering[1]).

The best n is used for each task based on crossvalidation.

3.2.3 Combined features

We can also combine text and word features into a single feature vector. A combined *tfidf-2000-combined* model has 2 051 features. First 2 000 values are the same as in the previous example. The last 51 features are the appended text features.

3.2.4 Feature Vector Normalization

The tfidf feature vectors are $l2$ normalized. For text features, when the algorithm requires normalized features, we normalize each feature using z-score.

3.3 Classification Task

We create a seperate model for each text-feature we want to predict. By doing this, each model can be specialized to accurately predict a single document attribute.

Naive Baseline Estimate

For each document attribute we want to predict, we first formulate a base classificator which predicts always the same class based on apriori distribution. For classification tasks, this predicts the largest class (modus). For regression tasks, it predicts a mean of

[8]The stop words are excluded after the word count as creating the Count vectorizers is a quite costly operation. The stop words set can be then adapted based on explorations.

[9]Bigrams are discarded if at least one word is on the stop word list. Otherwise, the feature vector would be polluted by pairs such as (determiner, noun).

the values of this document attribute. The base classificator helps us to estimate the real quality of the predictors.

Choosing Parameters

When training a model, we try out and compare Naive Bayes models and Decision Trees with different parameters. At the end, we choose the algorithm and feature type which performed the best on the crossvalidation set – out of all data, we save 10 % data for training and create the best possible classifier using these 90 % with 10-fold crossvalidation.

Naive Bayes Models

We can use two versions of NB. The multinomial version is used for the binary features, the gaussian for all others.

Decision Trees

For decision trees, we can include all possible features, as each feature is split based on its scale independently on other features. Therefore, we use often the combined features consisting of both text features and word features for decision trees and random forests.

3.4 Main Characters Extraction and Exploration

In this section, we introduce a Character Interaction model. For each character, it computes number of occurrences in the document and how often they interact with other characters. Two characters interact with each other every time when there are less than max_d sentences between the name occurrences. The process can be split into 3 tasks:

(a) Identify the character names in the book. Add titles *Mr.* and *Mrs.* if possible.

(b) Group different names of the same characte together.

(c) Compute the interaction strength between each two characters and visualize it.

3.4.1 Identifying Character Names

Python module spacy has an entity extractor. It tries to capture entities with a specific meaning – e.g. a person name, time details or street name. An entity can consist of multiple words. The entity extractor assigns each entity a type, the most common being:

- *Person* – Holmes, Sherlock Holmes
- *Organization* – Scotland Yard
- *Location* – London, Great Britain

- *Cardinal and Ordinal numbers* – half, two, first, 3rd
- *Date and Time* – yesterday, Monday, this morning, a few minutes ago

We are interested in the entities of the type *Person*. The entity extraction process is not perfect, of course – for example, it might consider *Scotland Yard* to be a *Location* instead of *Organization*, because it contains the word Scotland. The accuracy of the entity extractor is dependant on the similarity of the document to the corpus it was trained on. The entity extractor takes capitalization into account. Because of that, it can classify words with capital letters without a spelling reason as a named entity. As an example might serve a snippet from The Constitution of The United States of America, Article 1, Section 3:

```
But the Party convicted shall nevertheless be liable and
subject to Indictment, Trial, Judgment and Punishment,
according to Law.
```

Due to word capitalization, the entity extractor returns *Law* as a person entity and *Indictment, Trial, Judgment and Punishment* as an organization entity. Conversely some named entities being at the beginning of the sentence might not be recognized.

As the entity extractor doesn't consider titles Mr., Mrs. and Miss to be part of the name, we look at the preceding word of every person name and add the word to the name if it is a title. It is important to add these titles as some names are identical except for the title, e.g., Mr. Darcy and Mrs. Darcy. If we didn't include the title, we wouldn't know that these are actually two different people.

3.4.2 Character Grouping

One person usually occurs in the book under different named entities. *Sherlock Holmes* is called *Sherlock* by his friend Dr. Watson, his clients call him *Mr. Holmes* and a letter to him is addressed to *Mr. Sherlock Holmes*. To correctly capture the interactions between the characters, we need to know which named entities represent the same person.

The algorithm describing the grouping of names is shown in Listing 3.3. It takes all names one by one – sorted in the descending order by the name length[10] – and finds for the given name N all names that contain N in itself. These are called *super names*. Delete all super names which occur less than k times[11] in the whole document . After deleting super names with low occurrence, if there is only one super name left, replace all occurrences of the original name N by this super name. If there are more super names, we can't choose the right one, so leave N as is.

[10]The names have to be pre-sorted. As an example, if we had names *Watson*, *John Watson* and *Dr. John Watson* and the algorithm took *Watson* first, it doesn't know if *Watson* should be assigned to *John Watson* or *Dr. John Watson*. However, if we process them in the right order, at the time the algorithm first sees the name *Watson*, the name *John Watson* doesn't exist anymore as it was already replaced by *Dr. John Watson*.

[11]A reasonable k turned out to be somewhere between 3 and 10 based on the length of the document. If the k is too low, it might map names to too detailed or misclassified entities, e.g., *Sherlock* to *Sherlock the Wise*. If the k is set too high, some names are not grouped together as there are considered too rare.

```
1  def group_names(document,k):
2    S <- set of all names in the text, sort in desc. order by
        name length
3    for Name in S: # taking longest names first
4        # all names that contain more information than the Name
           and have at least k occurences (excl. self)
5        Super_names <- all names from S - {Name} which contain
           all words from Name and have at least k occurences
6        if size(Superset_of_name) ==1:
7            # only one candidat for being the same person
8            replace all occurrences of Name with the element of
               this set
9            remove Name from S
10       else:
11           # if there are 0 or 2 and more names that contain
               Name, we can't easily replace Name by another
               entity, so do nothing
12           continue
```

Listing 3.3: Algorithm describing the grouping of names of the same characters together.

3.4.3 Computing Interaction

To compute interactions between characters, we assign an id-sentence to each occurrence of the character. Next, for every occurrence of N, find all names which occur in the distance at most m sentences from N. Such pair of names are called *interaction*. We compute the strength of an interaction as a^d where a is a number smaller than 1 and $d \in [0, m]$ is a distance in number of sentences from the sentence having N. This function has a decay effect which makes the strength of interactions weaker with the higher distance between sentences. Finally, the strength of interaction between characters $N1$ and $N2$ is sum of all their interactions. The algorithm is described in Listing 3.4.

Summary:

In this chapter we saw the processing pipeline and how the features are created. Next, we briefly discussed some basis for the classification task and intorduced the Character Interaction Algorithm.

```
def compute_interactions(T, max_d, a):
  # T are tuples (of name,sent_id)
  # M is a matrix which sums up all interactions
  M=[]
  for N, sent_id in T:
      # find all interactions at most max_d sentences away
      # ignore self-interactions
      Interact <- {(name,id) in T : abs(id, sent_id) <= max_d
                                    & name != N}
      for I in Interact:
        # compute the strength of the interaction and add to M
        M[N,I.name] += a^(I.dist_from_N)
  return M
```

Listing 3.4: Algorithm describing the grouping of names of the same characters together.

4 Implementation

4.1 Downloading the Data

4.1.1 Documents

Project Gutenberg's website doesn't enable users to robot the site directly but redirects to a repository[5] where all data can be freely downloaded wihout receiving a ban. Several addresses are tried out if the book text is not find as books using different encodings uses slightly different URLs[1]. We download all documents defined as *Text* and *English* by the metadata catalog. After downloading data, we stripped each book of its Project Gutenberg's headers and footers using the *gutenberg* for *python* package which includes an exhaustive list of headers and footers available in different documents.

4.1.2 Metadata

The metadata file can be obtain from Project Gutenberg's website as well. It is updated daily to include the newest books added to the project. The content of the metadata file was described in detail in Section 3.1.1. We stick to the data from the catalog as the only source of metadata as obtaining reliable data by either scrapping *Wikipedia* or online book databases would be to costly. The catalog contains few mistakes[2], but they are not important in the scope of the whole analysis.

4.2 Used python Modules

The implementation of the prediction task and Character Interaction model is done in python3. In the analysis, we used mainly python modules relating text processing and machine learning.

4.2.1 nltk

Nltk is a popular module for text analysis. It provides user with various tokenization processes, stemming algorithms as well as with POS tagging. However, the lemmatization process was not as good as the one in the module spacy.

[1] e.g. instead 1.txt, they are stored as 1-0.txt or 1-8.txt
[2] e.g. wrong language field, or bad author assigned

4.2.2 spacy

Spacy is module for text processing which focuses on efficiency and robustness, as many parts of the module are programmed in cython to achieve better performance. It is available for English and German language. For a given text, it creates tokens and computes many kinds of token properties, such as POS tags, entity extraction[3] and others. Its drawback is a bit longer loading times as it needs the whole English dictionary to perform accuratelly, which is rather large.[4]

4.2.3 sklearn

Sklearn provides us with all machine learning algorithms we used. It covers decision trees, naive Bayes classifiers, but also ensemble methods , such as, random forests or boosting algorithms. Apart from that, we used the PCA module and K-clustering module. Last but not least, the algorithms needed to compute the BOW (CountVectorizer) or convert it to tfidf representation (TfidfTransformer or Vectorizer if the user wants to skip the CountVectorizer step) are also provided.

4.2.4 pandas

We used pandas to conveniently work with the feature vectors as it implements DataFrames and support indexing based the column and index names.

4.3 Own Implementation

In this section, we shortly introduce the included implementation. It is also available at *https://github.com/hobil/gutenberg* where the future updates will appear.

4.3.1 Prediction Model

To predict the author, epoch, category and genres of a text stored in [filename], run

```
1 python3 inspect_text.py -f [filename]
```

or

```
1 python3 inspect_text.py [text]
```

to hand over the text directly.

It requires spacy to be installed in order to be able to compute the feature vectors. The output is done to the standard output.

[3] Which we use in the Character Interaction Algorithm

[4] That is also the reason why a simple prediction in our implementation takes a bit longer then necessary, as the whole dictionary is loaded.

4.3.2 Character Interaction Model

The Character interaction model can be run in the same way as the prediction model:

```
1 python3 character_interaction.py -f [filename]
```

By default, it outputs the graph into a pdf file in the working directory.

5 Evaluation

This chapter contains the results of the analysis. First, in Section 5.1, we explore the documents and their attributes as well as the descriptive text features and their distributions. Section 5.2 introduces the feature vectors we use in the analysis and looks at correlation between features. In Section 5.3, we divide the documents into several clusters and inspect them. Next, in Section 5.4, we create classifiers for several document attributes. **??** *presents the results in character extraction and description. Finally, Section 5.6 discusses the overall runtime of the steps in the analysis.*

5.1 Document Exploration

We start the analysis with document exploration. The gained information can be then exploited in the prediction task and thus, obtain better results. As for the document metadata, it is interesting to know what kind of documents are actually in the Project Gutenberg. Before coming to the predictive analysis, it is also essential to know the sizes of the underlying classes. The document metadata we focus at are:

- author
- epoch (extracted from author's year of birth)
- category (described by LCC tag, e.g., *Q – Science*)
- genres, subjects and topics (described by LCSH tags, e.g., *Poetry* or *Fantasy*)

Next, we inspect the text features introduced in Section 3.2.1. We focus on their distributions, which might, apart from others, help us to find outliers among the documents, e.g., documents with too long or short words. Such documents are usually somehow special and not representative of the real texts we focus on. An example might be a children book full of illustrations. Even though the illustration book is classified by the PG catalog as an English text, it is of no use for us as those books contain only several sentences. Text features can also show the differences between the document classes. One could expect that scientific literature contains longer words and sentences than fiction literature or that it has different share of meaningful words (non stop words).

5.1.1 Document Selection

Project Gutenberg does not only contain books but also video and audio files. This makes the size of the project quite large. The *ftp mirror* of the project has 650 GiB.[1] As already mentioned, we are interested only in English text documents. Project Gutenberg has a *txt* file even in the non-text repositories – a readme file explaining the contents of the repository. Therefore it is not possible to rely on the *txt* file in the repositories to be the book's text. Some *txt* files only state that it doesn't make sense to transcript the book to plain text, as it may contain lots of tables or illustrations, and link to the *HTML* version instead. After filtering out the non-english and non-text repositories based on the PG catalog, we get 42 823 documents of a total size 16.07 GiB.

However, there are some documents classified by the catalog as English texts which are actually not a typical text of a book. We filter these out, as they are not representative for the actual documents we want to classify. Nevertheless, it is hard to draw a line what is still a text document and what is not – we filtered out documents which consist only of:

- illustrations and their captions
- tables (e.g. census results)
- decimal places of mathematical constants
- logs of chess games in the chess notation
- dictionaries to foreign languages and various lists

These documents are outliers due to having too long words (containing decimal places of mathematical constants), too short words (chess games logs as chess notation usually consists of two to three letters) or too long sentences (tables with census data or list of synonyms which miss full stops).

After filtering out such documents, we end up with 42 404 documents. We store them without headers and footers to speed up working with them as well as saving a bit of space – the total size is then 14.6 GiB. Table 5.1 shows the number of documents and their total sizes throughout the process. The analysis on document metadata (Section 5.1.2) and text features (Section 5.1.3) are performed on those 42 404 cleaned documents.

5.1.2 Metadata Exploration

In the following, we look at each document attribute one by one. Based on the observations, we determine the strategy for the classification task.

[1] The data in the ftp collection is redundant as it also contains ISO files aggregating the repositories to one downloadable format. Apart from that, historical versions of the files are also kept there for future references.

	Number of repositories	Size
English text documents of PG in zip format	42 823	6.0 GiB
Unpacked documents	42 823	16.1 GiB
Remaining non-text docs manually filtered out	42 404	15.8 GiB
Removed headers and footers	42 404	14.6 GiB

Table 5.1: Amount of data in the pipeline during document fetching

Author

In total, there are 14 215 unique author labels.[2] However, not all of them are an actual name of an author. When we sort the authors based the number of documents in PG, three most common author labels are *Various*, *N/A* and *Anonymous*. The most common author labels can be seen in Table 5.2.[3]

Documents written by *Various* authors are mainly collective works such as periodicals (60 %), dictionaries (7.5 %)[4] or encyclopedias (4.6 %). The most common LCC tags of documents written by *Various* authors can be seen in Table 5.3.

Documents written by *Anonymous* authors refer often to biblical topics (25 %). Another big group is the literature for the youth, which contains lots of oral literature and fairy tales (24 %). The most common LCC tags of documents written by *Anonymous* are shown in Table 5.4.

When creating the author classifier, we exclude documents with author labels *Various*, *N/A*, *Anonymous* and *Unknown*[5]. Even though these labels carry some information about the content (as we saw for *Various* and *Anonymous*), classifying a book as written by *Various* authors would be too general and not entirely satisfying.

Author name	No. of documents
Various	2874
N/A	1543
Anonymous	566
Lytton, Edward Bulwer Lytton, Baron	214
Shakespeare, William	178
Ebers, Georg	163
Twain, Mark	148
Kingston, William Henry Giles	132

Table 5.2: Eight most common author tags

[2]If an author published under multiple names, PG considers them as one author.

[3]PG contains different versions of some works, therefore the total number of documents of some authors might seem too high.

[4]PG contains even more dictionaries by *Various*, but they were filtered out as they did not match the criteria of having fluent text with sentences.

[5]The *Unknown* author tag is present for 100 documents. It is the same as *N/A* as some transcribers leave the author label blank rather than filling *Unknown* in there.

LCC tag	No. of documents
AP – Periodicals	1714
AG – Dictionaries and other general reference works	215
AE – Encyclopedias	133
PZ – Fiction and juvenile belles lettres	109
T – Technology General	89

Table 5.3: Five most common LCC tags of books written by *Various* authors

LCC tag	No. of documents
BS – The Bible	143
PZ – Fiction and juvenile belles lettres	136
PR – English literature	20
DA – History of Great Britain	16
BX – Christian Denominations	15

Table 5.4: Five most common LCC tags of books written by *Anonymous* author

After excluding these four author labels, there are 14 211 unique authors left. The distribution of documents written per author is captured in Figure 5.1. There are several authors with over 100 books (five of them were already shown in Table 5.2). However, the majority (13 639, which is 96 % of all authors) wrote less than 10 books. In order to see better the numbers of all authors who wrote only few books, Figure 5.2 shows the cumulative sum of authors who wrote only n documents or fewer. It can be seen that 9 908 authors, corresponding to almost 70 % of all authors, have only one document in Project Gutenberg. When adding authors with two (12 %) and three books (5 %), we see that 87 % of all authors wrote 3 books or fewer.

