FAKULTÄT FÜR !NFORMATIK

Faculty of Informatics

Language Model Driven Analysis

Simplifying text on an individual scale

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering & Internet Computing

eingereicht von

Alexej Strelzow, BSc.
Matrikelnummer 0926103

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Dr. Allan Hanbury
Mitwirkung: Dr. Mihai Lupu

Wien, 19. September 2016

Alexej Strelzow   Allan Hanbury

FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Language Model Driven Analysis

Simplifying text on an individual scale

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Software Engineering & Internet Computing

by

Alexej Strelzow, BSc.
Registration Number 0926103

to the Faculty of Informatics

at the TU Wien

Advisor: Dr. Allan Hanbury
Assistance: Dr. Mihai Lupu

Vienna, 19th September, 2016

Alexej Strelzow   Allan Hanbury

Erklärung zur Verfassung der Arbeit

Alexej Strelzow, BSc.
Heitzles 20/2, 3623 Kottes-Purk

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 19. September 2016

Alexej Strelzow

Acknowledgements

First and foremost, I would like to thank my supervisor Dr. Mihai Lupu, who has always supported and motivated me when my back was against the wall. Thank you for your guidance, great suggestions, extensive proof reading and kindness. I am deeply grateful to have worked with you and very satisfied with what we have accomplished.

I also want to thank Professor Hideaki Takeda from the National Institute of Informatics (NII) of Japan, who took me in as a research intern in summer 2014. Under your supervision, I was able to get in touch with research areas and data sources that have lead to this very work. Thank you for letting me explore those and for the very comprehensive frequent word list that you purchased on my behalf.

Furthermore, I want to express my deepest gratitude to my fiancée Yoko. You have always supported me in this very time consuming endeavor and made me your first priority. Thank you for your understanding, the delicious food you put on the table and your patience.

Finally, I want to thank everybody who supported me, especially my parents.

Abstract

The goal of this thesis is to provide a tool that individually supports people (users) to comprehend relatively challenging textual resources like a researcher's published papers. Therefore, based on a user's document collection, we introduce a novel approach to detect words in a new document he or she might be reading that are most likely to be unknown to the user. Furthermore, we explain those words by utilizing external data sources. Our tool visualizes the analyzed document page by page and provides the user with a list of detected, possibly unknown words and their meaning with respect to the currently viewed page.

We implemented a proof of concept application to generate language models (user and document models) from text, compare them with each other, and provide an explanation of the words identified as unknown. The user model is an abstraction of the user's language skills in terms of known vocabulary. We estimate this set of known words by considering the user's written documents as a domain specific component and a very comprehensive frequent word list of contemporary American English as a general component. The model comparison algorithm takes a user and a document model as input and identifies possible unknown words based on semantical and statistical methods. To explain the words that are considered to be unknown to the user, we use BabelNet, a large semantic dictionary.

To validate our approach, first, we have created a test set of user and document models and second, conducted quantitative and qualitative experiments based on them. The underlying document collection of our user models has been identified using the DBLP computer science bibliography, a database for bibliographic metadata. In our conducted experiments, we compare user models from the domain of computer science (CS) with document models from the domains of CS (equi-domain experiments) and medicine (cross-domain experiments). We obtained the medical journal articles from PubMed, a meta-database for the area of biomedicine.

After 360 completed experiments (180 for each domain), we witnessed that on average almost twice the amount of unknown words have been found in documents from the medical domain in contrast to documents from the CS domain. Furthermore, in contrary to the equi-domain experiments, the cross-domain experiments revealed that the majority of unknown words were domain specific words and not general terms. We also revealed a negative correlation (Kendall's τ = -0.82) between the estimated language level of the user and the sum of detected unknown words with respect to the user.

Kurzfassung

Das Ziel dieser Arbeit ist es ein Tool bereit zu stellen, das Personen (Benutzern) beim Verstehen von schweren Texten, z.B. Publikationen eines Forschers, individuell unterstützt. Basierend auf einer Sammlung von Dokumenten des Benutzers stellen wir eine neuartige Methode vor Wörter in einem neuen Dokument, welcher er oder sie lesen möchte, zu finden, welche dem Benutzer mit großer Wahrscheinlichkeit unbekannt sind. Außerdem erklären wir diese Wörter mit Hilfe von externen Datenquellen. Unser Tool stellt das analysierte Dokument Seite für Seite dar und zeigt dem Benutzer eine Liste von gefundenen, unbekannten Wörtern und deren Erklärung zu der jeweiligen Seite.
Wir implementierten einen Prototypen, welcher Sprachmodelle (Benutzer- und Dokumentenmodelle) aus Text generiert, diese miteinander vergleicht und die als unbekannt befundenen Wörter erklärt. Das Benutzermodell ist eine Abstraktion von seinen Sprachkenntnissen im Sinne von bekanntem Vokabular. Wir schätzen die Menge an bekannten Wörtern mit Hilfe der geschriebenen Dokumenten vom Benutzer (domänenspezifische Komponente) und einer sehr umfassenden Wortliste, bestehend aus zeitgenössischem amerikanischen Englisch (generelle Komponente). Der Algorithmus, der die Modelle vergleicht, nimmt die Sprachmodelle als Eingabe und identifiziert mögliche unbekannte Wörter basierend auf semantischen und statistischen Methoden. Um unbekannte Wörter erklären zu können verwenden wir BabelNet, ein großes semantisches Wörterbuch.
Um herauszufinden ob unsere Vorgehensweise funktioniert haben wir zuerst ein Testset aus Sprachmodellen erstellt und danach quantitative und qualitative Experiment durchgeführt. Dokumente für die Generierung von Benutzermodellen stammen von der DBLP Computerwissenschaftsbibliographie, einer Datenbank für bibliographische Metadaten.
In unseren Experimenten haben wir Benutzermodelle (aus der IT Domäne) mit den Dokumentenmodellen aus den Domänen der IT (equi-domain Experimente) und Medizin (cross-domain Experimente) verglichen. Die medizinischen Journalartikel stammen aus PubMed, einer Metadatenbank der Biomedizin. Nach 360 Experimenten (180 pro Domäne) haben wir beobachtet, dass im Durchschnitt beinahe zwei Mal die Menge an unbekannten Wörtern in Dokumenten aus der medizinischen Domäne, im Kontrast zu Dokumenten aus der IT Domäne, gefunden wurden. Im Vergleich zu den equi-domain Experimenten, haben wir in den cross-domain Experimenten beobachtet, dass die Mehrzahl an unbekannten Wörtern domänenspezifischer Natur sind und nicht generellen Ursprungs. Wir haben auch eine negative Korrelation (Kendall's τ = -0.82) zwischen dem geschätzten Sprachniveau des Benutzers und der Summe der unbekannten Wörter nachgewiesen.

Contents

CHAPTER 1

Introduction

1.1 Motivation

This work is related to the research areas of Information Retrieval (IR), Text Readability (or only Readability) and Natural Language Processing (NLP). According to Collins-Thompson [CT14], assessment of text readability or text difficulty has been addressed for over 70 years. Traditional measures like the Flesch-Kincaid score [KFRC75] from 1975, estimate readability based on word and syllable counts. They essentially state that longer sentences or words are more complex than shorter ones. Clearly, that is not always the only criteria because other important factors like complexity of words (semantics) and complexity of the sentence structure or grammar (syntax) have not been taken into account.

Advances in Machine Learning (ML) and Computational Linguistics opened the door for automated readability assessments [CTC04]. The basic idea is to extract features from text, which are then used in a supervised machine learning approach to classify other documents. For example, one can design a training set that contains documents with words that are most likely to be understandable by six year old children. An algorithm "learns" those words and, in case of binary classification, classifies an "unknown" document as "Yes - understandable by a six year old child" or "No - not understandable". By using ML methods one can check if educational material has the right difficulty level regarding students age or grade. This approach is more sophisticated than the first one, but the fundamental problem remains unsolved: By nature, readability is a highly individual and subjective task. To assess the text difficulty precisely, one must perform it on an individual scale. The importance of user models (UM) in this matter has been known for more than ten years [AAB$^+$03], but still remains as an open problem [CT14].

The goal of this thesis is therefore to individually support people (users) to comprehend relatively challenging textual resources like a researcher's published papers. For that purpose, we compute user models from text, e.g. a researcher's published papers, with the aim to find words in other textual documents that are most likely to be unknown to the user model (and therefore to the user). In the next step, we will explain those words by using glossary entries and images from an already existing data source. We address the problem of constructing language models from domain-specific text and of comparing them for the purpose of identifying dissimilarities. We provide experimental results on a collection of scientific articles from the Computer Science (CS) and the Medical domains. The proposed method can be useful in the area of readability research and E-Learning.

1.2 Problem Statement

The term "user model", adopted from computational linguistics [Chi88], has been used extensively in research areas like Information Retrieval [AAB$^+$03] and Text Readability [CT14]. A simple definition for it might be the following (from [Chi88]): "A user model (UM) contains information about users, such as users' goals, beliefs, knowledge, preferences and capabilities." Among them, knowledge, in terms of used words in textual documents, is the most important property for our purpose. We use those words to construct a "language model", therefore we will use "language model" and "user model" interchangeably. While we acknowledge the difference between the two concepts in general, in the context of the current thesis it is more practical to use the expression "user model" to distinguish between language models of different users and language models of individual documents.

There has been a considerable amount of work on using language models for estimating the readability of textual documents as early as 2001 [SC01]. Nevertheless, what happens after a document has been judged to be more or less readable is rarely considered. This is a significantly harder problem. Because first, we have to identify specific terms that are likely to be unknown to the user and find their proper definition, second, creating a test collection for assessing the effectiveness of the method is fundamentally dependent on the knowledge of the user and therefore highly subjective.

In order to address these problems, we report our experiments using language modeling on a collection of scientific articles. We consider, as Paukkeri et al. [POH13] before, documents from two domains, namely, Computer Science and Medicine. Unlike Paukkeri et al. however, we take the analysis a step further and look at individual terms selected by the models, rather than a general assessment of reader/writer skills.

1.3 Structure of the Work

This work is divided into a theoretical, a practical, and an evaluation part. The theoretical part essentially describes the construction of our language models (user and document models) and how we compare these to find words that are likely to be unknown to the user.

The implementation of the design outlined in the theoretical part (Chapter 3) as a software prototype, is the outcome of the practical part (Chapter 4). The proof of concept offers the following functionalities:

- Computation of document models and user models from a collection of scientific papers.
- By comparing a user model with a document model, we detect words that are most likely to be unknown to the user.
- Providing information about those unknown words by leveraging BabelNet[1], which contains data from different sources like (WordNet[2] and Wikipedia[3]).

In order to validate the proposed method, we have considered both quantitative and qualitative experiments (see Chapter 5). In the absence of user judgments for known or unknown terms, the quantitative experiment will observe whether the method identifies a significantly higher number of unknown terms in documents belonging to the medical domain (i.e. a domain different from that of the users'). In our qualitative analysis, we examined the identified unknown words from a single chosen experiment from each domain.

1.4 Methodology

Finding words that are likely to be unknown to the user requires knowledge about the user's vocabulary. By creating a model from a collection of text (written by the user), we are able to perform experiments and draw conclusions based on them. In addition, we need to evaluate our conclusions or results, to validate our proposed approach.

1.4.1 Design

Our design (see Chapter 3) based on a literature research (see Chapter 2) is created and it includes answers to the following questions:

- **Language modeling**: How can we create language models from text to model documents and users?
- **Comparing models**: How can we compare language models with each other to find words (from a document model) that are likely to be unknown to the user (model)?

[1]`http://babelnet.org/`
[2]`https://wordnet.princeton.edu/`
[3]`https://www.wikipedia.org/`

- **Explaining words**: How can we explain the identified unknown words to the user?
- **Evaluation:** How can we evaluate our approach?

Based on this design, a proof of concept implementation is created.

1.4.2 Implementation

Chapter 4 describes the proof of concept application, which implements the following functionalities, explained in the design from Chapter 3:

- The generation of document and user models from document collections.
- The comparison of language models (document vs. user model) to find words that are most likely to be unknown to the user model and therefore to the user.
- The explanation of those unknown words if the information is available.

This implementation is evaluated in Chapter 5.

1.4.3 Evaluation

The evaluation part reveals the quality of our approach to find words that are likely to be unknown to the user model and therefore to the user. We conduct quantitative and qualitative experiments on a collection of documents from the domains of computer science (CS) and medicine (Chapter 5). First, we compute user models from a document collection of published papers by scientific researchers from the field of CS to compare them afterwards with document models constructed from the domains mentioned above to find words that are most likely to be unknown to the user. Based on those user models, we evaluate the following two cases:

1. **Equi-Domain Experiments**: Documents from the same domain (CS) as the user, but not written by the user, must not contain more unknown words than documents from the medical domain.
2. **Cross-Domain Experiments**: Documents from the medical domain must contain more unkown words than documents from the CS domain.

Documents that have been used to generate user models are excluded from the documents that we use for our model comparison. This prevents us from having unrealistic good results. In terms of qualitative observations, we have manually considered the terms identified as unknown for one medical and one CS document.

CHAPTER 2

Related Work

This chapter covers related work from the research areas of Information Retrieval (IR) and Readability. The problem tackled in this thesis can be seen as a special IR problem. IR is primarily concerned with developing algorithms to identify relevant pieces of information in response to a user's information need [LC03]. Relevant information might be documents like scientific papers, articles, or even smaller parts or passages of documents. In our case, words that are unlikely to be known to the user are considered as highly important. As such, we are searching for information that is unknown to the user (or to its corresponding user model), by utilizing the "Language Modeling Approach" from Ponte and Croft [PC98].

Section 2.1 gives a brief introduction to IR and describes some probabilistic retrieval approaches like the "Classical Model" from Robertson and Sparck Jones [RJ76], and the "Generative Model" from Victor Lavrenko [Lav04]. We also explain the concept of language models, its technical realizations as unigram and n-gram models, and how their probabilities are calculated.

Our work is also related to readability, which is concerned with the difficulty of text and how to objectively measure it. The vocabulary load is one reason why text is felt to be difficult to comprehend and there are ways to determine whether words are likely to be unknown or not [WN08]. We explore this research area in Section 2.2, where we describe both the classical approaches which are based on formulas (traditional measures) and the modern ones which apply various machine learning techniques. Furthermore, we outline the concept of user models which we use in this thesis and which is still attached to many problems like a lack of reliable and scalable evaluation methods. To conclude, in this chapter, we explain the concepts on which our thesis is based upon, namely user models based on the language modeling approach.

2.1 Information Retrieval

Information retrieval can be a very broad term. Manning et al. define IR as an academic field of study, as follows [MRS$^+$08]:

> "Information retrieval (IR) is finding material (usually documents) of an unstructured nature (usually text) that satisfies an information need from within large collections (usually stored on computers)."

Basically this is what Google does for us when we enter a search query into the text input field. The retrieved documents are usually ranked or sorted in descending order by their suggested "relevance" for us. A formal notation would be that based on the query (Q), the retrieval system needs to decide which document (D), from the collection (C), is most likely to satisfy the user's information need (or is among the class of relevant (R) documents).

Over the past six decades, IR methods evolved from boolean retrieval to vector space models towards a probabilistic approach. Furthermore, different ways have been explored to leverage probabilities and to interpret or model the concept of relevance. This resulted in a language modeling approach to IR [PC98] and a more powerful "Generative Model" [Lav08].
This section explains the importance of relevance and covers the ideas behind the chosen probabilistic retrieval methods. Other retrieval methods like boolean retrieval and vector space model are well explained elsewhere [MRS$^+$08].

2.1.1 Relevance and Relevance Models

As mentioned above, an IR system is supposed to return documents that are relevant to the user. Within a probabilitic framework, Lafferty and Zhai elaborated the following question [LZ03]:

> "What is the probability that this document is relevant to this query?"

The answer depends on the chosen retrieval method and its underlying "relevance model". According to Lavrenko and Croft, a relevance model is defined as the probability distribution P(w|R), where w is a random word from a random document D from the collection C. For every word w, it states the probability that w can be observed in a document from the relevant class R [LC03].

How to formalize and to model relevance is quite hard and has been attempted by over 160 publications [Lav08]. In ad-hoc retrieval for example, relevance models need to get estimated without examples or prior knowledge. Lavrenko and Croft [LC01] described how to effectively estimate a relevance model based only on the user's query and without the use of heuristics.

Nevertheless, over time, the following three relevance models (see Figure 2.1), which follow different assumptions, have been established.

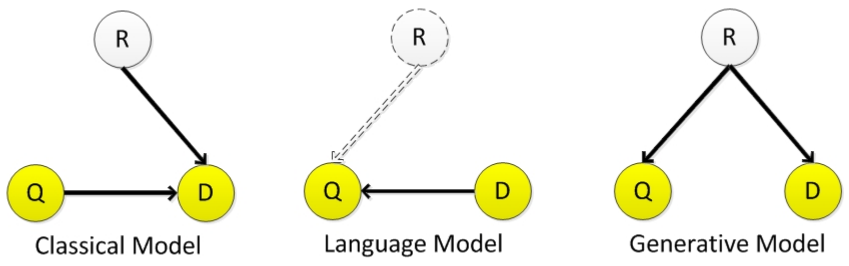


Figure 2.1: Graphical diagrams showing dependencies between the query Q, the document D, and relevance R variables in different probabilistic models of IR. Shaded circles represent observable variables (from [Lav08]).

The next sections will cover those models and explain their core ideas and assumptions. We look at them from a generative point of view and model relevance explicitly.

Generative Relevance Models

Generative models generate text or strings by using the probabilities assigned to its attributes or words [MRS+08]. In the classical probabilistic approach for example, we estimate the probability that a document is generated given a certain query and relevance (relevant or not relevant). And in the language modeling approach we go the other way and estimate the probability that the query is generated from the document.

2.1.2 Classical Probabilistic Approaches

Probability theory offers a principled foundation for reasoning under uncertainty. Uncertainty plays a key role in IR, because the system has to guess the relevance of documents based on a user's description of his or her information need in terms of a query [MRS+08].

Robertson and Sparck Jones [RJ76] estimated two models for each query, one modeling the relevant documents, the other modeling the non-relevant ones. Afterwards, documents are ranked according to the posterior probability of relevance. In other words, documents D get ranked by the odds of their being observed in the relevant class [Rob77]: $\mathrm{P}(D|R)/\mathrm{P}(D|N)$, where N is the non-relevant class or 1-R. This means that the IR system presents the documents in order of their probability of being relevant to the user's request or query. This concept is known as the Probability Ranking Principle (PRP). The Binary Independence Model (BIM) has traditionally been used with the PRP and is known to be the original probabilistic retrieval model [MRS+08].

Document Likelihood

Lafferty and Zhai [LZ03] argue that from a generative relevance model point of view, the probability of relevance p(R=r|D,Q) is estimated indirectly by invoking Bayes' rule: $p(R = r|D, Q) = \frac{p(D,Q|R=r)p(R=r)}{p(D,Q)}$. R, D and Q are random variables, where R denotes the relevance, which can either be r (relevant) or $\bar{r}$ (not relevant).

In the next step p(D,Q|R=r) is factored as p(Q|R=r)p(D|Q,R). After several transformations the equation is as follows: p(R=r|D,Q) $\simeq$ p(D|Q,R=r). The probability of relevance (R=r) given a document and a query (or simply D is relevant to Q) is proportional to the probability of observing the document given the query and relevance. The latter probability can be easily estimated under the Naive Bayes conditional independence assumption, which states that features are independent of each other given the parameter [MRS$^+$08]. This assumption is used to reduce the number of parameters, hence allows simpler computation. In reality, the conditional independence assumption does not hold for text data. Terms are conditionally dependent on each other [MRS$^+$08]. For example, the pairs "hong" and "kong" are highly dependent terms.

Given the Naive Bayes conditional independence assumption we estimate p(D|Q,R=r) with $\prod_{i=1}^{n} p(w_i|Q, R = r)$. We assumend, that the words w of D are independent given R and Q. The same method is used to estimate the probabiltiy for R=$\bar{r}$. The result is the document likelihood based on two parameters, the query and relevance. The term likelihood is just a synonym of probability. It is the probability of an event or data according to a model. The term is usually used when people are thinking of holding the data fixed, while varying the model [MRS$^+$08].

By estimating a second model for R=$\bar{r}$ (not relevant), documents are ranked by their log-odds ratio: $\log\left(\frac{p(D|Q,R=r)}{p(D|Q,R=\bar{r})}\right)$. Simply said, we are generating documents given a query and rank them according to their likelihood descending. The log-odds ratios are needed for document normalization. Ranking only based on document likelihoods p(D|r,Q) would not be effective, because it is inherently biased against long documents [LZ03]. In the next section (2.1.3) we will also start from the PRP, but choose another factorization of the same joint likelihod.

2.1.3 Language Modeling Approaches

The language modelling approach to information retrieval was introduced in 1998 by Ponte and Croft [PC98]. They took the idea of "Language Models" (LM) from other areas of human language technology (HLT), like speech recognition or statistical machine translation [GM03]. A language model is a probability distribution that captures the statistical regularities of the generation of language (generative model). The generated language or result contains more terms with higher probability than terms with lower probability [MRS$^+$08].

The two main ideas behind the model are first, to let the data speak for itself and second, to unite the indexing model, which assigns indexing terms to documents, and the retrieval model [PC98]. The first idea emphasizes the importance of probabilities of words as their manifestation of an underlying probability distribution [GM03]. This approach is different from classical methods like tf-idf weighting and has been shown to outperform these [PC98]. Section 2.2.6 describes how language models are implemented.

The original model, proposed by Ponte and Croft, has also been improved by Song and Croft [SC99] or Hiemstra [Hie01]. Still, it had no explicit notion of relevance as the classical probabilistic approach, which raised the question “Where is the relevance?” [SJR01]. The concept of relevance, while seemingly intuitive, is nevertheless quite hard to define and even harder to model in a formal fashion, as evidenced by over 160 publications attempting to deal with this issue [Lav04]. Lafferty and Zhai [LZ03] incorporate relevance into the LM approach to show its probabilistic equality to the classical approach. Robertson [Rob05] disputed that this document generation approach by Lafferty and Zhai is theoretically equivalent to the classical probabilistic retrieval model [RJ76] due to their different event spaces [LZ15]. However, Lafferty and Zhai’s work [LZ03] still presents a formal and widely-accepted way to connect the language modeling approach to the notion of “relevance” that could answer the question:“Where is the relevance?” [LZ15].

Query Likelihood

In the previous section (2.1.2), we explained how the probability that this document is relevant to this query (p(R|Q,D)), was computed. We factorized p(D,Q|R=r) as p(Q|R=r)p(D|Q,R). In this approach we factorize it as p(D|R=r)p(Q|D,R) [LZ03].

In this case two assumptions are made:

1. Conditioned on the event R=$\bar{r}$, the document D is independent of the query Q. This allows us to drop the “irrelevant” language model p(Q|R=$\bar{r}$).
2. D and R are independent.

