university of groningen / faculty of mathematics and natural sciences / artificial intelligence

MASTER'S THESIS

Medical Incident Report Classification using Context-based Word Embeddings

Steven Bosch
s1861948

Department of Artificial Intelligence
University of Groningen, The Netherlands

University Medical Center Groningen, The Netherlands

Primary supervisor:	Dr. M.A. Wiering (Artificial Intelligence, University of Groningen)
Secondary supervisor:	Dr. F. Cnossen (Artificial Intelligence, University of Groningen)
External supervisor:	Drs. S. Meijer (University Medical Center Groningen)

September 15, 2017

Abstract

The University Medical Center Groningen is one of the largest hospitals in The Netherlands, employing over 10.000 people. In a hospital of this size incidents are bound to occur on a regular basis. Most of these incidents are reported extensively, but the time consuming nature of analyzing their textual descriptions and the sheer number of reports make it costly to process them. Therefore, this thesis proposes ways of employing machine learning techniques to process the incident reports more efficiently and effectively. More specifically, we show how context-based word embeddings, vector representations of words, can be used to enhance searching capabilities within the report database and how they can be combined with classifiers to predict labels attributed to the incidents. For these purposes, we subjected several word embedding and classification techniques to a comparative analysis.

To evaluate word embedding techniques, we propose a method to measure the extent to which word embeddings that are similar in vector space, represent words that are associated with each other. Using this method, we find that the *continuous bag-of-words* architecture developed by Mikolov et al. performs best out of a selection of six different word embedding methods. Furthermore, we compare different methods for embedding entire incident reports, and find that simply averaging word embeddings in combination with an inverse document frequency weighting function yields better results than several recently introduced techniques.

For the prediction of class labels, the combination of such report embeddings with multilayer perceptrons proves most promising out of a selection of eleven different combinations of classifier and input type (bag of words, word embeddings or report embeddings). We find that three out of five tested categories are predicted well above the baseline, established by always selecting the most frequent label, whereas the other two maximally yield baseline performances.

Finally, to improve searching capabilities within the report database, we developed an application that makes use of the learned word embeddings by providing the user with search suggestions based on a search query. The Central Incident Commission, which is concerned with processing the incident reports, experiences this functionality to drastically increase the efficiency and effectiveness of search queries, both saving them time and helping them find more complete patterns within the data.

Acknowledgements

My gratitude goes to everyone who has helped me bring this project to a successful conclusion. In particular, I want to thank my primary supervisor, Dr. M.A. Wiering, for guiding me throughout the project. With his expertise in machine learning and his experience in research in general he helped me tackle the technical challenges of the project and write this thesis. I would like to thank my secondary supervisor, Dr. F. Cnossen. She set up this project, made sure the applied side of it was given the right amount of attention and helped bring this thesis to its final form.

My gratitude goes to Ko Jans and Stella Meijer, my supervisors at the UMCG, who taught me much about the workings of the hospital and brought me into contact with the right people. Their enthusiasm and positive attitude motivated me to go for this project a hundred percent. It was a delight to work for and with them and I was pleased to hear they were content with the things I was able to provide them with. I hope to have left them an application that can assist them with the good work they are doing and possibly might lead to future developments in this area.

Finally, I would like to thank Ruben Gabriëls, a medical graduate student, without whom my lunch breaks throughout the course of this project would have been quite less memorable.

Steven Bosch
September 15, 2017

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Over the past decades machine learning has taken on an increasingly important role in many companies and institutions. Both the development of machine learning techniques and the enormous expansion of available data have given a great impulse to its application for commercial and non-commercial purposes. One area that experiences an increasing interest in applying machine learning is the medical field. Many hospitals and other medical institutions have been using software for various purposes, over time accumulating extensive databases. These databases can be of great value in acquiring knowledge concerning the process with which the software is concerned. One such database is the center of interest in this research project.

The University Medical Center Groningen (UMCG) is one of the largest hospitals in The Netherlands, employing over 10.000 people in health care, research and education.[1] It offers basic care as well as specialized and exclusive treatments. In a hospital of this size it is not surprising that incidents occur on a regular basis. The nature and causes of these incidents can differ greatly, ranging from relatively minor events, such as medication dosages being slightly off, to incidents with greater consequences, such as surgeries going wrong.