Because of that, creating the author classifier might be very challenging. Therefore, we focus on authors who wrote at least 5 books, as we need several different documents to reliably capture the author's style. Otherwise, we would just identify the authors with the one book in PG wrote by them and compute the similarity to their one book.

Author's Birth Year

As already mentioned in Section 3.1.1, as a year entry, the author's year of birth is used. Therefore, there will be no year labels for the documents written by the 4 author groups mentioned above – *Various*, *N/A*, *Anonymous* and *Unknown*. Apart from those, year labels are also not available for documents written by groups of people such as *United States*.

In total, there are 30 684 documents with an author's birth year available. Figure 5.3 shows the number of documents written by authors born in the given century.

When we look closer at the year distribution, there are almost 200 documents from the Ancient Greek and Roman Era. After that, PG contains close to no documents from the Dark Age period (5th to 10th century). From the 15th century on, the number of

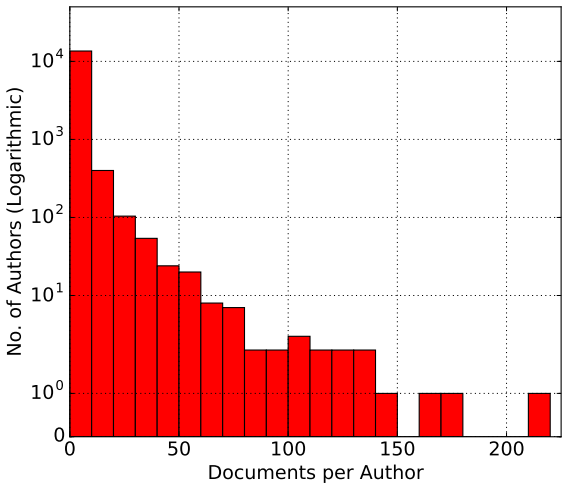

Figure 5.1: Histogram: Number of authors with the given number of documents written

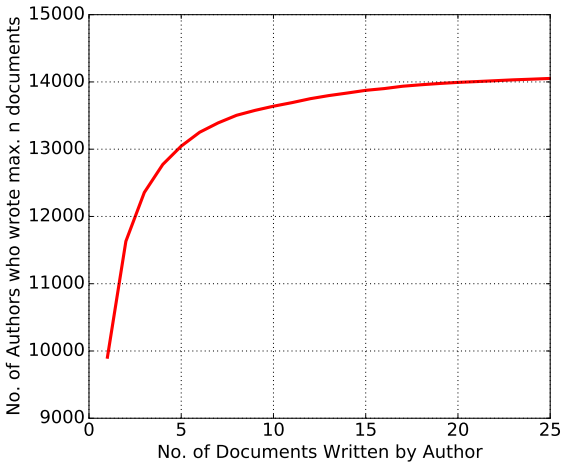

Figure 5.2: Cumulative number of authors based on number of documents written

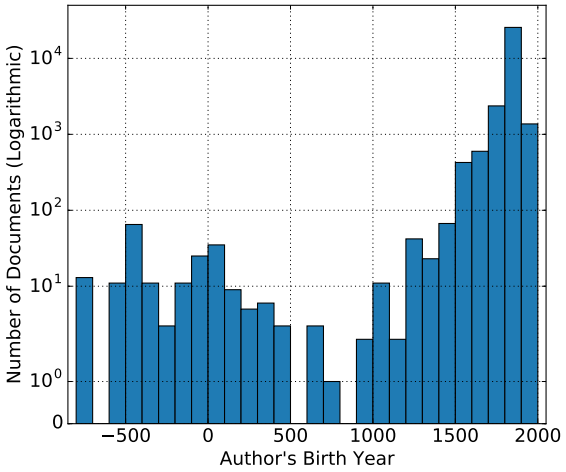

Figure 5.3: Histogram: Number of documents per century

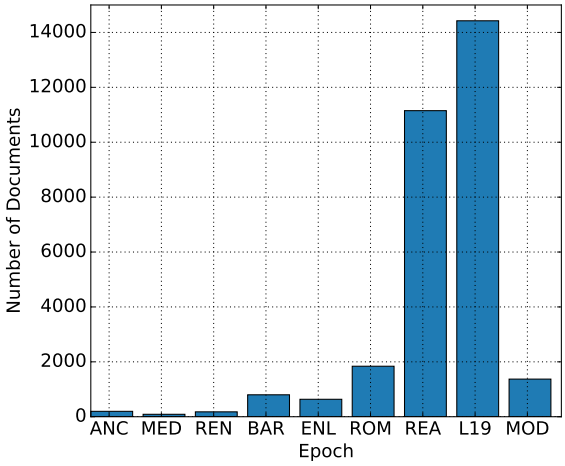

Figure 5.4: Number of documents per epoch

documents grows rapidly. The vast majority of documents come from the authors born in the 19th century (83 %).

Due to the unbalanced year distribution, we don't create a regressor which predicts a year label directly. Instead, we train a classifier which predicts one of the 9 epochs introduced in Section 3.1.1. Figure 5.4 shows number of documents in each epoch. We can see that the epochs *Realism* and *Late 19th century* each contain almost half of all documents. However, it doesn't make sense to divide them into classes smaller than 50 years period to reduced the class imbalance at all costs.

Document category (LCC)

We describe the document categories by the Library of Congress Classification tags which were already introduced in Section 2.2.1. Document can belong to multiple categories. Table 5.5 shows the number of LCC tags per document. For 10.9 % of documents, there is no category information available. Out of documents having at least one category assigned, 95.5% have exactly one LCC tag.

Table 5.6 shows number of documents in each category.[6] The majority of documents (57 % of documents with an assigned category) belong to the class *P*, which represents classic fiction books, such as novels, short stories or poetry. Table 5.7 shows the most common LCC tags (LCC classes including subclasses); the most documents come from subclasses of *P* – American literature, English literature and literature for children.

In Section 3.1.1, we grouped several LCC parent classes together and created 6 super categories. Table 5.8 shows the number of documents in each of them.

No. of LCC classes	No. of documents	% of all documents
0	4620	10.9 %
1	36070	85.1 %
2	1676	4.0 %
3	37	0.1 %
5	1	0.0 %
Total	42 404	100.0 %

Table 5.5: Number of documents with a given number of LCC classes

[6] If the document is classified into multiple categories, it is counted towards both of them.

LCC parent class	No. of documents
P – LANGUAGE AND LITERATURE	21 617
D – WORLD HISTORY	3 534
B – PHILOSOPHY, PSYCHOLOGY, RELIGION	2 722
A – GENERAL WORKS	2 146
E – HISTORY OF THE AMERICAS	1 436
Q – SCIENCE	1 235
H – SOCIAL SCIENCES	968
F – HISTORY OF THE AMERICAS	955
T – TECHNOLOGY	769
G – GEOGRAPHY, ANTHROPOLOGY	683
N – FINE ARTS	554
S – AGRICULTURE	428
M – MUSIC AND BOOKS ON MUSIC	272
R – MEDICINE	266
J – POLITICS	263
C – AUXILIARY SCIENCES OF HISTORY	232
L – EDUCATION	200
Z – BIBLIOGRAPHY, LIBRARY	186
K – LAW	125
U – MILITARY SCIENCE	80
V – NAVAL SCIENCE	53

Table 5.6: Number of documents in LCC parent classes

LCC class (incl. subcategory)	No. of documents
PS – American literature	7120
PR – English literature	6935
PZ – Fiction and juvenile belles lettres	5051
AP – Periodicals	1740
PQ – French, Italian, Spanish & Portuguese literature	1070
DA – History of Great Britain	977
PT – German, Dutch & Scandinavian literature	667

Table 5.7: Number of documents in LCC classes (incl. subclasses)

Super category	Number of documents
Language and Literature	21 617
History and Geography	6 735
Science and Technology	3 352
Philosophy, Psychology, Religion	2 722
General works	2 332
Social science and Arts	1 807

Table 5.8: Number of documents in each super category

Topics and genres (LCSH)

The best source of document topics and genres available for PG texts are LCSH tags, which were introduced in Section 2.2.2. There are no limitations on the number of tags per document – the detailed distribution can be seen in Figure 5.5. The most documents have one or two tags (61 % of documents), 10.8 % have no LCSH tag and 4.5 % have more than 5 tags.

In total, there are 23 407 unique LCSH tags. Table 5.9 shows that the LCSH tags are rather too specific as many of them occur in only one or two documents. Listing 5.1 shows the LCSH tags with most occurrences – *Short stories* followed by *Fiction* and *Science fiction*.

In order to simplify the too specific tags, we could split each tag into the set of tags on commas or double dashes, which signal the hierarchy separator. For example *English wit and humor – Periodicals* would be split into two tags *English wit and humor* and *Periodicals*. The total amount of unique tags reduces from 23 407 to 15 791. The most common simplified LCSH tags are listed in Listing 5.2. The tags which increased the most are those regarding general style of the book (Fiction, Biography) and tags regarding geographical location (United States, Great Britain, England).

To predict the subjects of the book, we create binary classifiers for several often used subjects, such as *Science fiction* or *Adventure stories*.

LCSH tag occurrences	No. of LCSH tags	% of all LCSH tags
1	14 969	64.0 %
2	3 266	14.0 %
3	1 520	6.5 %
4 - 5	1 392	6.0 %
6 - 10	1 106	4.7 %
11 - 20	590	2.5 %
21 - 100	494	2.1 %
101+	70	0.3 %
Total	23 407	100.0 %

Table 5.9: Number of occurrences per LCSH tag

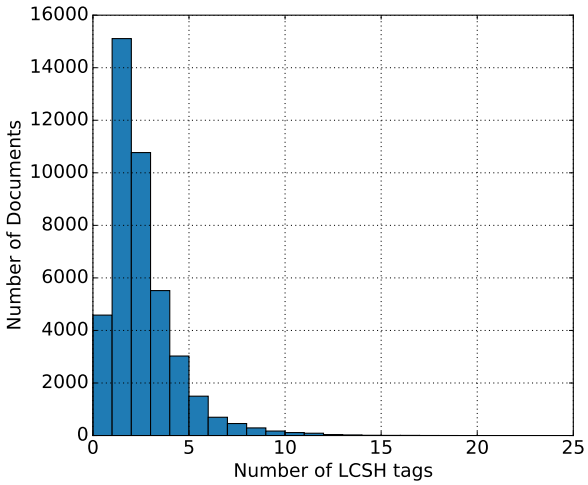

Figure 5.5: Number of LCSH tags per document

```
1430, Short stories
1242, Fiction
1211, Science fiction
705,  Adventure stories
632,  Conduct of life -- Juvenile fiction
565,  Love stories
555,  English wit and humor -- Periodicals
519,  Detective and mystery stories
470,  Historical fiction
420,  Western stories
```

Listing 5.1: Most used LCSH tags

```
9642, Fiction
4411, History
3645, Juvenile fiction
2216, United States
2051, Periodicals
1964, Social life and customs
1892, 19th century
1705, Great Britain
1644, England
1466, Biography
```

Listing 5.2: Most used simplified LCSH tags

5.1.3 Text Feature Exploration

In this section, we explore the text features that were described in Section 3.2.1. For selected text features, we look at their value distributions and try to explain the underlying variance in the distribution (e.g., why do some documents have so long sentences). The exploration helps us to identify outlier documents and provides some interesting pieces of information and insight into the English language. The cleaned versions of documents (i.e., without headers and footers) are used for the following analysis with only exception being the document sizes where we use the sizes of the original text documents as provided by PG.

Document Size

As stated in Table 5.1, the total size of all 42 404 raw documents is 15.8 GiB. There are some documents with size over 10 MiB. These are mainly collected works of one author – the largest document with the size of 15.3 MiB contains all works available in PG by Mark Twain, who, as we already saw by the author exploration, wrote almost 150 documents. Even though these documents are not a text of a single book as the rest, they could help us a lot with the author style classification. Another group of large files are the CIA World Factbooks. These books are almanacs released every year by the U.S. Government providing several pages of summary on every country in the world. However, the vast majority of documents have less than 0.5 MiB. To explore the sizes of these documents, we plot another histogram (see Figure 5.7) with only those books having less than 1 MiB (which corresponds to 94 % of all documents). It can be seen that the highest density of documents is between 50 and 100 kiB. The overall median size of a document is 300 kiB.

The document size feature is the only feature we extracted from the document which takes the actual text length into account. All other features are relative as we look at the proportions or normalized values based on the document length. Therefore, we won't use this feature in any classifier, as we don't want them to make decisions based on the length of the given sample.

Average Word Length

Next, we look at the document's average length of the word. For this purpose, the word is defined as an sequence of alpha-numeric symbols. The mean of the average word length is 4.32 with 95.8 % of values being within *half of word distance* from the mean. The distribution of the average word length can be seen in Figure 5.8

The average word length differs based on document category – the LCC class P has the shortest words. Interestingly, all other categories have the mean word length higher than the corpus average (see Listing 5.3) – it is possible due to the big size of class P with short words. This could greatly help to distinguish P from other classes. The longest words have documents from categories *Politics*, *Medicine* and *Law* (see Listing 5.4).

Figure 5.9 shows the distribution of the average word length based on document categories, which can show us more information than the mean statistics. We can see

that the average word length distribution for *Science and Technology*, *Social Science and Arts* and *History and Geography* is almost the same. The smallest variance have *General Works* and the biggest *Philosophy, Psychology and Religion*, which has an equal amount of documents for the average word length interval from 4.3 to 4.6. As we already expected from the mean statistics, the category *Language and Literature* has significantly smaller word lengths than other categories.

The average word length is also author-specific. Figure 5.10 shows the comparison for the 3 most frequent authors in PG: novelist *Mark Twain*, playwright *William Shakespeare* and novelist and politician *Lord Edward Bulwer-Lytton*. Shakespeare wrote the shortest words with 4.05 average word length per document. The reason might be the nature of drama genre, which usually contains lots of direct speech. Mark Twain's average word length per document is 4.16, which is slightly below the average of the *Language and Literature* category. The Lytton's word length per document is 4.36 in average, which is significantly higher than the mean for the category *Language and Literature*, which might indicate the usage of higher class English.

Average number of words in the sentence

The length of sentences could also help in distinguishing between document categories. We define sentence length as number of words in a sentence.

Figure 5.11[7] shows the histogram of the average sentence length per document. Although the mean is 20.9 words per sentence, there are several documents with the length of sentence more than 100 words. Such documents do not stick to the conventional style of writing and using full stops. An example are kinds of poetry which do not use dot at the end of verses. Another class are dictionaries or almanacs consisting of long entries without full stop. However, such an indicator that the document doesn't follow classic style of writing can be also useful.

When we compare the three categories with longest and shortest sentences (Listing 5.5 and Listing 5.6 respectively), we can see two interesting facts:

(a) Documents in category *J – POLITICS* and *K – LAW* seem to be written in quite a complicated language – these documents have the longest sentences as well as the longest words out of all LCC classes (see Listing 5.4).

(b) On the contrary, *P – LANGUAGE AND LITERATURE*, representing mainly ordinary fiction books, has by far both the shortest words and sentences.

```
28.1, J -- POLITICS
27.2, K -- LAW
26.3, D -- WORLD HISTORY
```

Listing 5.5: Three LCC classes with longest sentences

```
18.8, P -- LANGUAGE AND LITERATURE
19.7, A -- GENERAL WORKS
19.9, Z -- BIBLIOGRAPHY, LIBRARY
```

Listing 5.6: Three LCC classes with shortest sentences

[7]Only values up to 70 words per sentence are shown for better clarity.

