UNIVERSITÄT OSNABRÜCK

Institute of Cognitive Science

Bachelor's Thesis

A textual recommender system and other text mining applications for clinical data

Philipp Hummel

June 9, 2017

First supervisor: Dr. Frank Jäkel
Second supervisor: Prof. Dr. Gordon Pipa

A textual recommender system and other text mining applications for clinical data

In unusual clinical situations doctors would often like to review cases of similar patients to guide their decision making for the current one. To retrieve the relevant cases, however, is a hard and time consuming task. This thesis works on building a recommender system that automatically finds physician letters similar to a specified reference letter in an information retrieval like manner. We use a small dataset of free text physician letters of oncology patients to do exploratory work on this issue. We provide a prototypical system that gives recommendations on this dataset and verify its performance through a psychological experiment assessing doctor's similarity judgments of those letters. We find that the similarity measure of our recommender system correlates strongly with doctor's similarity judgments. Additionally we explore other text mining applications like automatic diagnosis extraction based on this dataset.

Acknowledgements

acknowledgements

Contents

Chapter 1

Introduction

1.1 Motivation

Publishing the case of a patient with a particularly interesting medical phenomenon in the form of a clinical case report has undergone a change in popularity in the medical community. The number of published case reports has been declining in standard journals. This has happened not only because case reports can hardly contribute to a good impact factor, but also because their scientific benefit has been questioned (Mason, 2001). However, several new journals dedicated only to case reports have emerged (Kidd and Hubbard, 2007)[1] and many people have argued for the value of case reports for research itself but also beyond (Williams, 2003; Dib et al., 2008; Sandu et al., 2016). Importantly case reports allow practitioners a more in-depth understanding of specific disease courses and provide educational material for students (Nissen and Wynn, 2014). Although case reports lack the scientific validity of large empirical studies, it is apparent that people have strong intuitions for the usefulness of them. During our collaboration with practicing doctors we also found that practitioners use the clinical records of similar patients to guide problem solving for the current patient. Especially when faced with hard or unusual cases doctors seek similar patient information from the hospital database. While this only shows that doctors think that the presentation of similar cases helps them in their work, we will argue that this can indeed improve their medical problem solving.

Cognitive Scientists have discussed the usefulness of examples for reasoning processes for long and found that at least in some experimental settings reasoning processes are based on earlier presented examples (Medin and Schaffer, 1978). More recently these reasoning processes have also been studied in more realistic scenarios. Klein (2008) reviews models for decision making under real world circumstances. According to him experts interpret a situation based on its resemblance to remembered situations. Once a sufficiently similar situation has been retrieved from mem-

[1] Additional Case Journals can be found at:
Journal of Medical Cases: http://www.journalmc.org/index.php/JMC/index
British Medical Journal Case Reports: http://casereports.bmj.com/site/about/index.xhtml
American Journal of Medical Case Reports: http://www.sciepub.com/journal/AJMCR

ory, experts apply the solution from the remembered situation to the current one in a thought experiment. They evaluate whether or not this solution strategy will lead to success and adjust it, if necessary. In case no way to adequately adjust the solution can be found, another situation is retrieved from memory. This process is repeated until a sufficiently good solution is found. Presentation of similar cases should therefore aid doctor's decision making in an actual clinical setting. The medical domain has also been directly addressed by research in cognitive science. Elstein and Schwarz (2002) have concluded that for medical problem solving reasoning processes can be divided into two distinct categories. For cases perceived as easy doctors apply a kind of pattern recognition based on the examples they have encountered before and use solutions stored in memory. For harder cases, however, doctors need to rely on a more elaborate reasoning process. They have to consciously generate and eliminate hypotheses to be able to solve the problem. It is plausible that hypothesis generation as well as hypothesis falsification is also guided by the doctor's experience of earlier patients. From a more theoretical perspective Kolodner and Kolodner (1987) have specifically argued that "[i]ndividual experiences act as exemplars upon which to base later decisions" in medical problem solving. Their research was partially driven by the desire to understand the way in which clinicians perform problem solving but also by the goal of building artificial systems that can aid in this process. They argue that both humans and machines can learn from specific examples and use them to reason about new problems.

Around the idea that artificial systems might learn from examples has evolved a whole branch of Artificial Intelligence (AI), which is called "Case-based reasoning". This domain has been greatly influenced by psychological findings, some of them mentioned above. Researchers have successfully built systems used in real world applications, that reason from the examples provided (Aamodt and Plaza, 1994). Within this domain of AI one of the greatest application areas is medicine. It seems that the medical area does not only offer straight forward usage of examples, but also has a need for automatic aids for problem solving (Begum et al., 2011).

Given the practical, psychological and theoretical reflections above we believe that it would be helpful for practitioners to be able to review cases of similar patients. One particularly well suited source for the retrieval of patient cases are databases of physician letters, as these letters provide concise summaries of the specifics of a patient that matter in practice. Search in these databases is, to our knowledge, usually limited to character matching procedures and therefore provides limited practical value for doctors. We therefore set out to build a prototypical recommender system on those physician letters to do automatic retrieval of only the relevant documents from a database.

1.2 Dataset

To get a dataset for exploratory work on the recommender system we collaborated with the university hospital Freiburg. The heamatology and oncology department has a database of approximately 190,000 German physician letters in PDF form (Spadaro, 2012) (!correct! noch nicht zufrieden, wie das item in der bibliography erscheint. Ich weiß allerdings auch nicht wie es richtig wäre.). These physician letters are free text documents written by the doctors to keep record of the patient's visit. They usually include information about the patient's age, sex, diagnosed diseases, therapy history, current complaints, many more medical details like blood counts, but also personal information like names and birth dates. The letters generally follow a rough structure. Almost all of them include a letter head (a greeting and introduction), a diagnosis (summarizing diagnosed diseases bullet point like), a therapy history (listing the past therapies with dates) and an anamnesis (free text about current complaints etc.) section, separated into individual paragraphs. In principal though, doctors are free to document this information in the way they please. The database does not, however, contain the information of 190,000 unique patients. For many patients several letters are included, as a new visit will often result in an updated letter, that is added to the database. We refer to these letters as "follow-up" letters.

To get permission to use a subset of those letters for our experiment, it was necessary to ensure that all personal information was removed from them. A medical student of the university hospital was therefore paid to manually anonymize 307 of the letters and forward them to us. The letters were given to us in Microsoft Word XML format.

1.3 Use Case Scenario

The recommender we envision is built into the clinical every day life of doctors. During the visit of a patient, physicians write a new or modify an existing letter for this patient. Already in this writing phase we wish to retrieve letters of similar patients and present them on demand. Thereby doctors decision making processes can be guided by these similar cases, if the doctor deems this necessary. The perceived fitness of the retrieved letters has to be exceptionally high. Doctor's additional time resources are usually very limited. Therefore the recommender system will only be used in practice, if almost all recommendations are deemed useful.

With the dataset mentioned above we set out to build this kind of recommender system. To achieve this goal we first clear our dataset from duplicates with the help of basic information retrieval methods. (!correct! die kapitel beschreibung ist obsolet.) This is described in the subsequent chapter. We elaborate on methods for hiding and showing specific information from the letters in chapter 3. For this goal we make extensive use of more advanced information retrieval methods and introduce them alongside there. Chapter 4 is concerned with classification of the physician letters based on qualities of interest of the corresponding patients. Chapter 5 describes the recommender system itself. Experimental assessments of the quality of the system are explained in chapter 6. Finally we review shortcomings, conceivable problems and possibilities for further work in the discussion.

Chapter 2

Dataset Cleansing

The dataset we acquired from the university hospital contains several duplicate letters. To ensure undistorted test results, we have to identify as many of them as possible. It is clear that finding all of these duplicates manually is not only error-prone, but also very time consuming as in principle $\frac{(n-1)^2+(n-1)}{2}$ letter comparison have to be made. With $n = 307$ this amounts to almost 50,000 comparisons. Personal information, like names and dates of birth, would be helpful for this task, but has been removed during anonymization. Therefore we make use of two semi-automatic procedures for duplicate identification. First we use a longest common subsequence matching method. This method would be sufficient for our problems, if we only had to deal with exact duplicates. However, we face the issue of follow-up letters with modified information in our dataset. To overcome this problem we use a procedure well known in the information retrieval community: The bag of words model for representing texts as vectors, with which it is possible to compute distances between documents. Duplicates are then found by their vector proximity.