The resulting estimation for the relevance ranking becomes $\log p(Q|D, R = r)$, which is basically the probability that the query is generated by the document, given relevance. As in the previous approach, we can decompose the query into words or attributes and view those as independent given the document and relevance. The resulting probability (p(Q|D,R=r)) is then the product of each words’ probability. This computation is quite simple but also dangerous, because a word that cannot be observed in the document, receives the probability of zero, leading to a total probability of zero. This problem is known as the problem of zero frequencies and it can be dealt with by applying a technique called “smoothing”.

There are simple ways to get rid of zero probabilities. The simplest one is to always add a small number to each probability. A more advanced method is to steal away some probability mass from the maximum-likelihood estimator and distribute it among the words that have zero frequency [LC03].

Query Likelihood vs. Document Likelihood vs. PRP

Above we have briefly mentioned two liklihood models for relevance. The first is generating the document from the query, whereas the second is generating the query from the document. Intuitively, it is easier to estimate a model for "relevant queries" based on a document than to estimate a model for relevant documents based on a query [LZ03]. The reason is that a document usually contains more words than a query and therefore provides a larger foothold for estimating a statistical model.

In the LM approach we want to compute the probability that this document generated this query. In contrast to the classical model relevance is not explicitly modeled here. Intuition tells us that the document that is most likely to generate the query is of capital importance. Therefore it must be the most relevant one [JRHZ03]. The classical probabilistic model on the contrary comes up with heuristics to approximate relevance [RJ76].

Another important difference between those two approaches is the need for document normalization. Ranking on document likelihoods is biased against long documents, hence log-odds ratios are used as a counter-measure. In the LM approach competing documents are scored using the same number probabilities, therefore document normalization is not a crucial issue [LZ03].

2.1.4 Generative Model

In his PhD thesis, entitled "A Generative Theory of Relevance", Victor Lavrenko worked on another view on relevance. He described relevance as a generative process, and hypothesized that both user queries and relevant documents represent random observations from that process (see Figure 2.1). This is formulated in the Generative Relevance Hypothesis (GRH) (from [Lav04]):

> "For a given information need, queries expressing that need and documents relevant to that need can be viewed as independent random samples from the same underlying generative model."

By treating both documents and queries as such, he was able to estimate all relevant probabilities without resorting to heuristics [Lav08].

Lavrenko showed that his model, which he named "Generative Model", could outperform prior retrieval models on the TREC ad-hoc retrieval and cross-language tasks [LC01]. The model is also applicable to other retrieval areas like handwriting retrieval, image retrieval or video retrieval. The reason is that nothing in the model is specific to language, to documents, and to queries. Its most important quality is that it makes no structural assumptions about the data.

2.1.5 Limitations of Probabilistic Retrieval Models

In previous sections we covered three probabilistic retrieval models. Each of them is limited in some way, but all of them share the closed-universe approach. That means, they all assume that there exists only a single information need R. For two information needs ($R1$ and $R2$), having a document D which is relevant to both of them, we run into a philosophical inconsistency.

According to the Lavrenko's GRH, D would have to be drawn from the relevance model from $R1$ and $R2$, but this is only possible if they are identical. Same inconsistency also exists in the classical probabilistic model [RJ76] and an even bigger one in the language modeling approach [RH01], where we work with the query likelihood. Assuming that the query Q is a sample drawn from a relevant document D_r, implies that among the whole collection C, there can only be one relevant document, namely D_r, the source of Q.

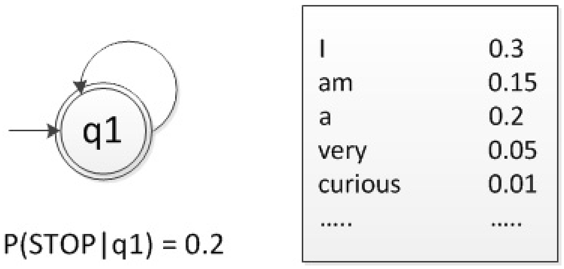
Lavrenko states that those inconsistencies are purely philosophical and that they have absolutely no practical impact, since we are dealing with probabilities and no hard yes-or-no decisions about the origin of the sample [Lav04].

2.1.6 Technical implementation of Language Models

Language models can be realized in different ways. Depending on the conditioning context a language model can be generally classified into a unigram language model or a n-gram language model. The simplest and most common one is the unigram language model [MRS+08].

Unigram Language Models

The unigram language model is the simplest form of a language model. It estimates the probabilities over each term independently. Therefore the order of terms is not relevant. The probability of a certain string of terms is calculated as follows: $\mathrm{P_{uni}}(\mathrm{t_1t_2t_3t_4}) = \mathrm{P}(\mathrm{t_1})$ $\mathrm{P}(\mathrm{t_2})$ $\mathrm{P}(\mathrm{t_3})$ $\mathrm{P}(\mathrm{t_4})$. Figure 2.2 illustrates a unigram language model as a one-state finite automation.



I	0.3
am	0.15
a	0.2
very	0.05
curious	0.01
.....	.....

Figure 2.2: Unigram language model as a one-state finite automation ([MRS+08]).

The probability of a term is typically calculated using a Maximum Likelihood Estimator, as the number of occurences of that term in the document divided by the total amount of terms in the document. For instance the probability distribution of a document with the content “I love Sushi. I love Japan.” is displayed in Table 2.1.

Table 2.1: Probability of terms in an unigram language model.

Term	Probability
I	$2/6 = 1/3$
love	$2/6 = 1/3$
Sushi	$1/6$
Japan	$1/6$

Language models, which take more than one term into account, are called “n-gram models”.

n-gram Language Model

The n-gram model, for $n > 1$, does not estimates the probabilities over each term independently, therefore the ordering is important. This is crucial in areas like speech recognition. For example, the probability to generate a sequence of four words gets computed as follows: $P(t_1 t_2 t_3 t_4) = P(t_1)\ P(t_2|t_1)\ P(t_3|t_2)\ P(t_4|t_3)$.

The probability distribution of the sentence “I love Sushi. I love Japan.” using a bigram language model is displayed in Table 2.2.

Table 2.2: Probability of terms in an bigram language model.

Term	Probability
I love	$2/4 = 1/2$
love Sushi	$1/4$
love Japan	$1/4$

In contrast to unigram models, n-gram models require more computational power and space, but they are able to capture dependencies between words. Also data sparsity becomes a greater issue, because the terms are considered in a compound and not standalone anymore. The probability to encounter another occurence of the word “love” is higher than to encounter an n-gram, where the word “love” is included. Probably the biggest disadvantage of n-gram models is that they are static in the sense that their parameters are fixed during their design [HAA+96]. For example, once a trigram model is created for a certain task, it cannot be transformed to a bigram model to fit another task. Therefore, unigram models are more flexible than n-gram models.

2.2 Text Readability

Readability of text can be judged by content, style, format, and features of organization [GL35]. Research in readability originated in the desire to grade textbooks and other materials for use in the elementary grades. Subsequently, the research activities were extended not only to demonstrate the lack of adequate reading materials for adults, but also to suggest how materials might be better prepared [Lor44]. In the 1920s, methods have been developed to measure the "vocabulary burden" or vocabulary load [LP30].

Twenty years later, Lorge confirmed that vocabulary load is the most important concomitant of difficulty [Lor44]. More than 60 years later, researchers still try to evaluate the vocabulary load of written text [WN08]. Also, methods have been developed to find words that are likely to be unknown [NB07] [NH02].

While we shall come back to these later in this section, it is worth pointing out already that in contrast to those methods, our approach does not require the attendance of the user, because we build a user model based on the user's written work and reason about words that are likely to be unknown, on that "virtual basis". Nevertheless, there are interesting ideas and concepts that we can learn from readability research and incorporate in our proof of concept implementation.

This section gives a brief historical overview, then introduces some concepts and ideas of how to find words that are likely to be unknown by measuring the vocabulary load and ends with user models, yet an open problem and regarded as future work by Collins-Thompson [CT14].

2.2.1 History

In his book "The Classic Readability Studies" [DuB07], William H. DuBay gives a brief historical overview covering the time span of 1893 to 1948. After we covered some of the traditional formulas, we will briefly mention advanced approaches based on machine learning techniques to present a more holistic picture of this research area.

The Early Times (1890 - 1930)

In 1893, a professor of English Literature at the University of Nebraska, Lucius Adelno Sherman, published a book with the title "The Analytics of Literature: A Manual for the Objective Study of English Prose and Poetry" [She93]. He analyzed literature from different time periods and found that the average sentence length became shorter over time [DuB07]:

- **Pre-Elizabethan times**: 50 words per sentence
- **Elizabethan times**: 45 words per sentence
- **Victorian times**: 29 words per sentence

- **Sherman's time**: 23 words per sentence.

He made literatue a subject for statistical analysis and proposed to shorten sentences to increase readability. His ground work set the agenda for a century of research in reading.

In 1921 Edward Lee Thorndike published the first extensive list of words entitled The Teacher's Word Book which initially listed 10,000 words in English. The alphabetical list of 10,000 words were chosen from a corpus with 41 different sources. They mainly contained literature for children and contained about 625,000 words [Tho21]. Each of those 10k words is listed with a "credit-number", a measure of the range (how many sources use the word) and frequency (how often is the word used) of each word's occurence (see Table 2.3). Common words have a higher number than uncommon ones.

Table 2.3: Thorndike's scheme from "The Teacher's Word Book" [Tho21].

Credit-Number	Position of Word
49 or over	1 to 1000
29 to 48	1001 to 2000
19 to 28	2001 to 3000
14 to 18	3001 to 4000
10 to 13	4001 to 5144
9	5145 to 5544
8	5545 to 6047
7	6048 to 6618
6	6619 to 7262
5	7263 to 8145
4	8146 to 9190
3	9191 to 10000

His work helped teachers to measure and adjust the difficulty of textbooks to the reading level of the students. This book is available online[1].

A few years later, Lively and Pressey of Ohio State University, tested the following three methods for measuring the vocabulary load of a thousand words of text [DuB07]:

1. Number of different words (vocabulary range)
2. Number of words not in "The Teacher's Word Book" (10,000 words version) [Tho21]
3. An average value of the credit numbers of Thorndike's list mentioned before. If a word was not part of that list it got the value 0 and got counted twice to give it some extra weight [BG16].

[1]https://babel.hathitrust.org/cgi/pt?id=coo1.ark:/13960/t9f48724f;view=1up;seq=15

They found that the third method was the best indicator of what they called the "vocabulary burden" of their analyzed reading materials [LP30]. Their work and Thorndike's frequent word list influenced future readability formulas.

In the late 1920s Mabel Vogel and Carleton Washburne of Winnetka, Illinois, published a paper in which they included structural characteristics as criteria for their readability formula [VW28]. This formula allowed to objectively match the grade level of a text with the reading ability of the reader or in other words, to classify the reader into grades. The Winnetka formula, which takes semantics and syntax into account, became the prototype of modern readability formulas [DuB07].

New Directions of Readability (1931 - 1950)

By using a formula, which determines the relative difficulty of textbooks using a combination of frequency and vocabulary diversity, W. W. Patty and W. I. Painter discovered that the sophomore year (second year of high school or college) is the year of the highest vocabulary burden [PP31]. They also used another way of creating a sample by using a percentage of words from each text instead of a big passage from one text.

In 1935, William S. Gray and Bernice Leary investigated the question:"What makes a book readable?". They identified 228 elements that affect readability and grouped them into four categories [GL35]:

1. **Content**: Propositions, Organization, Coherence
2. **Style**: Semantic and Syntactic Elements
3. **Format**: Typography, Format, Illustrations
4. **Features of Organization**: Chapters, Headings, Navigation

Their very comprehensive study revealed following findings:

- Content, with a slight margin over style, was most important.
- Less relevant was format, followed by features of organization.
- They could not measure content, format or organization statistically.

In 1944, Irving Lorge published a new formula consisting of a new combination of variables, which predicted readability with higher accuracy than the Gray-Leary formula. His formula used the "Dale list of 769 easy words" [Dal31], which was published in 1931. It took the average sentence length in words and the number of prepositional phrases per 100 words into account. He also published "The Semantic Count of the 570 Commonest English Words", which is basically a frequent word list of word-meanings, rather than just words. The Lorge Index was initially conceived for children's reading, but was also used for adult material as well [Lor44].

Only four years after Lorge published his Index, Edgar Dale and Janne Chall published the most reliable readability formula of that time, namely "The Dale-Chall Readability Formula" [DC48] [DuB07]. Its core component was a list that contained 3,000 easy words[2]. This is basically the extension of the "Dale list of 769 easy words". He developed this list, because he claimed that Thorndike's vocabulary lists failes to capture the familiarity of words accurately.

Over 70 years ago, in 1943, the Austrian Rudolf Flesch, published his first readability formula for measuring adult reading material. Publishers adopted his formula and witnessed an increase of readership by 40 to 60 percent [DuB04]. In 1948 he published another formula in his article "A New Readability Yardstick"[Fle48]. This formula used only two variables, the number of syllables and the number of sentences for each 100-word sample as follows: "Flesch Reading Ease (FRE)" = 206.835 - (1.015 x ASL) - (84.6 x ASW), where ASL is the average sentence length and ASW is the average number of syllables per word. The constant 206.835 adjusts most readability scores to the scale from 0 to 100, moreover the values 1.015 together with 84.6 are the weights for ASL and ASW [HS75]. The interpretation of the score is listed in Table 2.4.

Table 2.4: Interpretation table for Flesch Reading Ease scores (adapted from [HS75])

FRE score	Description of style	ASL	ASW	Typical magazine	Potential Audience school grade completed
0-30	Very difficult	$\geqslant$ 29	$\geqslant$ 1,92	Scientific	College
30-50	Difficult	25	1,67	Academic	H.S. or some college
50-60	Fairly difficult	21	1,55	Quality	Some H.S.
60-70	Standard	17	1,47	Digests	7-8 Grade
70-80	Fairly easy	14	1,39	Slick-fiction	6 Grade
80-90	Easy	11	1,31	Pulp-fiction	5 Grade
90-100	Very easy	$\leqslant$ 8	$\leqslant$ 1,23	Comics	4 Grade[3]

The FRE formula became the most widely used readability formula and one of the most tested and reliable [DuB07]. Nevertheless, traditional formulas have some weak points. Some of them are [CT14]:

1. They typically assume that text has little or no noise.

2. They require significant samples of text (> 300 words).

[2] `http://www.readabilityformulas.com/articles/dale-chall-readability-word-list.php`

[3] Students in Fourth Grade are usually 9–10 years old.

3. They do not perform well on non-traditional documents like web pages.

4. They are based only on surface characteristics of text and ignore deeper levels like cohesion, syntactic ambiguity, etc.

Advances in computational linguistic, machine learning and new data sources smoothed the way for new approaches.

Machine Learning based Approaches

The "AI" (Artificial Intelligence) approach to readability requires following three things:

1. A gold-standard training corpus of individual texts and their corresponding gold-standard labels or readability levels.

2. A set of features extracted from text, which will be later on used by the machine learning model.

3. The ML model that predicts (predefined) readability levels based on features, extracted from text.

The gold-standard corpus plus labels need human expertise to be made (choosing text and annotating it by hand). Labels or grades are usually numbers, which reflect the difficulty level. Complex features need to be extracted by using computational linguistic methods. Some of those features are [CT14]:

- **Lexico-semantic**: rare, unfamiliar or ambigious words

- **Morphological**: rare or more complex morphological particles

- **Syntax**: grammatical structure

- **Higher-level semantics**: use of unusual sense, idioms, domain knowledge, etc.

The ML model can be viewed as a function, which receives features as input and maps them to a number (the readability level). There are many types of learners, but it has been shown [KLP+10] that they are not as important as the features used by those learners. In general, machine learning models improved accuracy over traditional readability formulas [FM12]. The best prediction performance has been accomplished by using both, traditional and non-traditional features.

2.2.2 Evaluating the vocabulary load of written text

In 2002, Paul Nation[4] from the Victoria University of Wellington (New Zealand) published a paper [NH02] about a piece of software that could evaluate the vocabulary load of written text. RANGE, programmed by Alex Heatley, is freely available for download from Prof. Nation's homepage[4]. Before we describe its features, we introduce the concept of "word families", which is important for vocabulary size research in general.

Word families are groups of words that have a common feature or pattern. Bauer and Nation define word families as follows [BN93]:

> "From the point of view of reading, a word family consists of a base word and all its derived and inflected forms that can be understood by a learner without having to learn each form separately".

According to the "Word Family Framework of General English" (WFF)[5], a searchable resource that consists over 22,000 vocabulary items, developed for the British Council, the word "easy" is in the same word family as "ease", "uneasy", "unease", "uneasily" and "uneasiness". The headword[6] (or lemma) is "ease". It gets transformed by attaching affixes[7], e.g. prefixes, like "un-", or suffix, like "-ly", to it.

RANGE, which uses resources (baseword files) that are built upon the concept of word families, can be used to compare the vocabulary of up to 32 different texts at the same time. For each word, RANGE provides (see INSTRUCTIONS.doc of the downloaded archive file):

- a range or distribution figure (how many texts the word occurs in)
- a headword frequency figure (the total number of times the actual headword type appears in all the texts)
- a family frequency figure (the total number of times the word and its family members occur in all the texts)
- and a frequency figure for each of the texts the word occurs in.

It can be used for example to determine the vocabulary size necessary to understand the vocabulary in text, or to evaluate the vocabulary load of text and, most important for us, to determine the number of words in the text which are likely to be unknown. If RANGE gets multiple files as input, it lists the frequency of types from each base list for each file like below (`F1` is file 1 and `F2` is file 2).

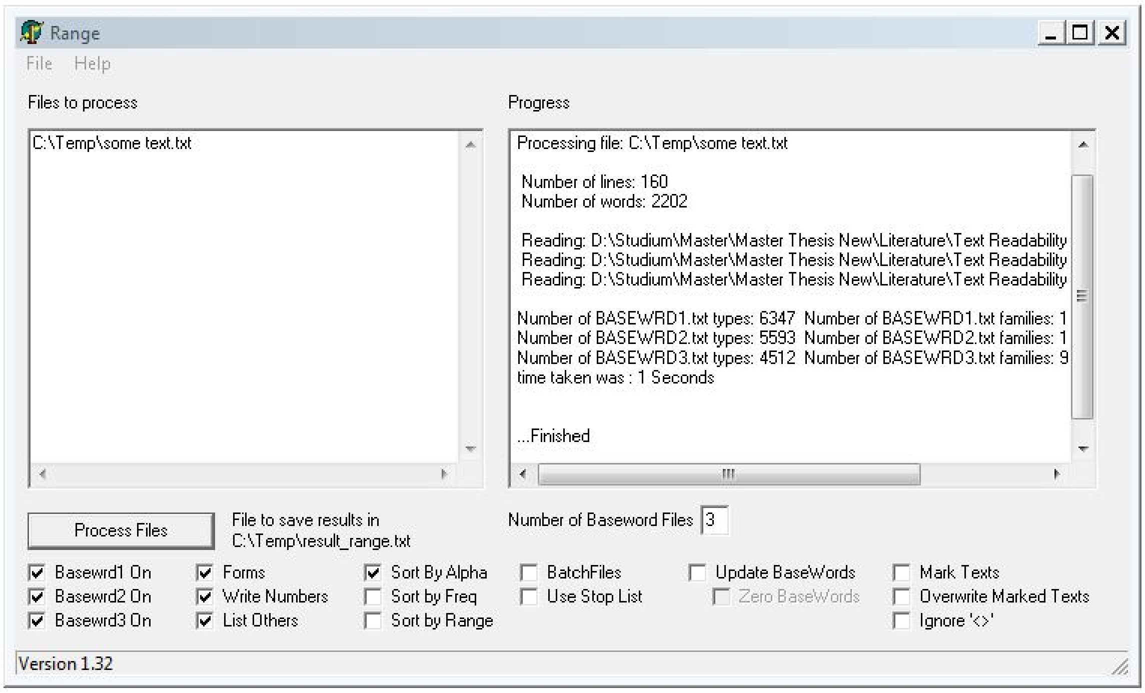
[4]http://www.victoria.ac.nz/lals/about/staff/paul-nation
[5]https://www.learnenglish.org.uk/wff/index.html
[6]https://en.wikipedia.org/wiki/Headword
[7]http://www.affixes.org/a/index.html

```
TYPE                            RANGE   FREQ     F1      F2
A                                   2    229    193      36
ABILITY                             1      3      3       0
ABLE                                2      4      1       3
ABOUT                               2      5      3       2
```

In that way, one can analyze which words have been used in document `F1` but not in document `F2` (see "ABILITY" above).

RANGE was used in the development of the 14 British National Corpus (BNC) 1000 word lists, which were created according to the frequency and range of occurence of word families [WN08].

It can be configured to use up to 16 baseword files, where files 1 to 14 are the 14 BNC word lists, the 15th contains proper nouns and the 16th shows marginal words such as "ah" and "oh" (see Figure 2.3).



C:\Temp\some text.txt

```
Processing file: C:\Temp\some text.txt

 Number of lines: 160
 Number of words: 2202

 Reading: D:\Studium\Master\Master Thesis New\Literature\Text Readability
 Reading: D:\Studium\Master\Master Thesis New\Literature\Text Readability
 Reading: D:\Studium\Master\Master Thesis New\Literature\Text Readability

Number of BASEWRD1.txt types: 6347  Number of BASEWRD1.txt families: 1
Number of BASEWRD2.txt types: 5593  Number of BASEWRD2.txt families: 1
Number of BASEWRD3.txt types: 4512  Number of BASEWRD3.txt families: 9
time taken was : 1 Seconds


...Finished
```

File to save results in C:\Temp\result_range.txt

Number of Baseword Files 3

[✓ Basewrd1 On] [✓ Forms] [✓ Sort By Alpha] BatchFiles · Update BaseWords · Mark Texts
[✓ Basewrd2 On] [✓ Write Numbers] Sort by Freq · Use Stop List · Zero BaseWords · Overwrite Marked Texts
[✓ Basewrd3 On] [✓ List Others] Sort by Range · Ignore '<>'

Version 1.32

Figure 2.3: Screenshot of the RANGE program.

Those baseword files contain word families. An example is the word "achieve", which is the headword of "achievable", "unachievable", "achieved", "achievement", "achievements", "achiever", "achievers", "achieves" and "achieving". RANGE can be used in conjunction with the Vocabulary Size Test, which measures the knowledge of word families, based on those 14 1000 word lists [NB07]. The idea is simple: if the learner is able to demonstrate knowledge of the headword in the test (“achieve” in our case), there is an assumption that he or she will also have receptive knowledge of the rest of that word family. There are ten questions per 1000 word level, which corresponds to a list. Each question represents knowledge of 100 word families. If one question (out of 10) gets answered wrong, then the learner demonstrated knowledge of 900/1000 word families from that level [NB07].

RANGE seems to use a very intuitive approach, which allows fast analysis due to simple computation like tokenizing words (from files) and looking them up in the BNC lists. By writing own baseword files for each user (this would substitute our user model), we could probably achieve a similar result as presented in this work. We could provide RANGE with documents (not known to the user) which contain the “unknown” words and match them with our custom basword files (user model replacements). The disadvantage of that approach is the time-consuming creation and maintenance of those files. Moreover, it is not possible to extend the program with additional functionality or data sources.