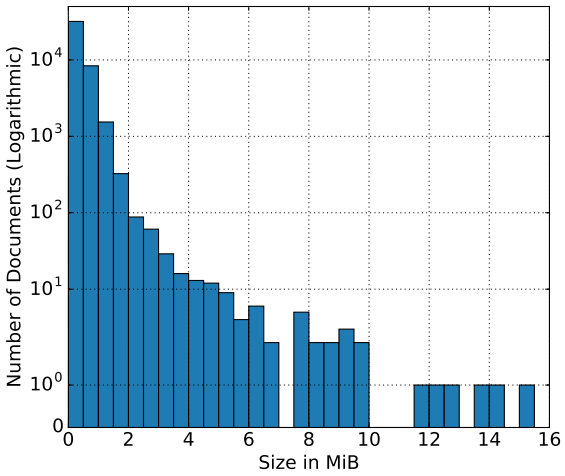

Figure 5.6: Number of documents per document size in MiB

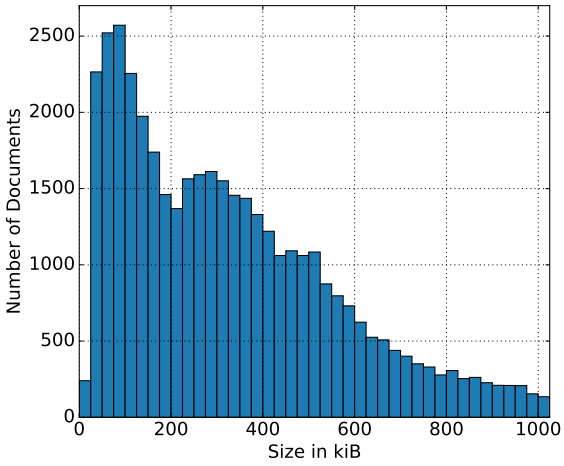

Figure 5.7: Number of documents under 1 MiB per document size in kiB

```
4.19, P -- LANGUAGE AND LITERATURE
4.36, G -- GEOGRAPHY, ANTHROPOLOGY
4.40, A -- GENERAL WORKS
```

Listing 5.3: LCC classes with shortest avg word length

```
4.71, J -- POLITICS
4.59, R -- MEDICINE
4.58, K -- LAW
```

Listing 5.4: LCC classes with longest avg word length

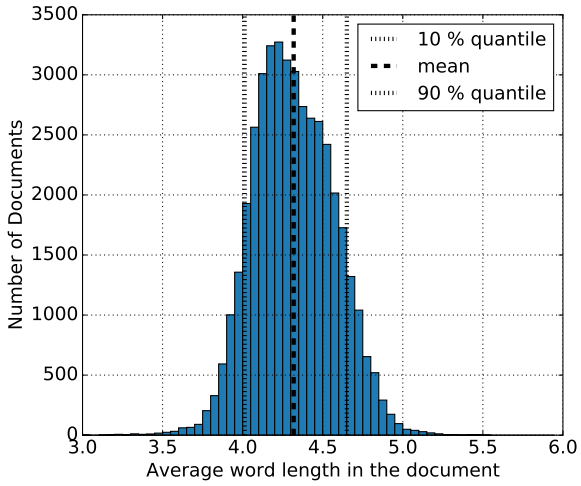

Figure 5.8: Average word length in the document

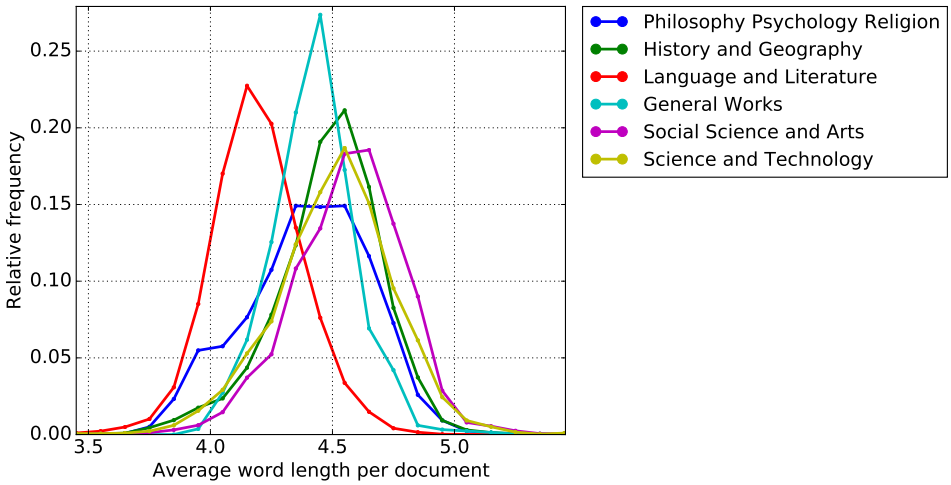

Figure 5.9: Average word length split by category

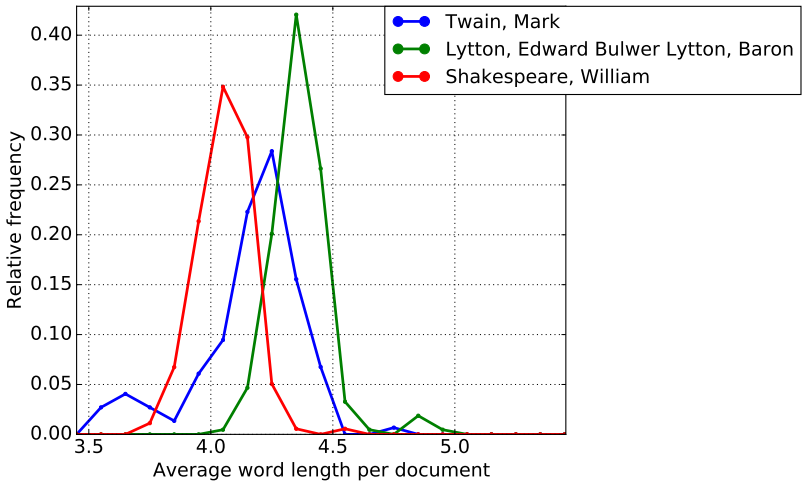

Figure 5.10: Average word length split by author

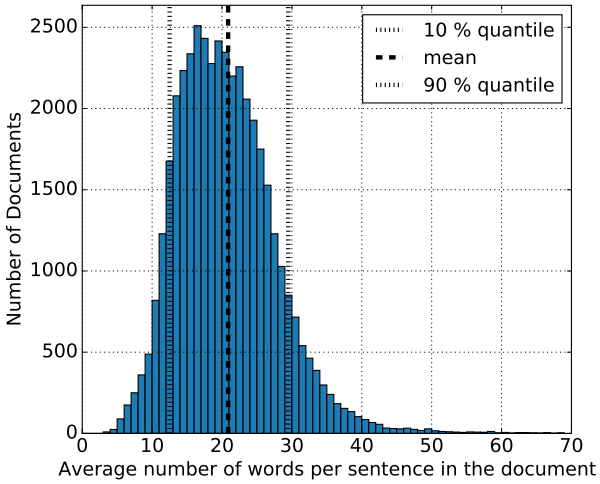

Figure 5.11: Histogram: Average sentence length per document

Figure 5.12 shows us the distributions of the average sentence length based split by the category. The category *Language and Literature* has its peak between 16 and 17 words per sentence. Other categories have all the peak between 22 and 24 word per sentence. The exception is the category *General Works* which has both of the maxima. It is caused by the periodicals written by *Various* authors. The documents with the LCC tag *AP – Periodicals* are very heterogeneous – from archived newspapers to humoristic stories.

Figure 5.13 shows the average sentence length for the three authors *Twain*, *Lytton* and *Shakespeare*. The distribution for Lytton and Twain looks almost identical with the highest frequency of sentence length ca. 20. On the contrary, Shakespeare's mean is only 11.9. The median is 10.3. The reason for the short average sentence length is the dramatic style of writing. Shakespeare writes the character's name which is speaking with full stop as in this example:

```
Hamlet.
To be, or not to be,--that is the question.
```

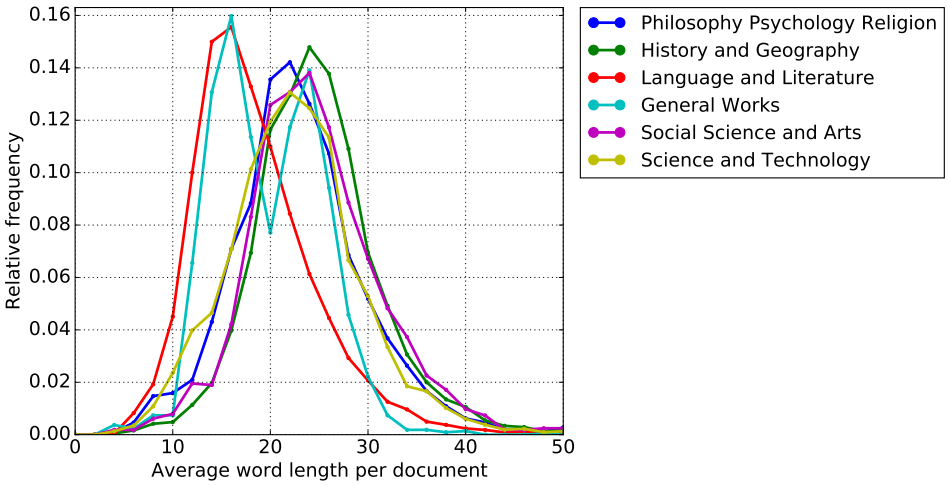

Figure 5.12: Average sentence length split by category

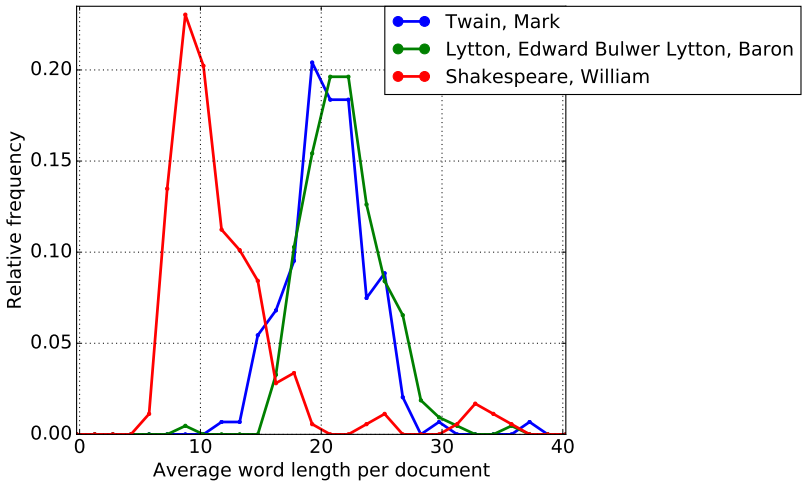

Figure 5.13: Average sentence length split by author

Stop words share

The next feature we explore is the share of stop words in the document. It is computed as the number of stop words divided by number of all words in the document. The stop words share distribution is shown in Figure 5.14. The average rate is 50.0 %, which means that every other word is a stop word. The median a bit higher at 50.8 %.

The stop words share can also help with identifying outlier documents. Documents which have low proportion of stop words are usually not written in English or don't use proper sentences. Documents with the stop words share (under 0.2) are from one of the following categories:

(a) Documents not written entirely in English. These were either misclassified by the PG catalog or big parts of the document consist of original untranslated book. The English stop words are then not found in the foreign language.

(b) Catalogs. They list facts without putting them into sentences. The documents have higher share of meaningful words than usual.

(c) Dictionaries. Similar to catalogs, dictionaries (e.g. English synonyms) don't contain whole sentences and therefore have low share of stop words.

(d) Documents with many lists or tables. In this case, the document contains many numbers or meaningful words which lowers the stop words share.

These documents can be seen in Figure 5.15, which shows the stop words share before filtering out the outlier documents. The local peak[8] near the value 0.16 is a rate typical for catalogs and dictionaries. There are also 19 documents with zero stop words share, which are all not in English. The highest share have the fiction books from LCC class *P* – 0.513. The lowest has the literature regarding Technology and Science with 0.445.

Part of speech (POS)

Distributions of different POS classes could be a good indicator of the text type. Apart from that, it enables further filtering of documents as well. The tokens in the text belong to one of the following POS classes:[9]

Based on the relative frequency rates of the POS classes, we can also filter out bad documents. The documents with high frequency of PROPN class are usually catalogs or lists of people. Documents with high frequency of PUNCT usually contain tables, illustrations or diagrams – the rate of PUNCT is high because of many brackets, dots and commas.

The Figure 5.16 shows the distributions of all POS classes. Most of the relative frequencies seem to be be normally distributed, however, we can observe several interesting phenomena:

[8]This peak can't be seen in the Figure 5.14 as that histogram shows only the 42 404 documents left for analysis.

[9]We use classes as defined in the module *spacy*. Punctuation is usually not considered a POS, but we include as it helps us to capture the usage of e.g. commas, exclamation marks or brackets.

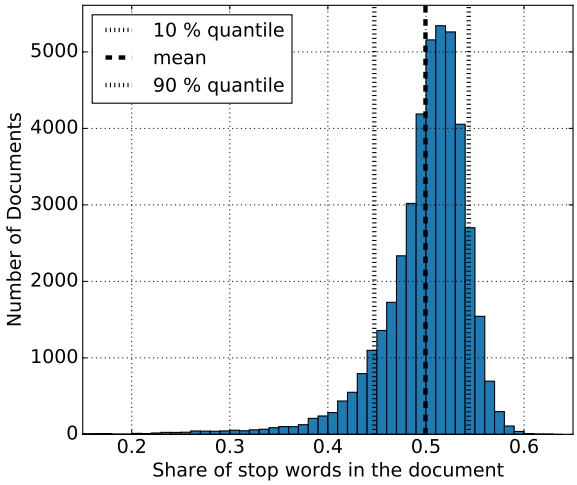

Figure 5.14: Stop words share per document (without filtered out docs)

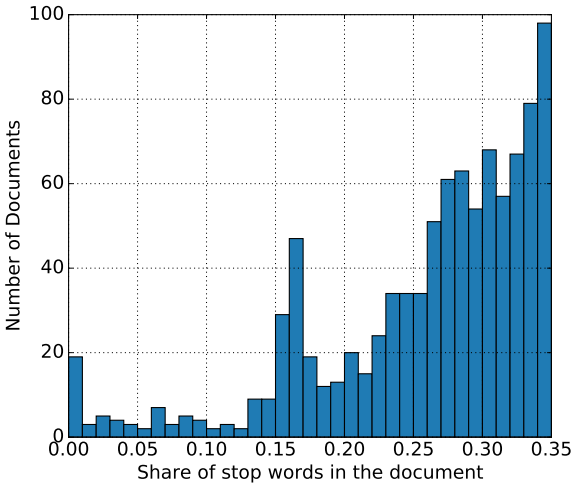

Figure 5.15: Stop words share (incl. filtered out documents)

(a) The relative frequency of proper nouns in the documents is skewed to the right – this might be caused by the corpus still containing some lists of names or catalogs. The POS tagger also takes the capitalization of words into account so if the author overuses capitalization, the tagger might misclassify some words as proper names.

(b) The numerals are also skewed to the right with some documents having over 5 % numerals in the text. LCC class *P* has the lowest rate (0.8 %). The highest rate have the documents regarding Science and Technology (3.3 %) followed by Geography as well as Naval and Military Science all around 2.5 %.

(c) The verbs are slightly skewed to the left. The highest verb share have the *P* documents with 16.5 % on average. All other categories have less than 15 % with minimum being Science with 12 %.

(d) The most extraordinary distribution out of the POS classes are the pronouns. There is no substantial peek. The most pronouns occur in the LCC class *P* – 6.9 %. It is significantly higher than the second category with most pronouns, *B* – *PHILOSOPHY, PSYCHOLOGY, RELIGION* with 4.7 %. The lowest pronoun share have the documents related to Science and Technology (2.5 %).