2.1 The Bag of Words Model

The bag of words model represents a text as the multiset (bag) of its words. That means all word order information is disregarded and only the information how often a word is present in a text is encoded (Manning et al., 2008*b*). (!correct! Mit der Art wie das Buch hier zitiert wird bin ich noch nicht glücklich. Es sollte eigentlich chapter 2 und 6 sein und nicht 2008 a und b. oder wenigstens a, b, c in derselben Reihenfolge wie die chpaters.) More concretely in the bag of words model documents are represented as fixed length feature vectors, where every feature is a word occurrence count. In the simplest approach every feature is the word or term frequency $\mathrm{tf}(t,d)$ of term t in document d. Where $\mathrm{tf}(t,d)$ is the occurrence count of word or term t in document d. To compute feature vectors for documents in a corpus the vocabulary V of the corpus needs to be established first. Documents are represented by a $1 \times |V|$ vector of the values $\mathrm{tf}(t,d)$ for all $t \in V$. To represent text in the bag of words model, however, the text needs to be preprocessed first.

Preprocessing

In computers, text is most often represented as sequences of characters. The process of extracting words from such a sequence is known as tokenization (Manning et al., 2008*c*). A first naive approach to tokenization might be to split the sequence of characters on every whitespace and regard everything in between as a word. This approach, however, can easily lead to unexpected results. Consider the example string “The doctor treats the patient.”. The naive approach yields the words “The”, “doctor”, “treats”, “the” and “patient.”. Note how the last word contains the punctuation character “.”. So even for very easy examples the naive approach does not produce expected results. Luckily many more sophisticated algorithms are implemented in ready-to-use open-source software packages.

Some words are more important or representative of a text than others and so it is a useful preprocessing step for the bag of words model to remove some frequently appearing but uninformative words, so called stop words. A list of English stop words comprises words like “the”, “a”, “just” or “some”. While they are necessary for the grammatical structure of a language, they do not convey much information about the similarity of documents (Manning et al., 2008*c*).

Additionally for the bag of words model we are not interested in the exact grammatical form in which a word is present in a text. Removing this unnecessary information is achieved with a procedure called stemming, that maps word appearances to their stem (Manning et al., 2008*c*). It maps words such as “doctors” and “doctor” both to their common stem “doctor”. Both removal of stop words and stemming seem at first glance to disregard information. This is true, but they allow to represent a text more compactly and thereby reduce the noise of the representation. We explain those preprocessing steps and the bag of words model with a more illustrative example.

Bag of Words Example

Assume the corpus consists of two documents d_1 = “The patients with disease A.” and d_2 = “The patients with disease B.”. After tokenization, removal of stop words, and stemming, the bag of words vectors representing the two texts are:

$$\mathbf{v}_{d_1} = \begin{array}{c} 1 \\ 1 \\ 1 \\ 0 \end{array} \quad \mathbf{v}_{d_2} = \begin{array}{cc} 1 & \text{patient} \\ 1 & \text{disease} \\ 0 & \text{A} \\ 1 & \text{B} \end{array}$$

Note how the punctuation character does not appear, as the tokenization has removed it from the list of words. The words “the” and “with” have been removed as stop words and the word “patients” has been stemmed to produce “patient”. Then the list of words has been converted to a bag of words vector.

Application and shortcomings

The vectors $\mathbf{v}_{d_1}$ and $\mathbf{v}_{d_2}$ live in a four dimensional vector space and we can either use them as feature vectors for a machine learning task or compute a measure of similarity between them. The standard similarity measure used for bag of words vectors is the cosine similarity $s_{\cos}(\mathbf{v}_1, \mathbf{v}_2) = \frac{\langle \mathbf{v}_1, \mathbf{v}_2 \rangle}{||\mathbf{v}_1|| \cdot ||\mathbf{v}_2||}$, (!correct! sollte ich hier auch $\mathbf{v}_{d_1}$ shcreiben oder ist $\mathbf{v}_1$ ok?) where $\langle \mathbf{v}_1, \mathbf{v}_2 \rangle$ is the scalar product of $\mathbf{v}_1$ and $\mathbf{v}_2$ and $||\mathbf{v}||$ is the norm or length of vector $\mathbf{v}$. The cosine similarity has the appealing property that it normalizes the scalar product by the norm of the vectors used. Thereby it ensures, that the resulting measure lies in the interval of $[-1, 1]$. For positive spaces (i.e. spaces, where all vectors have only non-negative elements, like the bag of words vector space) this measure lies in the interval $[0, 1]$. Two documents d_1 and d_2 are then considered to be more similar than d_3 and d_4, if $s_{\cos}(\mathbf{v}_{d_1}, \mathbf{v}_{d_2}) > s_{\cos}(\mathbf{v}_{d_3}, \mathbf{v}_{d_4})$.

Generally, however, the bag of words model has severe limitations. Consider for example the two sentences $d_3 =$ "The patients with disease A, but not B" and $d_4 =$ "The patients with disease B, but not A". Their representation in the bag of words model is equivalent, although they express quite different meanings. Still the bag of words model is a standard approach for information retrieval problems, as it has many desirable properties. Texts can be straight forwardly embedded in a vector space (i.e. represented as vectors) and these embeddings can be used as fixed length feature representations for machine learning tasks.

2.2 Duplicate Detection

With the bag of words model and the longest common subsequence matching method we semi-automatically identify duplicates in our data. We had a senior oncologist identify all duplicates present in a subset of 150 of these letters manually to have a baseline to which we can compare the methods' performances.

The longest common subsequence matching procedure scans two documents and finds the longest sequence of characters they have in common. If the length of this sequence exceeds a threshold the pair is marked as possibly duplicate and manually inspected. To get more hits we lower the threshold until the false positive rate becomes to high. Secondly we preprocess all texts as described above and map them into the bag of words representation. We use the cosine similarity as a measure of relatedness between pairs and mark all pairs with distance lower than a threshold as possibly duplicate and again iteratively decrease the threshold.

The bag of words approach identifies all manually detected duplicates and follow-ups while having a very low false positive rate. It detects one additional manually undetected follow-up pair. The string matching procedure is not as useful due to a high false positive rate and does not detect all manually found duplicates. Its performance is worse than the performance of the bag of words procedure, especially for follow-up letters. However, it finds a second manually undetected follow-up pair, that we have not identified with the bag of words procedure either. We use both approaches with thresholds from the "training" set on the remaining 157 letters as well. Overall we detect 18 exact copy pairs and 17 follow-up pairs. We additionally remove three letters from the dataset as they include almost no information (an artifact of the specific documentation procedure at the university hospital). This results in our final dataset consisting of 269 unique physician letters (follow-ups are only included, where mentioned explicitly).

Chapter 3

Fine-grained information presentation

Many physician letters are rather long documents, spanning several pages. To increase the usability of a recommender system used in practice it is desirable to be able to show specific information like the diagnosis or the therapy history on demand or hide unnecessary parts of the letters like the introduction. It might also be desirable to compare letters only based on a specific section like therapy history. A first step towards this goal is the automatic extraction of the relevant paragraphs.