It is worth exploring the opportunities it offers and consider the usage of those 14 1000 word lists. But our problem cannot efficiently be solved with RANGE. We need a fully automated approach that creates models from documents of the author (or user), compares them with each other, and explains the words which are most likely to be unknown to the user. In his survey regarding the current and future research of computational assessments of text readabiltiy, Collins-Thompson [CT14] describes the importance of individual assessments by leveraging “user-centric models”.

2.2.3 User Models

Current measures are not able to capture readabiliy adequately, because its nature is inherently individual. In an ideal world, readability is taylored to each user individually. One step in that direction is to explore the creation, evaluation, and validation of models that abstract the individual. User-centric models or just user models can be used for that purpose. Collins-Thompson and Callan [CTC05] built similar models to retrieve content appropriate for elementary and secondary school students. The main idea is to provide students with a search tool that can find reading material relevant to the students reading level. In a first step, they created 12 statistical language models to fit 12 American grade levels. Then, they built a classifier to categorize text from web pages according to those 12 grade levels. That classifier can categorize unknown content with an average root mean squared error of between one and two grade levels for 9 of 12 grades.

We can interpret the above language models as a model for a specific user group. In this thesis, we go a step further and create language models from text, e.g. a researcher’s papers, on an individual level. We call such a user specific model also user model, because it models the user’s knowledge in terms of used and therefore known words. Based on those models, we identify words that are most likely to be unknown to the user. Also, as several studies have shown, word difficulty is an excellent predictor of reading difficulty. Chall and Dale showed a correlation between 0.7 and 0.9 for word difficulty to reading difficulty [CD95]. According to Collins-Thompson [CT14] user models can also be used for the following two tasks:

- **Personalized training**: Adapt users to content.
- **Personalized simplification**: Adapt content to users.

In this thesis, we build a bridge between those two tasks by introducing additional content to the user. By performing experiments on a generated user model, we first identify words that are likely to be unknown to the user. In a next step, we provide additional content like definitions, images or translations to explain those "hard" words. We therefore simplify the content that has been identified as difficult for the user with the intention to adapt the user to that content.

A significant challenge regarding those models is the method of creation. In our approach, we do not assess the user's language skill in terms of known words a-priori, but use his or hers individual work (e.g. scientific articles) to create the user model directly and fully automated. The challenge in creating those models is to estimate the words which are not contained in the underlying documents, the basis of those models. We use statistical methods to compute the language level of the user from the user model and further use this level to estimate the words, not mentioned in the documents at hand. For this, we use an additional resource, which is based on the Corpus of Contemporary American English (COCA) corpus[8]. From a technical perspective, we create the user model as a smoothed unigram model. More details on that is provided in Section 3.1.

To conclude, user models, for the task of text readability, can be created using language modeling techniques. In contrast to traditional measures like the Flesch-Kincaid measure, they do not rely on any (reliable) syntactic features, but just on the words themselves (semantics). Therefore, they can also be applied on non traditional documents like web pages. Also, user models can be easily extended, combined with other models, and used to classify text (see [CTC05]). When dealing with user models, one thing is for sure, gold-standard approaches need to be adapted to provide reliable methods for evaluation.

[8]http://corpus.byu.edu/coca/

CHAPTER 3

Method

This chapter outlines the proposed method of how we detect specific dissimilarities between two probabilistic language models, denoted as document and user models. Specifically, we describe our approach of finding words in the document model that are most likely to be unknown to the user model.

In the first section, Language Modeling, we explain the concept of document and user models. Although, language models can be built from any kind of textual resource (book, web page, etc.), for our purposes, we limit ourselves to scientific articles from the two domains of computer science and medicine.

Section 3.2 covers the basics of natural language processing, like text normalization, tokenization and annotation. We use these techniques to clean, segment and enrich text before we perform the task of language modeling. In this way, we can e.g. remove unwanted content, which results in models of higher quality which consequently leads to better results when comparing these.

Section 3.3 describes how we compare a user model with a document model to detect words that are likely to be unknown to the user. We introduce an algorithm which takes two models, a user model (M_u) and a document model (M_d), as an input. Each model consists of words, which again carry properties like part of speech, lemma and statistical data, that are used to determine, whether a word from M_d is likely to be unknown to M_u.

What happens to a word after it has been identified as possibly unknown to the user is the topic of Section 3.4. There, we briefly outline the content needed to explain those identified terms to the user.

In the last section, we explain our evaluation method, which consists of two types of experiments. The equi-domain experiments take documents from a single domain into account, whereas the cross-domain experiments use documents from both domains, namely computer science and medicine.

3.1 Language Modeling

Language modeling is the process of transforming text into models. The texts we have chosen are scientific journal articles from the domains of computer science and medicine. We selected those documents for the following reasons:

1. **Quality**: Because journal articles are the traditional scientific dissemination medium, they are less prone to slang and typos.
2. **Quantity**: Over the past decades, research conducted in both areas resulted in millions of papers.
3. **Legality**: Among the quantity of published papers, we are able to acquire a subset of it without violating any legal rights.
4. **Authorship**: Each paper contains information about the author(s), therefore we can easily group documents by a specific person. This is inherently important for the task of creating a user model from multiple documents.
5. **Scalability**: By removing or adding documents, we are able to vary the size of the user model, which enables us to conduct a variety of experiments. We are able to evaluate the model comparison algorithm for “small” user models, as well as for “big” ones. Although, we have the opportunity to do so, for our evaluation part, we only consider user models of a certain “medium” size.

Based on the document collection resulting from both domains, we define two types of language models:

1. Document Models
2. User Models

Every model is realized as a unigram language model and therefore contains no information about the surrounding tokens. Each model consists of words and the number of their occurrences. In all models, words are considered together with their POS tag: e.g. *(book, NN)* is considered different from *(book, VBZ)*. For the user model, we additionally applied smoothing. Smoothing is a well known technique in language modeling to assign probabilites to words, which are not included in the language model. The basic idea is to redistribute parts of the probability mass from known words to rare and unseen words [CTC05].

Like Collins-Thompson and Callan [CTC05], we use Simple Good-Turing (SGT) smoothing. Gale and Sampson [GS95] described the Simple Good-Turing estimator as a new version of the "Good-Turing" algorithm, originally developed by Alan Turing and his assistant I.J. Good[1]. The SGT estimator has been tested on a variety of NLP data sets on which it performed well. A comparison of other smoothing techniques is given by Chen and Goodman [CG96]. More details regarding the used implementation of smoothing are provided in Section 4.3.5, where we describe the statistical computation step of the user model generation process.

3.1.1 Document Models

Document models (M_d) are abstractions of single documents and are self-contained. They are either matched with user models to find words that are likely to be unknown to the user or are embedded into the domain specific component of the user model.

3.1.2 User Models

This concept has already been introduced in Section 2.2.3. Our user model (M_u) consists of two components: a domain specific component (M_s) based on the scientific documents, and a general component (M_g) based on the list of frequent terms of American English (FWL, see Section 4.1.2).

The domain specific component is built from a collection of documents, which have been authored by the user. It contains the terms and their smoothed probabilities. To address the problem of missing common words, we needed to bring in another component. For example, a document (which we compare with M_u) might contain common words (like "dog" or "milk"), but because the user did not use those common words in his papers, they cannot be found in the domain specific component of the user model and are therefore marked as probably not known by the comparison algorithm. The purpose of the general component is to enrich the user model with those trivial words, which are most likely not included in the paper, but are most certainly known to the user. This component helps us to reduce the amount of too many potential false positives (words are classified as unknown, although they are most likely not).

Because the general component is based on a frequent word list (FWL) which contains over 100k entries, it is wrong to assume that the user might know all of them. Therefore, we introduced a threshold which we compute based on the domain specific component of the user model, but by using the rank order of words contained in the FWL. The rank order can be described as a measure for the user's language level, where a higher rank (1 is higher than 2, but 2 is greater than 1) denotes a lower level. Actually, the rank order of the word (token) is the manifestation of the raw frequency of the token in the 450 million word Corpus of Contemporary American English. By computing the average rank of words from the domain specific component of the user model, we can define the section

[1] `http://www.grsampson.net/RGoodTur.html`

of words (within the FWL) from rank 1 to the value of the average rank as probably known. Therefore, the size of the general component depends on the word complexity of the domain specific component. We will use the term “language level” and “average rank” interchangeably to refer to the general knowledge of English of the user.

Our approach to construct the domain specific component of a user model from an author’s document collection is as follows:

1. Extract the text from each document.
2. For each document, perform linguistic processing (e.g. tokenization) on the extracted content (see 3.2.1). The result is a document model with annotated tokens.
3. Merge all document models into one unified user model.
4. Compute the smoothed probabilities to handle yet unseen words.

Next section describes the linguistic processing.

3.2 Linguistic Processing

We summarize following steps as linguistic processing:

- **Text normalization**: For example cleaning the text from unwanted content, like html-markup or non-unicode characters.
- **Tokenization**: Segmenting the text into tokens.
- **Annotation**: Adding metadata to the tokens, like part of speech (POS) tags.

3.2.1 Text Normalization & Tokenization

To be able to construct a user model from documents, one must first preprocess or normalize the text. Tokenization is the task of segmenting a character sequence (e.g. text written in English) into pieces, called tokens [MRS+08]. Tokens are linguistic units such as words, punctuation, numbers, alpha-numerics, etc.

Text normalization can also include steps like:

- **Removing unwanted content**: Like special characters, HTML tags, spaces, line-breaks, digits, punctuation, etc.
- **Capitalization (case folding)**: Lowercase all content or just words after a full stop. Depending on the method side-effects might occur (“US” vs. “us”).

- **Defining token boundaries**: This is trivial for words like “dog” or “algorithm”, but not for words with apostrophes or hyphens like “isn’t” or “e-mail”. Those words do need to be further decomposed for subsequent processing. Defining boundaries is more complicated in other languages like Chinese or Japanese, where words are not separated by whitespaces.
- **Stemming**: This is a heuristic rule-based word reduction process that does not require any prior analysis of the word. As such, the result might not be a word anymore. The most common algorithm for stemming English is Porter’s algorithm [Por80]. More information is available here[2].
- **Lemmatization**: The aim of lemmatization is to return the base or dictionary form of the word. In contrast to stemming, it needs prior vocabulary and morphological analysis.
- **Stopword removal**: This method aims to remove the most frequent words, e.g. “the”, “and” or “a”. Those words usually do not carry much meaning and can be omitted as such. A very comprehensive list of 571 English words is available here [3].

Text normalization systems are considered to be very important [Zhu]. Mistakes made at this stage are very likely to induce more mistakes at later stages. Tasks listed above can usually be achieved by writing regular expressions (regex).

Tokens

The resulting tokens are sometimes referred to as terms, words, or types. The difference between these expressions is explained below [MRS+08] [Zhu]:

- **Token or Word Token**: An instance of a sequence of characters in some particular document that are grouped together as a useful semantic unit for processing.
- **Type**: Unique words or the class of all tokens containing the same character sequence.
- **Term**: This is used in the context of Information Retrieval systems. It is a type that is included in the IR system’s dictionary and may consist of one or more tokens.

3.2.2 Annotation

Annotations provide meta information about the underlying data. For example tokens can be annotated with POS tags, which describe if the token is a noun, verb or something else. Another example would be named entity (NE) annotations, which provide information about the type of the entity, like location, person or organization.

[2] `http://tartarus.org/~martin/PorterStemmer/`
[3] `http://www.lextek.com/manuals/onix/stopwords2.html`

Part of Speech Tagging

The part of speech (POS) for a word gives a significant amount of information about the word and is needed for us to look it up in our 100k frequent word list. POS can be divided into the following two categories [JM14]:

1. **Closed classes**: They have a relatively fixed membership, like prepositions. Closed class words are generally also function words (grammatical words), like “of”, “it”, “and”, or “you”, which tend to be very short, occur frequently, and play an important role in grammar.

2. **Open classes**: Nouns and verbs are of this category, because new nouns and verbs are continually coined or borrowed from other languages. The four major open classes that occur in the languages of the world[4], are nouns, verbs, adjectives, and adverbs.

For our purposes, it is also good to know if a noun is a proper noun or a common noun. Proper nouns, like “Arnold Schwarzenegger”, “Vienna” or “Red Bull”™, are names of specific persons or entities [JM14]. For our task of identifying words that are likely to be unknown to the user, we consider proper nouns as not important. We want to identify “hard words” of the general language domain with respect to the user model, which appear with a relatively low frequency, or domain specific words.

POS tagging for English is integrated into the Apache™OpenNLP library[5] and the Stanford CoreNLP distribution[6].

Named Entity Tagging

The tagging of named entities (NE), the names of particular things or classes, is regarded as an important component for many NLP applications. These applications include Information Extraction (IE), from which it was born, Question Answering (Q&A), Summarization and Information Retrieval (IR) [SSN02].

We perform NE tagging and use this information in our post-processing step (see Section 4.3.4). We also use this information from BableNet, which distincts between concepts and named entities. Because named entities are related to proper nouns, we have another way to check if the examined word is important for our purpose or not.

Named Entity Recognition (NER) is also embedded into Apache™OpenNLP[7] and the Stanford CoreNLP distribution[8].

[4]English has all four of these, but there are other languages, like, the native American language Lakhota, which have no adjectives.

[5]`https://opennlp.apache.org/documentation/manual/opennlp.html#tools.postagger`

[6]`http://nlp.stanford.edu/software/tagger.shtml`

[7]`http://opennlp.apache.org/documentation/1.6.0/manual/opennlp.html#tools.namefind.recognition`

[8]`http://nlp.stanford.edu/software/CRF-NER.shtml`

3.3 Comparing Language Models

Section 3.1 described the two types of language models (document and user models) that are matched by an algorithm, described in this section, to identify words that are most likely to be unknown to the user[9]. The user model (M_u) consists of a domain specific component (M_s), which we constructed from the document collection and a general component (M_g), which is a subset of the 100k frequent word list (FWL).

We define a function ($f_{\text{comp}}(M_u, M_d) : Set < w_u >$) that takes following input:

1. A user model (M_u), which is assumed to contain only known words (w_k).
2. A document model (M_d), which may contain both known words (w_k) and unknown words (w_u).

It returns a set (only unique entries) of unknown word types ($Set < w_u >$).

The idea of the method introduced below is that we first try a direct match of the word (w_d) with its POS tag against the domain specific component. If the match is successful, the word will be marked as known. Otherwise, we try to find it in the general component. If we cannot find the word in there either, we classify it as unknown. If the word is contained in the general component, then the user is most likely familiar with it and it is classified as known. Because we use the average rank order (computed from the domain specific component) as the language level of the user to compute the relevant subset of words (from the FWL) for our general component, we use the rank order (found in the FWL) of words from the document model to determine if they are known to the user or not. Therefore, to determine whether a word belongs to the general component of the user model, we compare its rank order with the average rank order of the user model. If the rank order of the word is higher than the average rank (or language level) of the user model, we consider the word to be known to the user. We already introduced the concept of the average rank in Section 3.1.2.

If the word is not contained in the general component, we obtain its lemma. We assume that the lemma is a more common word than the word at hand, hence we expect it to have a higher rank order or familiarity. To gain a more accurate or holistic picture of the comparison process, we abstained from using the lemma from the beginning. If the user uses a derived word from the lemma, e.g. “went”, then he or she most certainly knows the lemma, in this case “go”. Therefore, we perform the familiarity check with the lemma this time. If the lemma is contained in the general component, we classify the original word as known. Otherwise, in case of a verb or adjective, we retrieve the antonym (opposite meaning) of the word. In the other case (e.g. noun or adverb), we classify the word as unknown. The idea behind the antonym is that it might be a more familiar word than the word at hand. For example, the word “dissimilarities” is ranked at position 62,314 whereas the antonym “similarities” is ranked at position 7,916. Without considering the

[9]For the sake of simplicity, we denote those words as unknown words.

antonym, we would classify the word “dissimilarities” as unknown, although “similarities” is contained in the general component and as such probably known to the user. There are probably more word relations (like antonym) we could leverage. However, this is considered as future work.

Our intention behind using those concepts (lemma and antonym) is to keep the result set small. We want to avoid false positives (words classified as unknown, although they are probably known to the user). This is our design decision and we are aware of the tradeoff between false positives and false negatives. A brief description of the algorithm is given below.

We loop through each word (w_d) in the document model and for each word not in the domain specific component (M_s) of the user model make decisions based on the following decision tree:

1. w_d with its POS tag is not contained in the general component
   - 1.1. w_d without its POS tag in M_g: Add it to the result if the rank of w_d or its lemma is greater than the average rank of M_s.
   - 1.1. w_d without its POS tag not contained in M_g: Add it to the result if the word is a concept (not NE).
2. w_d with POS tag contained in the general component
   - 2.1. Add it to the result if:
     - 2.1.1. the rank of w_d or its lemma is greater than the average rank of M_s and
     - 2.1.2. if w_d is an adjective or verb, also compare it with the rank of the antonym

The POS tags that we use here are obtained from the linguistic processing step. The lemmas were taken from the 100k frequent word list, the average rank was computed from M_s and the antonym was taken from BabelNet. BabelNet contains further word relations (like WordNet relations and semantic relations from Wikipedia) that might be useful for our task. Exploring those relations in depth is part of the future work.

3.4 Explaining Unknown Words

Once we found the words which are most likely to be unknown to the user, we try to provide as much information to the user about those words as we can. We consider the following information useful to comprehend an unknown word:

- **Glossary entry**: It provides the user with an explanation of the word. This requires an embedded dictionary or glossary. Wiktionary[10] contains such information.

[10] https://en.wiktionary.org/wiki/Wiktionary:Main_Page

- **Translation**: A translation into the user's mother tongue requires a multilingual dictionary. This is probably relatively easy to realize for common languages. One project that provides such a serivce is Open Multilingual Wordnet[11].
- **Picture**: Sometimes a picture is worth a thousand words. Search engines like Google might be leveraged for this purpose. Also some Wikipedia[12] articles include pictures.
- **Further Information**: Pointer to web pages which contain further information about the word might provide deeper insights. Wikipedia contains a lot of information about all kinds of concepts. Also the BabelNet website[13] can be used for this purpose.

There exist probably additional resources, which we could add to this list, but the listed resources might be sufficient for a start.

3.5 Evaluation

We evaluate our work by conducting quantitative and qualitative experiments. We distinguish among the following two types of experiments, which we relate with each other to determine if our approach is working in general or not:

1. **Equi-Domain Experiments**: We construct document and user models from the same domain according to Section 3.1. We match M_u with M_d, for a document d not used to generate M_u and count words that have been identified as likely to be unknown, to compare them later on with the result of the cross-domain experiments.
2. **Cross-Domain Experiments**: We build document models from the medical domain and user models from the computer science domain, match them, and count words classified as unknonwn to the user (just as above).

By using quantitative methods, we expect to encounter a higher number of unknown words from the cross-domain experiments than from the equi-domain experiments. Also, among the identified unknown words, by using qualitative methods, we expect to encounter more domain-specific words from the cross-domain experiments. Those statements are sufficient to show that the proposed approach works in general. Little is known though, about its quality, because there is no reference whether the gap between both results is too high, too little or just right. Hence, more advanced evaluation methods are considered as future work.

Our detailed results and findings of our experiments are presented in Chapter 5.

[11]http://compling.hss.ntu.edu.sg/omw/
[12]http://en.wikipedia.org/
[13]www.babelnet.org

CHAPTER 4

Proof of Concept

This chapter describes the proof of concept implementation based on the proposed method from the previous chapter. In Section 4.1, we detail the three data sources we use. First, the DBLP computer science bibliography. Due to its structure and contents, DBLP has been extensively used to create expert models in the literature[DKL08]. We use it to create the domain specific component of our user model. Second, a very comprehensive frequent word list (FWL) which is the basis of the general component of the user model. Third, BabelNet, a large dictionary that we mainly use to explain the unknown words.

In Section 4.2, we describe our chosen software architecture, frameworks, and libraries. First, we introduce our frontend application, which visualizes our models and the outcome of our conducted experiments. Second, we describe the layers of our backend application, which consist of Maven modules. In those modules, we realized the functionality explained in the following sections.

Implementation details about the language model generation is explained in Section 4.3. There we describe the initial parsing of PDF documents, which basically extracts the text from files. This text is passed to our linguistic processing step where we filter and segment it into tokens. Those tokens are then processed again and finally used to generate our language models.

Generated models are then compared with each other in Section 4.4. There we explain the comparison function introduced in Section 3.3. Words identified to be most likely to be unknown to the user are then looked up in BabelNet, where we retrieve information (e.g. glossary entries or images) about them (see Section 4.5). Those entries are then saved to the database, where they can be retrieved by the frontend application.

An overall summary is given in the last section of this chapter.

4.1 Data Sources

The three data sources we utilize are:

1. **DBLP**: The DBLP computer science bibliography is a high-quality bibliographic metadata database lead by Dr. Michael Ley[1], lecturer at the Department of Computer Science of the University of Trier. We use it to identify authors (user model candidates) for the task of user modeling (see Section 4.3). The domain specific component is generated from the author's document collection.
2. **Frequent Word List (FWL)**: A very comprehensive frequent word list by Mark Davies[2], Professor of Linguistics at Brigham Young University. This resource is utilized to model the general component of the user model.
3. **BabelNet**: BabelNet [Bab] is a very large multilingual encyclopedic dictionary and semantic network desigend by Professor Roberto Navigli[3], head of the Linguistic Computing Laboratory of the Sapienza University of Rome. We use BabelNet for multiple tasks, like removing tokens which are most likely artefacts introduced by the document parser and explaining words, identified as possibly unknown, to the user.

4.1.1 DBLP Computer Science Bibliography

The DBLP computer science bibliography is an online reference for bibliographic information on major computer science publications [Ley05]. It provides free access to high-quality bibliographic metadata and links to the electronic editions of publications. The whole data set is accessible for download[4] as a single XML file [Ley09a]. Additionally, parts of it can be retrieved via the XML-based API [Ley09b]. DBLP contains more than 3 million publications from more than 1.6 million authors [Dbl].

The eight types of publications are[5]:

1. **Books and Theses**: Authored monographs, as well as PhD theses.
2. **Conference and Workshop Papers**: Papers published in peer-reviewed conferences or peer-reviewed workshops.
3. **Editorship**: All publications that have been edited by a person.
4. **Journal Articles**: Articles that have appeared in a peer-reviewed journal.

[1] `http://dblp.uni-trier.de/db/about/ml`
[2] `http://davies-linguistics.byu.edu/personal/`
[3] `http://wwwusers.di.uniroma1.it/~navigli/`
[4] `http://dblp.uni-trier.de/xml/`
[5] `http://dblp.uni-trier.de/faq/What+types+does+dblp+use+for+publication+entries`

5. **Parts in Books or Collections**: Research articles that have been published as a chapter of a monograph.
6. **Reference Works**: E.g. survey papers and encyclopedia entries.
7. **Data and Artifacts**: Evaluated and published research data and artifacts.
8. **Informal and Other Publications**: Publications that do not fall into one of the other categories above.

Publications (DBLP records) of those types above are represented as following XML elements [Ley09a]: `article`, `inproceedings`, `proceedings`, `book`, `incollection`, `phdthesis`, `masterthesis`, and `www`. Each XML entry represents a single publication. Those DBLP records can be understood as BIBTEX records [Lam86] in XML. For our purposes, we are only considering journal articles, because they are the traditional scientific dissemination medium. Figure 4.1 illustrates the growth of journal articles over time.