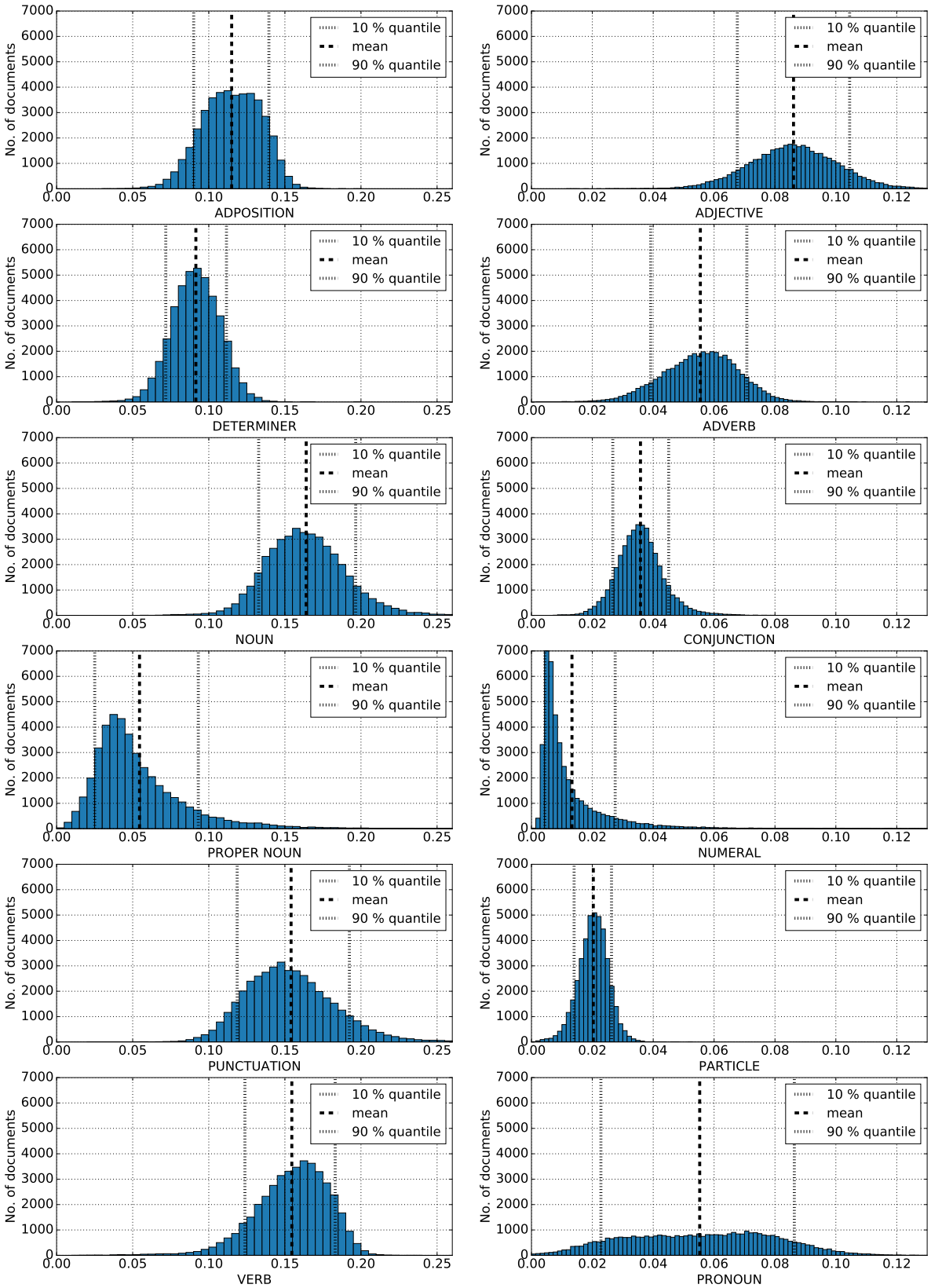

Figure 5.16: Distribution of POS classes

5.2 Feature Vectors

The feature vectors we are going to work with were already introduced in Section 3.2. They can be divided into two classes: text features, which we already explored in Section 5.1.3, and features relating to the frequence of a specific word or bigram. Mainly, we use the frequencies with tfidf term-weighting. In some parts of analysis or for some prediction models where it could make sense, we also try out the binary version of the word features. The binary word feature vectors just provide us with information if the word was available in the document, no matter how many times. Mainly, we work with the word feature vector of length 2 000 – carrying 2 000 most frequent (lemmatized) words without stop words. Extra features usually don't bring extra performance as the algorithms tend to overlearn on the document specific words.

5.2.1 Feature Vector Correlation

Some algorithms are more sensitive to the correlation of features than others. For the sensitive ones such as clustering with k-means, the highly correlated features can outweigh other features and determine the outcome. For example if we had following features:

- number of words in document
- number of sentences in document
- number of paragraphs in the document
- total size in kiB

All four features would probably be highly correlated as they all somewhat describe the document length. If we then compute the distance between feature vectors, the documents similar in length would be much closer to each other than documents with different lengths. To see if our features are highly correlated, we first do a principle component analysis and inspect if dimension reduction is useful. Then, we create correlation matrices for text features, tfidf word features and binary word features and explore the correlation of various pairs of words.

Principle Component Analysis(PCA)

The Principle Component Analysis shows us how the feature vectors are correlated overall. It transforms feature vectors into a lower dimensional space with maintaining the maximum original data variance. We look at text feature vectors, binary word feature vectors and tfidf feature vectors[10] and how they can be reduced with the PCA. Figure 5.17 shows us how many dimensions (relative to the original feature vector length) are needed to explain the certain amount of variance in the data. In other words, value 100 % means the feature vector can be reconstructed without any loss from the lower dimensional space. The rule of thumb is to take that many dimensions so that approx. 90 % of the original variance is kept.

[10]For both binary and tfidf feature vector, we consider the 2 000 most common words from the vocabulary, i.e., these feature vectors have length 2 000.

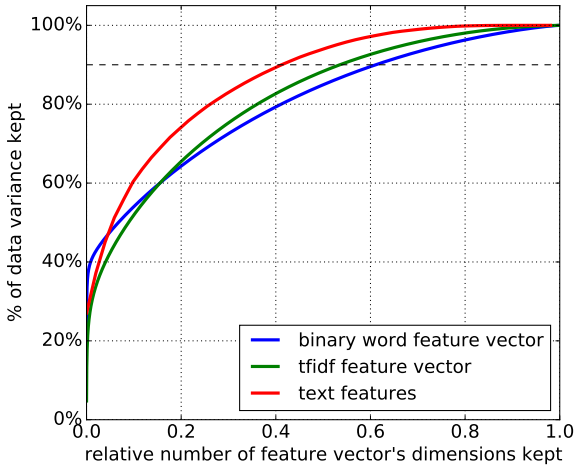

Figure 5.17: Variance explained based on the number of principle components taken (relatively to the original dimension)

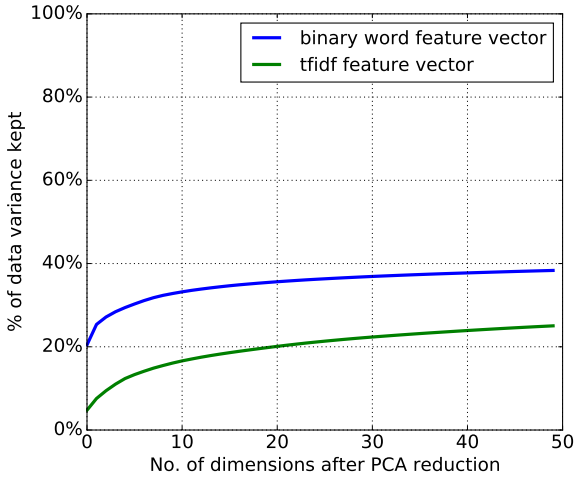

Figure 5.18: Variance explained showing the first 30 principle components for the binary and tfidf word feature vectors

All three feature vectors matrices can be transformed into space with ten times smaller dimension than the original one (relative dimensionality 0.1[11]) with keeping approx. 50 % of the original variance. To keep 90 % of variance, the matrix with text features has to be transformed from 51 to 21 dimensions (relative dimensionality 0.41). For the BOW features, the relative dimensionality for 90 % variance is 0.53 (1 060 dimensions) for tfidf and 0.61 (i.e. 1 220 dimensions) for binary word feature vectors.

Unfortunately, the popular 80-20 rule[12] does not apply here as 20 % of dimensions keep only about 65 % of the variance in the data. If we wanted to keep the 90 % of variance, which is usual practice, we would reduce the dimensionality of the word feature vectors from 2 000 to a space with more than 1 000 dimensions.

We can also see in the Figure 5.17 that there is around $30-40$ % variance kept for very few dimensions. Figure 5.18 shows the kept variance for the first 30 principle components for both word feature vectors. It can be seen that the first few principle components cover a lot of variance. The first component covers already 20% of the variance. The first principle components then correspond to words that occur at least ones in most of the documents. The first principle component for tfidf feature vector explains only 5% of the variance.

We could also visualize the data based on the first two principal components. It is already enough to distinguish books from the category *Language and Literature* from the category *History and Geography* as shown in Figure 5.19. When splitting vectors based on epoch, we can also visually distinguish individual epochs – see Figure 5.20. Baroque

[11]For example the value 0.1 on the x axis corresponds to 200 dimensions for the word features and 5 dimensions for the text features (10 % out of 51).

[12]It is also known as Pareto principle. It says that roughly 80 % of the effects come from 20 % of the causes. In this case, it would mean that 20 % of the vector dimensions explain 80 % of variance.

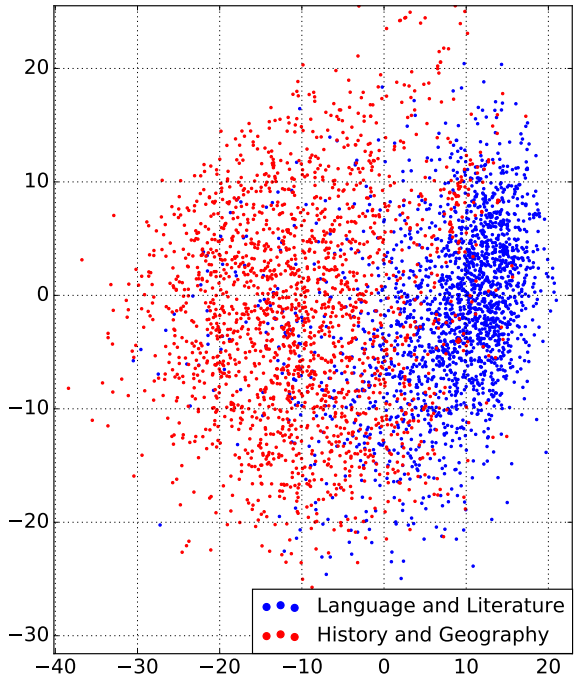

Figure 5.19: Tfidf feature vector with 2 000 words transformed to the first two principal components

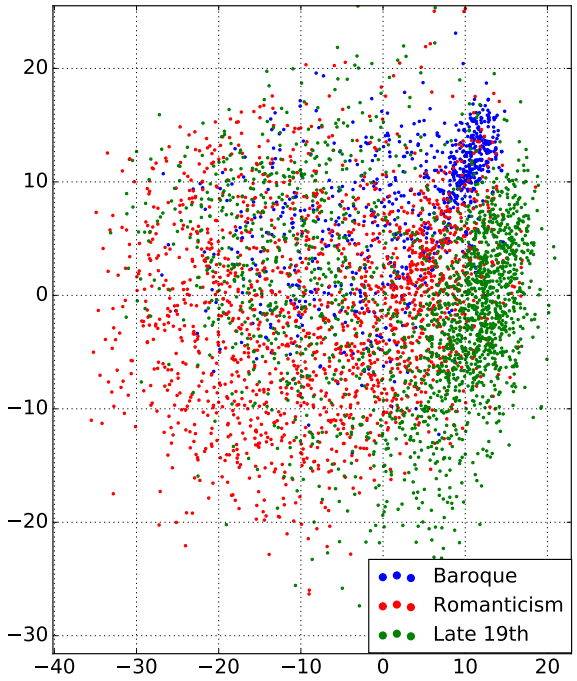

Figure 5.20: Tfidf feature vector with 2 000 words transformed to the first two principal components

literature and the literature from the end of the 19th century tends to stay both in an own cluster. On the contrary, for Romantic literature, we can't capture any pattern as the data points are spread quite evenly along both axes.

Through the PCA, we saw there are some correlations between features, even though not too high. In the following, we look at the correlation between individual features. The inspected feature vectors are once again: text feature vector, tfidf word feature vector and binary word feature vectors. For each feature vector, we analyze strong positive and negative correlations and show a correlation plot for them.

Correlation of Text Features

First, we look at the text features. In Figure 5.21, we can see the correlation between the majority of text features[13].

Observations – positive correlations:

(a) The features with highest correlation are those regarding quotation marks. It is not a big surprise that documents which start or end the sentence with a quotation mark have higher proportion of quotation marks in the whole document.

[13]For better clarity, out of the text features relating to individual POS tags, we included only the subclasses of verbs.

(b) The other highly correlated features relate to verbs and might not appear that intuitive – the higher the proportion of verbs in the document, the higher the proportion of pronouns.

(c) Documents with high frequencies of verbs tend to contain more stop words – this makes sense in context with the previous pair as pronouns are mostly on the stop word list.

(d) Verbs in past participle (VBN) appear more in documents with longer words.

(e) Adverbs are positively correlated mainly with verbs and pronouns.

(f) Last but not least, documents with high proportion of modal verbs have also high proportion of verbs in the infinitive form. This makes sense from the grammatical point of view as it is how are they used together.

Observations – negative correlations:

(a) Complementary features are usually negatively correlated – the higher the proportion of sentences ending with dot, the lower the proportion of sentences ending with question mark or exclamation mark. However, the proportions of sentences ending with *?* and *!* are positively correlated. The reason might be that they usually occur in the similar kind of documents – with lots of direct speech.

(b) Documents with high proportion of pronouns contain less verbs in the infinitive form (VB).[14]

(c) Documents with longer words have lower share of pronouns – this seems logical as the pronouns are usually very short

(d) The higher share of adpositions (ADP) class is negatively correlated with the frequency of punctuation. The reason could be that the class ADP contains, apart from prepositions, also some subordinating conjunctions, which are conjunctions connecting sentences – often without needing a comma (e.g. *as* or *if*).

Correlation of Word Features

Next, we explore the correlations between **tfidf features**. Figure 5.22 shows us how many pairs have the correlation coefficient lower than a given value. We sorted all 4 000 000 correlation pairs[15]. It means that if the curve goes through point [0.2,-0.035], 20 % of the correlation pairs have the correlation coefficient lower than -0.035. It can be seen that approx. 60% of correlation pairs have correlation coefficient higher than zero. However, only about top 0.5 to 1% are correlated so that we can speak about a real correlation. The same is true for bottom 0.5 to 1% and negative correlation.

[14]It doesn't contradict the high positive correlation of PRON with the whole VERB class, which we saw in **b**]. The VB tag is the proportion of infinitives out of all VERBS, not all words in the document.

[15]We inspect a vector with 2 000 features – $2000 * 2000 = 4000000$.