3.1 Paragraph Extraction

The XML data format of the letters allows to automatize inspection of rather fine-grained structures present in the letters. One can automatically determine boldfaced characters for example. Because of this and because the documents are similar in structure, we use a rule based approach for extracting the individual paragraphs. A simplified rule to find the beginning of the diagnosis paragraph is shown in pseudocode:

```
diagnosisRegex = '[dD]iagnose(n)?'
text = thisXmlNode.text()
if regex.match(text, diagnosisRegex) and boldface(text) and
 precededByNewline(thisXmlNode) then
    diagnosisStart = thisXmlNode
end
```

A regular expression is defined that matches the word diagnosis (the German word for diagnosis is "Diagnose"). The text of every node in the XML tree is checked for a regular expression match and whether it is boldfaced. If an XML node matches those criteria and is preceded by a newline, it is marked as the beginning of the diagnosis paragraph. With a set of rules like the one above we automatically extract

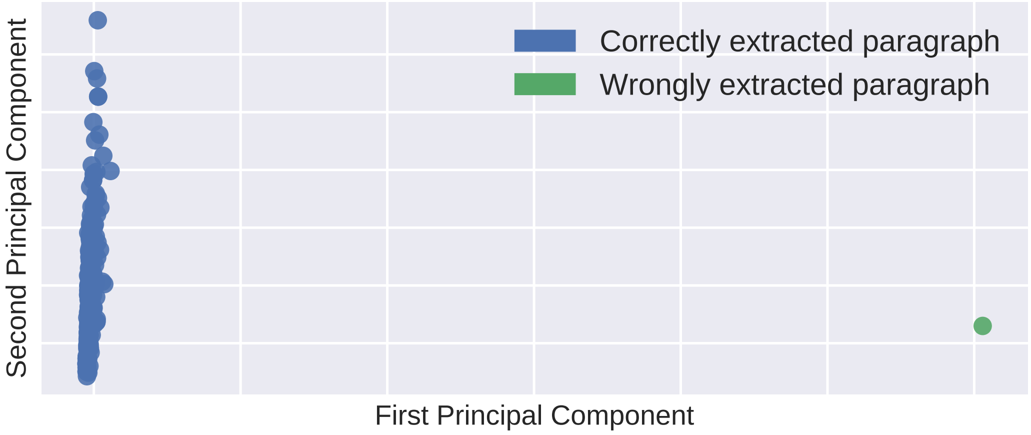

Figure 3.1: 2D PCA projection of bag of words representation of one incorrectly extracted diagnosis paragraph and several correctly extracted ones.

the paragraphs of interest from the documents. This approach, however, is not completely reliable, as the doctors are free to write the documents in the way they please. Indeed we find several wrongly extracted paragraphs, that e.g. include the subsequent paragraph as well. For our dataset it is possible to check the extraction process by hand. However, this is tedious work and is not scalable to bigger datasets. We therefore explore whether we can in principle make use of other automated methods to find paragraphs for which the rule based extraction process does not produce desired results. We therefore take several correctly extracted and one incorrectly extracted diagnosis paragraphs and convert them to their bag of words representation. To get a feeling for how these vectors behave we use Principle Component Analysis (PCA) to get a lower dimensional approximation of the vectors. PCA finds the linear subspace with desired dimensionality of the original space that preserves as much of the variance of the vectors in the original space as possible. Thereby one can gain a low dimensional approximation of the high dimensional data and use this approximation for visual inspection. See figure 3.1 for a 2D PCA plot of the bag of words representation of the correctly and incorrectly extracted diagnosis paragraph. As is apparent from the figure, it would not be a hard task to automatically detect the outlier. In this case the incorrectly extracted paragraph includes not only the diagnosis, but also the therapy history. In cases like this with additional text present, it is an easy task to identify the incorrect ones. A harder problem arises, when only parts of the paragraph of interest have been extracted. However, we believe that this problem is of little concern, due to the way our rules are made. Indeed, we did not find a case

3.2 Paragraph Classification

In our sample dataset, we can automate paragraph extraction as shown above. We can also detect for which paragraphs the procedure produces incorrect results. This approach works well only because the documents in our dataset generally adhere to a rough structure. For datasets from other clinics constructing a rule based extraction procedure is not only time consuming, it might not be possible at all. We therefore test an approach of classifying the extracted paragraphs into three categories—greeting, diagnosis and anamnesis. Our findings show that, surprisingly, this is not a hard problem. On unseen datasets it might therefore be possible to split text into unlabeled paragraphs with a basic rule based approach. One can possibly define a new paragraph to begin after a blank line and automatically label the resulting paragraphs with a predefined category. This way one would be able to hide or show specific information on demand even on datasets from other clinics.

We approach the problem again from a vector space based view-point. We compute the vector representation for every paragraph and use a classifier trained on these representations to predict the correct paragraph label. To get the best results we compare different text embedding methods. We test the standard bag of words, term frequency—inverse document frequency (tf-idf) and paragraph vector models. We also use Latent Semantic Analysis (LSA) and Latent Dirichlet Allocation (LDA) to get more condensed feature vector representations based on the tf-idf vector space. Before presenting the results of the different approaches we subsequently introduce these embedding methods.

Additional vector space models:

Term Frequency—Inverse Document Frequency

An extension of the bag of words model, that creates feature vectors based on term frequencies $\mathrm{tf}(t, d)$ of term t in document d , is the term frequency—inverse document frequency model (tf-idf). A common problem with the bag of words model is that some terms (even after filtering stop words) appear often across texts in a corpus, i.e. have a high term frequency, yet do not constitute a good feature for discrimination between texts. Therefore a scaling factor for the term frequencies is desired which captures the intuition that words appearing often in a few texts but rarely in others are good discriminative features for those texts. tf-idf refers to a specific scaling scheme, that downscales the importance of frequent words, while upscaling the importance of rare words.

Term frequency $\mathrm{tf}(t, d)$ usually refers to the standard word count. Inverse document frequency $\mathrm{idf}(t, C)$ of term t and corpus C can be computed as

$$\mathrm{idf(t,C)} = \log_2\left(\frac{|\mathrm{C}|}{|\{\mathrm{d} \in \mathrm{C} : \mathrm{t} \in \mathrm{d}\}|}\right)$$

where $|C|$ is the total number of documents in the corpus and $|\{d \in C : t \in d\}|$ is the number of documents in which term t appears at least once.

Term frequency—Inverse document frequency tfidf(t, d, C) is then calculated as

$$\mathrm{tfidf(t, d, C) = tf(t, d) \cdot idf(t, C)}$$

tf-idf can be used as a feature for the representation of the document that is more robust to uninformative changes in the distribution of common words and more expressive for rare words (Manning et al., 2008*b*).

Latent Semantic Analysis

An often occurring machine learning (ML) problem is that very high dimensional feature vectors, as occurring in bag of words and tf-idf models, tend to generalize poorly in subsequent tasks like classification. In those cases it is desirable to have a more condensed lower dimensional feature representation of documents that still captures most of the variance in the bag of words representation. Latent semantic analysis (LSA) of Deerwester et al. (1990) in its simplest form takes the plain bag of words vectors of all documents in the corpus and constructs a term-document matrix $\mathbf{M}$, where $\mathbf{M}[i, j] = \mathrm{tf(t_i, d_j)}$, i.e. row i represents the relation of term i to all documents, while column j represents one document and the relation to all it's terms. So column j represents document d_j's vector representation $\mathbf{v}_{d_j}$.

$$\begin{array}{cc} & \begin{array}{c} \mathbf{v}_{d_j} \\ \downarrow \end{array} \\ \mathbf{v}_{t_i}^T \rightarrow & \begin{bmatrix} \mathrm{tf(1,1)} & \dots & \mathrm{tf(1,|C|)} \\ \vdots & \ddots & \vdots \\ \mathrm{tf(|V|,1)} & \dots & \mathrm{tf(|V|,|C|)} \end{bmatrix} \end{array}$$

Singular value decomposition is then used on the term document matrix $\mathbf{M}$, allowing to find a k-dimensional ($k < dim(\mathbf{M})$) linear subspace of $\mathbf{M}$, $\hat{\mathbf{M}}$, that still captures as much of the original information as possible, i.e. that captures as much of the variance in the original space as possible. Column j of $\hat{\mathbf{M}}$ contains a k-dimensional, approximate representation of the document j's feature vector $\mathbf{v}_{d_j}$, called $\hat{\mathbf{v}}_{d_j}$. The document representations $\hat{\mathbf{v}}_{d_j}$ in $\mathbf{M}$'s linear subspace spanned by $\hat{\mathbf{M}}$ do no longer have an intuitive interpretation as word counts, but can still be used as feature vectors for subsequent ML tasks or can be analyzed for similarity. Deerwester et al. (1990) argue that the resulting features have several appealing properties. The LSA features can simply be viewed as a noise-reduced version of the original features, but according to the original paper they can also better deal with linguistic issues such as synonymy and polysemy. This works because every original observation (i.e. document) is represented as a linear combination of k hidden (or latent) semantic concepts. Terms that often appear together (i.e. are semantically related) will then be mapped onto similar LSA representations and can thereby for example capture some aspects of synonymy.

The value of k is a parameter of the model and has to be specified by the researcher. LSA can be used not only with the bag of words space as a basis, but also with the tf-idf space. Throughout this thesis we use LSA in the latter way.