Since 2008 the UMCG has been using software with which employees can report any incident that occurred. Such a report involves a description of the incident, as well as categorical information concerning the incident, such as causes, risks and department of occurrence, and meta information on the report and reporter. The incident reports are processed by committees on either a decentralized or centralized level, and depending on the evaluation of these committees steps can be taken to prevent similar incidents in the future. While the different categorical variables in the data contain helpful information in processing the incidents, most of the useful information is stored within their textual descriptions. Unfortunately, the time consuming nature of processing these descriptions and the sheer number of incident reports make it costly to effectively process all of them. Moreover, since not all reports are centrally processed, recurring patterns can be missed or picked up on in a relatively late stadium.

To process the incident reports more efficiently and effectively, a machine learning approach is sought after. Specifically, the Central Incident report Commission (CIM) of the UMCG would be benefited in two ways: (1) improving the effectiveness of search queries by incorporating content-related words in the search results and (2) providing a means of knowing whether new reports contain content that has not been encountered in the database before.

To address these issues, we concern ourselves with a central issue distinguished in natural language processing (NLP): how are words represented in such a way that their semantic properties are captured and machine learning techniques can leverage these properties? This is difficult, because the symbolic representation of a word conveys no meaning, other than what we, humans, ascribe to it. We learn their meaning by connecting their representation to a concept that exists in the real world, whereas the sole features of words that can be used by statistical models are their occurrence frequency in documents or corpora as a whole, and their co-occurrence with other symbolic entities. Traditional NLP systems often exploit the former, treating words as discrete atomic symbols. These can be used in statistical models, but capture no information about the relationship between words and therefore contain little semantic information. On the other hand, conventional methods of representing these relationships, such as word webs, are usually unfit for the machine learning techniques that we aim to apply.

[1] https://www.umcg.nl/NL/UMCG/overhetumcg/Paginas/default.aspx

A concept that can combine both of these features is called *word embeddings*. Word embeddings are vector representations of words that can be mapped in a continuous vector space by means of a *vector space model* (VSM) [92], a model that has been widely used in NLP methods [3, 41, 59, 96, 97]. Many such VSMs make use of the property that semantically similar words tend to have similar contextual distributions. This is called the *distributional hypothesis* [34, 77]. These *distributional semantic models* (DSM) are a type of statistical language model that uses vectors to keep track of the contexts in which words appear. In other words, looking at contexts tells the model something about the relationship between words, which is useful information and is therefore captured in word embeddings, or in vectors embedding entire paragraphs or documents.

Therefore, this research project aims to find the method that generates the 'best' word embeddings, as well as utilizing these embeddings to address the requests of the CIM through classification of reports and finding similarities between reports. The problem this research concerns itself with can thus be framed into the following research question:

How can context-based word embeddings be combined with machine learning techniques to make predictions about and find similarities between medical incident reports?

This question can be decomposed into the following subquestions:

1. Which technique yields the word embeddings that best represent their semantic properties?
2. How are single word embeddings combined into meaningful report representations?
3. Which combination of paragraph representation and classification technique performs best?
4. Can word embeddings be used to construct a similarity measure between different paragraphs?

Simultaneously, with respect to the applied side of this research, the following questions are investigated regarding the concrete process of extracting useful information from the incident report database:

5. Using word embeddings, can a search query implicitly be expanded to incorporate both the target word and words similar in content, such as synonyms?
6. How can word embeddings contribute to the process of detecting words that have not themselves, nor any content-related word, been encountered before in the report database?

Classification using medical incident reports has been applied in several cases [112, 56, 54, 11, 83], but these mostly use rule-based classifiers and manually picked keywords instead of utilizing word embeddings. On the other hand word embeddings have been used for text classification [111, 110, 75, 21, 52, 121, 126], but these cases were concerned with proof-of-concept problems, such as sentiment classification in Twitter messages. We have not found any literature dealing with classification in medical incident reports, or medical texts in general, that make use of word embeddings.

Word embeddings have become increasingly popular for machine learning purposes. As a result, many techniques have been developed for generating them. While most of these techniques have been proven to achieve outstanding results when trained on large, English corpora and tested on general topics, it is unclear how they perform given a highly characteristic corpus and test set such as the UMCG incident report database, which additionally is in Dutch. Therefore it is important that these techniques are assessed and compared to gain a clear understanding of their merits and drawbacks with respect to such databases. Furthermore, while word embeddings have already received a fair amount of attention, this is less the case for sentence or paragraph embeddings. Developing and testing these techniques further can be of importance for several types of text processing systems, such as machine translation, human-machine communication and document retrieval, classification and clustering.