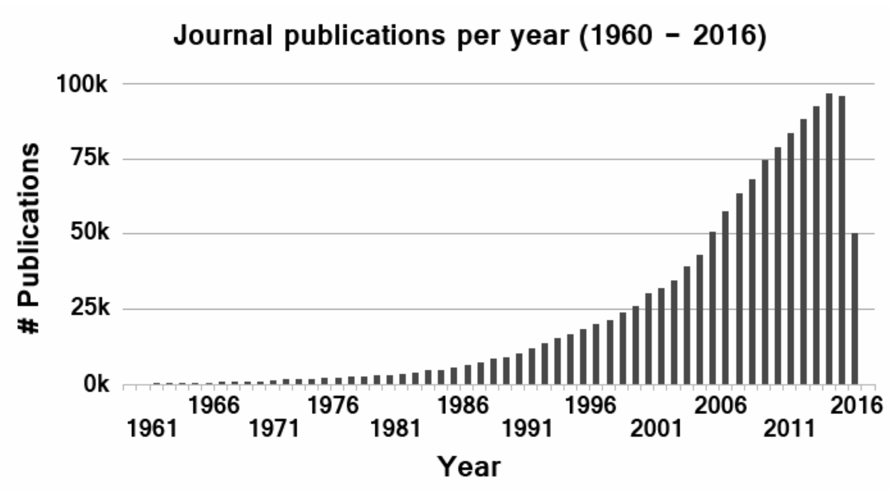


Figure 4.1: Total number of Journal publications from 1960 to 2016[6]

We can further expand our document collection by considering other types of papers. The Proximity DBLP database[7], based on data from the DBLP Computer Science Bibliography with additional preparation performed by the Knowledge Discovery Laboratory, University of Massachusetts Amherst, denotes following DBLP record types as the type `paper`:

- **`articles`**: An article from a journal or magazine.
- **`incollection`**: A part of a book having its own title.
- **`inproceedings`**: An article in a conference proceedings.

[6]`http://dblp.uni-trier.de/statistics/recordsindblp.html`
[7]`https://kdl.cs.umass.edu/display/public/DBLP+README`

4.1.2 100,000 Frequent Word List

The frequent word list used by this implementation is the most accurate one for English data [wor]. It is based on the 520 million word Corpus of Contemporary American English[8] (COCA)[9]. Relevant columns (from the 100k spreadsheet[10]), which are used in our implementation are listed below:

- **ID**: This is the rank order. Terms are ordered decreasingly by their frequency in the English language. The word "the" is ranked as the first one in the list, wheras "ovules" is ranked as number 100,813.
- **Word**: This is the actual word.
- **Lemma**: That is the base form of the word, e.g. "go" for the words "went" and "gone".
- **Part of Speech**: POS according to the CLAWS7 tags[11].
- **Raw Frequency**: This is the word count in the 450 million word Corpus of Contemporary American English. For the most frequent word amongst them ("the") it is 25,131,726 and 16 for the least frequent word ("ovules").

The frequent word list also contains frequencies (per million words) in other corpora like the British National Corpus[12], Corpus of American Soap Operas [13] and Corpus of Historical American English for 1950-1989, 1900-1949 and 1810-1899[14].

Before we can use this list to filter out words when we compare language models with each other, we have to convert the POS tags from CLAWS7 to Penn Treebank tags. This is necessary because the POS tagger we use in Section 4.3.3 is trained on the WSJ Penn Treebank. The conversion table is shown in the Appendix (see Table1).

4.1.3 BabelNet

BabelNet is both a multilingual encyclopedic dictionary with lexicographic and encyclopedic coverage of terms and a semantic network, which connects concepts and named entities in a very large network of semantic relations, called "Babel synsets". Each synset represents a given meaning and contains all the synonyms which express that meaning in a range of different languages. In its current form (version 3.7), it contains 14 data sources and covers 271 languages [Bab].

[8]http://corpus.byu.edu/coca/
[9]http://www.wordfrequency.info/comparison.asp
[10]http://www.wordfrequency.info/100k.asp
[11]http://ucrel.lancs.ac.uk/claws7tags.html
[12]http://corpus.byu.edu/bnc
[13]http://corpus2.byu.edu/soap
[14]http://corpus.byu.edu/coha

We use the BabelNet version 2.5.1, which was available for download in winter 2014, but is not anymore at the time of writing. This software is based on approximately 26 GB of unpacked resources (Apache Lucene^{TM}index files) from Wikipedia, Wiktionary, Wikidata, WordNet, Open Multilingual WordNet and OmegaWiki (see Figure 4.2). From version 2.5.1 on, users have to use the BabelNet API to query the data source.

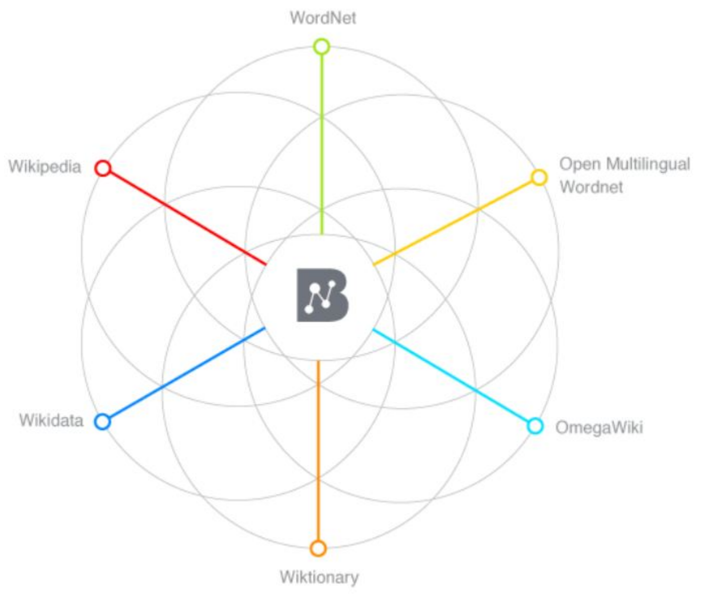


Figure 4.2: BabelNet: A composition of data sources (from[15])

BabelNet 2.5.1 offers the following features (see [Rob]):

- 50 languages covered
- 9.3 million Babel synsets (**syn**onym **set**s of specific meanings in different languages)
- 67 million Babel senses (synonyms which belong to a synset)
- 21.7 million textual definitions
- 262 million semantic relations
- 7.7 million synset-associated images
- 1.1 billion RDF[16] triples

[15]http://babelnet.org/about
[16]https://www.w3.org/RDF/

More statistics and comparisons between versions are available online[17].

The BabelNet Java^{TM}API provides methods to query the data. The following information can be retrieved:

- Senses or Synsets by language, word, part of speech[18], data source (Wikipedia, WordNet, OmegaWiki, etc.)
- Relations from synsets and senses
- Images from synsets
- Glossaries from synsets
- Translations from synsets

Using these resources, it is possible to explain words to the user in many ways, e.g. by providing the glossary entry, translation to the mother tongue, or showing an image if available. Furthermore, BabelNet classifies words into the following two categories:

- **Concepts**: According to Merrill et al. [MT77], a concept is a set of specific objects, symbols, or events which are grouped together on the basis of shared characteristics and which can be referenced by a particular name or symbol. Some examples for concepts are the words “company” or “person”.
- **Named Entities**: Named entities (NE) are phrases that contain the names of persons, organizations, locations, times, and quantities [TKSDM03]. For example, “Apple” is a company and “Michael Jackson” a person. One can say that NE are instances of concepts [Web].

This distinction is important for our task, because we are interested in unknown concepts and known NE. We use BabelNet as a semantic filter and as an information provider to explain unkown words to the user. Its anatomy (underlying RDF structure/graph) allows us to follow the relations of synsets (like “Antonym” - the opposite meaning) and as such provides more possibilities to filter words.

4.2 Software Architecture

This section describes the architecture of the proof of concept implementation. At first, we give an overview about the overall structure of the application. In Sections 4.2.2 and 4.2.3, we explain the used technologies and their features.

[17]`http://babelnet.org/stats`

[18]In version 2.5.1 BabelNet uses only following four part of speech tags, namely nouns, verbs, adjectives and adverbs.

4.2.1 Overview

The proof of concept has been implemented as a Client-Server application (see Figure 4.3). Such an architecture provides us with loose coupling between the data, which are stored on the server-side, and their presentation, which resides on the client-side which is the web browser. The server and client application use the http protocol[19] to transfer data.

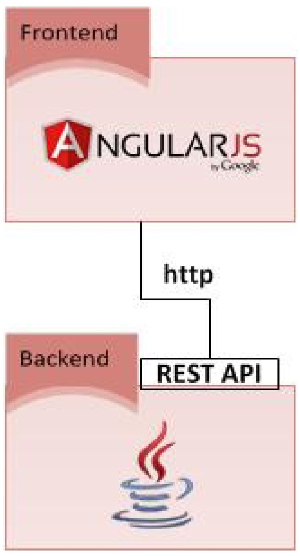


Figure 4.3: Architecture Overview: Client-Server (C/S) Application.

For a variety of reasons, which we explain in the next section, we chose AngularJS 2[20] as our frontend technology.

The entire backend application is written in Java™, where we structure the logic in services. Those services are exposed via REST[21] and are accessed by the client over the network via http.

4.2.2 Frontend

AngularJS 2 or just Angular2 (the successor of AngularJS), is a web framework for single page applications (SPA). A SPA consists of a shell page (index.html) which displays different HTML templates or views, based on the applications state. Those fragments are retrieved asynchronously, hence the user does not have to wait for the request to finish. This enriches the user experience. A downside of this approach is that the browser has

[19]Hypertext Transfer Protocol: `https://tools.ietf.org/html/rfc2616`
[20]`https://angular.io/`
[21]`https://en.wikipedia.org/wiki/Representational_state_transfer`

a lot of work to do (e.g. compiling HTML elements, evaluate data bindings, execute directives). This is not the case in the traditional round-trip-model [Fre14], where the browser is essentially a rendering engine and data and logic reside on the server side.

Angular2 (as its predecessor) can be used with plain JavaScript[22], but we use it with TypeScript[23], which is a superset of JavaScript. In contrast to JavaScript, TypeScript offers type safety, which helps developers to write maintainable code. Because browsers are not able to run TypeScript, it has to be compiled to JavaScript. The TypeScript compiler (tsc) might display errors during compilation, which is an additional help for web developers. The official website of AngularJS[24] recommends the usage of TypeScript.

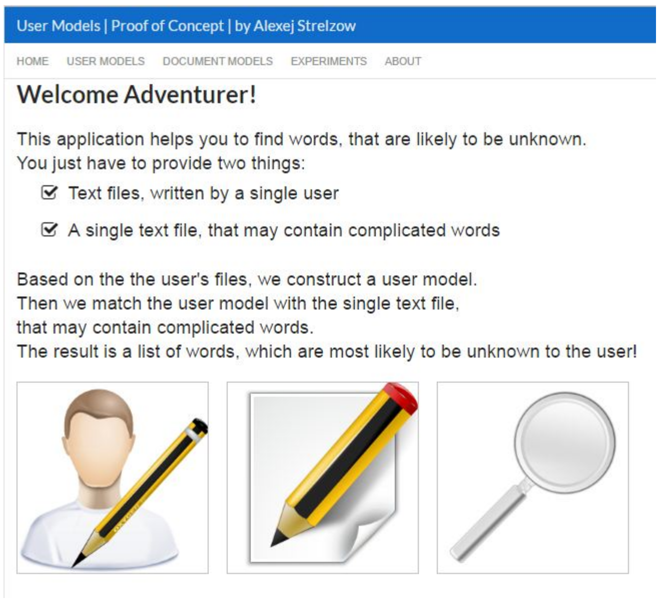
We use Angular2 for the following reasons:

- It is a state of the art web framework supported by major actors in the field.
- We can use it with HTML5[25] and CSS[26], hence we can build it as a responsive application[27].
- Due to the Node Package Manager[28] (NPM), the integration of other libraries is simple.
- We can visualize our data with the help of Angular Animate, which is one of the numerous features[29] from Angular2.
- Asynchronous and event-based programming is very convenient, because it is based on The Reactive Extensions for JavaScript[30] (RxJS), a project developed by Microsoft™.
- Because one of the core concepts of Angular2 is dependency injection (DI)[31], testing the services, which contain the logic, is very easy.
- One of the most powerful features of Angular2 is the concept of components. Components are self-contained structures that are intended to be reused by other components. This can be a navigation bar, table, or any composition of HTML elements.

[22]https://en.wikipedia.org/wiki/JavaScript
[23]https://www.typescriptlang.org/
[24]https://angular.io/
[25]https://www.w3.org/TR/html5/
[26]https://www.w3.org/Style/CSS/
[27]https://en.wikipedia.org/wiki/Responsive_web_design
[28]https://www.npmjs.com/
[29]https://angular.io/features.html
[30]https://github.com/Reactive-Extensions/RxJS
[31]https://en.wikipedia.org/wiki/Dependency_injection

Another positive aspect of Angular is its community. There are numerous open source projects hosted on Github[32], a public git[33] repository, which can be easily integrated into the web application via NPM. Also, there are a few project seeds[34] available, which ease the project setup. They are already configured, contain many useful libraries and ways to build and deploy the application.

With our frontend application, the user can visualize the computed language models, run experiments on them and display words that are likely to be unknown. The model generation is only possible directly with the backend application. Figure 4.4 shows the home screen of the application, which consists of the navigation bar (menu) and the welcome screen. The welcome screen shows the user a brief introduction to the application and offers three image buttons (representing "User Model", "Document Model" and "Experiment"), which navigate the user to the respective page.



Welcome Adventurer!

This application helps you to find words, that are likely to be unknown.
You just have to provide two things:

- Text files, written by a single user
- A single text file, that may contain complicated words

Based on the the user's files, we construct a user model.
Then we match the user model with the single text file,
that may contain complicated words.
The result is a list of words, which are most likely to be unknown to the user!

Figure 4.4: Home screen of the frontend application.

4.2.3 Backend

The backend is written entirely in Java^{TM}, because there are several useful NLP libraries available. Due to the Java Virtual Machine (JVM^{TM}) a Java^{TM} application offers also the advantage of platform independence.

[32]https://github.com/
[33]https://git-scm.com/
[34]We used the following Angular2 seed: https://github.com/mgechev/angular2-seed

We structure our application by using Maven[35] modules. Maven is a tool for software project management. We use it for dependency management and building the backend application (for deployment). Figure 4.5 shows our modules and their dependencies:

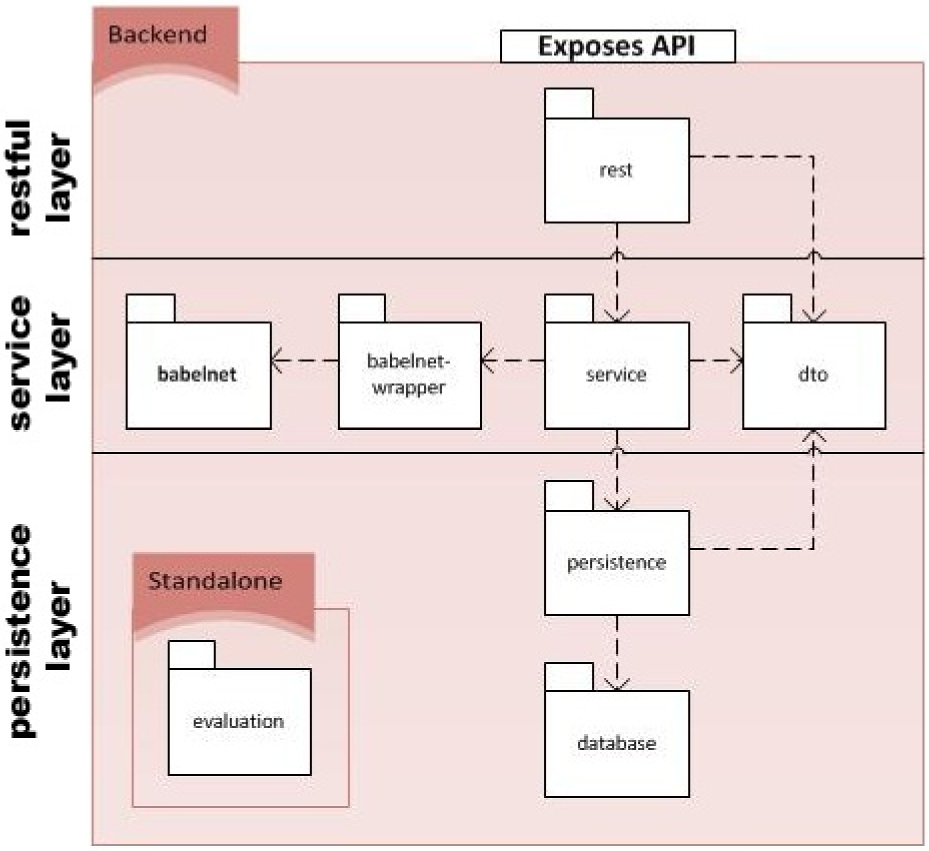


Figure 4.5: Backend Architecture: Dependencies of Maven modules.

The backend application is divided into the following three layers:

1. **RESTful web service layer:** It provides the API to external applications, e.g. our frontend written in Angular2.
2. **Service layer:** This layer implements all the logic, like linguistic processing, model comparison and word explanation.
3. **Persistence layer:** The persistence layer contains the Object-Relational (O/R) Mapper[36] and database.

The DTO-module, which contains data transfer objects[37], is shared accross all layers.

[35]https://maven.apache.org/
[36]https://en.wikipedia.org/wiki/Object-relational_mapping
[37]https://en.wikipedia.org/wiki/Data_transfer_object

Spring Boot Framework

Throughout this application, we utilize the Spring Boot[38] framework (or just Spring). We use it with the starter POM[39] for Maven, which simplifies the configuration. Spring Boot comes with an integrated web server, e.g. Tomcat[40], Jetty[41] or Undertow[42], which makes deployment straightforward.

Additional functionality or libraries can be added to Spring via Maven-Dependencies[43] as in the following example:

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
```

Maven downloads the libraries specified by the `artifactId`, and adds them to the classpath[44], where they are found by the JVM™, the part of the Java™runtime environment (JRE) that executes the bytecode (runs the program).

The first dependency enables support for full-stack web development, including Tomcat and spring-webmvc[45]. We used this to build our RESTful web service layer. Each starter artifact contains a bundle of consistent dependencies, which helps developers to get started.

The latter dependency enables support for the Java Persistence API (JPA), including spring-data-jpa, spring-orm and Hibernate[46], which is a widely used O/R-Mapper. We implemented our persistence layer on top of that.

RESTful Web Service Layer

The purpose of this layer is to expose functionality (API) to other applications via RESTful web services, which use http to transport data and JavaScript Object Nation[47] (JSON) as

[38] `http://projects.spring.io/spring-boot/`
[39] `https://maven.apache.org/pom.html`
[40] `http://tomcat.apache.org/`
[41] `http://www.eclipse.org/jetty/`
[42] `http://undertow.io/`
[43] Maven queries repositories for libraries, which are defined in dependency-tags. One of the most important ones is `https://mvnrepository.com/`.
[44] `https://en.wikipedia.org/wiki/Classpath_(Java)`
[45] Further information is available here: `http://docs.spring.io/spring-boot/docs/current/reference/html/using-boot-build-systems.html`
[46] `http://hibernate.org/orm/`
[47] `http://www.json.org/`

the data format. JSON is a lightweight, language independent data-interchange format, that is widely used by modern web applications, which perform many asynchronous calls to the backend.

Service Layer

The service layer contains the following three modules:

1. **babelnet**: This is the BabelNet source code from version 2.5, which constructs BabelNet from Lucene[48] indices.
2. **babelnet-wrapper**: The wrapper project overwrites configuration files from the babelnet module to make it work in a web environment.
3. **service**: This module is the very core of this proof of concept implementation. It contains the logic to generate and compare a document and a user models and utilizes BabelNet to explain words that are likely to be unknown. The work outlined in the future Sections 4.3, 4.4 and 4.5 are implemented in this module.

Persistence Layer

The persistence layer consists of the database and the O/R-Mapper, which serves as an abstraction of the database. One of many advantages of an O/R-Mapper is that we can exchange the database easily, e.g. we can use a different database during development and production. The database is completely transparent to the service layer, which only needs to call the API of the O/R-Mapper. Java defines the Java Persistence API (JPA), which is implemented by Hibernate, the library we use.

As a database we use H2[49], a relational open source database that has been written entirely in Java$^{\mathrm{TM}}$. It can be run standalone as a server or as an in-memory database, which is convenient for application testing. Our main entities, which are represented by tables, are “RawDocument”, “DocumentModel”, “UserModel” and “Experiment”. Both document models and user models have been built from text files, e.g. PDF files, which are represented by the “RawDocument” entity. The “Experiment” entity links one document model to one user model and contains the result as a byte-array. The result is a serialized data structure, which can be deserialized using Java Object Serialization (JOS)[50].

Evaluation Module

The standalone evaluation module has dependencies to the service and to the DTO module, but cannot be accessed via RESTful web services. Because it only serves

[48] https://lucene.apache.org/
[49] http://www.h2database.com/html/main.html
[50] https://www.cs.cornell.edu/Info/People/chichao/ccc-ch4.pdf

(internal) experimental purposes, it is not part of the deployed application. We have conducted our experiments using this module, which offers the following functionality:

1. Processing the DBLP xml file and downloading papers from user model candidates (see 4.1.1).
2. Generating document and user models (see 4.3) used for evaluation purposes by using services exposed by the service layer.
3. Performing experiments on user models (see 4.4) and writing statistics to the file system.

We will explain more about the evaluation in Chapter 5.

4.3 Generating Language Models

This section outlines the process of language model generation. It is different for user and document models, because user models are composed of multiple documents, therefore generating them is more complex. The model generation process consists of up to seven steps:

1. **Document parsing**: Parsing the text from the file, e.g. PDF-file[51], or use the already parsed text directly for further processing.
2. **Preprocessing**: Manipulating the raw text to increase quality.
3. **Tokenization**: Segmenting the text into single tokens.
4. **Annotation**: Enriching the tokens with part of speech tags and identifying the named entities (NE) from the text.
5. **Merging**: Merging all document models into a single user model.
6. **Post-Processing**: Performing filtering on a higher level to increase the quality of the language model.
7. **Statistical Computation**: Computing the counts and smoothed probabilities.

Table 4.1 provides an overview about the neccessary steps needed to generate each model. Both models rely on the same algorithms for document parsing, preprocessing, tokenization, annotation and post-processing. In contrast to the user model, we omit the merge operation and the statistical computation step when generating the document model.

[51]We use Apache PDFBox® (`https://pdfbox.apache.org/`)

Table 4.1: Processing steps needed for each language model.

Processing steps	User Model	Document Model
Document Parsing	yes	yes
Preprocessing	yes	yes
Tokenization	yes	yes
Annotation	yes	yes
Merging	yes	no
Post-Processing	yes	yes
Statistical Computation	yes	no

4.3.1 Document Parsing

We parse our PDF documents with a sophisticated open source Java™ solution from The Apache Software Foundation[52] (Apache), namely Apache PDFBox®. This library is also used by Apache Tika™, a software framework focused on automatic media type identification, text extraction, and metadata extraction [MZ11]. Although Apache Tika™ offers more functionality than Apache PDFBox® (it can be used to extract text from over a thousand different file types [Tik]), we cannot use it to extract the content by the pageful in a reliable way. Therefore, we have to use Apache PDFBox® directly.

Listing 4.1 shows how we extract data from an input stream of a PDF file.

Listing 4.1: Text extraction from PDF using Apache PDFBox®