Latent Dirichlet Allocation

A popular probabilistic method for finding latent semantic features of documents in a corpus is latent dirichlet allocation (LDA). As with LSA the rationale is that it would be useful to find a shorter or lower dimensional description for documents in the bag of words vector space format for subsequent tasks. In the LDA context the latent features are called topics and texts are represented as probabilistic mixtures of these topics. A topic is represented by a probability distribution over words and so a document is represented as a probabilistic sample from several topic distributions over words. More specifically that means that documents in LDA space are represented by vectors of dimensionality k. The elements of one vector then represent the strength of association of the document with the respective topic. As in the case of LSA the number of topics k is a parameter and needs to be specified by the researcher (Blei et al., 2003).

Word Vectors

In the standard bag of words model no relationship between words is encoded, i.e. every word is dissimilar from every other word. Thereby a lot of information is lost, as e.g. the words “car” and “automobile” are considered by the system to be as similar to each other as to the word “blue”. One way to encode relationships between words is to embed them in a vector space. Just like the way document similarities can be computed from their distances in the vector space model, similarities of words can be computed, if the words are represented as vectors. A first successful approach to this problem used the LSA model. Here a single word can be represented as a pseudo document and thereby be mapped into the LSA space. The resulting vector in the LSA feature space can be considered a vector representation for the word and be used for comparisons with other words (Deerwester et al., 1990).

Bengio et al. (2003) introduced a neural language model that uses word vectors to predict subsequent words in a text and updates the model as well as the vectors with gradient descent. It seems that by being useful for prediction of subsequent words, the vectors represent statistical information about word contexts and capture meaning of the words to some extent.

Recently Mikolov, Chen, Corrado and Dean (2013) proposed their influential Word2Vec model. This model utilizes a neural network for the prediction of word vectors, given other nearby word vectors. However, the authors focused on the refinement of the word vectors only and were able to simplify the net substantially, while gaining word vector “performance” (i.e. vectors that perform better on a task designed to measure how well the vectors capture human intuitions about the words). Their model works simply with scalar products of input and output word vectors (it has two vectors per word) and a softmax output layer for normalization. This model comes in two architectural types: “Continuous Bag of Words” (CBOW) and “Continuous Skip-gram” (skip-gram). The former predicts the output vector of a center word, given the n previous and n subsequent input word vectors. The input vectors of the $2n$ surrounding words are averaged, the scalar product of this average with every output word vector is computed and a softmax normalization produces final output probabilities. The skip-gram model utilizes the input vector of the center word to predict the output vectors of surrounding words in a left and right context window of maximal size w. The current window size c is sampled uniformly from $\text{range}(1, w)$ at every iteration. This results in $2c$ predictions for every center word considered. The model parameters are updated with gradient descent. After training, either only the input word vectors or the input and output word vectors are used. In the latter case they are either averaged or concatenated to produce the final word vectors used for subsequent tasks.

A note on computational cost: The softmax output layer is very inefficient as it requires the computation of $|V|$ scalar products (every word in the vocabulary). As a first speed up a hierarchical softmax (Morin and Bengio, 2005) is used as an approximation to the real softmax, that reduces the number of scalar product computations to $\log_2(|V|)$ (Mikolov, Chen, Corrado and Dean, 2013). In a subsequent publication the authors introduced a simplified variant of Noise Contrastive Estimation (Gutmann and Hyvärinen, 2012), called Negative Sampling, that further reduces the computational complexity, while still giving useful word vectors (Mikolov, Sutskever, Chen, Corrado and Dean, 2013).

With these architecture and approximation simplifications the Word2Vec model is able to train on a vast amount of text data (billions of words) in a matter of hours. Mainly through those speed ups (and the concomitant larger amounts of processable data) word2vec is able to produce better word vectors, than previously possible.

Paragraph Vector—An extension of Word2Vec

A simple extension of the word2vec model allows to obtain vectors for sentences or longer passages of text. An additional vector is introduced for every piece of text, that we regard as a separate entity (a sentence, a paragraph or a document). This so called paragraph vector is then used in two different ways in the extensions of CBOW and skip-gram. In the Distributed Memory (DM) model, an extension of CBOW, the paragraph vector is used in combination (average or concatenation) with the vectors of surrounding words to predict a center word. In the Distributed bag of words model, an extension of skip-gram, the paragraph embedding is used to predict words randomly sampled from the paragraph.

After the initial training phase a second step, called inference phase, is necessary to gain paragraph embeddings. A paragraph vector is initialized randomly, the rest of the model is kept fixed and the normal training procedure of word prediction and gradient descent on the paragraph vector is done. Thereby the final document embeddings are produced, which can be used like the embeddings of the methods described above. Note, however, that the time until the model reaches convergence both in the training and in the inference phase is unknown and a number of running epochs for both phases must be specified beforehand (Le and Mikolov, 2014).

Classification Results

After having introduced the additional embedding methods tf-idf, LSA, LDA and paragraph vector, we now use their embeddings for classification of the paragraphs into respective categories. Therefore we take all the extracted greeting, diagnosis and anamnesis paragraphs and map them to one of the embedding representations. Then logistic regression is trained on a training portion of the dataset and the performance evaluated on a testing portion. We use leave-one-out cross-validation to obtain a good estimate of the performance even on our limited dataset.

As vector embedding methods we test all methods described above. For LSA and LDA we tune the hyperparameter of dimensionality k to obtain best results. The paragraph vector method has several rather unintuitive hyperparameters, that need tuning. We start with hyperparameters reported as working well in the literature (Lau and Baldwin, 2016). From there we use a randomized search strategy to find better fitting parameters for our specific problem. Results of our evaluation can be found in table 3.1. Several things are noteworthy about the results. First it is surprisingly easy in general to use a small number of training paragraphs (less than 300 per category) to predict its label with very high accuracy. Second all methods are indeed outperformed by the more recent paragraph vector approach. However, the paragraph vector performance comes with the cost of needing to tune many hyperparameters, whose influence is not intuitively clear. Third LSA performance is always smaller or equal to tf-idf performance. As the tf-idf vector space has several thousand dimensions, but we only have several hundred texts, all these texts must fall into a linear subspace with dimension no greater than the number of texts. We assume the dimension is even substantially smaller, as LSA vectors produce the same results in classification accuracy when reducing the number of dimensions even much further.

Embedding Method	BOW	TF-IDF	LSA	LDA	Para2Vec
Classification Accuracy	0.995	0.997	0.997	0.992	<u>1.0</u>

Table 3.1: Mean paragraph classification accuracy of logistic regression measured with leave-one-out cross-validation for different vector embedding methods.

To gain a more intuitive understanding of the performance of these approaches we use PCA to get a 2D approximation of the vectors of the extracted paragraphs. In figure 3.2 one can compare the 2D PCA projections of the tf-idf and the paragraph vector models. While it is obvious that both methods can produce good results even just using a linear classifier, it is also easy to see that the paragraph vectors are easier separable (although not linearly separable in the 2D projection). We

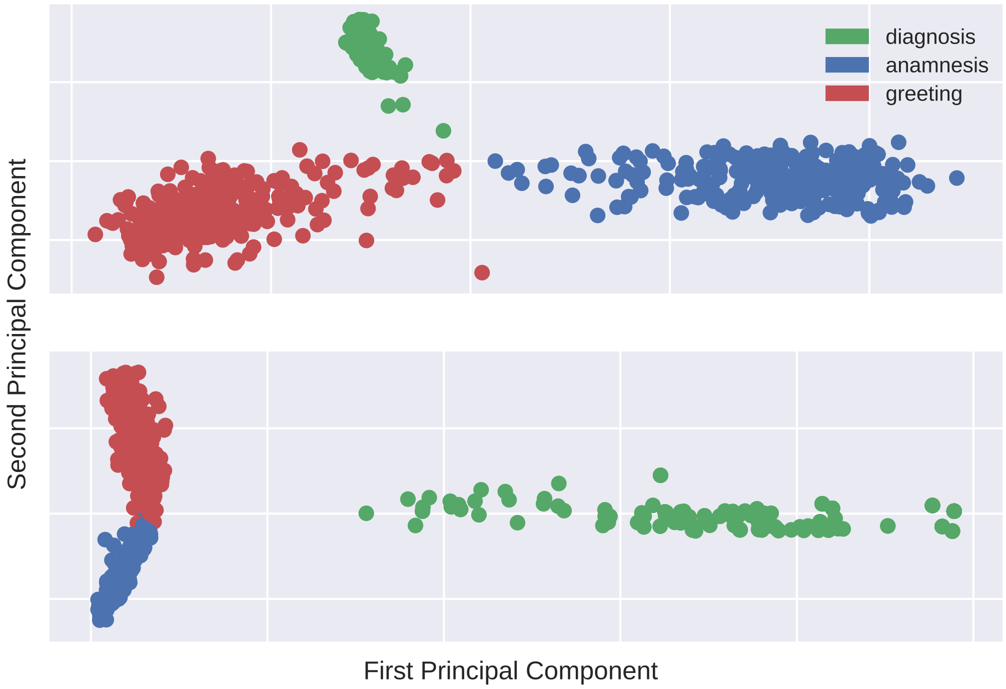

Figure 3.2: 2D PCA projections of a vector space embedding of the physician letter paragraphs. Colors encode the respective paragraph category for each vector. **Top:** Paragraph vector space. **Bottom:** Tf-idf vector space.

conclude that paragraph vector is the best suited method for this classification task and surprisingly performs well even with very limited training data.