This thesis is organized as follows. Chapters 2, 3 and 4 treat the theoretical background of word embeddings, paragraph embeddings and classifiers, including the particular techniques that are subjected to a comparative analysis. Chapter 5 specifies the used training data, the method configurations and the experiments to evaluate these methods, after which the application of these techniques in the form of a search tool is elaborated in chapter 6. Finally, the results of the performed experiments are provided in chapter 7 and subsequently discussed in chapter 8.

Part I

Theoretical Background

Chapter 2

Word Embedding

As was explained in the introductory chapter, *distributional semantic models* (DSM) are statistical language models that use vectors to keep track of the contexts in which words appear, possibly learning representations for these words, *word embeddings*, in the process. While many such models have proven successful, they carry with them one intrinsic problem: the *curse of dimensionality*. Languages can produce huge numbers of word combinations given a finite sequence size. For example, just a five word sequence using a vocabulary of 100.000 words can already provide 10^{25} possible combinations. Because of this, systems can encounter word combinations that have not been trained on. This problem has been countered in different ways.

Traditional DSMs such as *latent semantic analysis* (LSA) [23] or *hyperspace analogue to language* (HAL) [68] capture co-occurrences of words with a particular type of information, such as the type of document the words occur in, and use dimensionality reduction techniques such as *singular value decomposition* (SVD) [26] to create a statistical model of reduced complexity. Such *matrix factorization methods* [89] often make use of *n-grams* [47, 50, 12] to factor in a part of dependency information within word sequences while keeping the dimensionality limited. *Shallow window-based methods* on the other hand do not use the entirety of co-occurrences in the corpus as basis for a statistical model, but apply a window-based approach. These methods tend to be less computationally expensive than the former, but cannot leverage the information gained from global corpus statistics, although it has been argued that window-based approaches are implicitly performing matrix factorization as well [60].

The effective learning of representative word embeddings has also been achieved by using bilingual corpora, where words of multiple languages that represent the same meaning have similar vectors [32, 114, 36, 2, 37]. Such methods are however not usable for this research, as we are dealing with a dataset containing highly specialized medical jargon. To our knowledge there are no available bilingual corpora that contain enough words used in the data set.

Another distinction is proposed by Baroni et al. [4] as *count-based* versus *context-predicting models*. Count-based methods rely on word embeddings for which co-occurrence counts form the basis. These embeddings are often subject to several transformations, such as reweighing the counts based on informativeness or smoothing with dimensionality reduction techniques. This is generally done unsupervised. In the past years however, a new generation of DSMs has been developed, in which the weights in a word vector are set to maximize the probability of the contexts in which the word is observed. These methods apply the idea of distributed representations in combination with neural networks, first proposed by [5]: representing a word within several 'meaningful' dimensions. In these methods, instead of first collecting context vectors and then reweighing them, the vector weights are directly set to optimally predict the contexts in which the corresponding words tend to appear. As such, these DSMs are learning in a supervised way, as opposed to the count-based methods.

Baroni et al. conclude their comparison in favor of the predict models, particularly the highly popular Word2Vec method proposed by Mikolov et al. [71, 75, 76], which has produced good results in word analogy and classification tasks [118, 119, 123]. However, other research has since indicated that count-based models such as Singular [106] and PSDVec [65] or hybrid systems such as GloVe [89] can still outperform predict models such as Word2Vec.

The different techniques have since been compared, with different conclusions as a result. In [61] Levy et al. argue that much of the performance gains of the word embeddings produced by different techniques

can be ascribed to system design and hyperparameter optimizations, rather than the algorithms themselves. Alternatively, Akata et al. [1] suggest that the embeddings are task-specific, as they show that one technique outperforms the other in a particular task, whereas the opposite happens in a different task. In [82] and [127] this is also found true for some tasks, whereas for other tasks their performances were near equal, which is in line with the conclusions made by Levy et al. Finally, combinations of the techniques have also been suggested [104, 121, 126], which might prove to be the best solution of all. As such, it seems that no consensus has been reached as to the question whether count- or predict-based models should be favored in the case of large English corpora. We will extend the discussion by comparing how the techniques perform in our small and specialized database.