```java
private DocumentDTO extractPdfWithPdfBox(InputStream is) throws IOException
    {
  DocumentDTO documentDTO = new DocumentDTO();

  PDDocument document = PDDocument.load(is);
  Map<Integer, String> pageToContent = new LinkedHashMap<>();
  final int pages = document.getNumberOfPages();

  for (int i = 0 ; i <  pages; i++) {
    PDFTextStripper stripper = new PDFTextStripper();
    stripper.setStartPage(i);
    stripper.setEndPage(i + 1);
    final String text = stripper.getText(document);
    pageToContent.put(i, text);
  }

  documentDTO.setSourceType(SourceType.PDF.name());
  documentDTO.setTitle(document.getDocumentInformation().getTitle());
  documentDTO.setPageCount(pages);
  documentDTO.setText(new PDFTextStripper().getText(document));
  documentDTO.setPages(pageToContent);

  return documentDTO;
```

[52] http://www.apache.org/

```
23 }
```

Below is the explanation of the code:

- **Line 2**: Definition of the Java^{TM} DTO that contains the result of the method.
- **Line 4**: Loading the document stream into a data structure.
- **Line 6**: Obtaining the number of pages of the PDF document.
- **Line 8-14**: Stripping the content of each page separately, so we know which words are on which page. This is important for the PDF-Viewer we introduce in Section 4.5. Without the page information we are not able to display the identified unknown words with respect to the currently viewed page.
- **Line 16-20**: Document meta data like title and pages from the meta data object are set into the DTO. We also set the raw extracted text from the document, as well as the data structure that contains the pages separately.

Now that we have the raw text from the input stream of the PDF file, we can preprocess the text to increase the text quality and therefore the quality of the generated language models.

4.3.2 Preprocessing

The concept of text preprocessing or text normalization has already been introduced in Section 3.2. This step prepares the text for tokenization and includes the following steps:

1. **Removing the bibliographic chapter**: Paper references contain no valuable content and as such must be removed.
2. **Lowercase first word in sentences**: Without this operation we would get many wrong tagged words, because the tagger[53] would for example identify a noun (NN) as a proper noun (NNP)[54].
3. **Handle line breaks and whitespaces**: Reduce all line breaks and multiple spaces to a single white space character.
4. **Remove digits**: Digits are irrelevant for the purpose of finding words that are most likely to be unknown to the user.
5. **Remove markup**: HTML markup will most likely lead to unusable tokens.

[53]`http://www-nlp.stanford.edu/nlp/javadoc/javanlp/edu/stanford/nlp/tagger/maxent/MaxentTagger.html` with the model: english-left3words-distsim.tagger

[54]Example:“I am Bob. Apples are my favorite fruits!”

6. **Handle syllables**: Words that are connected by hyphens because of formating need to be reassembled by omitting the hyphen.

7. **Remove non Unicode characters**: As a final cleanup, we remove any non Unicode characters.

Now the text is ready for the next step, tokenization and annotation.

4.3.3 Tokenization & Annotation

The term tokenization has already been defined in Section 3.2. It is the process of dividing text into a sequence of tokens, which roughly correspond to words.

For tokenization, we use a Standford NLP pipeline consisting of a fast, rule-based tokenizer (the `PTBTokenizer`[55]), which has been already introduced in Section 3.2.1. The `PTBTokenizer` mainly targets formal English writing rathern than SMS-speak, hence it is suitable for our documents. It was initially designed to mimic the Penn Treebank 3[56] (PTB) tokenization [Gro]. The Penn Treebank project selected 2,499 stories from a three year Wall Street Journal (WSJ) collection of 98,732 stories for syntactic annotation [Con].

After we obtained the tokens, we tag them with part of speech tags using a POS tagger (`MaxentTagger`[57]). The POS tagger is the bidirectional dependency network tagger trained on the WSJ Penn Treebank provided with the Stanford NLP distribution. While the training data is considerably different from our own data, to the best of our knowledge there is no POS model specifically created for scientific articles, nor an annotated corpora sufficiently large to train such a model.

Listing 4.2 shows how we combine tokenization and POS tagging.

Listing 4.2: Combination of tokenization and POS tagging using the Stanford CoreNLP library

```java
public Bag<TaggedWordWrapper> articleToTaggedTokens(String text) {
    Bag<TaggedWordWrapper> bag = new HashBag<>();
    final PTBTokenizer<Word> tokenizer = new PTBTokenizer(new StringReader(
        text), new WordTokenFactory(), getOptions());
    final List<Word> tokens = tokenizer.tokenize();
    for (TaggedWord word : tagger.apply(tokens)) {
        bag.add(new TaggedWordWrapper(word));
    }
    return bag;
}
```

[55]http://nlp.stanford.edu/nlp/javadoc/javanlp/edu/stanford/nlp/process/PTBTokenizer.html

[56]https://catalog.ldc.upenn.edu/LDC99T42

[57]http://www-nlp.stanford.edu/nlp/javadoc/javanlp/edu/stanford/nlp/tagger/maxent/MaxentTagger.html

Below is the explanation of the code:

- **Line 2**: Instantiation of the resulting data structure, which is a Bag[58], a collection that counts the number of times an object appears, of `TaggedWordWrappers`. Latter class is a custom class that wraps the POS taggers resulting class `TaggedWord`[59], but overrides the `equals` and `hashcode` functions, so that the `uniqueSet` function of the Bag can be used.
- **Line 3**: Creation of the `PTBTokenizer`, which needs the text, a word token factory and optional options.
- **Line 4**: Triggers the tokenization process, which results in a list of words.
- **Line 5-7**: Application of the `MaxentTagger`, which adds the POS tag to each of those tokens, wrapping the result and saving it into the bag.

In addition to tokenization and POS tagging, we also identify named entities. Again, we use a solution from the Stanford University, namely the Stanford Named Entity Recognizer [FGM05]. It includes the following models[60]:

- **3 class**: Location, Person, Organization
- **4 class**: Location, Person, Organization, Misc
- **7 class**: Location, Person, Organization, Money, Percent, Date, Time

We use the 7 class model trained on the MUC 6 and MUC 7 training data sets, which are available through the Linguistic Data Consortium[61] (LDC). The Message Understanding Conference (MUC) is designed to promote and evaluate research in information extraction [GS96]. One of the tasks of MUC-6 was to identify named entities of the types "People", "Organizations", and "Geographic Locations".

Listing 4.3 shows how we perform named entity recognition (NER).

Listing 4.3: Named Entity Recognition performed with Stanford NER

```java
public Set<CoreLabel> identifyNER(String text) {
    Set<CoreLabel> ners = new LinkedHashSet<>();

    for (List<CoreLabel> coreLabels : classifier.classify(text)) {
        for (CoreLabel coreLabel : coreLabels) {
```

[58]`https://commons.apache.org/proper/commons-collections/javadocs/api-release/org/apache/commons/collections4/Bag.html`

[59]`http://nlp.stanford.edu/nlp/javadoc/javanlp/edu/stanford/nlp/ling/TaggedWord.html`

[60]`http://nlp.stanford.edu/software/CRF-NER.shtml`

[61]`https://www.ldc.upenn.edu/`

```
6               String category = coreLabel.get(CoreAnnotations.
                    AnswerAnnotation.class);
7               if(!SeqClassifierFlags.DEFAULT_BACKGROUND_SYMBOL.equals(
                    category)) {
8                   ners.add(coreLabel);
9               }
10          }
11      }
12      return ners;
13  }
```

Below is the explanation of the code:

- **Line 2**: Defines the result, a set of `CoreLabel`[62] instances. A `CoreLabel` represents a single word and its metadata, like the POS tag or lemma.
- **Line 4**: Identifies the NE and returns them as a list of `CoreLabels`. We use the `CRFClassifier`[63] which classifies a sequence using a Conditional Random Field model. A CRF is a statistical modelling method[64] which can take a context into account and therefore is used in pattern recognition and machine learning (ML) to predict structures. In our case a sequence of lables (NE tags) are predicted for a sequence of input samples (the words in the text). To detect the maximum amount of named entities, we use the 7 class model (english.muc.7class.distsim.crf.ser.gz).
- **Line 6**: Retrieves the category (NE type) from the label. The `CoreLabel` class is internally a typesafe map which contains `Classes` as keys. Metadata is added to the `CoreLabel` via specific annotations (see CoreAnnotations[65]). The NE classifier stores the NE type using the `AnswerAnnotation`.
- **Line 7**: Filters out the lables which contain the default symbol, which is defined as a `String` with the value “0”.
- **Line 12**: Returns the resulting data structure which contains `CoreLabel`, classified as one of the seven classes mentioned above.

We hold those NE in a separate list and use them in conjunction with our tokens in the post-processing step.

[62] `http://nlp.stanford.edu/nlp/javadoc/javanlp/edu/stanford/nlp/ling/CoreLabel.html`
[63] `http://nlp.stanford.edu/nlp/javadoc/javanlp/edu/stanford/nlp/ie/crf/CRFClassifier.html`
[64] `https://en.wikipedia.org/wiki/Conditional_random_field`
[65] `http://nlp.stanford.edu/nlp/javadoc/javanlp/edu/stanford/nlp/ling/CoreAnnotations.html`

4.3.4 Post-Processing

Before we construct a language model, we want to make sure that its foundation (the bag of words) is sound. Therefore we perform sanity checks, which remove entries that do not satisfy them. The post-processing step takes a bag of `TaggedWordWrappers` as input, performs the following operations (which are listed below) in the given order and returns a smaller bag of words, which only contains approved entries.

1. **Remove short character sequences**: Removes tokens with a length less than 4 letters. This operation removes all sorts of punctuation, which are irrelevant for our task, and short words (like “I”, “am” or “the”), which are most likely known to the user.
2. **Remove words not in BabelNet**: Words, that cannot be found in the almost 10 million BabelNet synsets, are considered to be artefacts, which we omit.
3. **Remove proper nouns/NE**: We remove proper nouns (words tagged with “NNP” or “NNPS”) and NE in two steps. First, we remove NE identified in the previous section if we cannot find them in the FWL. For example, we might encounter the token “Francisco” (from “San Francisco”), which is not part of the FWL, therefore we discard it. The organization “Apple” on the other hand is included as a noun in our FWL, hence we keep this entry. Second, because our FWL does not contain any proper nouns, we remove them if we cannot find them in the list without their tag.

After those operations, the bag contains only words that are at least four characters long, can be found in BabelNet, and are not considered as proper nouns or named entities. In the next section, we describe the creation of our probabilistic language models.

4.3.5 Statistical Computation

The goal of this step is to smooth the unigram language model and to compute the average word rank order, which we interpret as the language level of the user. The probability of a word in our user model is the word frequency divided by the sum of words contained in that model. Because the user model does not reflect all the known words from the user, we have to take possibly missing words into account. Smoothing is the technique that takes some probability mass away from events (e.g. word occurrences) and reserves it for future events to avoid zero estimates. We use `SimpleGoodTuring`[66] (SGT), which is implemented in the Stanford CoreNLP library. We already mentioned it in Section 3.1.

After we computed the smoothed probabilities, we calculate the average rank order of words within the model to estimate the models' difficulty. Based on this value, we decide

[66]`http://nlp.stanford.edu/nlp/javadoc/javanlp/edu/stanford/nlp/stats/SimpleGoodTuring.html`

whether words from the document model are below or above the user's language level. The word rank order is taken from the frequent word list (see Section 4.1.2).

4.4 Comparing Language Models

The model comparison algorithm has already been introduced in Section 3.3. Because the implementation contains more than 100 lines of code, we will break it down in two pieces, each denoted as a placeholder. Listing 4.4 shows the method with two placeholders, which we explain later.

Listing 4.4: Model comparison algorithm with two placeholder

```java
public ComparisonResultDTO compareWithLemma(UserModelDTO userModel,
    DocumentModelDTO documentModel) {
    Bag<TaggedWordWrapper> unknownWords = new HashBag<>();
    Bag<TaggedWordWrapper> knownWordsUserModel = new HashBag<>();

    final double avgRank = userModel.getAvgRank();

    for (TaggedWordWrapper docWrapper : documentModel.getModel().uniqueSet
        ()) {
        final String word = docWrapper.get().value().toLowerCase();
        final String pos = docWrapper.get().tag();

        final boolean isInMu = userModel.getModel().contains(docWrapper);

        if (isInMu) {
            knownWordsUserModel.add(docWrapper);
            continue;
        }

        <placeholder_1> -> see Listing 4.5

        <placeholder_2> -> see Listing 4.6
    }

    final Set<TaggedWordWrapper> allWords = new HashSet<>(documentModel.
        getModel().uniqueSet());
    allWords.removeAll(new HashBag<>(unknownWords));
    allWords.removeAll(new HashBag<>(knownWordsUserModel));

    ComparisonResultDTO result = new ComparisonResultDTO(unknownWords);
    result.setKnownWordsUserModel(knownWordsUserModel);
    result.setKnownWordsCollection(new HashBag<>(allWords));

    return result;
}
```

Each placeholder has its own set of variables, which it needs to make decisions whether to put the observed word from the document model into the set of known or unknown words.

Below is the explanation of the relevant parts of the code:

- **Line 1**: Definition of the method, which accepts a user model and a document model as input.
- **Line 5**: Retrieval of the user model's underlying computed average rank (language level). This number helps us to distinguish possible known words from unknown ones. In future code sections, we will compare this value with the rank of words from the document model, which we retrieve from the FWL.
- **Line 11-15**: If M_u contains exactly the same word (word + POS tag) as the `TaggedWordWrapper` (w), we add w to the known words and skip further processing. We will proceed to the next word in M_d.
- **Line 18+19**: Will be shown in Listing 4.5 and 4.6.
- **Line 23-29**: For evaluation purposes, we document which component (M_s or M_g) contains which known words. We compute known words from the general component (our FWL M_g) as all words minus the unknown words minus the known words from the user model M_u.

Before we explain the code of the placeholders, we introduce our `ProtocolService`. We use it to protocol unknown words and their reason for being unknown to identify anomalies. The `protocolUnknownWord` method takes three arguments, the `Reason`, word or lemma and the POS tag or blank in case of a lemma.

The first placeholder handles the case that w cannot be found in M_g with its POS tag.

Listing 4.5: Model Comparison Algorithm - w cannot be found in M_g with its POS tag

```java
final boolean isInFwlListWithPoS = fwlService.isInFwlList(word, pos);
final boolean isInFwlList = fwlService.isInFwlList(word);
final boolean isWordsLowestRankGtAvgRank = fwlService.getLowestRank(word) >
    avgRank;

String lemma;
boolean isLemmasLowestRankGtAvgRank;
boolean isLemmaInFwlList;
lemma = getLemma(word);
isLemmaInFwlList = fwlService.isInFwlList(lemma);
isLemmasLowestRankGtAvgRank = fwlService.getLowestRank(lemma) > avgRank;

final boolean useBabelNet = Configuration.USE_BABELNET;

if (!isInFwlListWithPoS && isInFwlList && isWordsLowestRankGtAvgRank &&
        isLemmasLowestRankGtAvgRank) {
    unknownWords.add(docWrapper);
    protocolService.protocolUnknownWord(Reason.LEMMA_RANK_GT_AVG_RANK,
        lemma, "");
        continue;
18 }
19
20 if (!isInFwlListWithPoS && !isInFwlList) {
21     if (isLemmaInFwlList) {
22         if (isLemmasLowestRankGtAvgRank) {
23             unknownWords.add(docWrapper);
24             protocolService.protocolUnknownWord(Reason.
                   LEMMA_RANK_GT_AVG_RANK, lemma, "");
25         }
26     } else {
27         unknownWords.add(docWrapper);
28         protocolService.protocolUnknownWord(Reason.NOT_IN_COLLECTION_NO_NE,
               word, pos);
29     }
30     continue;
31 }
```

Below is the explanation of the relevant parts of the code:

- **Line 1-3**: Preparing variables for decision making. A word can be contained in the FWL multiple times, e.g. “walls” as noun (NN2) with the rank 1,520 or as verb (VVZ) with the rank 89,672.
- **Line 5-10**: Similar to lines 1-3, we gather that information for the lemma of the word, which we obtain from the FWL.
- **Line 14-18**: The word w is added to the collection of unknown words if it cannot be found with its POS tag in the FWL, but it’s in general contained (without tag) and the rank of the word and its lemma is greater than the average rank. This condition is very weak, because if the rank of the lemma is higher than the average rank, we do not add the word to the collection of unknown words.
- **Line 20-31**: Here we know that the word is not contained in the word column of the FWL. If we cannot find the word’s lemma in the FWL, we add it to the list of unknown words. We do the same if we can find the lemma, but its rank is greater than the average rank (i.e. the language level of the user).

The next placeholder handles the case that we can find the word (with its POS tag) in the FWL, but even the rank order of its lemma is bigger than the average rank.

Listing 4.6: Model Comparison Algorithm - w can be found in M_g with its POS tag but the lemmas rank is greater than the average rank

```
final long wordRank = fwlService.getRank(word, pos);
final boolean wordRankWithPoSGtAvgRank = wordRank > avgRank;
final boolean wordLowestRankGtAvgRank = fwlService.getLowestRank(word) >
    avgRank;
lemma = fwlService.getLemma(word.toLowerCase(), pos);
isLemmasLowestRankGtAvgRank = fwlService.getLowestRank(lemma) > avgRank;
```