Chapter 4

Recommender System

The main objective of this thesis is to build a prototypical recommender system for the physician letters and assess its quality. We do so based on the document embedding methods introduced in chapters 2 and 3. Once documents are embedded in a vector space similarities between documents d_1 and d_2 can be computed as the cosine similarity of the corresponding vectors $\mathrm{sim}_{\cos}(\mathbf{v}_{\mathrm{d}_1}, \mathbf{v}_{\mathrm{d}_2})$. To find the most relevant letters, given a reference letter, we compute the similarity of the reference letter to all other letters in the database. Thereby we get a ranking of all other letters, given the reference letter. The "best" fitting letter or the letter with rank one is considered the most relevant according to the algorithm. The n most relevant letters are the ones on the n highest ranks or the n letters with highest cosine similarity to the current reference letter. They can be retrieved from the database and presented to a user.

4.1 Fine-tuning

Based on the cosine similarity between vectors of the corresponding texts all document embedding methods can in principle be used as a base for the recommender system. To find out which hyperparameters and which embedding method works best we use supervised similarity information. The most desirable information is an expert rating of similarity between letter pairs. As this information is expensive to come by, because experts working hours are expensive, we use a different, easier acquirable dataset for the task of fine-tuning. This data consists of a grouping of 135 of the letters into 50 non-overlapping groups of similar patients done by an expert. Some groups only contain a single patient (if he/she was dissimilar to all others), some contain several and the average group consists of 2.7 patients. This grouping is not equivalent to a correct measure of similarity, but we can still use it as an approximate measure of similarity to tune our algorithm. Thereby letters from the same group are considered similar and letters from different groups are considered dissimilar. One measure of goodness for a real recommender system is how often the system ranks the truly similar letters into the top n. I.e. given one reference letter,

Embedding Method	BOW	TF-IDF	LSA	LDA	Para2Vec
Continous Measure Score	0.794	0.870	0.868	0.634	0.830

Table 4.1: Performance of different embedding methods (with tuned hyperparameters) on the grouping dataset evaluated with the continous measure.

are the similar letters of the grouping dataset considered to be among the top n most relevant by the recommender system. We call this the top-n measure. This measure, however, does not take all available information into account. Say we specify $n = 5$, the top-n measure gives the same score, if a truly similar letter is ranked to be the 6th most similar or the least similar of all. A measure that assigns a higher score in the first situation than in the second would be preferable for the fine-tuning of the algorithm. We therefore develop a continuous measure, that assigns a score from the interval $[0, 1]$ for all possible ranking situations. A score of 1 is given, if the truly similar letters occupy the foremost positions, a score of 0, if the truly similar letters occupy the last positions, a score of 0.5, if the truly similars occupy the positions in the centre. A score of 0.5 is expected, if the algorithm sorted the letters by chance.

Based on the continous measure we first do hyperparameter tuning for the embedding methods as applicable. Afterwards we select the best performing embedding method the same way. Table 4.1 shows the scores obtained by each embedding method. Generally the performance is high above chance level. It is noteworthy that the simple and hyperparameterless tf-idf method outperforms all other methods including LDA and the recent and hard-to-tune paragraph vector. We therefore choose tf-idf as the embedding method for the recommender system.

There are good reasons that tf-idf works so well for our problem. It is plausible that the main features for human similarity judgments of letter pairs are based on the diagnosis of the patients. The diagnosis and diseases words are features that the tf-idf method will judge as very important, as they appear only in few documents overall. An additional problem, however, is that of different spellings of the same disease. Consider for example the disease "chronic lymphocytic leukemia". Spellings range from "CLL" over "B-CLL" to "chronic-lymphocytic-leukemia". All these spellings are regarded by all our models as separate entities. However, the tf-idf vectors are still useful document embeddings, as the overall word statistics still differ depending on the diagnosis. Words describing medication for leukemia for example are still present in all letters of patients with CLL. These medication words, however, are again present only in few documents overall and are thereby considered more important features by the tf-idf method.

4.2 Recommendation Quality—Experimental Setup

Having fine-tuned the recommendation procedure as described above the next step consists of assessing the quality of the recommendation. We test this quality in a psychological experiment. To this end we probe the similarity ratings that subjects give to pairs of letters and compare them to the similarity measure of our algorithm.

We construct an experiment with 32 “trials”, in which subjects have to compare letter pairs for similarity. More precisely that means we select 32 letters as “reference letters” out of our database and let subjects rate the similarity of these 32 reference letters to five other letters each. Thereby we gain ratings for 160 letter pairs. 16 of the reference letters have a follow-up letter in our database. Trials with this kind of reference letter are called “follow-up trials” and letter pairs of follow-ups are called “follow-up pairs”. The other 16 reference letters are selected randomly among the letters without follow-ups present in our dataset. The five letters that are compared to one reference letter are called the comparison letters. Four of them are selected based on the cosine similarity between the reference letter and all other letters. These four are the ones with highest cosine similarity to the reference letter according to our algorithm. That means they are the best matches to the reference letter according to our algorithm—or equivalently, they are ranked on places one to four given a reference letter. The fifth comparison letter is randomly selected among all other letters and then fixed. Subjects are presented with one reference and one comparison letter at a time. After rating their similarity they are presented with the next comparison letter. Once a trial is done, i.e. five comparison letters are rated, the next reference letter is presented. The order of the reference and comparison letters is random, but fixed. Subjects are forced to give a rating in the range of 1 (very dissimilar) to 7 (very similar) for each letter pair.

We design the experiment such that the first trial is a follow-up trial and the second one is a non-follow-up trial. Thereby we ensure that subjects can adjust for the upper similarity bound of having to compare letters of the same patient. These two trials are excluded for the later analysis. We have six subjects performing the experiment, four experts (oncologists with at least five years of practical experience) and two novices (medical students more than halfway through their study course).

4.3 Recommendation Quality—Results

Inter-Rater Agreement

We first analyze the inter-rater agreement among subjects. We believe it is not trivial for them to rate physician letter pairs for similarity as it is not directly clear along which dimension similarity is to be judged, a problem well known from the cognitive science literature (Medin et al., 1993). One might for example judge similarity based on diagnosis or based on therapy. We also believe that experts and novices (or students) base their judgments on different features. Experts generally have higher agreement when categorizing stimuli, than novices do, and usually use more abstract, less accessible features (Chi et al., 1981; Linhares and Brum, 2007; León-Villagrá and Jäkel, 2013). In line with these results we find that the inter-rater agreement is higher among the experts than among the students (expert agreement: 0.76; student agreement: 0.59). We measure this agreement with the spearman rank correlation coefficient, that produces a number between -1 (perfect anti-correlation) and 1 (perfect correlation). Spearman's coefficient is used instead of the often used cohen's kappa coefficient, because the former can deal with ordinal data, whereas the latter can only deal with nominal data (Spearman, 1904; Cohen, 1960). Our number of subjects is quite low for concluding that students rate letter pairs worse than experts. Still as our findings are in line with well established research, we discard the student rating data for further analysis.