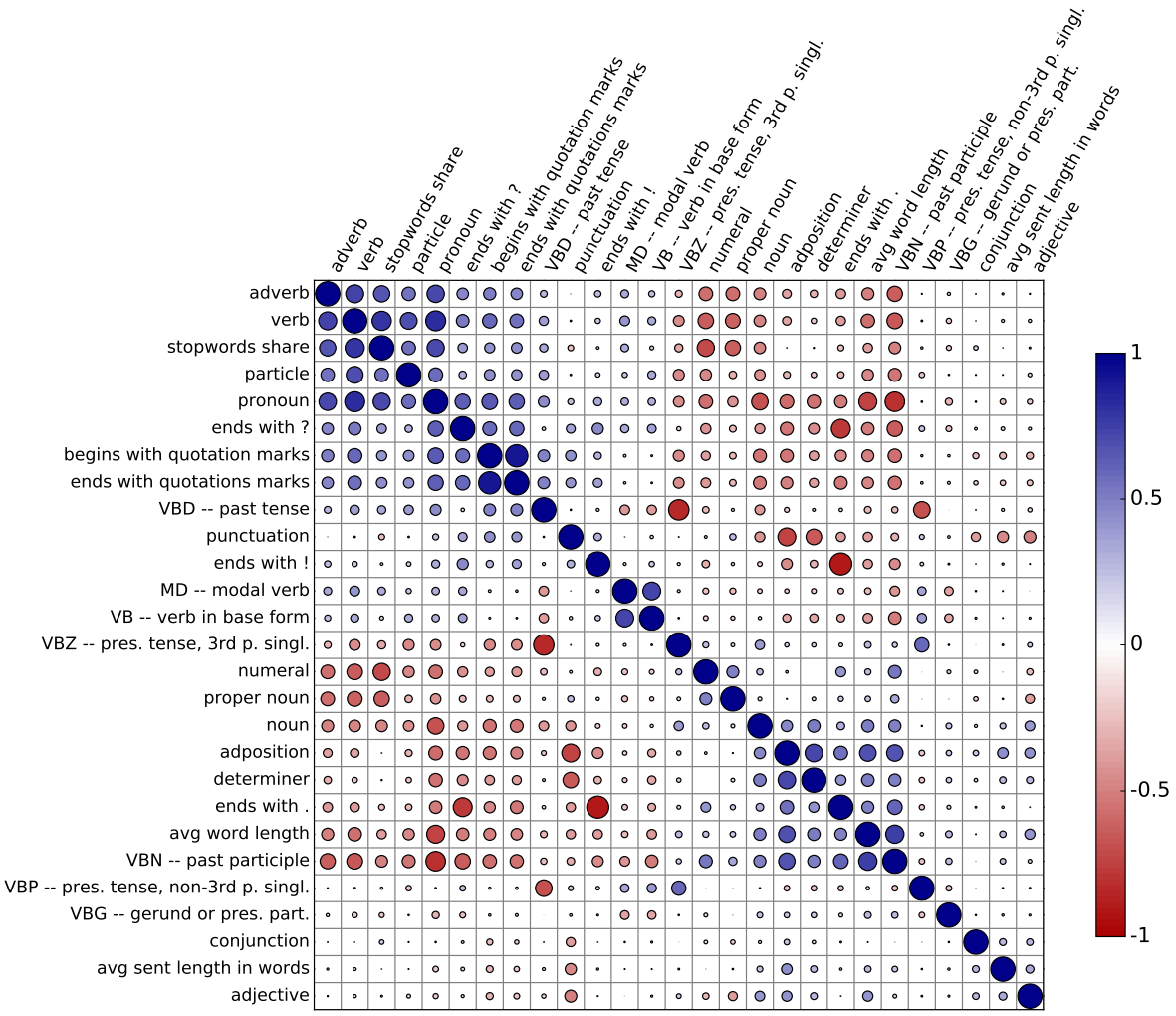

Figure 5.21: Correlation plot of text features

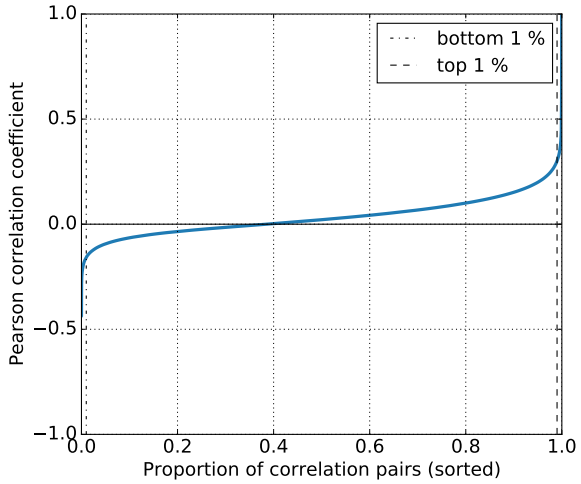

Figure 5.22: The portion of correlation pairs with the correlation coefficient value lower than y

As it is not possible to show the correlation matrix for all 2 000 features, we select the features which appear in the extreme pairs with strong positive or negative correlation and show the correlation plot for them (see Figure 5.23).

Observations – positive correlations:

(a) If bigrams are included, the most correlated features doesn't come as a surprise as the feature pairs usually consist of a bigram – a well known phrase or name – and unigram which is one of the two words in the bigram – example: *states* and *united* are both highly correlated with their combination being *united states* (0.98 for *united* ans 0.90 for *states*.

(b) Another group with high very high positive correlation are old English words. *Thee*, *thou* and *hast* are all correlated with coefficients between 0.85 and 0.90.

(c) Other group are pairs relating to the same entity – *me* and *my*, *she* and *her* or relating to country, *irish* and *ireland*

Observations – negative correlations:

(a) There are no bigrams among the highly negatively correlated pairs.

(b) The negative correlation pairs seem to somewhat capture the formality of the documents. On the one side, there are words such as *you*, *look* or *go*, which occur mainly in fiction literature. On the other side, words such as *thus* or *various* which mainly appear in the technical literature.[16]

[16]That's the reason we didn't include personal pronouns and verb *go* in stop words as it can greatly help with distinguishing fiction and technical or formal literature.

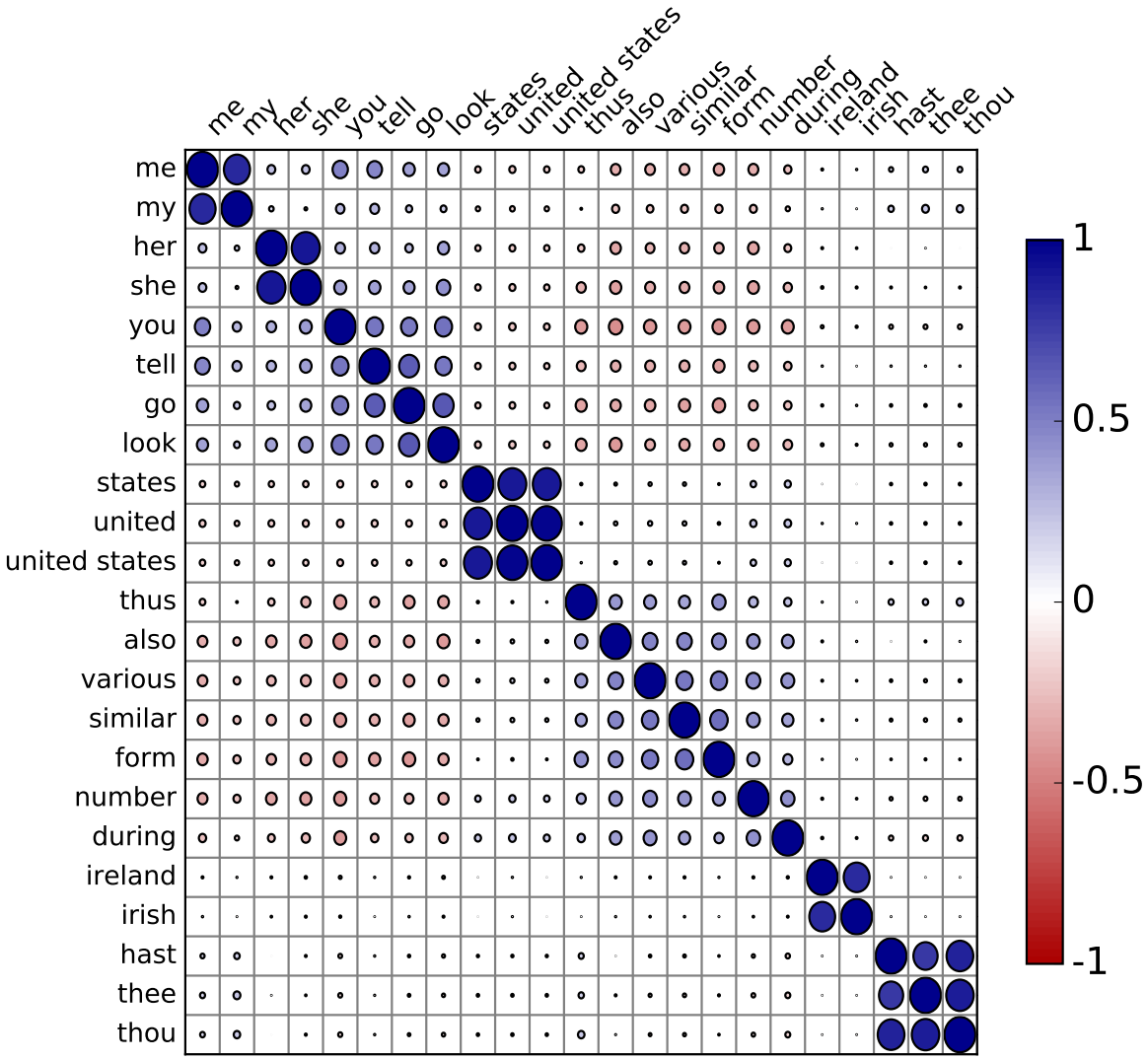

Figure 5.23: Correlation matrix for selected tfidf word features

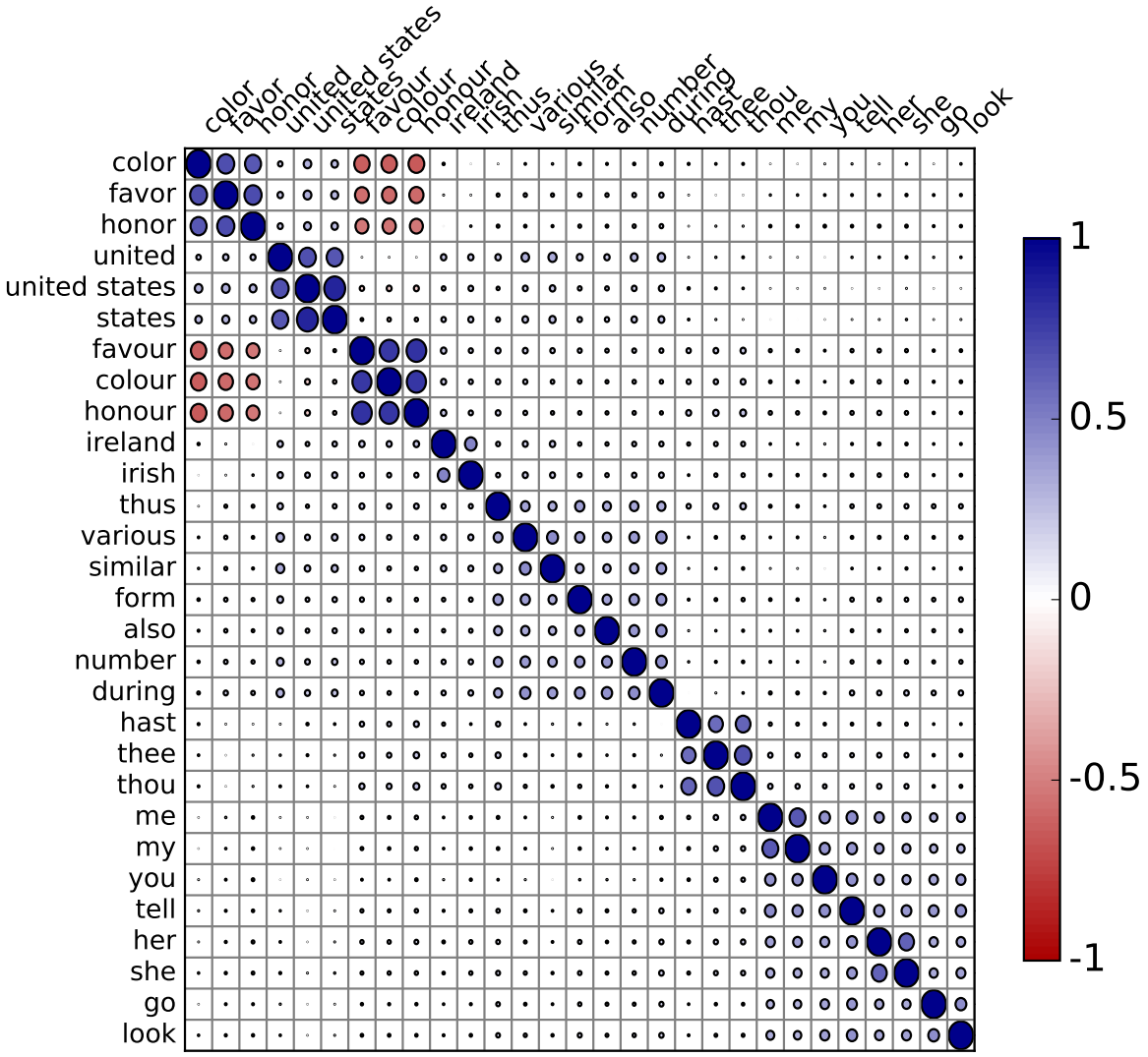

Figure 5.24: Correlation matrix for selected binary word features

If we plot the correlation matrix for **binary word features** (see Figure 5.24), most of the trends we saw in tfidf feature correlation stays the same. However, for words that have more ways of spelling, American and British (color vs colour, favor vs favour etc.), words spelled in British English are highly positively correlated with each other but highly negatively correlated with the American versions of words.

5.3 Document Clustering

When clustering, we want the feature vectors to be in lower dimensional space. However, as we saw in Section 5.2.1, the word features are not much correlated (to keep 90 % of the data variance, the transformed feature vector matrix would have around 60 % of the

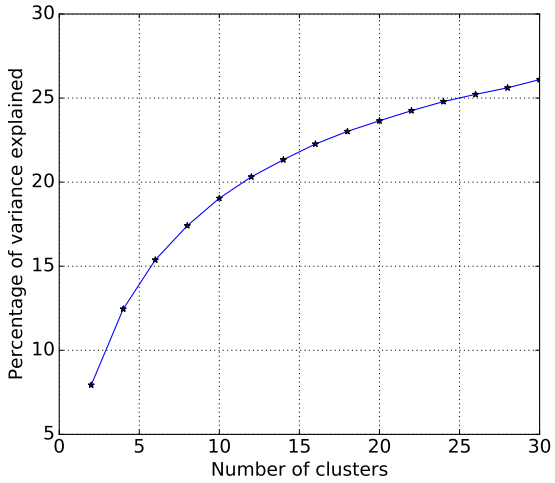

Figure 5.25: Variance explained based on number of clusters in the clustering –k (tfidf, 1 000 features)

original number of dimensions). To keep the dimensionality as small as possible, we take only 1 000 most frequent word features. Then, we transform them to 600-dimensional space defined by first 600 principal components. It is still a very high-dimensional space for clustering, but it is as much as we can get without losing too much information from the feature vectors.

We use the elbow method to determine the number of clusters. It is a visual method which chooses k so that higher ks don't bring that much extra explained variance. The Figure 5.25 shows the variance explained based on the number of clusters. We can see that the curve has no apparent elbow and is not curved a lot. The best k seems to be somewhere around 8 or 10. We try the k-mean clustering for $k = 8$.

5.3.1 Evaluating the Clusters

As the clustering in this case is a pure unsupervised task, it is hard to say if it actually performed good. We look at each cluster and the distribution of documents based on categories and epochs. The quality criteria would be then if the clusters are heterogeneous and differ in epochs and categories. For each cluster we show the matrix of occurrences where rows correspond to the text categories and columns to the epochs.