```
 6
 7 if (isInFwlListWithPoS && wordRankWithPoSGtAvgRank &&
       wordLowestRankGtAvgRank && isLemmasLowestRankGtAvgRank) {
 8     if (useBabelNet) {
 9         final POS posForTag = BabelNetUtils.getPosForTag(pos);
10         if (POS.ADJECTIVE.equals(posForTag) || POS.VERB.equals(posForTag))
               {
11             final String antonymWord = getAntonym(Language.EN, word,
                   posForTag);
12             if (antonymWord != null && fwlService.getLowestRank(antonymWord
                   ) > avgRank) {
13                 unknownWords.add(docWrapper);
14                 protocolService.protocolUnknownWord(Reason.
                       ANTONYM_RANK_GT_AVG_RANK, antonymWord, pos);
15             } else {
16                 unknownWords.add(docWrapper);
17                 protocolService.protocolUnknownWord(Reason.
                       LEMMA_RANK_GT_AVG_RANK, lemma, pos);
18             }
19         } else {
20             unknownWords.add(docWrapper);
21             protocolService.protocolUnknownWord(Reason.
                   LEMMA_RANK_GT_AVG_RANK, lemma, "");
22         }
23     } else {
24         unknownWords.add(docWrapper);
25         protocolService.protocolUnknownWord(Reason.LEMMA_RANK_GT_AVG_RANK,
               lemma, "");
26     }
27 }
```

Below is the explanation of the relevant parts of the code:

- **Line 7**: We pursue further investigation if we can find the word (with its POS tag) in the FWL, but even its lemma is greater than the average rank.
- **Line 9**: Converting the words POS tag into the BabelNet POS tag. BabelNet only maintains four POS tags (noun, verb, adverb and adjective).
- **Line 10-18**: In case of a verb or adjective, we retrieve the antonym (if exists) and check if its rank is greater than the average rank. For example, the rank of the word “nonspecific” is 38,128, whereas the rank of “specific” is 867. If latter rank is lower than the average rank, then we assume that the related word “nonspecific” is most likely to be known to the user.
- **Line 19-22**: In case we cannot find the lemma, we add the word to the set of unknown words.
- **Line 23-26**: In case we do not use BabelNet for comparison, we add the word to the set of unknown words.

After we run an experiment, we generate a protocol using the `ProtocolService` which we briefly explained above. For example, we compared a user model containing around 70,000 tokens and 3,500 types with a document model which contains around 2,250 tokens and 700 types. By examining the protocol, we know that we found exactly 173 possible unknown word tokens, from which 145 result from the reason "LEMMA_RANK_GT_AVG_RANK", 2 from "ANTONYM_RANK_GT_AVG_RANK", and 26 from "NOT_IN_COLLECTION_NO_NE". It further shows that from 147 unknown words, 21 were found in the FWL without the POS tag and the rest (126) with the POS tag. The remainder (26 words) were not part of the FWL. Based on that information, we can analyze why words could not be found with their POS tag in the FWL. Maybe some words were not correctly classified, e.g. as a proper noun instead of an ordinary noun. If that is the case, we can try to improve our preprocessing step to handle such cases. Next section covers the explanation part of possible unknown words.

4.5 Explaining Unknown Words

In this section, we explain how we support the user in understanding the analyzed document. Based on the identified words in the previous section, we query our local BabelNet instance for additional information like:

- Glossary entries
- Images
- Translations
- The BabelNet word identifier, which we use to link to the word on the Babelnet website[67].

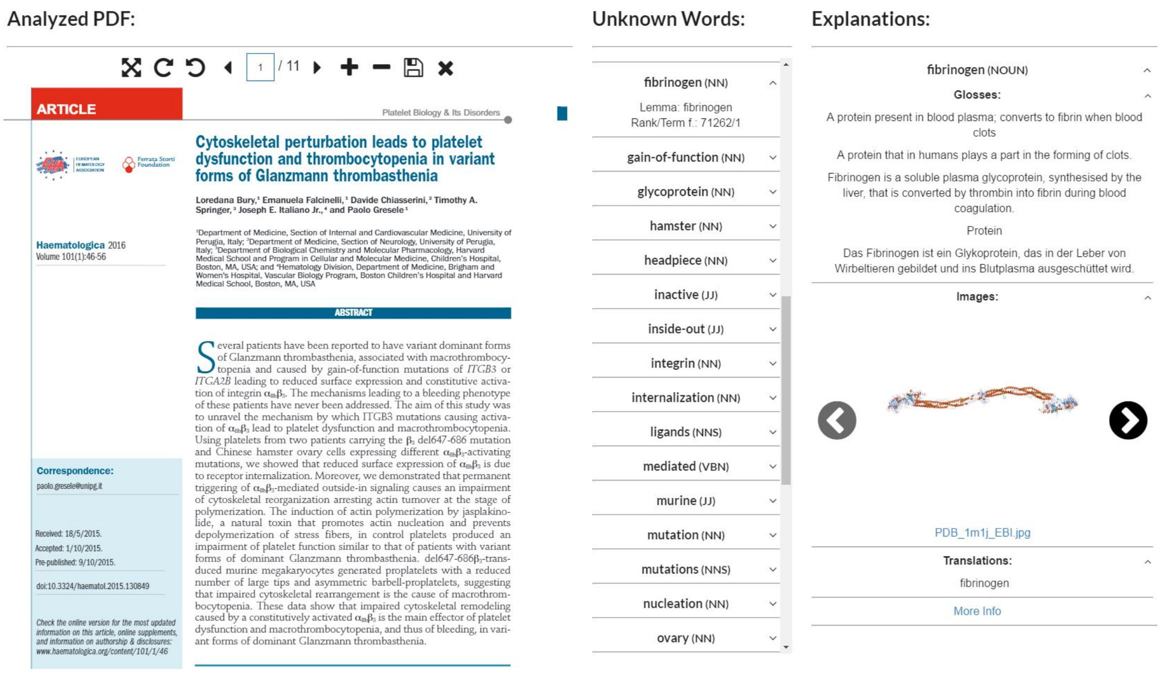
BabelNet returns us the content in the requested language. In the following presented case, we chose the languages of German and English. In addition to those resources, we also provide information from the frequent word list (FWL) like the lemma, part of speech (POS), rank order and also the frequency of the word within the analyzed document (term frequency). Figure 4.6 shows the tool we provide to the user to support him or her in understanding a challenging document (in this case a medical journal article). The illustration is divided into the following three sections:

1. **Analyzed PDF**: We use PDF.js, a general-purpose, web standards-based platform for parsing and rendering PDFs[68], to display the analyzed document. We extended the viewer to provide basic operations like page rotation and go to next or previous page.

[67] `www.babelnet.org`
[68] `https://mozilla.github.io/pdf.js/`

2. **Unknown Words**: The second section displays a list of unknown words from the current page. Entries in this list can be expanded to show information from the FWL and the number of occurences (as described above). In the given example, we chose the word “fibrinogen”, which lemma is “fibrinogen”, the rank is 71,262 and the term frequency is 1. As in a dictionary, the unknown words are sorted alphabetically.

3. **Explanations**: If the user clicks on an unknown word, we request the above mentioned resources from BabelNet. First, we display the glossary entries in the requested languages (if available), followed by the images, which we present in a space saving image viewer, then the direct translation of the term and in the end, the pointer to the word in BabelNet. BabelNet provides additional information like pronunciation (sound) and it visualizes relations to other words[69].


Analyzed PDF:

1 / 11

ARTICLE

Platelet Biology & Its Disorders

Cytoskeletal perturbation leads to platelet dysfunction and thrombocytopenia in variant forms of Glanzmann thrombasthenia

Loredana Bury,[1] Emanuela Falcinelli,[1] Davide Chiasserini,[2] Timothy A. Springer,[3] Joseph E. Italiano Jr.,[4] and Paolo Gresele[1]

[1]Department of Medicine, Section of Internal and Cardiovascular Medicine, University of Perugia, Italy; [2]Department of Medicine, Section of Neurology, University of Perugia, Italy; [3]Department of Biological Chemistry and Molecular Pharmacology, Harvard Medical School and Program in Cellular and Molecular Medicine, Children's Hospital, Boston, MA, USA; and [4]Hematology Division, Department of Medicine, Brigham and Women's Hospital, Vascular Biology Program, Boston Children's Hospital and Harvard Medical School, Boston, MA, USA

Haematologica 2016
Volume 101(1):46-56

ABSTRACT

Several patients have been reported to have variant dominant forms of Glanzmann thrombasthenia, associated with macrothrombocytopenia and caused by gain-of-function mutations of *ITGB3* or *ITGA2B* leading to reduced surface expression and constitutive activation of integrin $\alpha_{IIb}\beta_3$. The mechanisms leading to a bleeding phenotype of these patients have never been addressed. The aim of this study was to unravel the mechanism by which ITGB3 mutations causing activation of $\alpha_{IIb}\beta_3$ lead to platelet dysfunction and macrothrombocytopenia. Using platelets from two patients carrying the β_3 del647-686 mutation and Chinese hamster ovary cells expressing different $\alpha_{IIb}\beta_3$-activating mutations, we showed that reduced surface expression of $\alpha_{IIb}\beta_3$ is due to receptor internalization. Moreover, we demonstrated that permanent triggering of $\alpha_{IIb}\beta_3$-mediated outside-in signaling causes an impairment of cytoskeletal reorganization arresting actin turnover at the stage of polymerization. The induction of actin polymerization by jasplakinolide, a natural toxin that promotes actin nucleation and prevents depolymerization of stress fibers, in control platelets produced an impairment of platelet function similar to that of patients with variant forms of dominant Glanzmann thrombasthenia. del647-686β_3-transduced murine megakaryocytes generated proplatelets with a reduced number of large tips and asymmetric barbell-proplatelets, suggesting that impaired cytoskeletal rearrangement is the cause of macrothrombocytopenia. These data show that impaired cytoskeletal remodeling caused by a constitutively activated $\alpha_{IIb}\beta_3$ is the main effector of platelet dysfunction and macrothrombocytopenia, and thus of bleeding, in variant forms of dominant Glanzmann thrombasthenia.

Correspondence:
paolo.gresele@unipg.it

Received: 18/5/2015.
Accepted: 1/10/2015.
Pre-published: 9/10/2015.

doi:10.3324/haematol.2015.130849

Check the online version for the most updated information on this article, online supplements, and information on authorship & disclosures: www.haematologica.org/content/101/1/46

Unknown Words:

fibrinogen (NN)
Lemma: fibrinogen
Rank/Term f.: 71262/1

gain-of-function (NN)
glycoprotein (NN)
hamster (NN)
headpiece (NN)
inactive (JJ)
inside-out (JJ)
integrin (NN)
internalization (NN)
ligands (NNS)
mediated (VBN)
murine (JJ)
mutation (NN)
mutations (NNS)
nucleation (NN)
ovary (NN)

Explanations:

fibrinogen (NOUN)

Glosses:

A protein present in blood plasma; converts to fibrin when blood clots

A protein that in humans plays a part in the forming of clots.

Fibrinogen is a soluble plasma glycoprotein, synthesised by the liver, that is converted by thrombin into fibrin during blood coagulation.

Protein

Das Fibrinogen ist ein Glykoprotein, das in der Leber von Wirbeltieren gebildet und ins Blutplasma ausgeschüttet wird.

Images:

PDB_1m1j_EBI.jpg

Translations:

fibrinogen

More Info

Figure 4.6: Explaining unknown words with data from BabelNet

[69]Fibrinogen: `http://babelnet.org/synset?word=bn:00032665n&details=1&lang=EN`

4.6 Summary

This chapter covered the technical realization of our proposed method from the previous chapter. We started with a description of the three data sources we use, namely, the DBLP computer science bibliography, a very comprehensive frequent word list of over 100k words based on American English and BabelNet. After that, we described our software architecture, which consists of a client and a server application. The frontend and backend application communicate over http via RESTful services. And, we further outlined our strict structuring of the backend into layers and Maven modules, where each module is responsible for a specific area. The core module is the service module, which contains services that create our language models and compares those to identify the unknown words.

Due to our modular approach and utilization of services, we are very flexible with adding and removing data sources or functionality. We are aware that this proof of concept has some limitations, which we outline in Section 6.1. Nevertheless, it demonstrates the integration of linguistic and semantical sources and libraries into a mature software framework built for business applications. It contains a workflow to automatically generate language models from text, like PDF files, compare them with each other and write the resulting outcome into the database, where it can be retrieved by the client application. Our introduced web client displays the analyzed document page by page and shows the detected unknown words and their explanation with respect to the current page. How well our approach works in terms of identified words which are likely to be unknown to the user is the topic of the next chapter.

CHAPTER 5

Evaluation

This chapter describes the evaluation of our proposed method to find words that are most likely to be unknown to the user. We describe the used data (our computed test set) in Section 5.1.

In order to validate our work we have considered both quantitative and qualitative experiments. In the absence of user judgments for known or unknown terms, the quantitative experiment will observe whether the method identifies a significantly higher number of unknown terms in documents belonging to the medical domain (i.e. a domain different from that of the users) than in documents belonging to the computer science (CS) domain (see Section 5.2).

In terms of qualitative observations, we have manually considered the terms identified for one document from each domain and for one fixed user model (see Section 5.3). For each experiment, we display the whole list of words which have been identified to be most likely unknown to the user and comment on the result.

We discuss our findings and the data at hand in Section 5.4. First, we explore the properties of our user models, such as tokens, types and the computed average word rank. Second, we investigate the correlation of types and the average word rank within the user model and their influence on the amount of unknown words.

The main observations are summarized in Section 5.5.

5.1 Test Set

This section describes the test set that is used to evaluate this work. We divide this set into user models and documents models. Throughout the evaluation, which consists of two kinds of experiments, namely, the equi-domain experiments and the cross-domain experiments, we use the same ten user models. Those models are introduced in Section 5.1.1. In Section 5.1.2, we describe the used document models.

5.1.1 User Models

Before we describe the chosen user models, we explain how we chose their underlying document collection. We selected the documents, used to generate this test set, from the DBLP Computer Science Bibliography (see Section 4.1.1). Due to its structure and contents, DBLP is a good starting point to choose authors, from which we generate user models. A user model consists of a preprocessed (normalized) collection of documents from a certain author. We filtered the DBLP XML records by the following three criteria:

1. As explained earlier in Section 4.1.1, we only consider journal articles.
2. To avoid legal problems when releasing our data, documents must be freely available for download.
3. For our purposes, only authors with at least 25 publications are appropriate candidates for generating user models. We assume that more documents, hence more words, lead to a better model.

In the first step, all 1,409,695 `article` XML elements have been extracted from the main XML file and written into a separate file. Articles itself contain other elements[1], like `author` or `ee`. The `author` element contains the name of the author and can appear multiple times in an article element. In the next step, we grouped articles by the domain name of their `ee` element. The `ee` element (short for electronic edition) provides a link to an online version of the paper or to a DBLP web page for the corresponding entry [Ley09a]. Table 5.1 shows the top 10 domains and whether it is possible to download articles from those domains for free.

Table 5.1: Grouped content of `ee` element by domain.

Domain	Count	Free download available
http://dx.doi.org/	2,270,860	No
http://doi.ieeecomputersociety.org/	346,315	No
http://doi.acm.org/	274,414	No
http://arxiv.org/	96,820	Yes
http://dl.acm.org/	36,010	No
http://ieeexplore.ieee.org/	27,458	Yes
http://www.isca-speech.org/	21,002	Yes
http://aisel.aisnet.org/	16,936	No
http://www.aaai.org/	16,807	Yes
http://search.ieice.org/	11,023	No

In total, 20 authors were selected, each of them was listed as an author for at least 25 articles. For our evaluation purposes, we chose ten authors randomly and downloaded their papers (see Table 5.2).

[1]For more details download dblp.dtd from `http://dblp.uni-trier.de/xml/`

Table 5.2: Chosen user model candidates from DBLP.

Author's Name	# Articles	Author's Name	# Articles
Amir K. Khandani	58	Lei Chen	29
Andrea Montanari	88	Mita Nasipuri	67
Bertrand Meyer	28	Philippe Wenger	86
Chin-Chen Chang	28	Sanjeev R. Kulkarni	29
Jim Gray	52	Stefan Szeider	46

The whole collection of 511 articles in PDF format has been made available to the public and can be downloaded[2]. From each author, we possess at least 28 documents which we can use to build our user models. To create this test set, we have randomly selected 25 of their publications (see Table 5.3). We provide the mapping from the user model (M_u#1-10) to the author in Table 2 of the Appendix).

Table 5.3: User models statistics generated from 25 random papers.

Author	Total initial tokens	Tokens after preprocess.	Total removed tokens	Total tokens in M_s	Total types in M_s	Average word rank
M_u#1	328,293	198,800	257,935	70,358	4,567	12,740
M_u#2	523,962	306,025	415,686	108,276	7,323	13,548
M_u#3	453,248	230,200	309,029	144,219	10,010	14,597
M_u#4	183,281	85,400	138,078	45,203	4,461	10,944
M_u#5	218,356	99,675	130,161	88,195	10,729	14,079
M_u#6	376,807	188,450	262,556	114,251	7,609	12,499
M_u#7	133,771	53,725	87,846	45,925	5,210	11,764
M_u#8	168,859	82,225	125,462	43,397	4,806	12,132
M_u#9	273,245	146,350	198,005	75,240	6,636	12,667
M_u#10	427,962	221,625	336,354	91,608	4,956	12,154
Average	308,778	161,247	226,111	82,667	6,630	12,711
σ	127,194	76,865	102,128	31,709	2,163	1,041

Table 5.3 contains the following information about our created user models:

- **Total initial tokens**: This is the amount of tokens we get from the `PTBTokenizer` for the parsed document. We do not directly use those tokens, because we perform preprocessing on the text.
- **Tokens after preprocessing**: The preprocessing has already been described in Section 4.3.2. We remove many tokens by discarding the bibliographic chapter, merging multiple occurences of whitespaces into a single one and removing digits.

[2]`https://public@bitbucket.org/alexej_strelzow/thesis.git`

- **Total removed tokens**: This includes all the removed tokens from the preprocessing step, the removed named entities and tokens removed during execution of the post-processing step (see Section 4.3.4).

- **Total tokens in M_s**: The total tokens in the user model are the total initial tokens minus the total removed tokens. In most of our cases, this is less than one third of the initial size of the model.

- **Total types in M_s**: Those are the unique tokens (or distinct words) of the user model. In most of our cases, this is less than 10 percent of the total tokens.

- **Average word rank**: This value is crucial for the model comparison algorithm as it describes the user's language level, which we introduced in Section 4.4. The higher the value, the more complex (in terms of frequent words) the model is.

In the next section, we briefly describe the used document models.

5.1.2 Document Models

For the equi-domain experiments, we use the same documents as in the previous section. From the collection of 511 PDF documents, we randomly choose two documents from each author, resulting in a total amount of 18 documents. The 18 medical papers used for the cross-domain experiments can be found on PubMed[3] using the digital object identifier[4] (DOI) from Table 5.4.

Table 5.4: The eighteen DOI of medical papers we used.

10.1056/NEJMoa1503479	10.1186/s13148-016-0196-3
10.1056/NEJMoa1504542	10.1186/s13148-016-0182-9
10.1056/NEJMoa1506119	10.1186/s13148-016-0197-2
10.1056/NEJMoa1506583	10.1186/s13148-016-0191-8
10.3322/caac.21333	10.1186/s13148-016-0192-7
10.4049/jimmunol.1500696	10.1186/s12929-015-0180-9
10.1093/jnci/djv387	10.1186/s13148-016-0174-9
10.3324/haematol.2015.130849	10.1186/s13148-016-0175-8
10.3324/haematol.2015.135921	10.1007/s00280-015-2850-4

PubMed is a service of the US National Library of Medicine[5] that provides free access to medical journal articles[6].

[3]`http://www.ncbi.nlm.nih.gov/pubmed/`
[4]`https://www.doi.org/`
[5]`https://www.nlm.nih.gov/`
[6]For more information: `https://www.nlm.nih.gov/services/pubmed.html`

5.2 Quantitative Results

This section outlines the results from the equi-domain and the cross-domain experiments separately.

5.2.1 Equi-Domain Experiments

This experiment compares each user model with a set of 18 randomly chosen papers (2 from each author) from the same domain. Those 18 document models have on average 3,611 tokens and 1,032 types. Table 5.5 shows the results of the first experiment. The colums have the following meaning:

- **Avg. tokens in M_d**: This denotes the average value of tokens of the document model (after preprocessing and post-processing), computed over 18 documents. The standard deviation shows that the documents vary around 50 percent in size. We have some papers with more than 5,000 tokens and some with less than 2,000 tokens.
- **Avg. types in M_d**: This is analogue to the previous column, only for unique tokens.
- **Avg. unknown words**: The amount of words which are probably unknown to the user are shown here. The standard deviation shows us, that this is highly variable ($\sigma > 0.5$ x word-count).
- **Avg. known words (M_s)**: In Section 4.4, we outlined how we distinguish between known words. Our user model is based on two components, the domain specific one (M_s), which we build from the PDF document collection and the general component (M_g), which we construct from our FWL. This column contains the quantity of words from M_d, which we found in M_s. We only considered an exact match (word + POS tag).
- **Avg. known words (M_g)**: The amount of words, we found in the FWL (sometimes with its lemma), which have a higher rank than the average rank of M_d, are outlined here.

Table 5.5 shows that the proportion of unknown word types is on average below ten percent. The domain specific component (M_s) of our user model, which we built from the user's document collection, identified on average three times more known words than the general component (M_g), which we use to patch M_s with missing general terms. This is the case, because we first look up the words in M_s, before we query M_g for it. We do this, because we know that words found in M_s are truly known to the user. In contrary to M_s, M_g is an estimated component which is derived from M_s and therefore not genuinely user specific. Words identified as known by M_g might be unknown words.

Table 5.5: Comparison results of M_u and M_d, both from the same domain (CS). The shown values are averages and standard deviations (σ), computed over 18 runs (one M_u and 18 M_d).

Author/ Standard deviation	Average tokens in M_d	Average types in M_d	Average unknown words	Average known words (M_s)	Average known words (M_g)
M_u#1	3,617	1,037	102	678	256
σ	1,834	423	64	230	170
M_u#2	3,631	1,026	81	777	168
σ	1,828	424	58	277	123
M_u#3	3,426	972	72	775	124
σ	1,631	374	35	278	76
M_u#4	3,805	1,076	140	667	267
σ	1,725	400	70	190	156
M_u#5	3,460	964	95	726	142
σ	1,767	356	50	246	68
M_u#6	3,632	1,042	95	798	148
σ	1,797	416	52	281	101
M_u#7	3,873	1,091	126	705	260
σ	1,639	381	63	199	140
M_u#8	3,698	1,057	120	647	289
σ	1,751	404	64	200	165
M_u#9	3,408	1,013	90	754	169
σ	1,656	416	59	255	132
M_u#10	3,560	1,038	108	683	246
σ	1,739	415	61	235	149
Average	3,611	1,032	103	721	207
σ	1,744	403	62	246	145

5.2.2 Cross-Domain Experiments

This experiment compares the same user models as before with a fixed set of 18 papers from the medical domain. The 18 document models had on average 3,199 tokens and 1,024 types (see the identical values for the document properties in Table 5.6). By comparing the average word tokens (3,611 vs. 3,199) and word types (1,032 vs. 1,024) of both experiments, we can see that the setting is similar. The standard deviation on the other side, tells us that the medical documents we picked do not vary as much in size, as the documents chosen for the first experiment.

Table 5.6 shows that the number of unknown words is significantly higher (almost double) than in the prior experiment, where the documents belong to the same domain. It also shows that the proportion of unknown word types is on average slightly below twenty percent, which is almost twice the amount compared to the first experiment. Furthermore,

Table 5.6: Comparison results of M_u from the CS domain and M_d from the medical domain. The shown values are averages and standard deviations (σ), computed over 18 runs (one M_u and 18 M_d).

Author/ Standard deviation	Average tokens in M_d	Average types in M_d	Average unknown words	Average known words (M_s)	Average known words (M_g)
M_u#1	3,199	1,024	202	461	361
σ	763	199	47	103	80
M_u#2	3,199	1,024	187	579	257
σ	763	199	44	129	58
M_u#3	3,199	1,024	171	627	225
σ	763	199	39	144	51
M_u#4	3,199	1,024	217	462	345
σ	763	199	50	98	77
M_u#5	3,199	1,024	192	591	240
σ	763	199	46	133	52
M_u#6	3,199	1,024	193	592	239
σ	763	199	45	134	53
M_u#7	3,199	1,024	205	532	286
σ	763	199	48	117	68
M_u#8	3,199	1,024	204	461	359
σ	763	199	48	102	81
M_u#9	3,199	1,024	194	554	275
σ	763	199	45	132	56
M_u#10	3,199	1,024	201	471	351
σ	763	199	48	114	72
Average	3,199	1,024	197	533	294
σ	763	199	48	136	84

the domain specific component (M_s) identified on average a bit less than two times more known words than the general component (M_g). In the previous experiment, we observed a ratio slightly above 3 to 1.

Without any knowledge of the qualitative results, these observations can have two causes. First, a significant increase of unknown domain specific words (medical terms) increased the unknown word count and lead to a shift of the known word ratio. Second, a significant increase of unknown general terms increased the unknown word count and lead to a shift of the known word ratio. Our intuition tells us that the first cause is the most likely one, because we are dealing with another domain than before. A qualitative analysis of those results, which is presented in next section, will provide more insights.

5.3 Qualitative Results

This section analyzes the result of one experiment from one domain each. For both experiments, we chose the user model we computed from M_u#1. The average word rank we have computed for this model is 12,740. We present the log entry, which contains statistics about the compared models and the words, which were classified as possibly unknown to the user model. Words are presented with their POS tag. Their rank, if available, is always greater than the average rank of the user model. The presentation of the unknown word and its metadata follows following format: <word>#<POS> (rank).

5.3.1 Equi-Domain Experiments

For the qualitative analysis of this experiment, we used the paper with the title "Distance-join: pattern match query in a large graph database"[7]. The results are displayed below.