The subsequent sections give an overview of the ideas and mathematics behind statistical and neural network language models in general, and in particular explain the details of the word embedding architectures that will be compared in this thesis.

2.1 Statistical Language Models

Statistical language models capture the probability distribution of the occurrences of word sequences, $P(w_1, w_2, w_3, ..., w_t) = P(w_1^{t-1}, w_t)$. Following the general probability chain rule [86], the probability of word sequence W of length N can be expressed as:

$$P(W) = \prod_{t=1}^{N} P(w_t|w_1^{t-1}) \tag{2.1}$$

where w_1^{t-1} denotes the context words. Knowing the probability distribution can be useful for making predictions about the class of documents a sequence belongs to (e.g. in spam detection) or the next word in a sequence (e.g. in speech and handwriting recognition). The goal of language modeling is thus to find a way to accurately estimate $P(W)$. There are two problems with accomplishing this. First, an objective prior probability of a word does not exist, since the use of language is variable: the parameters of the distribution are not set such as with a dice roll. Second, the curse of dimensionality prevents the knowledge of all word sequences of a given length, since there are simply too many possibilities.

As a consequence, statistical models can only approximate the probability distribution. To do this, typically large training sets are required, since more data means more training sequences and therefore less sparsity (given that the vocabulary does not increase with the size of the training set). However, even if huge training sets are available, the parameter space $P(w_t|w_1^{t-1})$ is simply too big for reasonable computation times. This forces methods to put restrictions on the context w_1^{t-1} in the form of an equivalence class, given by a function $\Phi(w_1^{t-1})$:

$$P(W) = \prod_{t=1}^{N} P(w_t|\Phi(w_1^{t-1})) \tag{2.2}$$

The most used and arguably most successful equivalence class paradigm, is to give the context size of the words a fixed length $n-1$:

$$P(W) \approx \prod_{t=1}^{N} P(w_t|w_{t-n+1}^{t-1}) \tag{2.3}$$

This restriction, called *n-grams*, limits the dimensionality of the context and as such makes calculations on it possible, at the cost of information. *N-gram models* then calculate the conditional probability of a word w_i based on sequence occurrence counts:

$$P(w_t|w_{t-n+1}^{t-1}) = \frac{c(w_{t-n+1}^{t})}{c(w_{t-n+1}^{t-1})} \tag{2.4}$$

where $c(w)$ is the count of w in the given corpus. While n-gram models have been successful, their biggest problem is dealing with unencountered sequences, since these can result in zero division. To deal with this, many different smoothing functions have been applied, such as Good-Turing frequency estimation [27], Katz's back-off trigram model [50] or interpolated trigram models [47]. The latter two use small context

sizes, usually of $n = 3$ (trigrams), which are less sparse than large context sizes. They then combine these to make predictions about larger contexts. More tricks and combinations have been used to cope with the problem, often producing good results [13, 28].

However, Bengio et al. [5] distinguished two characteristics of this approach that needed to be improved upon: (1) words are related to a much bigger context than one or two words, and (2) the semantical and syntactical similarity between words is not taken into account (e.g. having seen a training example with the word `cat`, should make a sentence with the word `dog` almost as likely, due to their similar use). The next section elaborates on a type of language model that introduces a way to deal with these shortcomings.

2.2 Neural Network Language Models

Statistical neural network language models (NNLM) using distributed feature vectors were introduced by Bengio et al. in 2003 [5]. Just as more traditional probabilistic language models they attempt to find a model $f(w_t, w_{t-n+1}^{t-1})$ for the probability of a word given its context words and a training set $w_1, ..., w_T$ of words $w_t \in V$, with V being the vocabulary, a finite and preferably large set of words:

$$f(w_t, w_{t-n+1}^{t-1}) = P(w_t | w_1^{t-1}) \tag{2.5}$$

The probability chain rule can subsequently be used to get the probability of word sequences (equation 2.1). The only constraint Bengio et al. put on the model is that for any context w_{t-n-1}^{t-1}:

$$\sum_{i=1}^{|V|} f(w_i, w_{t-n+1}^{t-1}) = 1, \text{ with } f(\cdot) > 0 \tag{2.6}$$

meaning that a sequence of words always expects a succeeding word to be in V, and that every word in the vocabulary w_i is possibly the target word w_t. They then decompose their model $f(\cdot)$ into two parts (as is shown in figure 2.1):

1. Each word w_i in V is mapped to a real m-dimensional feature vector $\mathbf{C}_{w_i} \in \mathbb{R}^m$: the word embedding. Every such vector contains the learned features for the word and is a column in $|V| \times m$ parameter matrix C.