In the following, we list the clusters and provide what is typical for them or interesting about them:

cluster 0 - 3319 documents (7.8 %)

- majority class Language and Literature
- 5 times higher share of philosophy, religion, psychology
- 3 times higher share of ancient literature

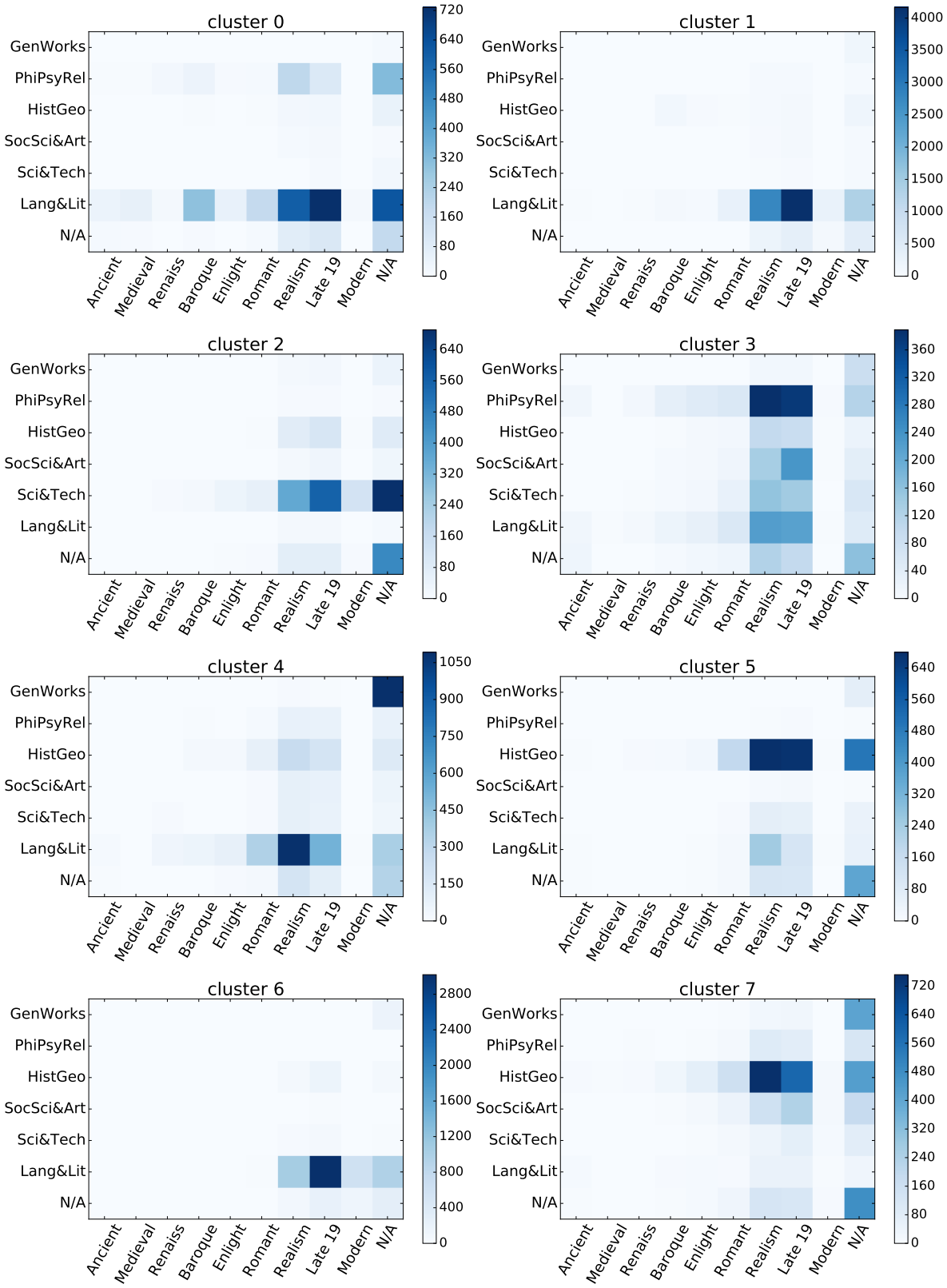

Figure 5.26: VisualizationVariance explained based on k (text features)

cluster 1 - documents 11437 (27 %)

- classical fiction books
- mainly end of the 19th century

cluster 2 - documents 2975 (7 %)

- science and technology
- 45 % the author's birth year not known

cluster 3 - 3914 documents (9.2 %)

- ld literature before the 19th century
- High coverage of biblical topics.
- high share of English and Germanic literature

cluster 4 - 4594 documents (10.8 %)

- encyclopedias, dictionaries, historical books
- law and political science

cluster 5 - 3405 documents (8 %)

- history and geography
- turn of 19th century

cluster 6 - 6952 documents (16.4 %)

- classical fiction books
- probably less complicated language (30 % children books)
- 19th - 20th century

cluster 7 - 5808 documents (13.7 %)

- general mixture of literature from 16th to 18th century
- 19 % periodicals, 36 % no author

Another interesting measure is to look if documents by one author are all clustered into one cluster or rather spread across multiple clusters. **William Shakespeare** seems to have very distinctive word choice[17] – clustering labels almost all of his books to the same cluster:

```
cluster 0 : 167 documents
cluster 1 :  10 documents
cluster 7 :   1 document
```

Edward Bulwer Lytton's work is bit more spread among the clusters, when compared to Shakespeare:

```
cluster 0 :  46 documents
cluster 1 :  67 documents
cluster 3 :   3 documents
cluster 4 : 102 documents
cluster 6 :   2 documents
cluster 7 :   6 documents
```

As Lytton wrote very diverse works, it makes sense that his documents are in multiple clusters. In cluster 0 are his historical novels and in cluster 7 are his history non-fiction documents about Athens and Ancient Greece. The rest of his work is distributed mainly between clusters 1 and 4 with cluster 1 covering rather domestic topic and family topics and cluster 4 novels.

Jane Austen wrote mainly novels focusing on social criticism. All her work by PG was assigned into one cluster:

```
cluster 1 : 13 documents
```

All in all, the k-means clustering on tfidf vectors with 1 000 most common words with 8 clusters delivers surprisingly reasonable results.

5.4 Predictive Model Evaluations

In this section, we create classification models and evaluate them. The document attributes we want to predict are mainly those explored in Section 5.1.2:

- author – domain in the order of thousands
- epoch – 9 classes
- category – 6 classes[18]
- topics – independent binary classifiers for each topic such as *Science Fiction*

[17] Mainly beacuse of old English words like *thy* or *thou*.

[18] One of the 6 aggregated LCC classes – Language & Literature, History & Geography, Science & Technology, Philosophy & Psychology & Religion, Social Science & Art or General Works.

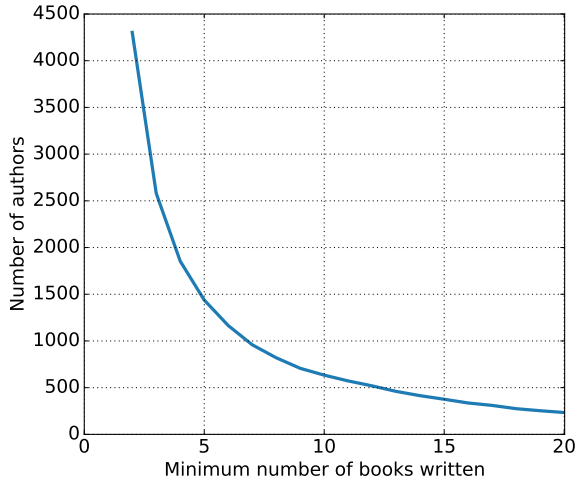

Figure 5.27: Number of authors who wrote n or more books

Each classification task needs a different approach. When creating classifier, we choose between two basic models – Naive Bayes Classifier and Decision Tree Classifier. Model parameters are chosen through 10-fold crossvalidation. Results of these basic predictive models can be then further improved by bagging – decision trees are combined into a Random Forest Classifier. However, the cost for higher accuracy by the Random Forests is losing the insight into the prediction and the reasoning behind it.

As some classes are very unbalanced, accuracy is not always the metric to be optimized. Instead, we use the macro F_1 score defined as:

$$F_1 = \frac{2PR}{P+R},$$

where P and R are precision and recall respectively averaged over all classes:[19]

$$P_{macro} = \frac{1}{|C|}\sum_{i=1}^{|C|}\frac{TP_i}{TP_i+FP_i}, R_{macro} = \frac{1}{|C|}\sum_{i=1}^{|C|}\frac{TP_i}{TP_i+FN_i}$$

5.4.1 Author Prediction

Prediction of the document author is a very challenging task. First, the author can write multiple types of literature - e.g. a poem and dictionary which makes it very hard to capture the uniform author style from both works. Apart from that, the author domain is much bigger than domain of any other attribute we are going to predict. In total, there are 14 211 authors in the Project Gutenberg. How the number of authors reduces when we filter out authors with less than n books can be seen in Figure 5.27. We decided to focus only on authors with at least 5 books in Project Gutenberg to be able to capture the author's style. This reduces the amount of authors to 1 438.

[19]TP and TN stand for true positives and negatives, FN means false negatives.

Figure 5.28 shows the comparison of the author prediction accuracy for the Naive Bayes Classifier with different feature vectors:

(a) Multinomial naive Bayes classifier with binary word features

(b) Gaussian naive Bayes classifier trained on tfidf features

(c) Gaussian naive Bayes classifier with all 51 text features

(d) Modus prediction – this model just predicts the most frequent label from the training set

We can see that all three models perform quite good if we take into account that random guess has accuracy less than 1 %.

The multinomial model (binary features) doesn't improve much after 5 000 features. The accuracy is the best for 8 000 features – 74 %. When increasing the number of features further, the accuracy starts to drop.

The Gaussian naive Bayes model with tfidf features performs surprisingly good for small lengths of feature vectors. The accuraccy grows when increasing the feature vector length – up to 57 % for 500 features. After that, the accuracy drops down to 20 % for 5 000 features. When adding more features, the accuracy grows slowly again. The reason is probably that the seldom author-specific words, such as character names and cities, are also included in the vocabulary.

Based on the figure, it seems that only very few tfidf features are needed to make a reasonable author prediction using Gaussian naive Bayes model. Accuracy for 100 features is 54 %. Surprisingly, based on even only 10 words, the classifier performs significantly better than random (mode) guessing. Based on tfidf scores of words *go*, *good*, *her*, *know*, *make*, *me*, *my*, *she*, *time* and *you*, the accuracy of the prediction is 12 times higher than the mode prediction.[20]

The author can also be predicted using decision trees on text features. A simple decision tree yields 24.1 % accuracy. Using random forest of such decision trees, we can boost the accuracy up to almost 70 %. The accuracy based on the number of trees in the forest can be seen in Figure 5.29. Although the accuracy still slowly grows for 1 000 trees, we didn't try forests with more trees as the training process gets very computationally intensive and around 1 GiB is needed for storing such a forest.

5.4.2 Epoch Prediction

In Section 3.1.1 we divided the documents in 9 groups based on the birth year of the author. We wanted to approximately capture the literary styles. The classes are very uneven – 47 % documents are in the Late 19th Century epoch and 36 % in Realism period.

Therefore, we optimalize the F_1 score instead accuracy – we were able to achieve accuracy of 80 % for random forest classifier, however, several epochs were almost ignored.

[20]The tfidf scores are computed out of the ten words in the vocabulary, ignoring all other words.

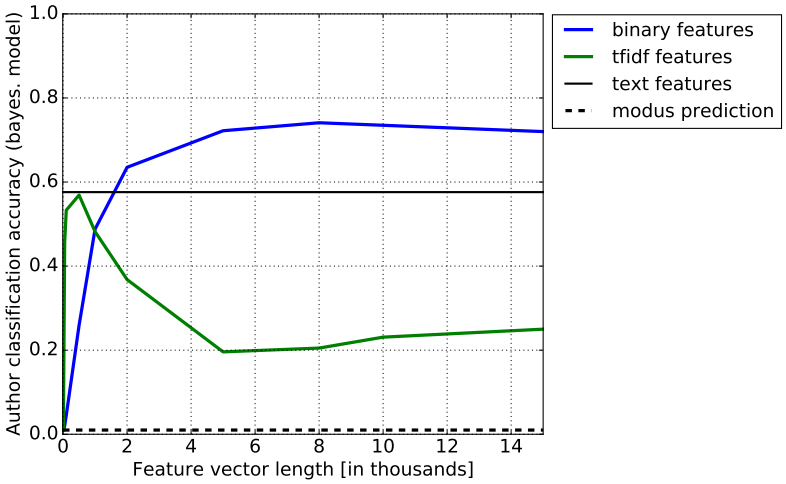

Figure 5.28: Accuracy of naive bayes model in author prediction (authors with at least 5 books)

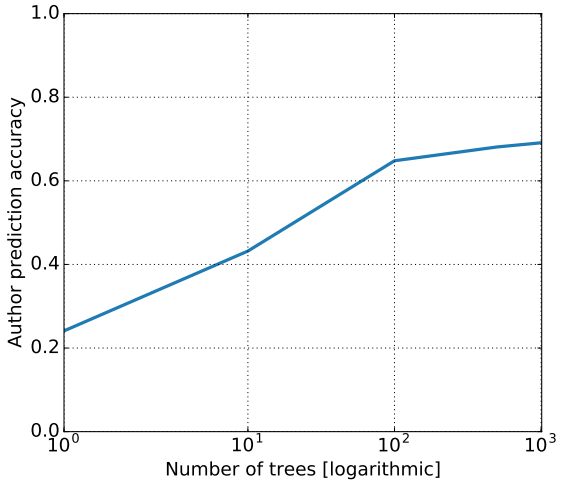

Figure 5.29: Accuracy of random forest with text features in author prediction (authors with at least 5 books)

Model	Feature vector type	Feature vector length	F_1 score
Mode guess	-	-	15.2 %
Decision Tree Classifier	Text Features	50	56.9 %
Random Forest (200 trees)	Text Features	50	58.6 %
Multinomial Naive Bayes	Binary BOW	15 000	46.1 %
Decision Tree Classifier	Binary BOW	15 000	48.3 %
Random Forest (200 trees)	Binary BOW	15 000	61.3 %
Gaussian Naive Bayes	Tfidf	15 000	54.3 %
Decision Tree Classifier	Tfidf	2 000	48.1 %
Random Forest (200 trees)	Tfidf (combined)	2 000	**65.6** %

Table 5.10: Era prediction F1 score

Using F_1 score, the best classifier was the random forest with tfidf-2000-combined features with F_1 score 65.6 % and accuracy 78.2 %. Complete table is shown in Table 5.10.

The confusion matrix can be seen in Figure 5.30. Medieval era was not recognized and classsified as realism. Romanticism was often misclassified to realism as well. However, these misclassifications are in this case not that bad as the epoch borders were created artificially and romanticism and realism co-existed together for a few decades.

Listing 5.7 shows the features that bring the highest contribution to the classes. It means the higher the value of the feature, the higher probability the document comes from the given class. We can definitely see the evolution of the language as words typical for Medieval epoch don't exist in the modern English anymore. The words typical for Enlightenment come from the philosophy and include rather complicated words, such as, circumstance or consequence.

5.4.3 LCC class prediction

The category prediction task has unbalanced classes again. However, the performance was better than for the epochs. The Random forest classifier achieved F_1 score of 81.6 % and accuracy 88.7 %. That is about three times higher than just blindly guessing the category *Language and Literature*.

Figure 5.31 shows the confusion matrix. All classes tend to predict about 10 to 20 % in favour of the major classes *Language and Literature* and *History and Geography*. The worst performance has the smallest class *Social Science and Arts*, which acheived the accuracy 62 %.

Features with the most contribution for each class are listed in Listing 5.8. We can see that all of them make sense in their context.

(a) Featers of General Works refer to literature in general (e.g. word *volume* or *reader*).

(b) History books use lots of past tense as well as military words, such as *troop* or *war*.