```
Size of tokens/types in user model:     70358 / 4567
Size of tokens/types in document model:  2266 /  699
Number of unknown words identified:        45

Of the 45 words identified as unknown:
1 word not in the general model (FWL):
  shortest-path#NN

16 words found in the general model (FWL) without considering
the POS tag:
  NULL#NN          (19000)     NULL#NNP          (19000)
  QUERY#NNP        (18048)     Query#NNP         (18048)
  Theorems#NNPS    (48005)     Vertex#NNP        (57064)
  Vertices#NNPS    (62477)     hash#JJ           (23658)
  hash#VB          (23658)     his/her#NN        (15856)
  query#VB         (18048)     redistribute#VB   (31921)
  republish#VB     (78225)     s/he#NN           (32364)
  traversal#JJ     (68860)     vertex#VB         (57064)

28 words found in the general model (FWL) considering also
the POS tag:
  Estimation#NN    (13806)     Lout#NN           (47582)
  Query#NN         (18048)     Selectivity#NN    (31040)
  datasets#NNS     (54849)     embedding#NN      (37457)
  fictitious#JJ    (21520)     hash#NN           (23658)
  informative#JJ   (14224)     metabolic#JJ      (14460)
  nested#JJ        (34745)     operand#NN        (64186)
```

[7] http://dl.acm.org/citation.cfm?id=1687727

operands#NNS	(50545)	predicates#NNS	(61170)
prohibitive#JJ	(26167)	pruned#VBN	(46606)
pruning#NN	(20475)	queries#NNS	(18734)
query#NN	(18048)	reachability#NN	(84041)
relational#JJ	(13835)	selectivity#NN	(31040)
tamer#JJR	(47396)	traversal#NN	(68860)
twig#NN	(21169)	usefulness#NN	(13095)
verification#NN	(14492)	well-studied#JJ	(58446)

In terms of qualitative observations, we can note that in the case of the document belonging to the same domain, the unknown words (e.g.*shortest-path*, *query*, *hash*, *Vertex* or*datasets* tended to be domain specific terms. General terms in the set of frequent American English terms (M_g) that rank below the author's average rank are for example *his/her*, *s/he*, *fictitious*, *informative*, *prohibitive*, *tamer*, *twig*, *Lout*, *Selectivity*, *metabolic*, *usefulness*, and *well-studied.*

We would group the 35 unique words (types) as follows into general and domain specific terms:

- **General terms**: *embedding*, *estimation*, *fictitious*, *his/her*, *informative*, *lout*, *metabolic*, *nested*, *operand*, *operands*, *predicates*, *prohibitive*, *reachability*, *redistribute*, *relational*, *republish*, *s/he*, *selectivity*, *tamer*, *theorems*, *twig*, *usefulness*, *verification*, *well-studied*
- **Domain specific terms**: *datasets*, *hash*, *null*, *pruned*, *pruning*, *queries*, *query*, *shortest-path*, *traversal*, *vertex*, *vertices*

We could probably move some more words from the domain specific group to the general group or vice versa, but the message is clear to us: The majority of unknown words tend to be originating from the general domain. Another observation we could make is that the domain specific unknown words are describing the topic of the paper well enough to guess what it might be about. The analyzed paper addresses pattern match problems over a large data graph. Shortest-path queries and pruning strategies are also mentioned. Indeed, it has something to do with the above mentioned domain specific words like: graph, queries and the shortest-path between vertices.

5.3.2 Cross-Domain Experiments

For the second analysis, we used the medical paper with the title "Telomerase Inhibitor Imetelstat in Patients with Essential Thrombocythemia"[8]. Below is the result of this experiment.

[8]http://www.nejm.org/doi/full/10.1056/NEJMoa1503479#t=article

```
Size of tokens/types in user model:     70358 / 4567
Size of tokens/types in document model:  2797 /  905
Number of unknown words identified:       176

Of the 176 words identified as unknown:
26 not in the general model (FWL):
  CALR#NN                Hydroxyurea#NN        Hypokalemia#NN
  JAK-STAT#NN            calreticulin#NN       clin#NN
  covalently#RB          cytopenia#NN          cytoreduction#NN
  gabriela#NN            hepatotoxic#JJ        hepatotoxicity#NN
  hydroxyurea#NN         hype#R                Bilirubinemia#NN
  insel#NN               megakaryocyte#NN      megakaryocytes#NNS
  megakaryocytic#JJ      multipotent#JJ        myelofibrosis#NN
  myelosuppression#NN    nasopharyngitis#NN    reticulin#NN
  steatosis#NN           thrombocythemia#NN    thrombopoietin#NN

23 words found in the general model (FWL) without considering
the POS tag:
  Alkaline#NN            (28418)    Diagnostics#NNPS     (26059)
  Hematology#NNP         (72693)    Inhibitor#NNP        (30048)
  Leukemia#NNP           (15121)    Neutropenia#NNP      (88293)
  Oncology#NNP           (25817)    Supplementary#NNP    (22236)
  Telomerase#NNP         (41622)    Telomere#NNP         (62500)
  Thrombocytopenia#NNP   (76659)    Upstate#NNP          (14717)
  alkaline#NN            (28418)    coexisting#NN        (40592)
  elicit#VB              (14230)    esophageal#NN        (34612)
  inhibit#VB             (15598)    lipid#NN             (31448)
  mutant#JJ              (20917)    recur#VB             (28616)
  reprint#VB             (31545)    vitro#FW             (95775)
  vouch#NN               (31839)