Note that for the following analyses we exclude data of follow-up pairs, except where explicitly stated otherwise. Subjects rate the similarity of these pairs very highly and almost any information retrieval system will find them to be similar. Thereby they would improve positive correlation statistics in our analysis, although retrieving them is useless in practice. We will show later that our recommender can easily distinguish them from normal pairs.

Ranking and Subject Ratings

The first analysis concerning our recommender system asks the question whether the recommendations are better than chance. Therefore we compare the average subject rating of the "best fitting" letter (as computed with our algorithm) and the random comparison letter for each trial. The best fitting comparison letter is the one with highest cosine similarity to the reference letter. Figure 4.1a shows a histogram of the differences in subject rating of the best fitting and the randomly selected letter pairs. The figure also shows the mean difference and a 95% interval for this mean. The confidence interval was calculated using the standard error of the mean (sem). The lower and upper bound is then calculated as $bound_{1/2} = mean \pm 1.96 * sem$ respectively. This estimation of the confidence interval, however relies on the assumption that sample means are normally distributed. To verify this assumption we use bootstrapping (Efron, 1979).

Bootstrapping is a procedure in which many random resamples of the dataset are computed. If our collected dataset represents the true distribution of differences in ratings of best fitting and random letter pairs well enough, then the resamples are to our sample as our sample is to the true distribution. Thereby the mean of the average difference of the resampled datasets give an estimate of the average differences, when collecting many samples from the real distribution. We take 100.000 resamples with replacement from our data and compute the average difference of the resamples for every one of them. The distribution of these average differences of the resamples can be seen in figure 4.1b. From this figure it is apparent, that the assumption of normality for the distribution of average differences indeed holds. The figure also shows bootstrap estimates for the mean and the 95% confidence interval. As expected these estimations match closely with the ones estimated with the standard error of the mean as can be seen from the following table.

Estimated Statistic	Mean	Confidence Interval
Standard Estimation	1.93	[1.17, 2.70]
Bootstrap Estimation	1.93	[1.14, 2.68]

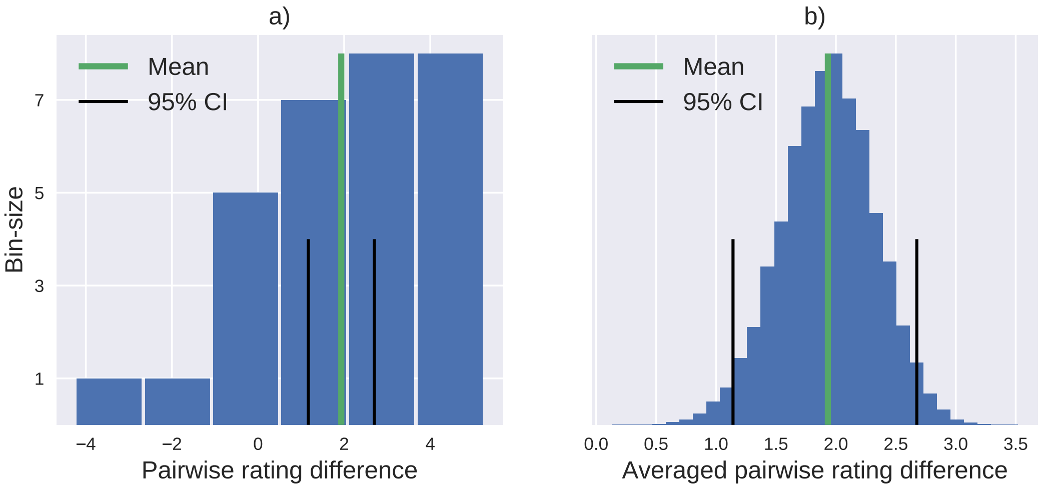

Figure 4.1: (a): Histogram of differences in rating of the "best" and the random comparison letter for each trial. **(b):** Histogram of the average differences in averaged subject ratings of the two groups for 100.000 bootstrap resampled datasets.

Cosine Similarity and Subject Ratings

After analyzing whether ranking letter pairs with our algorithm works better than chance, we analyze the relationship of the cosine similarity and the average subject ratings of letter pairs more directly. We visualize the mean cosine similarity, that our algorithm assigns to pairs, as a function of the subject rating in figure 4.2. The data shows a positive, close to linear correlation (pearson correlation: 0.96) of those two variables, suggesting that our recommendation method captures at least some aspects of perceived similarity.

Next we analyze the relationship of the cosine similarity and the subject ratings more thoroughly. We therefore draw all letter pairs as points in a plot of average rating of a pair versus cosine similarity of a pair. See figure 4.3 for this visualization. We mark follow-up pairs green and the random comparison letter pairs red. From the figure it is apparent that follow-up letters can be easily distinguished with the cosine similarity from other letter pairs. The randomly selected pairs mostly have a low cosine similarity as expected, however several of them are still rated rather highly by subjects. Some well fitting random letter pairs

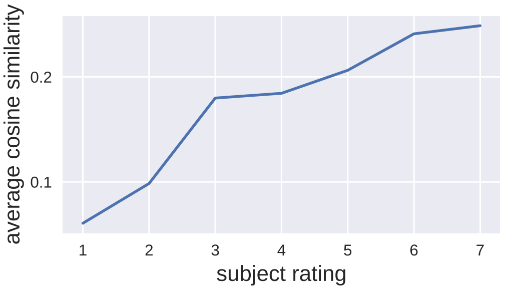

Figure 4.2: Averaged cosine similarity of all letter pairs that were rated into one of the seven possible rating categories.

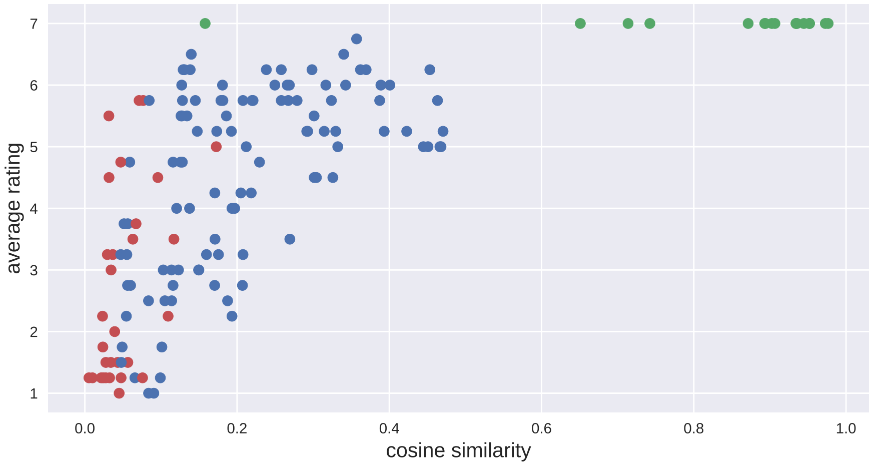

Figure 4.3: Average subject rating versus cosine similarity of all letter pairs. Follow-up pairs are marked green, random ones red.

are expected by chance and after manual inspection of these pairs we settle on this conclusion.

Considerations of Precision and Recall

A standard way to assess the quality of an information retrieval system uses two measures called precision and recall. Precision and recall are used as measures of the goodness of an information retrieval system, when all documents are binarily classified as either relevant or irrelevant given a current information need or query (Manning et al., 2008*a*). In our case a query is a reference letter and we ask the question of whether a comparison letter is relevant or irrelevant given this reference letter. Precision is then a measure of the correctness of the retrieved results. In probabilistic terms it is the probability that a document is relevant given that it is retrieved P(relevant|retrieved). Recall is a measure of completeness of the retrieved results and can be expressed as the probability of being retrieved given that the document is relevant $\mathrm{P}(retrieved|relevant)$. Both quantities usually exhibit an inverse relationship. One can trade higher precision for lower recall or vice versa. Both precision and recall always lie in $[0, 1]$. Note that we classify letter pairs with an average rating of five or higher as relevant and others as irrelevant. This threshold is somewhat arbitrary, but was set after several personal discussions with the contributing physicians.