(c) For Language and Literature, the usage of quotation marks is very important, as other categories probably don't use quotation marks a lot. The word typical for

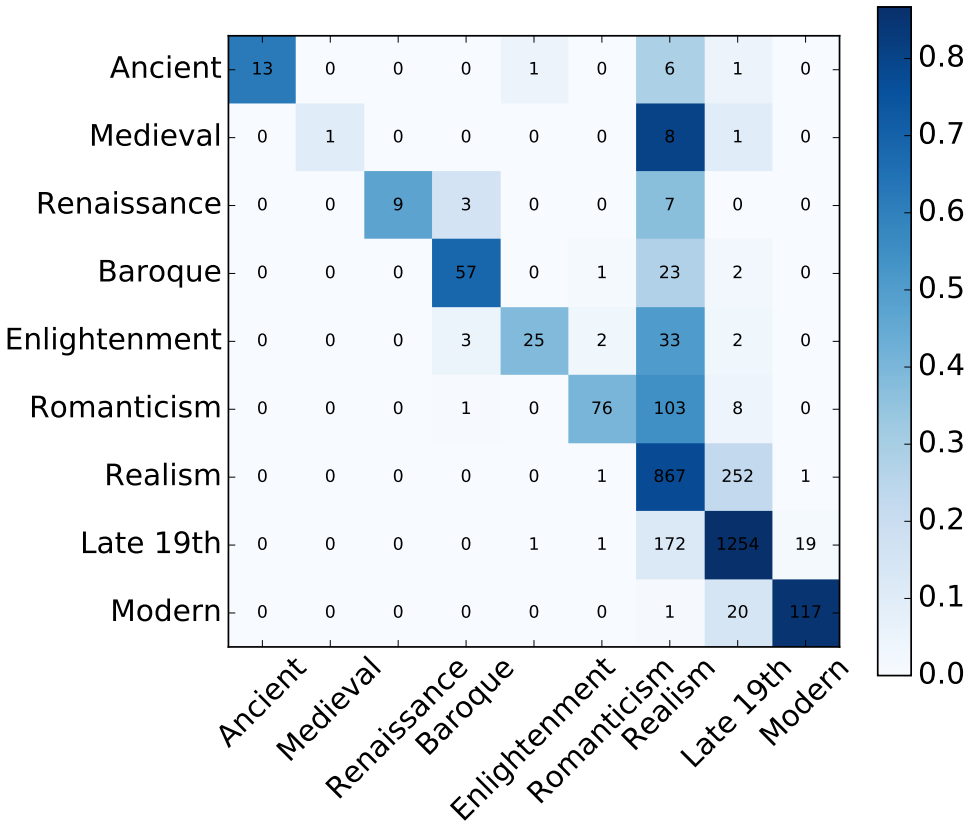

Figure 5.30: Confusion matrix for the epoch prediction – Random forest with tfidf-2000-combined features

```
Ancient: nor, whom, shall, greek, thus

Medieval: whence, thus, thou, whom, thee

Renaissance: avg sent length, comma proportion, don, nor, god

Baroque: my lord, shall, verbs in base form proportion, hath,
   sir

Enlightenment: oblige, situation, comma proportion,
   circumstance, consequence

Romanticism: feeling, circumstance, situation, oblige, comma
   proportion

Realism: week, excitment, adjective usage, england, look

Late 19th cent,: dot proportion, later, across, week, adjective
   usage

Modern: dot proportion, around, across, later, nod
```

Listing 5.7: Features with the highest contribution to the individual epochs – tfidf-2000-combined features

Model	Feature vector type	Feature vector length	F_1 score
Mode guess	-	-	27.6 %
Decision Tree Classifier	Text Features	50	49.9 %
Random Forest (200 trees)	Text Features	50	72.4 %
Gaussian Naive Bayes	Tfidf	15 000	68.1 %
Decision Tree Classifier	Tfidf	2 000	53.1 %
Random Forest (200 trees)	Tfidf (combined)	2 000	**81.6** %

Table 5.11: LCC class prediction accuracy

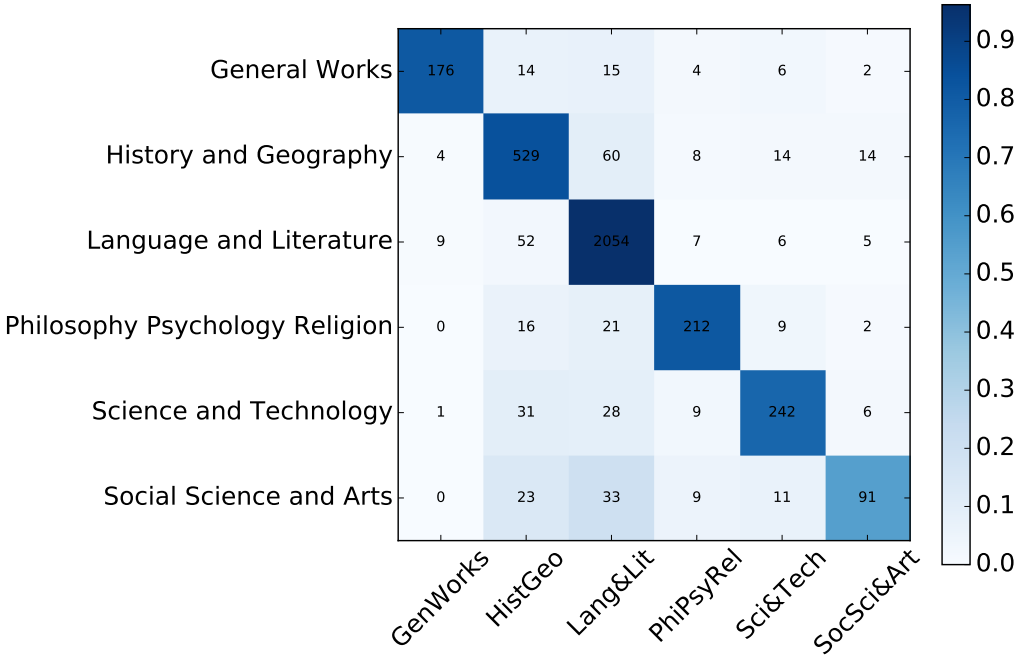

Figure 5.31: Confusion matrix for the category prediction – Random forest with tfidf-2000-combined features

this category are *smile* and *laugh*, as they perhaps don't occur in scientifical or historical literature.

(d) Philosophy, Psychology and Religion Category gets dominated by the religious vocabulary as the top 5 words somehow relate to the God.

(e) For Science and Technology, words relating to measures are the most important. These words include: *size*, *small* or *inch*.

(f) Social Science and Arts relate to the words such as *public*, *government* or *law*.

```
General Works: volume, vol, publish, reader, literary

History and Geography: verbs in past tense, troop, full stop
   proportion, war, people

Language and Literature: begins with ``, ends with ``, smile,
   quotation marks proportion out of punctuation, laugh

Philosophy, Psychology and Religion: god, christ, jesus, sin,
   holy

Science and Technology: size, full stop proportion, surface,
   small, inch

Social Science and Arts: public, government, full stop
   proportion, pay, law
```

Listing 5.8: Features with the highest contribution to the individual categories – tfidf-2000-combined features

5.4.4 Subject prediction

Next, we want to capture the main topics of the document. Each topic has an individual binary model telling the probability of the document covering the corresponding subject. To make the models more robust, we focus just on the subset of the whole dataset. The models are trained on belles-lettres only – we keep around 20 000 documents with following LCC classes:

- PR – English literature
- PS – American literature
- PQ – French, Italian, Spanish & Portuguese literature
- PT – German, Dutch & Scandinavian literature
- PZ – Fiction and juvenile belles lettres

The major challenge here is the imbalance in size between the positive and negative classes. For every topic, only few percent of documents cover it. We need to create a classifier so that it doesn't just predict the negative class everytime. One way to deal with imbalanced classes is to sample the train set to make it more even. Undersampling makes the train set balanced by selecting only some data points from the bigger class. Oversampling, on the contrary, takes data points from the smaller class several times to make both classes of comparable size. Another popular approach is the SMOTE[4] algorithm which combines undersampling and oversampling with creating synthetic data points for the smaller class. For the topic prediction, we use the random forest

Genre classifier	Genre frequency	Accuracy	F_1 score
adventure stories	1.8 %	95.9 %	62.3 %
love stories	1.2 %	93.4 %	58.5 %
short stories	4.3 %	96.5 %	74.8 %
historical fiction	1.3 %	97.9 %	65.5 %
poetry	2.1 %	97.3 %	72.3 %
drama	1.3 %	99.0 %	83.8 %
detective and mystery stories	1.4 %	98.9 %	78.9 %
science fiction	3.1 %	98.1 %	84.5 %

Table 5.12: Genre prediction F_1 score using Random Forests with tfidf-2000-combined features

classifier, which is prone to overlearn if the size of the individual trees is not too large. Therefore, the individual trees in the random forest will be just simple decision trees with significantly limited maximum depth – the best values were achieved for depth $10 - 20$ – with higher depth, the classifier started to overlearn on the bigger class.

In Table 5.12 we show the results for the genre classifiers. The success mainly correlated with the frequency of the genre[21] – if the genre was more frequent, the overlearn wasn't so bad, and thus the score better. The best performance achieved the classifier for the most frequent genre, science fiction, with $F_1 = 0.845$. Second best was the drama classifier with $F_1 = 0.838$, even though this genre is not the most frequent with only 1.3 % occurrence. The worst score had love stories with $F_1 = 0.585$.

If we mainly care about false negatives (genre present but not predicted) and false positive don't matter, we could predict the genre as present even with confidence less than usual 0.5. In the implementation, we deal with this problem by showing the probabilities directly (a probability by random forest classifier is the proportion of decision trees in the ensemble predicting the given label).

In Listing 5.9, the typical terms for each genre are shown. As the text features, relating mainly to the proportions of different punctuation symbols, were similar to all of them, we include this time only the word features. It can be seen that the individual words describe the genre as good as a human could. The most notable are the words *adventure* being the most typical for *adventure stories* and *police* being the most typical word for *detective and mystery stories* genre. As for text features, for *poetry*, the possesive *'s* POS tag was the most important feature. For *drama*, lots of dots, brackets and verbs in 3rd person are typical, which probably doesn't come as a surprise.

[21] Computed out of the belles-lettres only.

```
science fiction: around, star, evidence, space, control

adventure stories: adventure, rope, shout, shot, catch

love stories: her eye, she, she look, her, her hand

short stories: look, sit, face, eye, you

historical fiction: her hand, cry, voice, sword, speak

poetry: thy, song, sweet, breast, fair

drama: enter, let me, scene, my heart, pardon

detective and mystery stories: police, tell you, room,
   suspicion, door
```

Listing 5.9: Features with the highest contribution to the individual genres – tfidf-2000-combined features

5.5 Character Interactions

In Section 3.4, we already described all the steps needed to compute the interactions between characters. In the following, we visualize the results by showing a graph of interactions. We show the visualization for 3 books:

- The Adventures of Sherlock Holmes by Arthur Conan Doyle
- Harry Potter and the Sorcerer's Stone by J. K. Rowling
- A Storm of Swords[22] by George R. R. Martin

We analyze the latter two books even though they are not in PG. These books were chosen as they are quite popular and the character interactions make more sense to the reader if they know the books.

5.5.1 Character Interaction Visualization

Following graphics display the interaction strength between characters. The width of an edge corresponds linearly to the sum of interaction scores of the two characters. The size of a node states is linearly proportional to the number of occurrences of the character in the book – the occurrences and interactions are not necessarily related as a character can be mentioned a lot but never with other characters, conversly, some characters might occurred rather rarely but interact with many other characters.

[22] *A Storm of Swords* is the third novel of *A Song of Ice and Fire* fantasy series. The TV show *Game of Thrones* is based on this series.

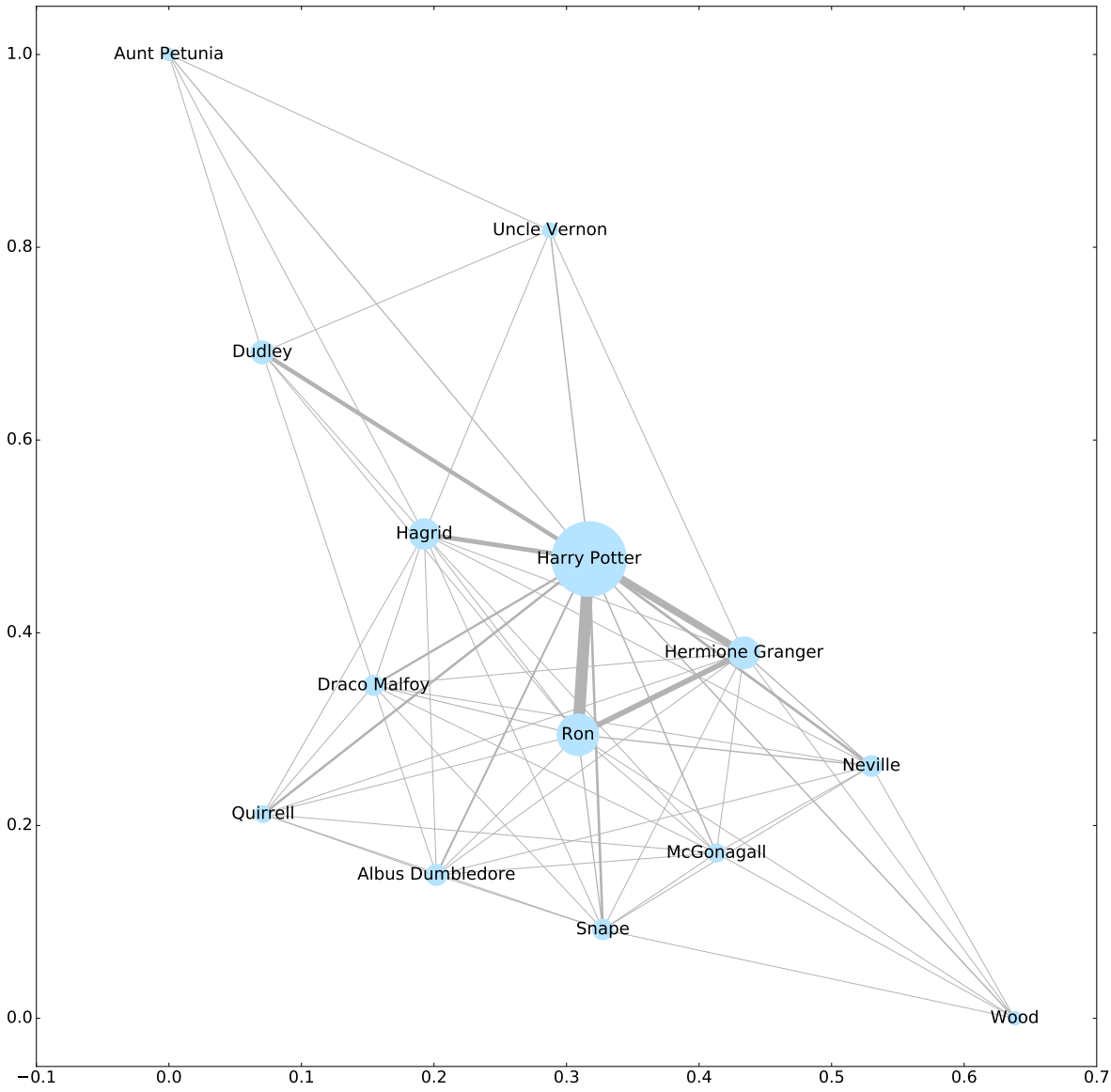

Figure 5.32: visualization of character interactions in the book Harry Potter and the Sorcer's Stone

Harry Potter and the Sorcer's Stone

The visualization of character interactions in Harry Potter and the Socer's Stone can be seen in Figure 5.32.

Observations:

- Harry is the main character. He interacts with every characters in the book.
- There is a strong triangle of interactions between Harry, Ron and Hermione.
- Harry's foster family Aunt Petunia, Uncle Vernon and Dudley interact only with Harry and make up therefore an own cluster.

The Adventures of Sherlock Holmes

The visualization of character interactions in the book The Adventures of Sherlock Holmes can be seen in Figure 5.33.

Observations:

- Sherlock is the main character. He interacts with everyone.
- The majority of Sherlock's interactions is with Watson. However, Watson has very few interaction with someone else (except Sherlock).
- The reason for very few interactions between other characters (Sherlock's *clients*) is that the book consists of several independent stories. The client is different in each story.

A Storm of Swords

The visualization of character interactions in the book A Storm of Swords can be seen in Figure 5.34.

Observations:

- There is no clear main character as in the previous two books.
- As there are multiple parallel storylines in the book, we can see several clusters which refer to each of them – an example might be the cluster in the upper left corner consisting of Ygritte, Jon, Mance, Sam and Grenn who all interact strongly with Jon (Snow).
- Robb Stark is somehow the main character as everyone interacts with him, however, he is not the dominant person in the number of occurrences in the book.
- The character *Jon Snow* occurs usually only as *Jon*. However, there are also other people called *Jon* in the book. That means if the parameter for minimum occurrence of the super name is not high enough, the algorithm doesn't know to which *Jon* does the name refer. As a result, there would be two character names *Jon* and *Jon Snow* even though they are the same character.