2. A function g that projects an input sequence of context word feature vectors $(\mathbf{C}_{w_{t-n+1}}, ..., \mathbf{C}_{w_{t-1}})$ to a conditional probability distribution over V for the subsequent word w_t:

$$f(w_i, w_{t-n+1}^{t-1}) = g(w_i, \mathbf{C}_{w_{t-1}}, ..., \mathbf{C}_{w_{t-n+1}}) \tag{2.7}$$

The function $f(\cdot)$ is thus composed of the matrix containing the feature vectors $\mathbf{C}$ and the function $g(\cdot)$. The function $g(\cdot)$ is typically implemented by a feed-forward or recurrent neural network and its output is a vector with $|V|$ elements, where every element i estimates the probability of word w_i given the word sequence w_1^{t-1}: $P(w_t = w_i | w_{t-n+1}^{t-1})$.

The implementation of $g(\cdot)$ as proposed in [5] relies on a feed-forward neural network. In this type of network there are no cycles. All the information in the input nodes is processed in one direction and no output is fed back to the network, as can be the case in recurrent neural networks. The input layer of the network consists of n context word feature vectors of size m which are projected onto a projection layer x of $(n-1)m$ nodes by concatenation, giving:

$$x = (C_{w_{t-1}}, C_{w_{t-2}}, ..., C_{w_{t-n+1}}) \tag{2.8}$$

Furthermore the network uses one layer of h hidden units connected to the projection layer by $(n-1)mh$ weights w given in weight matrix W_1. These hidden units are connected to $|V|$ output units by $|V| \times h$ weights in weight matrix W_2. Using a hyperbolic tangent activation function, the unnormalized log-probability u_i for each output word w_i is then computed as:

$$u_i = b_i + \sum_{k=1}^{h} W_2^{ik} \tanh(d_k + \sum_{l=1}^{(n-1)m} W_1^{kl} x_l) \tag{2.9}$$

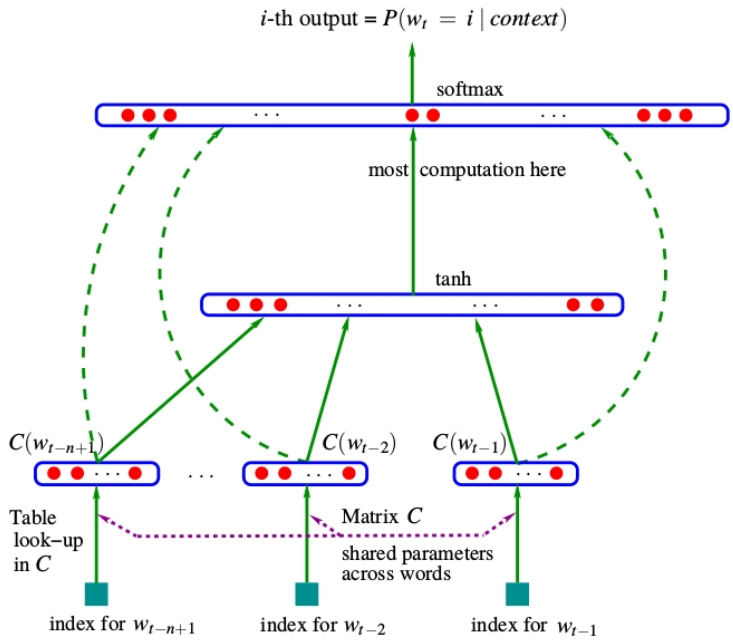


Figure 2.1: The neural network language model as proposed in by Bengio et al. (image taken from [5]).