We can assess the precision of our system in different scenarios. Recall, however, is impossible for us to measure. We do not know the set of relevant letters for a given information need (i.e. reference letter) and therefore cannot compute $\mathrm{P}(retrieved|relevant)$ (i.e. recall). Precision is a quantity of high interest for the

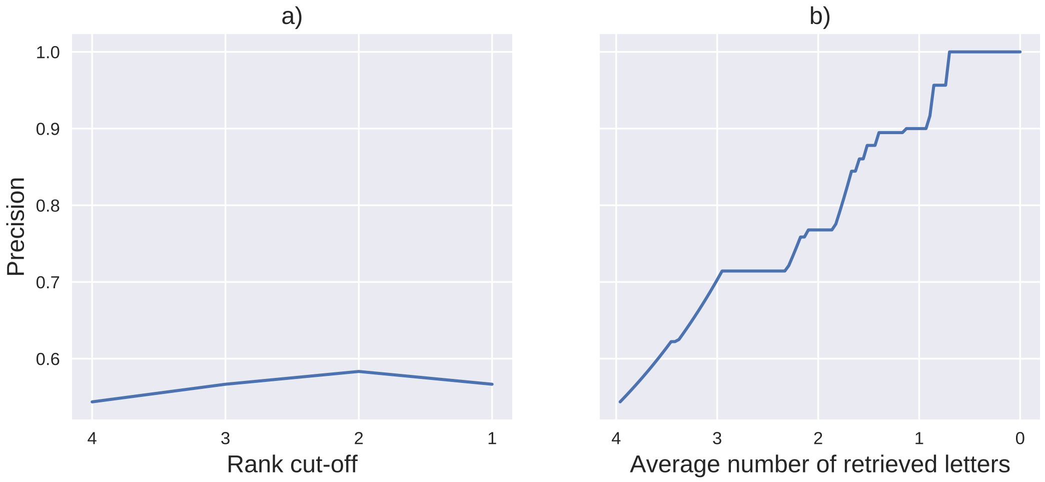

Figure 4.4: (a): Precision as a functin of rank cut-off. **(b):** Precision as function of a varying cosine similarity threshold. The x-axis shows the average number of retrieved letters per reference letter at the threshold instead of the threshold itself.
Note that both plots start at the same point, where all letters with rank four or lower are retrieved, and share a common y-axis.

recommender system in the use-case we envision. It is very important for a setting in clinical everyday life, that a large fraction of recommended letters is deemed useful by doctors. If this were not the case, physicians would quickly stop using the system. Recall on the other hand is not an interesting quantity in this situation. Doctors will only look at very few recommended letters and it is of no concern whether or not a large fraction of relevant documents is retrieved. However, an interesting question, somewhat related to recall, is whether or not the most relevant letters are retrieved first. We will address this question after examining the precision of our system. Note that for another use-case recall might be more important than precision.

In our usage scenario a possible implementation of the system could always retrieve the four highest ranked letters to a given reference letter. Therefore we ask the question of what precision we can achieve, when recommending the "best" four or even fewer letters. Figure 4.4a shows what level of precision the system can achieve, when stopping retrieval of letters at a given rank. As is apparent from this figure not much increase in precision can be gained by presenting a fixed number of fewer letters and the precision in all those cases is rather low at around 0.55. This means that little more than every second retrieved letter is relevant. From figure 4.3 it is apparent, however, that we can do better, when incorporating information about the absolute cosine similarity and not only the ranking of the letters. In this figure it is apparent, that we can get up to a precision of 1, if we present only letters with a cosine similarity higher than 0.4. This in turn means that only for few reference letters some comparison letters are retrieved. In figure 4.4b we examine this relationship more closely. We restrict ourselves to letters of rank four or lower and plot the precision of retrieved letters as a function of a varying cosine similarity threshold. Instead of labeling the x-axis with this threshold, however, we show the average number of retrieved letters. Thereby we can visualize the trade-off between higher precision and fewer retrieved letters. Increasing precision this way is much easier achievable. This though comes with the cost that for some reference letters no comparison letters might be retrieved.

Next we turn to the question whether the most relevant letters are retrieved first. This question is hard to answer, since we only know of five letters how relevant they are to a reference letter. It remains unclear what rating subjects would have assigned to the other letters from our database. A related question though can provide some insight into this problem. We look at how well the ranking of the algorithm corresponds to the ranking of the experts, given by their rating. We compare these two rankings with the spearman rank correlation coefficient. The correlation between the ranking given by the average rating and the ranking of the algorithm is 0.39. While this is not particularly high, it is still far better than chance (spearman coefficient of 0). The inter-rater agreement among the experts when calculated accordingly (i.e. without follow-up pairs) is 0.72. Summarizing one can state that our systems ranking is well above chance, but still quite far below human gold standard.

Considerations for further Work

We have two reasons to believe that our system performs even better in reality than expected from the considerations above. First, when extending our recommender to work on a much larger database, we expect that the best fitting letters to a given reference letter will have much higher cosine similarity, than is the case now. This is because in a much larger database more well fitting documents are expected for each reference letter. For figure 4.3 this means, that we would expect almost no blue points on the left and many more in the region around a cosine similarity of 0.4. These points most likely will be rated highly by subjects, as are all points with such a cosine similarity in our experiment. In figure 4.4a and 4.4b this probably shifts both curves up higher. Second we have reason to believe that some of the letter pairs with low cosine similarity, but high subject rating are due to psychological adjusting. If during one trial only rather badly fitting letters are presented, we believe that subjects adjust their rating scale and rate partially fitting letters higher than normally (!correct! Frank nach Paper fragen). Therefore we believe that points in the upper left corner of figure 4.3 are not as problematic as it might seem. In the figure it looks like some are highly relevant although they have very low cosine similarity and will therefore probably not be retrieved in practice. If the assumption, that subjects adjust their rating scales, is correct, then some of the letters in this region are probably less relevant than it seems from our data.

Finally one can ask whether there are possibilities to improve the system further. The tf-idf embedding method does not have hyperparameters, so no further tuning of those is possible. We believe, however, that combining similarity measures from

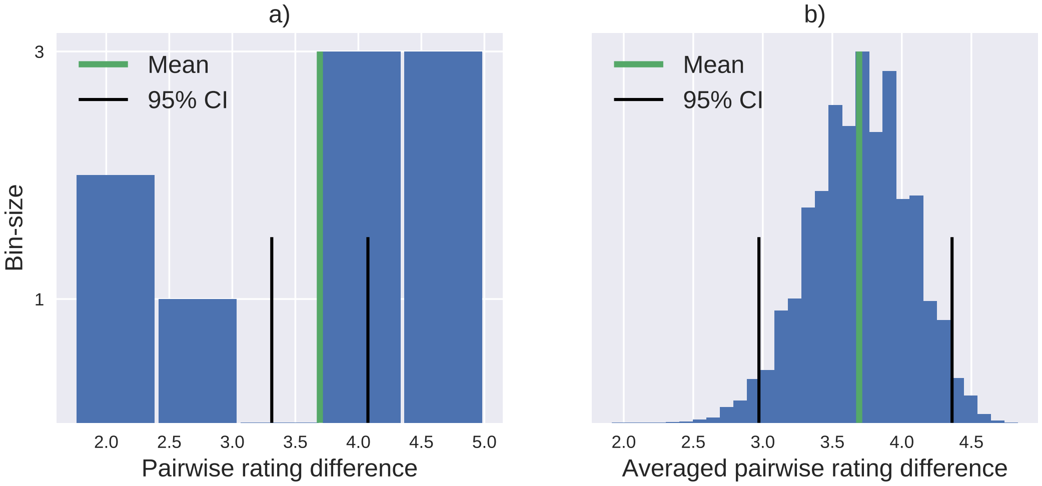

Figure 4.5: **(a):** Histogram of differences in rating of the "best" and the random comparison letter for each trial. **(b):** Histogram of the differences in the means of averaged subject ratings of the two groups for 100.000 bootstrap resampled datasets.

several embedding approaches is useful. Of the five embedding methods we tested only paragraph vector is a promising option for improving the results of the retrieval. Bag of words is more or less the simple basis for tf-idf, LSA is highly correlated with tf-idf as it is a feature compressed version of the latter, LDA performed severely worse than all other methods, leaving only paragraph vector. This method performed almost as well as tf-idf during our fine-tuning and computes feature vectors in a very different way, possibly complementing tf-idf well. Indeed we find cues that combining these two methods yields substantially better results. We again compare the ratings given to letter pairs from two groups. One of the groups consists of the letters from our experimental dataset, that paragraph vector ranks on the first place out of all letters from our complete database. I.e. letters in this group are ranked among the top four by tf-idf and top one by paragraph vector in our whole dataset. We only find 9 of these letters out of the 30 possible trials. The other group consists of the 9 randomly selected letters from the corresponding trials. Again we look at a histogram of differences in rating in figure 4.5a and the distribution of average differences for 100.000 bootstrap resamples in figure 4.5b. Comparing theses with figures 4.1a and 4.1b, where only tf-idf information was used, we see that additionally using information from the paragraph vector embedding improves results substantially. The mean of the average difference in ratings for example is improved from 1.93 to 3.69 units on the rating scale from 1 to 7. We therefore believe that work addressing how to combine information from different embedding methods is a worthwhile task and should be undertaken by further research.