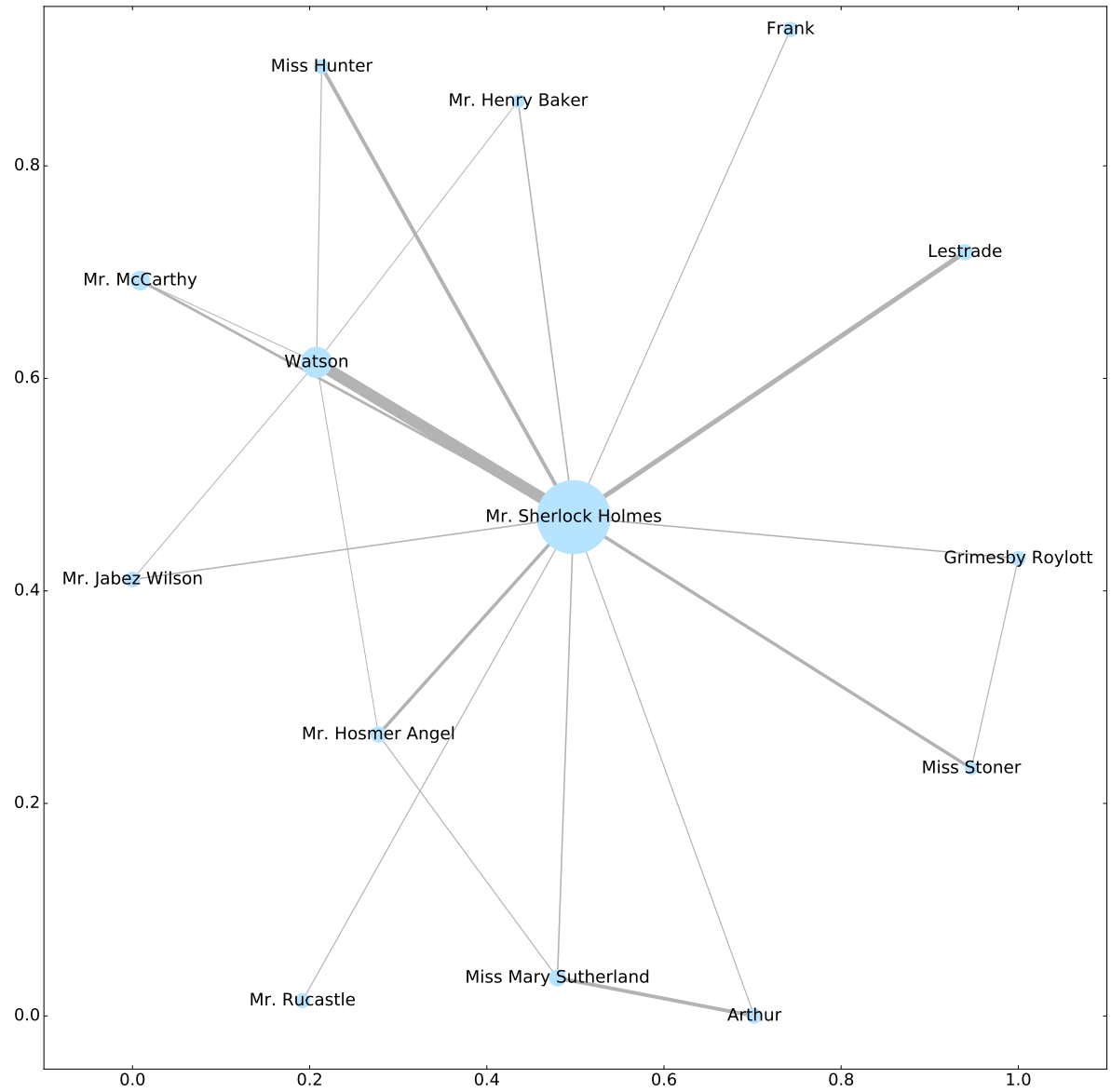

Figure 5.33: visualization of character interactions in the book The Adventures of Sherlock Holmes

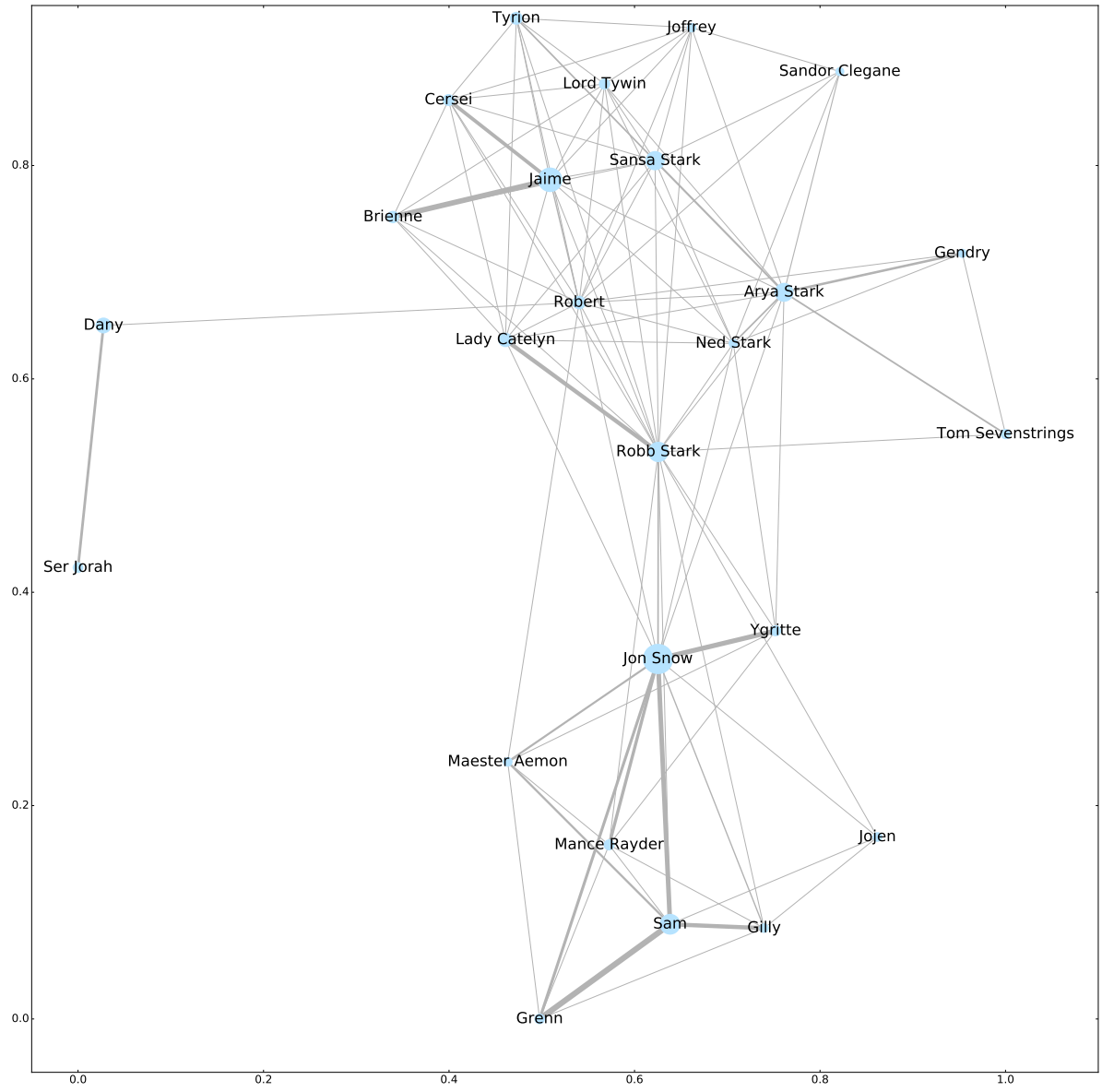

Figure 5.34: visualization of character interactions in the book A Storm of Swords

5.6 Performance Analysis

In this section, we evaluate the performance of the processing pipeline. We dicuss if the step can be done parallel to speed the process up. We also compare the runtimes for different document sizes and the whole document consisiting of 42 404 documents of the total size of 15.8 GiB. Unless stated others, the tasks were serial running one CPU with 4 cores @ 2.6GHz.

Preparing the Data

Downloading and preprocessing the catalog takes 117 seconds and is needed just ones, or when the update to the new nightly built is desired. Fetching the documents depends purely on the net throughput as we need to download roughly 16 GiB of data.

The preprocessing task removing headers and doing minor formatting changes takes 90 minutes for the whole dataset. It can be paralelized. The execution times grows approximately linearly with the size of the document. It is shown in Table 5.13:

Document size in kiB	time
100 kiB file	.050 s
500 kiB file	0.170 s
1000 kiB file	0.349 s
whole dataset (16 GiB)	5428 s

Table 5.13: Document preprocessing run time

Tokenization, Lemmatization, POS Tagging

The process can be paralelized and the execution time grows approx. linearly with the size of the document. The total run time of the tokenization, lemmatization and tagging step was approx. 6.2 hours as stated in Table 5.14.

Document size in kiB	time
100 kiB file	.198 s
500 kiB file	0.711 s
1000 kiB file	1.408 s
whole dataset (16 GiB)	22 386 s

Table 5.14: Tokenization, lemmatization and preprocessing run time

Computing the Count Matrix

In this step, we compute the frequencies of terms in the documents across the whole corpus. We use the CountVectorizer from the module *sklearn*. This module holds during

the computation the whole vocabulary in the memory, which might be costly, when using bigrams. That is the reason why the performance is not linear as can be seen in Table 5.15. However, the count matrix could be paralelized and computed without using CountVectorizer using spark, for example.

Document size in kiB	time
1 000 documents	2.3 min
10 000 documents	48 min
all 42 404 documents	384 min

Table 5.15: Computing matrix of term occurrences run time

Training Random Forest (200 trees)

Training the classifier which was often used in the predictions – random forest classifier with 2051 features and 200 trees with max tree depth and variable with 6 classes, took in average 23.5 s (including splitting the set to train and test). However, we had theoretically up to 48 cores available. When using single core, the execution time was 8.1 minutes.

Conclusion

The task which took longer time had to be all done only once. Therefore, spark is not necesarrily needed, even though it would decrease the preprocessing time.

6 Summary

6.1 Summary and Conclusions

In this thesis, we explored the books in Project Gutenberg and created a model which classifies a text based on author, year and category. The model also estimates the topics covered by the book. In the following, we list the observation we made throughout the exploration and prediction process.

Exploration

For the majority of observed features, the class *P – Language and Literature*, representing mainly fiction books, behaved differently than other types of literature. This category had shortest average words and sentences as well as the highest proportion of stop words or usage of question marks and exclamation marks. It confirms what one would expect –that these books are written in a more simple language than technical or scientifical texts.

Proportion of stop words is a very useful measure. Based on the proportion, we can determine if the book is written in the given language or if it contains proper sentences. Using stop words, we were able to filter out many documents consisting of list or dictionary entries, which were not the are of interest for this work.

Word feature vectors are not much correlated – to maintain 90 % of the data variance using PCA, the number of dimensions has to be around 60 % of the original feature vector length.

Predictions

The author classification was surprisingly successful. Even though we had over 1 400 authors, the accuracy of the classifier was 74 %. The best performance reached the naive Bayes model with binary features, which captures only if the word occur in the document or not. Using random forest classifier with the 10 most common words (excl. stop words), we could predict the author significantly better (12 %) than the random guess would (1 %).

Usually, naive Bayes classifier performed better than decision trees, where it is hard to find the balance between underlearning and overlearning. However, random forest classifier consisiting of small decision trees outperformed the naive Bayes classifier for all tasks except for author prediction, where NB classifier was 5 % better.

Decision trees and models based on them don't profit a lot from the tfidf term weighting. The reason behind that is that decision trees look at each feature individually. Therefore,

it doesn't matter if the feature value domain was transformed.

For different predicted metadata, we showed the most important features for the given prediction. It gives us an idea of what is typical for the given epoch and literature style. For example, the typical word for the genre *adventure stories* turned out to be the word *adventure*. For the detective stories, it was the word *police*.

6.2 Future Work

The whole analysis was performed only on *English* books. This saved a lot of work, because the English grammar is not as complicated as in other languages. Moreover, most of the natural language processing research was done related to English language. As a result, the lemmatization and POS tagging processes of English texts have satisfactory results, which is not the case for other languages. However, the analysis of books in other languages could be very interesting – it would provide us with some extra knowledge of the language specifics.[1]

As the corpus we worked with was quite heterogeneous, it might be worth it to focus only on a small part of it, e.g., poetry or historical novels, and do more in-depth analysis. We did this partially as we extracted the character interactions or trained the models for different topics. Instead of taking all books and filtering out the not interesting ones, which is how we proceeded, one could start with an empty set and include only documents having some properties. This would provide much cleaner data for the cost being smaller book working set.

The bottleneck of the feature extraction process was the *CountVectorizer*, which had to keep the whole vocabulary including all bigrams in the memory during the term counting process. Using parallelization (e.g. spark), the top n terms from the corpus could be computed using simple word counts. However, as this process had to be done just once, it was not necessary.

There are other promising approaches for representing text data. It would be interesting to see if other features, such as word2vec or doc2vec, give better results and new insight.

[1] We saw some grammar specific results when analyzing distributions and correlations of POS.

Bibliography

[1] Simona Balbi. Beyond the curse of multidimensionality: High dimensional clustering in text mining. *Statistica Applicata*, 22(1):53–63, 2010.

[2] Tom Betts, Maria Milosavljevic, and Jon Oberlander. The utility of information extraction in the classification of books. In *Proceedings of the 29th European Conference on IR Research*, ECIR'07, pages 295–306, Berlin, Heidelberg, 2007. Springer-Verlag.

[3] Leo Breiman and E. Schapire. Random forests. In *Machine Learning*, pages 5–32, 2001.

[4] Nitesh V. Chawla, Kevin W. Bowyer, Lawrence O. Hall, and W. Philip Kegelmeyer. Smote: Synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, 16:321–357, 2002.

[5] Project Gutenberg. Information About Robot Access to our Pages. `http://www.gutenberg.org/wiki/Gutenberg:Information_About_Robot_Access_to_our_Pages`.

[6] Project Gutenberg. Motion Pictures of the Apollo 11 Lunar Landing. `http://www.gutenberg.org/ebooks/116`.

[7] Project Gutenberg. The Complete Project Gutenberg Catalog. `http://www.gutenberg.org/wiki/Gutenberg:Feeds#The_Complete_Project_Gutenberg_Catalog`.

[8] Michael Hart. Free ebooks by Project Gutenberg. `http://www.gutenberg.org/`.

[9] Michael Hart. The History and Philosophy of Project Gutenberg. `http://www.gutenberg.org/wiki/Gutenberg:The_History_and_Philosophy_of_Project_Gutenberg_by_Michael_Hart`, 1992. [Online; version from 7-April-2010].

[10] Scott Hemphill. Pi to 1,000,000 places. `http://www.gutenberg.org/ebooks/50`.

[11] Karen Spärck Jones. A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation*, 28:11–21, 1972.

[12] Marie Lebert. *Project Gutenberg (1971-2008)*. 2008. `http://www.gutenberg.org/ebooks/27045`.

[13] Christopher D. Manning. Part-of-Speech Tagging from 97% to 100%: Is It Time for Some Linguistics? 2011.

[14] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781, 2013.

[15] David Nadeau and Satoshi Sekine. A survey of named entity recognition and classification.

[16] The Library of Congress. About the Library. `http://www.loc.gov/about/`.

[17] The Library of Congress. Library of Congress Classification Outline. `http://www.loc.gov/catdir/cpso/lcco/`.

[18] The Library of Congress. Library of Congress Subject Headings. `http://id.loc.gov/authorities/subjects.html`.

[19] University of Pensylvania. Penn Treebank P.O.S. Tags. `https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html`.

[20] Martin F .Porter. An algorithm for suffix stripping. *Program*, Vol. 14 Issue: 3, pp.130 - 137, 1980.

[21] Mehran Sahami, Susan Dumais, David Heckerman, and Eric Horvitz. A bayesian approach to filtering junk e-mail, 1998.

[22] Dr. Afşar Saranli, Stuart Russel, and Peter Norvig. Artificial intelligence: A modern approach, 2nd ed.

Eidesstattliche Versicherung

Hiermit versichere ich an Eides statt, dass ich die vorliegende Arbeit im Studiengang Msc. Informatik selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel – insbesondere keine im Quellenverzeichnis nicht benannten Internet-Quellen – benutzt habe. Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen entnommen wurden, sind als solche kenntlich gemacht. Ich versichere weiterhin, dass ich die Arbeit vorher nicht in einem anderen Prüfungsverfahren eingereicht habe und die eingereichte schriftliche Fassung der auf dem elektronischen Speichermedium entspricht.

Ort, Datum	Unterschrift

Veröffentlichung

Ich bin damit einverstanden, dass meine Arbeit in den Bestand der Bibliothek des Fachbereichs Informatik eingestellt wird.

Ort, Datum	Unterschrift