127 words found in the general model (FWL) considering also
the POS tag:
  Alanine#NN             (96361)    Anemia#NN            (20057)
  Cellulitis#NN          (63804)    Constipation#NN      (29308)
  Diarrhea#NN            (14425)    Discontinuation#NN   (54793)
  Dizziness#NN           (20914)    Epistaxis#NN         (48427)
  Hematologic#JJ         (72281)    Hematopoietic#JJ     (86881)
  Inhibition#NN          (21438)    Interferon#NN        (41190)
  Malignancies#NNS       (44977)    Marrow#NN            (13218)
  Mutation#NN            (16950)    Myalgia#NN           (83824)
  Neutropenia#NN         (88293)    Platelet#NN          (33616)
  Syncope#NN             (66314)    Transplantation#NN   (21610)
```

abnormalities#NNS	(18840)	activator#NN	(56824)
alanine#NN	(96361)	allele#NN	(37241)
aminotransferase#NN	(93421)	anemia#NN	(20057)
assay#NN	(29487)	assays#NNS	(39753)
attainment#NN	(13121)	bilirubin#NN	(54465)
biochemical#JJ	(20193)	biologic#JJ	(41031)
cirrhosis#NN	(34609)	clinically#RB	(15466)
clinician#NN	(22615)	clonal#JJ	(49405)
clone#NN	(17435)	clones#NNS	(19619)
colony-forming#JJ	(84717)	competitively#RB	(29235)
conclusively#RB	(26283)	concomitant#JJ	(20581)
confounded#VBD	(47679)	contributory#JJ	(47351)
deciliter#NN	(62048)	diarrhea#NN	(14425)
discontinuation#NN	(54793)	discontinued#VBN	(25159)
dosing#NN	(33100)	elicit#VBP	(14230)
enzymatic#JJ	(53297)	febrile#JJ	(44240)
fibrosis#NN	(25396)	hematologic#JJ	(72281)
hematopoietic#JJ	(86881)	hemiparesis#NN	(98111)
hemoglobin#NN	(30725)	hemorrhage#NN	(25840)
hemorrhagic#JJ	(48392)	hepatic#JJ	(38157)
hepatitis#NN	(14690)	histologic#JJ	(34824)
hyperplasia#NN	(43642)	influenza#NN	(13264)
infusion#NN	(15536)	inhibited#VBD	(45877)
inhibited#VBN	(29471)	inhibition#NN	(21438)
inhibitor#NN	(30048)	inhibitory#JJ	(42945)
inhibits#VBZ	(25187)	interferon#NN	(41190)
intermittent#JJ	(16831)	intravenously#RB	(43508)
ischemia#NN	(41617)	kilogram#NN	(35028)
kinase#NN	(58578)	leukemia#NN	(15121)
leukocytosis#NN	(90728)	malignant#JJ	(15763)
marrow#NN	(13218)	millimeter#NN	(23735)
mutated#VBN	(51613)	mutation#NN	(16950)
mutations#NNS	(17271)	neoplasm#NN	(45335)
neoplasms#NNS	(49843)	neoplastic#JJ	(59382)
neutropenia#NN	(88293)	neutrophil#NN	(76674)
nonspecific#JJ	(38128)	oligonucleotide#NN	(93481)
osteomyelitis#NN	(51547)	palpable#JJ	(13610)
pathway#NN	(13045)	phosphatase#NN	(69332)
platelet#NN	(33616)	postsurgical#JJ	(64793)
progenitor#NN	(37795)	progenitors#NNS	(47700)
prognostic#JJ	(44788)	rash#NN	(13444)
reassessed#VBN	(53046)	receptor#NN	(20653)
red-cell#JJ	(96498)	refractory#JJ	(40873)

remissions#NNS	(72251)	retinal#JJ	(29450)
reversibility#NN	(51542)	side-effect#JJ	(97640)
side-effects#NNS	(38875)	subgroup#NN	(20833)
syncope#NN	(66314)	telomerase#NN	(41622)
telomere#NN	(62500)	thrombocytopenia#NN	(76659)
thromboembolic#JJ	(70770)	thrombosis#NN	(31757)
thrombotic#JJ	(76582)	transcriptase#NN	(65901)
transcription#NN	(19849)	transducer#NN	(28539)
transfusion#NN	(24259)	transfusions#NNS	(29899)
transient#JJ	(14702)	treatment-related#JJ	(85700)
urinary#JJ	(20950)		

The result of this experiment contains 176 word tokens and 153 word types. This is more than four times the amount of words from the previous experiment. By looking at them, we notice that they are mainly from the medical domain. Due to their high word rank (most words are ranked widely above 20k), it is unlikely that they are contained in the general component M_g. Therefore, to mark them as known, medical terms have to be part of the domain specific component (M_s) of the user model. But because our user models are based on another domain (computer science), they are unlikely to be contained. Hence, the difference of unknown words we witness is because of different general terms in M_s and a lower rank order of words contained in the document model.

5.4 Discussion

In this section, we take a closer look at the data at hand. First, we explore the properties of our user models, such as tokens, types and the computed average word rank. Second, we investigate whether there are relationships between those properties. We investigate the correlation of types and the average word rank within the user model and their influence on the amount of unknown words.

Figure 5.1 shows the proportion of word tokens to types from each user model. Computed over all models, the ratio of tokens to types is on average $1 : 12.5$. Because the actual frequency of types in our user models varies quite strongly, the value of 12.5 is only useful to compare the height of the bars displayed in the discussed histogram. We explored the most and least frequent words (together with their POS tag) of the model $M_u\#3$. Common words like “that”, “this” and “with” occur over one thousand times. On the contrary, out of 10,010 word types, we have counted 3,467 types (~1/3) that appear only once, 1,442 types that appear twice, and 837 types that appear thrice within this user model. More than half of the types (5,746) contained in this user model appear less than four times. To emphasize the sparsity of data, we calculate the probabiltiy to randomly select one word contained in the 3,467 types from the user model, which is ~0.7×10^{-5} (1/144,219). The ratio of types to tokens presented above ($1 : 12.5$) would be even higher, if we would have included short words, like “I”, “a” or “and”, which we removed during the preprocessing step.

Another fact that this histogram reveals, is that the chosen 25 documents vary significantly in size. M_u#3 contains around three times more tokens than M_u#8, but only around twice the amount of types. Also interesting is that although M_u#3 contains the most tokens, it does not contain the most types. This model contains 144,219 tokens and 10,010 types, but M_u#5 contains 88,195 tokens and 10,729 types (see Table 5.3).

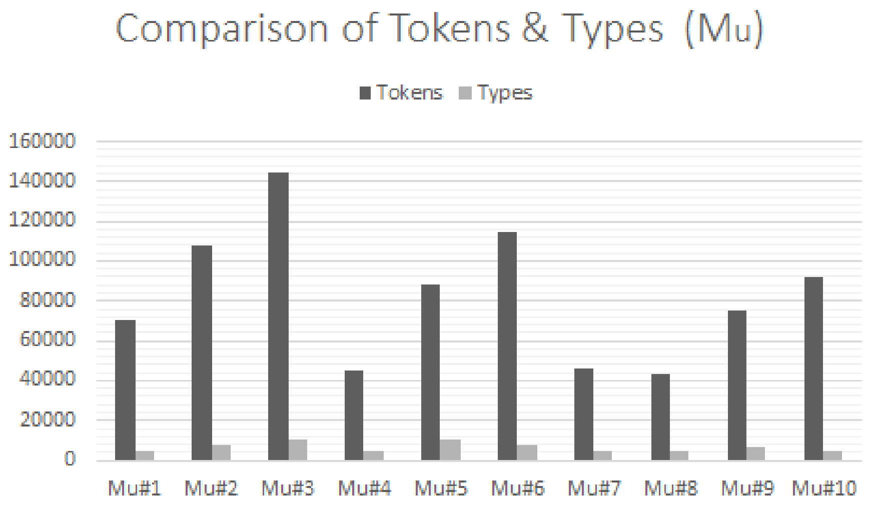


Figure 5.1: Histogram of word tokens and types of user models.

Besides the amount of tokens, an additional relevant aspect (to finding unknown words) is the language level of the user (i.e. the average rank of the words of the user model). Figure 5.2 shows that M_u#3 has the highest average rank, followed by M_u#5.

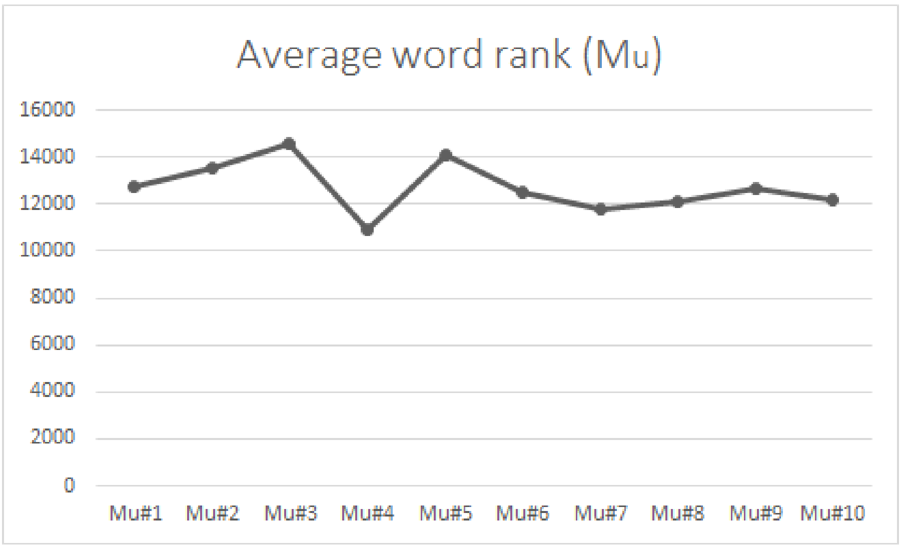


Figure 5.2: The average word rank of the user models.

To further explore the relationship between word tokens and the average rank of words, we calculated the Kendall rank correlation coefficient, also known as Kendall's τ coefficient. Kendall's τ[9] is a measure of non-parametric (distribution free) rank correlations. The correlation coefficient takes values between +1 and -1. For example, given some values of two variables (x_1 to x_n and y_1 to y_n), a value closer to +1 indicates more concordant pairs ($x_1 > x_2 \implies y_1 > y_2$) and a value closer to -1 indicates more discordant pairs ($x_1 > x_2 \implies y_1 < y_2$). A value of zero (0) implies no correlation between the variables x and y. The value for Kendall's τ can be interpreted as a probability of observing the agreeable pairs.

The null hypothesis is that there is no association between the amount of word tokens and the average rank of words. The computed value for τ is 0.56, which indicates a rather weak relationship. Therefore, we cannot claim that user models with more types have a higher average rank. Figure 5.3 visualizes the relationship of both properties of the user models. To better display the weak correlation, we sorted the data in this figure by the amount of word types in ascending order.

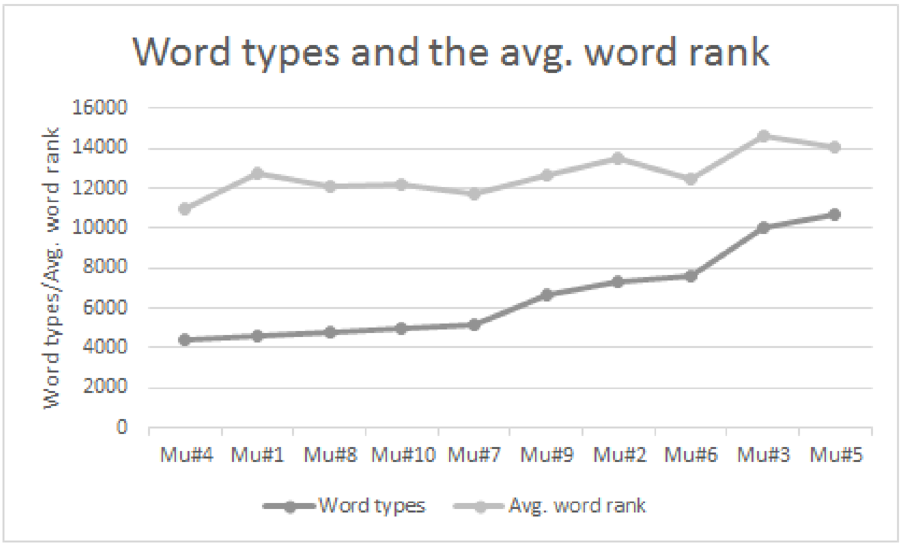


Figure 5.3: The average word rank and the amount of word types of the user models sorted by the amount of types.

In a next step, we explore the outcome of our conducted experiments and investigate the relationship between the amount of word types, the average rank and the amount of the resulting unknown words. Figure 5.4 shows a comparison of unknown and known words from both experiments for each user model. It states that with respect to the user model, we found less unknown words in the equi-domain experiments than in the cross-domain experiments.

[9] `http://www.statisticssolutions.com/kendalls-tau-and-spearmans-rank-correlation-coefficient/`

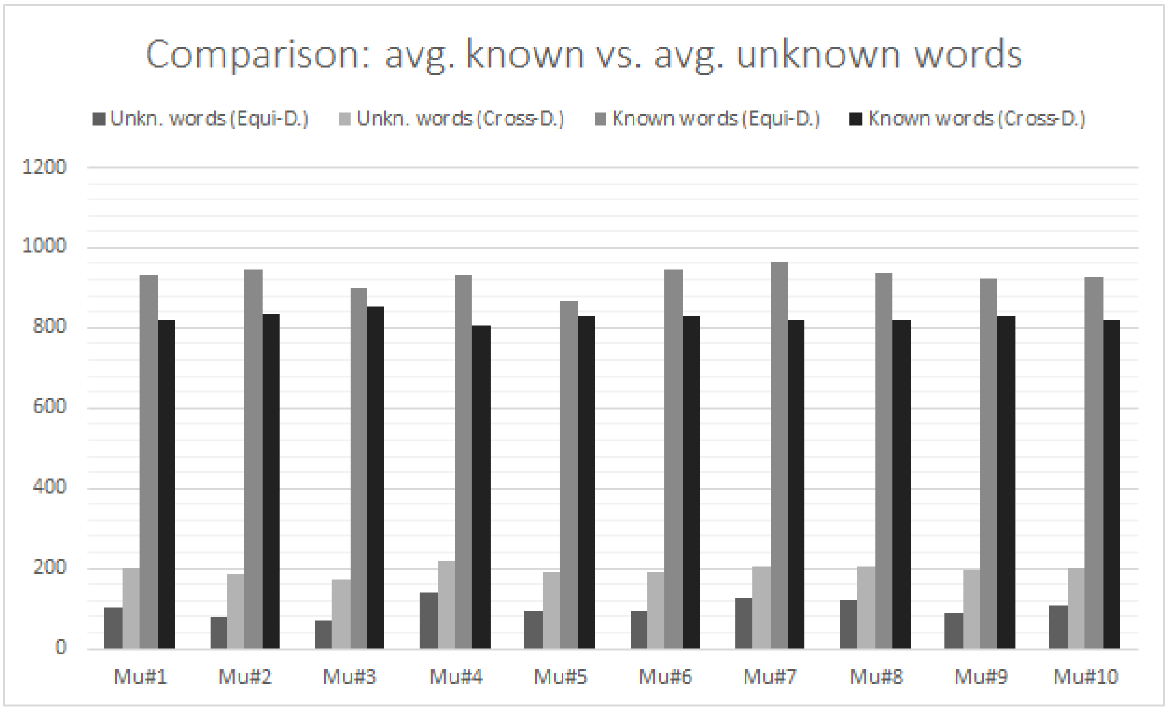


Figure 5.4: Comparison of the known and unknown words of both experiments.

Also the inverse proportion of known and unknown words is clearly visible. For the user model with the highest average word rank (M_u#3), we identified the least amount of unknown words for both experiments, namely 72 and 171 words (see Table 5.7).

Table 5.7: Relevant statistics about our conducted experiments. M_s are ordered by their sum of unknown words in descending order.

Author	Types of the 1. experiment in M_d	Sum of types in M_s	Average word rank in M_s	Sum of unknown words
M_u#3	972	10010	14597	243
M_u#2	1026	7323	13548	268
M_u#9	1013	6636	12667	284
M_u#5	964	10729	14079	287
M_u#6	1042	7609	12499	288
M_u#1	1037	4567	12740	304
M_u#10	1038	4956	12154	309
M_u#8	1057	4806	12132	324
M_u#7	1091	5210	11764	331
M_u#4	1076	4461	10944	357

Table 5.7 displays the most relevant data of our experiments. It contains the types of the document models (M_d) of the equi-domain experiment, but not of the cross-domain experiment. That is because we used different documents for the former experiment, but always the same for the latter, thus we treat it as a constant and omit it in this table. The less types M_d contains, the less words can be classified as unknown. Furthermore, the table displays the amount of types contained in the domain specific part of the user model (M_s), its average word rank and the sum of unknown words. We sorted the table by the sum of unknown words in descending order.

We observe that a higher amount of types and a higher average word rank leads to less unknown words. To investigate the relationship between those three variables, we performed two correlation analyses. The Kendall's τ coefficient for the sum of types in M_s and the sum of unknown words is 0.56 and -0.82 for the average word rank of M_s and the sum of unknown words. The latter correlation coefficient suggests a quite strong relationship between the average word rank and the sum of unknown words. After all, based on the example of $M_u\#2$ and $M_u\#9$, which both have a significantly lower amount of types and average word rank, the content is what really matters.

5.5 Summary

This chapter covered the evaluation part of this thesis, which consists of two kinds of experiments. The first type compares user models with documents from the same domain (equi-domain experiments), whereas the second one compares them with documents from another area (cross-domain experiments), namely the medical domain. We conducted the experiments based on a test set of ten user models, each consisted of 25 randomly chosen documents (see Table 5.3).

After 360 completed experiments (180 for each domain), we witnessed that on average more than four times the amount of unknown words have been found in documents from the medical domain in contrast to documents from the CS domain. Furthermore, in contrast to the equi-domain experiments, the cross-domain experiments revealed that the majority of unknown words were domain specific words and not general terms. Also, we described a relationship between the average word rank of our user models and the sum of detected unknown words. The Kendall's τ coefficient of -0.82 quantifies the intuition that a higher language level of a user likely implies a lower amount of unknown words.

CHAPTER 6

Summary

6.1 Limitations and Future Work

This section judges the limitations of our work from a technical and a philosophical stand point. The technical limitations describe the shortcomings of our implementation and are therefore not as severe as the boundaries we face in our methodology. We also suggest some possible solutions, regarded as future work, for the described limitations.

Our implementation is limited to the English language only, which has practical reasons. First of all, it is only a proof of concept and not a mature software product. We chose English as our language, because most of the work conducted on NLP has been done on English text. Despite the fact that Stanford CoreNLP also supports other languages when it comes to POS tagging or NER[1] (e.g. German), there are other areas which lack of a mature technical solution. Also the FWL we use is not available in such a quality for other languages. The same is true for the information contained in BabelNet.

A less important limitation, but yet present, is the support for other file formats except PDF and plain text. We limited them for our proof of concept, because we only deal with journal articles which are in the PDF format. This can be an issue, if we allow the public to use our application, because there exist other popular text formats (e.g. .docx from Microsoft Word or .odt from Apache OpenOffice^{TM}).

From a philosophical point of view, our biggest limitation would be the lack of knowledge about the user and therefore about his general and domain knowledge. The only information we have is what is written in those papers we process. Based on that, it is hard or maybe even impossible to tell which domain specific words from other domains might be known by the user. For example, according to our FWL the noun “infarct” has a rank of 82,944, altough it is clearly a common medical term. We need another measure

[1] `http://stanfordnlp.github.io/CoreNLP/`

for domain specific words, which tells us more about the importance of those words. The FWL we use does not provide us with an adequate measure for that information need. One way to approach this problem would be to count semantic relations of that sense, which we can do by resolving them with BabelNet.

We tried to keep the set of unknown words small by using the lemma instead of the word itself if the average rank of the user model was lower than the rank of the word. This approach might be justified when we deal with authors who have an already proficient language level. For beginners, we might not use the lemma, but just the word, because often there is a big morphological difference between the two of them (see “go” and “went”). Also our concept of the language level (average rank order) of the user as an index for the model complexity needs to be reconsidered. Our user models, which we computed from highly technical text, could not exceed a rank of 15,000 of the ~100,000 in the FWL. Also, we experienced the necessity of a well designed protocol service, which offers insights about model computation and comparison. After all, text comes in many forms and often brings new processing challenges with it, but that is the beauty of it.

We also do not include knowledge of other languages in our method. The German translation for the above word “infarct” is “Infarkt”. If we would compute an English language model for a user, whose mother tongue is German, we should include “infarct” in the collection of known words, because both words are very similar. We can compute a value for similarity by analyzing the structural difference of the word or, to be on the safe side, define a mapping between those words, because we might have too many false positives.

Another issue is the dynamic nature of languages. By the time of writing, the term “Brexit” is very popular, but it is not part of our FWL because the list is static and we do not receive any updates. We need to find a way to catch up with changes in the human languages. Those changes will only occur to words, where the POS is among the open classes. A possible solution would be to use words reported as new from web corpora like the Now Corpus[2] because it is updated on a daily basis.

Also worth mentioning is that due to our evaluation approach, we only consider relatively clean text from journal articles. Those papers have probably been proofread and do not contain misspelled text or slang. Also, our POS tagger only considers clean text, as opposed to SMS speak or text from the web.

As already mentioned in Section 3.5, we do not know if our approach finds too many, too few, or the right amount of unknown words. Only conducting individual assessments with the user would provide an accurate answer to this question. This is an option for building a training set which can be further used to judge the correctness of the comparison functions. Nevertheless, a semi-automated approach is impractical on a global scale. We must be able to learn from the user’s created or used resources and construct the model automatically from it.

[2]`http://corpus.byu.edu/now/`

6.2 Conclusion

In this thesis, we presented a fully automated approach for identifying words which are most likely to be unknown to the user and providing the user with the proper word explanations. We use the following three data sources: First, BabelNet, a multilingual encyclopedic dictionary with lexicographic and encyclopedic coverage of terms. Second, a very comprehensive frequent word list which is based on the 520 million word Corpus of Contemporary American English. Third, the DBLP computer science bibliography, a high-quality bibliographic meta-data database. Furthermore, we leverage state of the art libraries for natural language processing (NLP) from the well known Stanford Universiy.

We have integrated both, the data sources and libraries into our proof of concept (POC) implementation. The POC consists of a backend server application, which we use to create the language models from text, to compare user models with document models, and to explain words which have been identified as most likely to be unknown to the user. The frontend application, written in Angular 2, supports the user to understand a certain document, which has been submitted for analysis. It displays the submitted PDF file which can be viewed page by page, a list of identified unknown words with respect to the currently viewed page (context sensitive), and a list of word explanations, which the user can fill by clicking on an unknown word entry. We evaluated our approach by using a collection of over 500 scientific papers from exactly ten authors from the Computer Science domain, which we identified through the DBLP computer science bibliography. For each author, we construct a smoothed probabilistic unigram language model, which we refer to as user model, because it models the knowledge of the user in terms of known words. In a next step, we construct simpler language models from single documents, which we name document models. The construction of both high quality language models requires techniques like text parsing, text normalization, tokenization, part of speech (POS) tagging, named entity recognition (NER) and post-processing. We investigated and adapted these concepts for the task at hand. One important task was to reduce the amount of artefacts (tokens that are not words), for which we used custom regular expressions (regex), our FWL and BabelNet. Another challenge was to identify and remove named entities (like locations, organizations and people), because we were only aiming to find and explain concepts. With those measures, we effectively reduced the amount of possible false positives (words identified as unknown words).

The evaluation compared our generated user models with documents models. The results verified our assumptions regarding the amount of unknown words we found. Our comparison algorithm detected on average fifty percent less unknown words for documents from the same domain than for document from the medical domain. The qualitative analysis revealed that we were able to classify a big amount of the domain specific medical terms as possibly unknown to the user. Also, we identified a relationship between the language level (average word rank order) of our user models and the sum of detected unknown words. We computed a Kendall's τ coefficient of -0.82, which indicates a negative correlation. A higher language level led to a lesser amount of detected unknown words.

Appendix

Table 1: POS conversion table between CLAWS7 and Penn[3]. We added the rules (C7 -> Penn): PP -> PRP and VV -> MD.

C7	Penn	C7 Explanation
APPGE	PRP$	possessive pronoun, pre-nominal (e.g. my, your, our)
AT	DT	article (e.g. the, no)
AT1	DT	singular article (e.g. a, an, every)
BCL		before-clause marker (e.g. in order (that),in order (to))
CC	CC	coordinating conjunction (e.g. and, or)
CCB	CC	adversative coordinating conjunction (but)
CS	IN	subordinating conjunction (e.g. if, because, unless, so, for)
CSA	IN	as (as conjunction)
CSN	IN	than (as conjunction)
CST	IN	that (as conjunction)
CSW	IN	whether (as conjunction)
DA		after/post-determiner capable of pronominal function (e.g. such)
DA1		singular after-determiner (e.g. little, much)
DA2		plural after-determiner (e.g. few, several, many)
DAR		comparative after-determiner (e.g. more, less, fewer)
DAT		superlative after-determiner (e.g. most, least, fewest)
DB	PDT	before/pre-determiner capable of pronominal function (all, half)
DB2	DT	plural before-determiner (both)
DD	DT	determiner (capable of pronominal function) (e.g any, some)
DD1	DT	singular determiner (e.g. this, that, another)
DD2	DT	plural determiner (these,those)
DDQ	WDT	wh-determiner (which, what)
DDQGE	WP$	wh-determiner, genitive (whose)
DDQV	WDT	wh-ever determiner, (whichever, whatever)
EX	EX	existential there
FO		formula
FU		unclassified word
FW	FW	foreign word
GE	POS	germanic genitive marker - (' or's)

IF	IN	for (as preposition)
II	IN	general preposition
IO	IN	of (as preposition)
IW	IN	with, without (as prepositions)
JJ	JJ	general adjective
JJR	JJR	general comparative adjective (e.g. older, better, stronger)
JJT	JJS	general superlative adjective (e.g. oldest, best, strongest)
JK	JJ	catenative adjective (able in be able to, willing in be willing to)
MC	CD	cardinal number,neutral for number (two, three..)
MC1	CD	singular cardinal number (one)
MC2	CD	plural cardinal number (e.g. sixes, sevens)
MCGE		genitive cardinal number, neutral for number (two's, 100's)
MCMC	CD	hyphenated number (40-50, 1770-1827)
MD	JJ	ordinal number (e.g. first, second, next, last)
MF	NNS	fraction,neutral for number (e.g. quarters, two-thirds)
ND1	NN	singular noun of direction (e.g. north, southeast)
NN	NN	common noun, neutral for number (e.g. sheep, cod, headquarters)
NN1	NN	singular common noun (e.g. book, girl)
NN2	NNS	plural common noun (e.g. books, girls)
NNA		following noun of title (e.g. M.A.)
NNB		preceding noun of title (e.g. Mr., Prof.)
NNL1		singular locative noun (e.g. Island, Street)
NNL2		plural locative noun (e.g. Islands, Streets)
NNO		numeral noun, neutral for number (e.g. dozen, hundred)
NNO2		numeral noun, plural (e.g. hundreds, thousands)
NNT1		temporal noun, singular (e.g. day, week, year)
NNT2		temporal noun, plural (e.g. days, weeks, years)
NNU		unit of measurement, neutral for number (e.g. in, cc)
NNU1		singular unit of measurement (e.g. inch, centimetre)
NNU2		plural unit of measurement (e.g. ins., feet)
NP	NNP	proper noun, neutral for number (e.g. IBM, Andes)
NP1	NNP	singular proper noun (e.g. London, Jane, Frederick)
NP2	NNPS	plural proper noun (e.g. Browns, Reagans, Koreas)
NPD1	NNP	singular weekday noun (e.g. Sunday)
NPD2	NNPS	plural weekday noun (e.g. Sundays)
NPM1	NNP	singular month noun (e.g. October)
NPM2	NNPS	plural month noun (e.g. Octobers)
PN	NN	indefinite pronoun, neutral for number (none)
PN1	NN	indefinite pronoun, singular (e.g. anyone, everything, nobody, one)
PNQO	WP	objective wh-pronoun (whom)
PNQS	WP	subjective wh-pronoun (who)
PNQV	WP	wh-ever pronoun (whoever)
PNX1		reflexive indefinite pronoun (oneself)

PP	PRP	all PP* below
PPGE		nominal possessive personal pronoun (e.g. mine, yours)
PPH1	PRP	3rd person sing. neuter personal pronoun (it)
PPHO1	PRP	3rd person sing. objective personal pronoun (him, her)
PPHO2	PRP	3rd person plural objective personal pronoun (them)
PPHS1	PRP	3rd person sing. subjective personal pronoun (he, she)
PPHS2	PRP	3rd person plural subjective personal pronoun (they)
PPIO1	PRP	1st person sing. objective personal pronoun (me)
PPIO2	PRP	1st person plural objective personal pronoun (us)
PPIS1	PRP	1st person sing. subjective personal pronoun (I)
PPIS2	PRP	1st person plural subjective personal pronoun (we)
PPX1	PRP	singular reflexive personal pronoun (e.g. yourself, itself)
PPX2	PRP	plural reflexive personal pronoun (e.g. yourselves, themselves)
PPY	PRP	2nd person personal pronoun (you)
RA	RB	adverb, after nominal head (e.g. else, galore)
REX	RB	adverb introducing appositional constructions (namely, e.g.)
RG	RB	degree adverb (very, so, too)
RGQ	WRB	wh- degree adverb (how)
RGQV	WRB	wh-ever degree adverb (however)
RGR	RBR	comparative degree adverb (more, less)
RGT	RBS	superlative degree adverb (most, least)
RL	RB	locative adverb (e.g. alongside, forward)
RP	IN	prep. adverb, particle (e.g about, in)
RPK	IN	prep. adv., catenative (about in be about to)
RR	RB	general adverb
RRQ	WRB	wh- general adverb (where, when, why, how)
RRQV	WRB	wh-ever general adverb (wherever, whenever)
RRR	RBR	comparative general adverb (e.g. better, longer)
RRT	RBS	superlative general adverb (e.g. best, longest)
RT		quasi-nominal adverb of time (e.g. now, tomorrow)
TO	TO	infinitive marker (to)
UH	UH	interjection (e.g. oh, yes, um)
VB0	VBP	be, base form (finite i.e. imperative, subjunctive)
VBDR	VBD	were
VBDZ	VBD	was
VBG	VBG	being
VBI	VB	be, infinitive (To be or not... It will be ..)
VBM	VBP	am
VBN	VBN	been
VBR	VBP	are
VBZ	VBZ	is
VD0	VBP	do, base form (finite)
VDD	VBD	did

VDG	VBG	doing
VDI	VB	do, infinitive (I may do... To do...)
VDN	VBN	done
VDZ	VBZ	does
VH0	VBP	have, base form (finite)
VHD	VBD	had (past tense)
VHG	VBG	having
VHI	VB	have, infinitive
VHN	VBN	had (past participle)
VHZ	VBZ	has
VM	MD	modal auxiliary (can, will, would, etc.)
VMK	MD	modal catenative (ought, used)
VV	MD	wouldst
VV0	VBP	base form of lexical verb (e.g. give, work)
VVD	VBD	past tense of lexical verb (e.g. gave, worked)
VVG	VBG	-ing participle of lexical verb (e.g. giving, working)
VVGK	VBG	#NAME?
VVI	VB	infinitive (e.g. to give... It will work...)
VVN	VBN	past participle of lexical verb (e.g. given, worked)
VVNK		past participle catenative (e.g. bound in be bound to)
VVZ	VBZ	-s form of lexical verb (e.g. gives, works)
XX	RB	not, n't
ZZ1		singular letter of the alphabet (e.g. A,b)
ZZ2		plural letter of the alphabet (e.g. A's, b's)

Table 2: Mapping from the user models (M_u) to the author.

User Model	Author
M_u#1	Khandani
M_u#2	Montanari
M_u#3	Meyer
M_u#4	Chang
M_u#5	Gray
M_u#6	Chen
M_u#7	Nasipuri
M_u#8	Wenger
M_u#9	Kulkarni
M_u#10	Szeider

[3]https://github.com/magnusnissel/pos-tag-conversion/blob/master/conversion.csv

List of Figures

List of Tables

List of Listings

Bibliography

[AAB$^+$03] J Allen, Jay Aslam, Nicholas Belkin, Chris Buckley, Jamie Callan, WB Croft, Sue Dumais, Norbert Fuhr, Donna Harman, David J Harper, et al. Challenges in information retrieval and language modeling. In *SIGIR Forum*, volume 37, pages 31–47. ACM Press, 2003.

[Bab] BabelNet. http://babelnet.org/. Accessed: 2016-02-12.

[BG16] Alan Bailin and Ann Grafstein. *Readability: Text and Context.* Springer, 2016.

[BN93] Laurie Bauer and Paul Nation. Word families. *International journal of Lexicography*, 6(4):253–279, 1993.

[CD95] Jeanne Sternlicht Chall and Edgar Dale. *Readability revisited: The new Dale-Chall readability formula.* Brookline Books, 1995.

[CG96] Stanley F Chen and Joshua Goodman. An empirical study of smoothing techniques for language modeling. In *Proceedings of the 34th annual meeting on Association for Computational Linguistics*, pages 310–318. Association for Computational Linguistics, 1996.

[Chi88] David N Chin. User models and discourse models. *Computational Linguistics*, 14(3):86–87, 1988.

[Con] Linguistic Data Consortium. https://catalog.ldc.upenn.edu/ldc99t42. Accessed: 2016-04-22.

[CT14] Kevyn Collins-Thompson. Computational assessment of text readability: A survey of current and future research. *International Journal of Applied Linguistics*, 165(2):97–135, 2014.

[CTC04] Kevyn Collins-Thompson and James P Callan. A language modeling approach to predicting reading difficulty. In *HLT-NAACL*, pages 193–200, 2004.

[CTC05] Kevyn Collins-Thompson and Jamie Callan. Predicting reading difficulty with statistical language models. *Journal of the American Society for Information Science and Technology*, 56(13):1448–1462, 2005.

[Dal31] Edgar Dale. A comparison of two word lists. *Educational Research Bulletin*, pages 484–489, 1931.

[Dbl] DblpWebsite. http://dblp.uni-trier.de/. Accessed: 2016-03-19.

[DC48] Edgar Dale and Jeanne S Chall. A formula for predicting readability: Instructions. *Educational research bulletin*, pages 37–54, 1948.

[DKL08] Hongbo Deng, Irwin King, and Michael R Lyu. Formal models for expert finding on dblp bibliography data. In *2008 Eighth IEEE International Conference on Data Mining*, pages 163–172. IEEE, 2008.

[DuB04] William H DuBay. The principles of readability. *Online Submission*, 2004.

[DuB07] William H DuBay. The classic readability studies. *Online Submission*, 2007.

[FGM05] Jenny Rose Finkel, Trond Grenager, and Christopher Manning. Incorporating non-local information into information extraction systems by gibbs sampling. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 363–370. Association for Computational Linguistics, 2005.

[Fle48] Rudolph Flesch. A new readability yardstick. *Journal of applied psychology*, 32(3):221, 1948.

[FM12] Thomas François and Eleni Miltsakaki. Do nlp and machine learning improve traditional readability formulas? In *Proceedings of the First Workshop on Predicting and Improving Text Readability for target reader populations*, pages 49–57. Association for Computational Linguistics, 2012.

[Fre14] Adam Freeman. *Pro AngularJS*. Apress, 2014.

[GL35] William Scott Gray and Bernice Elizabeth Leary. What makes a book readable. 1935.

[GM03] Warren R Greiff and William T Morgan. Contributions of language modeling to the theory and practice of information retrieval. In *Language Modeling for Information Retrieval*, pages 73–93. Springer, 2003.

[Gro] Stanford NLP Group. http://nlp.stanford.edu/software/tokenizer.shtml. Accessed: 2016-04-22.

[GS95] William A Gale and Geoffrey Sampson. Good-turing frequency estimation without tears*. *Journal of Quantitative Linguistics*, 2(3):217–237, 1995.

[GS96] Ralph Grishman and Beth Sundheim. Message understanding conference-6: A brief history. In *COLING*, volume 96, pages 466–471, 1996.

[HAA+96] X Huang, Alex Acero, F Alleva, M Hwang, L Jiang, and Milind Mahajan. From sphinx-ii to whisper—making speech recognition usable. In *Automatic Speech and Speaker Recognition*, pages 481–508. Springer, 1996.

[Hie01] Djoerd Hiemstra. *Using language models for information retrieval.* Taaluitgeverij Neslia Paniculata, 2001.

[HS75] Shelley A Harrison and Lawrence M Stolurow. *Improving instructional productivity in higher education.* Educational Technology, 1975.

[JM14] Dan Jurafsky and James H Martin. *Speech and language processing.* Pearson, 2014.

[JRHZ03] K Sparck Jones, Stephen Robertson, Djoerd Hiemstra, and Hugo Zaragoza. Language modeling and relevance. *Language Modeling for Information Retrieval*, 13, 2003.

[KFRC75] JP Kincaid, RP Fishburne, RL Rogers, and BS Chissom. Derivation of new readability formulas. Technical report, Technical report, TN: Naval Technical Training, US Naval Air Station, Memphis, TN, 1975.

[KLP+10] Rohit J Kate, Xiaoqiang Luo, Siddharth Patwardhan, Martin Franz, Radu Florian, Raymond J Mooney, Salim Roukos, and Chris Welty. Learning to predict readability using diverse linguistic features. In *Proceedings of the 23rd international conference on computational linguistics*, pages 546–554. Association for Computational Linguistics, 2010.

[Lam86] Leslie Lamport. Latex: User's guide & reference manual. 1986.

[Lav04] Victor Lavrenko. *A Generative Theory of Relevance.* Ir, University of Massachusetts, 2004.

[Lav08] Victor Lavrenko. *A generative theory of relevance*, volume 26. Springer Science & Business Media, 2008.

[LC01] Victor Lavrenko and W Bruce Croft. Relevance based language models. In *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 120–127. ACM, 2001.

[LC03] Victor Lavrenko and W Bruce Croft. Relevance models in information retrieval. In *Language modeling for information retrieval*, pages 11–56. Springer, 2003.

[Ley05] Michael Ley. Dblp computer science bibliography. 2005.

[Ley09a] Michael Ley. Dblp: some lessons learned. *Proceedings of the VLDB Endowment*, 2(2):1493–1500, 2009.

[Ley09b] Michael Ley. Dblp xml requests, 2009.

[Lor44] Irving Lorge. Predicting readability. *The Teachers College Record*, 45(6):404–419, 1944.

[LP30] Bertha A.. Lively and Sydney Leavitt Pressey. *A method for measuring the" vocabulary burden" of textbooks.* 1930.

[LZ03] John Lafferty and Chengxiang Zhai. Probabilistic relevance models based on document and query generation. In *Language modeling for information retrieval*, pages 1–10. Springer, 2003.

[LZ15] Yuanhua Lv and ChengXiang Zhai. Negative query generation: bridging the gap between query likelihood retrieval models and relevance. *Information Retrieval Journal*, 18(4):359–378, 2015.

[MRS$^+$08] Christopher D Manning, Prabhakar Raghavan, Hinrich Schütze, et al. *Introduction to information retrieval*, volume 1. Cambridge university press Cambridge, 2008.

[MT77] M David Merrill and Robert D Tennyson. Concept teaching: An instructional design guide. *Englewood Cliffs, NJ: Educational Technology*, 1977.

[MZ11] Chris Mattmann and Jukka Zitting. *Tika in action.* Manning Publications Co., 2011.

[NB07] ISP Nation and David Beglar. A vocabulary size test. *The language teacher*, 31(7):9–13, 2007.

[NH02] ISP Nation and A Heatley. Range: A program for the analysis of vocabulary in texts [software]. *Retrieved April*, 4:2011, 2002.

[PC98] Jay M Ponte and W Bruce Croft. A language modeling approach to information retrieval. In *Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 275–281. ACM, 1998.

[POH13] Mari-Sanna Paukkeri, Marja Ollikainen, and Timo Honkela. Assessing user-specific difficulty of documents. *Information Processing and Management*, 49(1):198–212, 2013.

[Por80] Martin F Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, 1980.

[PP31] WW Patty and Wm I Painter. A technique for measuring the vocabulary burden of textbooks. *The Journal of Educational Research*, 24(2):127–134, 1931.

[RH01] SE Robertson and Djoerd Hiemstra. Language models and probability of relevance. 2001.

[RJ76] Stephen E Robertson and K Sparck Jones. Relevance weighting of search terms. *Journal of the American Society for Information science*, 27(3):129–146, 1976.

[Rob] Roberto Navigli. http://www.kdnuggets.com/2014/05/babelnet-25-multilingual-encyclopedic-dictionary-semantic-network.html. Accessed: 2016-04-22.

[Rob77] Stephen E Robertson. The probability ranking principle in ir. *Journal of documentation*, 33(4):294–304, 1977.

[Rob05] Stephen Robertson. On event spaces and probabilistic models in information retrieval. *Information Retrieval*, 8(2):319–329, 2005.

[SC99] Fei Song and W Bruce Croft. A general language model for information retrieval. In *Proceedings of the eighth international conference on Information and knowledge management*, pages 316–321. ACM, 1999.

[SC01] Luo Si and Jamie Callan. A statistical model for scientific readability. In *Proceedings of the tenth international conference on Information and knowledge management*, pages 574–576. ACM, 2001.

[She93] Lucius Adelno Sherman. *Analytics of literature: A manual for the objective study of English prose and poetry.* Ginn, 1893.

[SJR01] K Sparck-Jones and S Robertson. Lm vs pm: Where's the relevance? In *First Workshop on Language Modeling and Information Retrieval, Pittsburgh, PA*, 2001.

[SSN02] Satoshi Sekine, Kiyoshi Sudo, and Chikashi Nobata. Extended named entity hierarchy. In *LREC*, 2002.

[Tho21] Edward L Thorndike. The teacher's word book, 1921.

[Tik] Apache Tika. https://tika.apache.org/. Accessed: 2016-04-22.

[TKSDM03] Erik F Tjong Kim Sang and Fien De Meulder. Introduction to the conll-2003 shared task: Language-independent named entity recognition. In *Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003-Volume 4*, pages 142–147. Association for Computational Linguistics, 2003.

[VW28] Mabel Vogel and Carleton Washburne. An objective method of determining grade placement of children's reading material. *The Elementary School Journal*, 28(5):373–381, 1928.

[Web] WebKnow. http://webknox.com/p/named-entity-definition. Accessed: 2016-04-22.

[WN08] Stuart Webb and Paul Nation. Evaluating the vocabulary load of written text. *TESOLANZ Journal*, 16:1–10, 2008.

[wor] wordfrequency. http://www.wordfrequency.info/100k.asp. Accessed: 2016-02-12.

[Zhu] Xiaojin Zhu. http://pages.cs.wisc.edu/ jerryzhu/cs769/text_preprocessing.pdf. Accessed: 2016-03-19.