Chapter 5

Intelligent Search

With the set of methods used throughout this work, we can address a related, but distinct problem. Say for research purposes it is necessary to find as many documents of patients with a certain feature as possible. Traditionally this can be done with the help of string matching algorithms, that search the database. Even in easy scenarios, however, these algorithms might fail to retrieve all relevant documents. As mentioned earlier diseases are spelled differently in different documents. Additionally one might be interested in all patients with a specific disease of themselves, but a string matching algorithm returns documents where this disease is mentioned in the family anamnesis, as well.

The question arises whether we can do better, when using the embedding methods described earlier. We therefore take a subset of 135 physician letters and label them manually for five prevalent diseases—“Chronic lymphocytic leukemia”, “multiple myeloma”, “breast cancer”, “follicular lymphoma” and “diffuse large B-cell lymphoma”. In the subset of 135 letters these diseases appear between 9 and 14 times. We start out with one letter from one of the groups, retrieve the best match according to our retrieval method, classify this retrieved letter as either relevant or non-relevant, update our retrieval method and present the next candidate. Thereby we ask with how many false positives we have to deal until we find a prespecified fraction of relevant letters.

We specifically address the question with how many false positives we have to deal, when trying to retrieve different fractions of the relevant letters to one of the information needs given by the diseases. We always start with one disease letter, present the best match according to our retrieval method, classify the first retrieved letter as relevant or non-relevant, update our retrieval method and present the next candidate.

Bibliography

Aamodt, A. and Plaza, E. (1994), ‘Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches’, *AI COMMUNICATIONS* **7**(1), 39–59.

Begum, S., Ahmed, M. U., Funk, P., Xiong, N. and Folke, M. (2011), ‘Case-Based Reasoning Systems in the Health Sciences: A Survey of Recent Trends and Developments’, *IEEE Transactions on Systems, Man, and Cybernetics* **41**(4), 421–434.

Bengio, Y., Vincent, P. and Jauvin, C. (2003), ‘A Neural Probabilistic Language Model’, *Machine Learning Research* **3**, 1137–1155.

Blei, D. M., Ng, A. Y. and Jordan, M. I. (2003), ‘Latent Dirichlet Allocation’, *Machine Learning Research* **3**, 993–1022.

Chi, M. T. H., Feltovich, P. J. and Glaser, R. (1981), ‘Categorization and Representation of Physics Problems by Experts and Novices’, *Cognitive Science* **5**(2), 121–152.

Cohen, J. (1960), ‘A Coefficient of Agreement for Nominal Scales’, *Educational and Psychological Measurement* **20**(1), 37–46.

Deerwester, S., Dumais, S. T., Furnas, G. W., Landauer, T. K. and Harshman, R. (1990), ‘Indexing by Latent Semantic Analysis’, *American Society for Information Science* **41**(6), 391–407.

Dib, E. G., Kidd, M. R. and Saltman, D. C. (2008), ‘Case reports and the fight against cancer’, *Journal of Medical Case Reports* **2**(1), 39.

Efron, B. (1979), ‘Bootstrap Methods: Another Look at the Jackknife’, *The Annals of Statistics* **7**(1), 1–26.

Elstein, A. S. and Schwarz, A. (2002), ‘Clinical problem solving and diagnostic decision making: selective review of the cognitive literature’, *British Medical Journal* **324**(March), 729–732.

Gutmann, M. U. and Hyvärinen, A. (2012), ‘Noise-Contrastive Estimation of Unnormalized Statistical Models, with Applications to Natural Image Statistics’, *Machine Learning Research* **13**, 307–361.

Kidd, M. and Hubbard, C. (2007), ‘Introducing Journal of Medical Case Reports’, *Journal of Medical Case Reports* **1**, 1.

Klein, G. (2008), ‘Naturalistic Decision Making’, *Human Factors: The Journal of the Human Factors and Ergonomics Society* .

Kolodner, J. L. and Kolodner, R. M. (1987), ‘Using Experience in Clinical Problem Solving: Introduction and Framework’, *IEEE Transactions on Systems, Man, and Cybernetics* **17**(3).

Lau, J. H. and Baldwin, T. (2016), ‘An Empirical Evaluation of doc2vec with Practical Insights into Document Embedding Generation’, *CoRR* .

Le, Q. and Mikolov, T. (2014), Distributed Representations of Sentences and Documents, *in* ‘Proceedings of the International Conference on Machine Learning’, Vol. 32, pp. 1188–1196.

León-Villagrá, P. and Jäkel, F. (2013), ‘Categorization and Abstract Similarity in Chess’, *CogSci* pp. 2860–2865.

Linhares, A. and Brum, P. (2007), ‘Understanding Our Understanding of Strategic Scenarios: What Role Do Chunks Play?’, *Cognitive Science* **31**(6), 989–1007.

Manning, C. D., Raghavan, P. and Schütze, H. (2008*a*), Evaluation of unranked retrieval sets, *in* ‘Introduction to Information Retrieval’, Cambridge University Press, chapter 8, pp. 154–157.

Manning, C. D., Raghavan, P. and Schütze, H. (2008*b*), Scoring, term weighting and the vector space model, *in* ‘Introduction to Information Retrieval’, Cambridge University Press, chapter 6, pp. 117–120.

Manning, C. D., Raghavan, P. and Schütze, H. (2008*c*), The term vocabulary and postings lists, *in* ‘Introduction to Information Retrieval’, Cambridge University Press, chapter 2, pp. 22–27, 32–34.

Mason, R. A. (2001), ‘The case report - an endangered species?’, *Anaesthesia* **56**(2), 99–102.

Medin, D. L., Goldstone, R. L. and Gentner, D. (1993), ‘Respects for Similarity’, *Psychological Review* **100**, 254–278.

Medin, D. L. and Schaffer, M. M. (1978), ‘Context Theory of Classification Learning’, *Psychological Review* **85**(3), 207–238.

Mikolov, T., Chen, K., Corrado, G. and Dean, J. (2013), Efficient Estimation of Word Representations in Vector Space, *in* ‘Proceedings of Workshop at the International Conference on Learning Representations’.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. and Dean, J. (2013), Distributed Representations of Words and Phrases and their Compositionality, *in* 'Advances in Neural Information Processing Systems', pp. 3111–3119.

Morin, F. and Bengio, Y. (2005), Hierarchical Probabilistic Neural Network Language Model, *in* 'Proceedings of the international workshop on artificial intelligence and statistics', pp. 246–252.

Nissen, T. and Wynn, R. (2014), 'The clinical case report: a review of its merits and limitations', *BioMedCentral Research Notes* **7**(1), 264.

Sandu, N., Chowdhury, T. and Schaller, B. J. (2016), 'How to apply case reports in clinical practice using surrogate models via example of the trigeminocardiac reflex', *Journal of Medical Case Reports* **10**(1), 84.
URL: *http://dx.doi.org/10.1186/s13256-016-0849-z*

Spadaro, S. (2012), ClinicOn Kurzbeschreibung für neue und interessierte Anwender, Technical report.

Spearman, C. (1904), 'The Proof and Measurement of Association between Two Things', *The American Journal of Psychology* **15**(1), 72–101.

Williams, D. D. R. (2003), 'In defence of the case report', *The Royal College of Psychiatrists* **184**, 84–88.

Declaration of Authorship

I hereby certify that the work presented here is, to the best of my knowledge and belief, original and the result of my own investigations, except as acknowledged, and has not been submitted, either in part or whole, for a degree at this or any other university.

Osnabrück, June 9, 2017