where b and d are the bias vectors. The output layer uses a *softmax* function [7] to guarantee a positive probability for every word w_i in the vocabulary V, with all probabilities summing to 1:

$$P(w_i|w_{t-n+1}^{t-1}) = y_i = \frac{e^{u_i}}{\sum_{l=1}^{|V|} e^{u_l}} \tag{2.10}$$

The free parameters of the model $\theta = (b, d, W_1, W_2, C)$ are then trained by maximizing the log-likelihood of the complete training set:

$$L(\theta) = \sum_{t=1}^{T} \log y_t \tag{2.11}$$

Here y_t denotes the likelihood of predicting target output word w_t, as calculated with equation 2.10. The training is done using stochastic gradient descent and backpropagation [96], in which the parameters are changed on the basis of the partial derivative of the error, the difference between the target output and the actual output:

$$\theta \leftarrow \theta + \frac{\partial L(\theta)}{\partial \theta} \tag{2.12}$$

These are often recurring steps in neural networks generally and NNLMs more specifically. Mikolov et al.'s [71] Word2Vec makes use of similar steps, which are more thoroughly explained in section 2.3.

NNLMs offer the amazing feat that randomly initialized vectors that are trained to perform a context word prediction task ultimately capture semantics of words themselves. Since the introduction of the NNLM by Bengio et al. improvements have been achieved in several areas. The computational complexity can be decreased by using hierarchical versions of softmax [81, 78, 72] or by not normalizing the model [16, 45]. Recurrent neural networks have been proposed to overcome the need to specify the context length, which is required in feedforward NNLMs, and to efficiently represent more complex patterns [6, 73, 48]. This has been extended in the form of long-short term memory neural networks [42] which have yielded promising results for sentence prediction [108, 25]. While many of these tweaks have shown improvements, the most significant breakthrough shows that decreasing the simplicity of the NNLM can boost the performance while decreasing the computational cost significantly, which will be discussed in the next section.

2.3 Word2Vec

Mikolov et al. introduced Word2Vec in [71] and extended it in [75]. They propose two architectures that together make up Word2Vec (see figure 2.2): continuous bag-of-words (CBOW) and continuous skip-gram (SG). Both of these architectures work on the basis of an NNLM architecture proposed in their earlier work [70, 74], which followed the introduction of neural network language models as explained in section 2.2. Since most of the complexity in those models was caused by the non-linear hidden layer, Mikolov et al. decided to simplify their model by making it linear.

2.3.1 Continuous Bag-of-Words Model

In the CBOW architecture (figure 2.2a) a target word w_t is predicted based on a given context of C words w_I. Given a vocabulary of size V, the input layer consists of C one-hot encoded vectors x_c of size V, containing value 1 at the position of their respective context word and 0 at all other positions. These vectors are each multiplied by the (randomly initialized) $V \times N$ weight matrix W, after which the average of these products is taken as the activation of the N-dimensional hidden layer h:

$$h = \frac{1}{C} \sum_{C} W^T x_c \tag{2.13}$$

where the i-th row of W represents an N-dimensional word embedding v_{w_i} for word w_i in the vocabulary. As such, the hidden layer does not use a non-linear activation function such as a sigmoid or tanh. This decreases the complexity of the model considerably, compared to preceding NNLMs such as the one introduced by Bengio et al.

Just as the input and hidden layer, the hidden and output layer are fully connected. To get an output score u_j for word w_j in the vocabulary, the hidden layer activation values are multiplied by the j-th column of the $N \times W$ weight matrix W':

$$u_j = {w'_j}^T h \tag{2.14}$$

Finally, a softmax function is used in the output layer to arrive at the word estimates (equation 2.10).

As with other NNLMs, the goal is then to find the weights for which the likelihood of observing the target output word w_t given the input context w_I is maximized. To realize this, a loss function is defined using the log probabilities of the softmax function defined in equation 2.10:

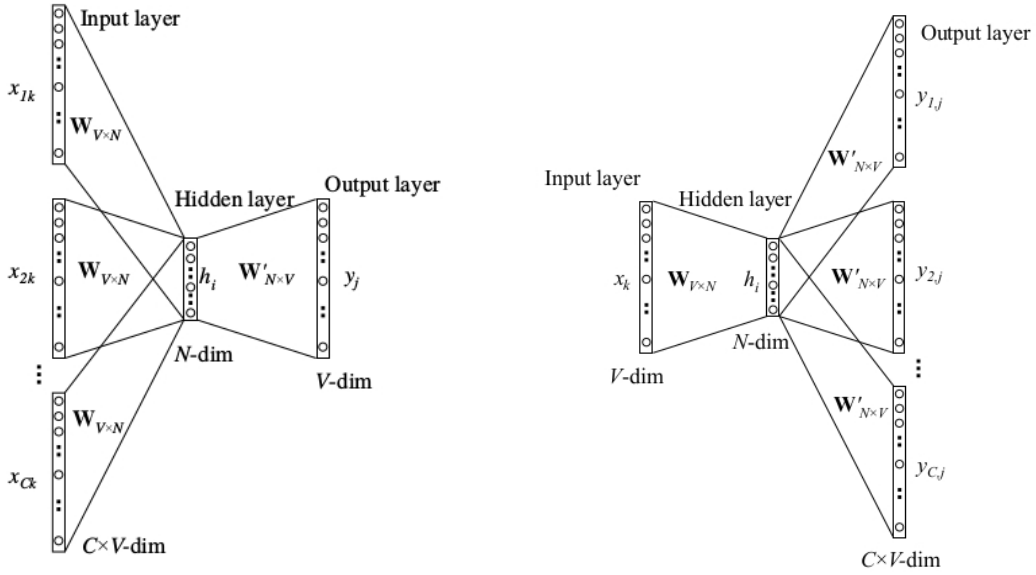
$$\max \log y_{j*} = u_{j*} - \log \sum_{j'=1}^{V} e^{u_{j'}} := -E \tag{2.15}$$

where $j*$ is the index of the target output word in the output layer and we want to minimize the loss function $E = -\log p(w_t|w_I)$. Just as in the NNLM designed by Bengio et all., stochastic gradient descent and backpropagation is used to adjust the weights. This means that the weights are updated according to the partial derivatives of E on the weights. In other words: the weights are changed according to the rate of change that minimizes the error. Using the chain rule, weight matrix W' is updated as:

$$\begin{aligned} w_{ij}^{\prime(new)} &= w_{ij}^{\prime(old)} - \eta \frac{\partial E}{\partial w_{ij}^{\prime(old)}} \\ &= w_{ij}^{\prime(old)} - \eta \frac{\partial E}{\partial u_j} \cdot \frac{\partial u_j}{\partial w_{ij}^{\prime(old)}} \\ &= w_{ij}^{\prime(old)} - \eta \cdot (y_j - t_j) \cdot h_i \end{aligned} \tag{2.16}$$

where η is a learning rate between 0 and 1 and t_j is 1 if $j = j*$ and 0 otherwise.

Next, weight matrix W is updated in a similar fashion, being adjusted to minimize the error E. For this we need to obtain the partial derivative of E to W, which is again accomplished using the chain rule. First, the partial derivative of E to the hidden layer output is given as:



(a) The CBOW architecture.

(b) The SG architecture.

Figure 2.2: The architectures proposed in [71] (images taken from [94]).

$$\frac{\partial E}{\partial h_i} = \sum_{j=1}^{V} \frac{\partial E}{\partial u_j} \cdot \frac{\partial u_j}{\partial h_i} = \sum_{j=1}^{V} (y_j - t_j) \cdot w'_{ij} := EH_i \tag{2.17}$$

where h_i is the i-th output of the hidden layer and EH_i is the sum of the output vectors generated for every word in the vocabulary, weighted by their error $e_j = y_j - t_j$. Following the chain rule, we then take the derivative of h_i to w_{ki} for a specific k:

$$\frac{\partial h_i}{\partial w_{ki}} = \frac{\partial \sum_{k=1}^{V} x_k \cdot w_{ki}}{\partial w_{ki}} = x_k \tag{2.18}$$

Multiplying 2.17 and 2.18 then yields:

$$\frac{\partial E}{\partial w_{ki}} = EH_i \cdot x_k \tag{2.19}$$

Finally, since the input context word vectors x_c are one-hot encoded, the update for every word embedding $v_{w_{c,i}}$ of input context word $w_{c,i}$ is updated in matrix W as:

$$v_{w_{c,i}}^{(new)} = v_{w_{c,i}}^{(old)} - \frac{\eta EH^T}{C} \tag{2.20}$$

where η is the learning rate.

As such, we can observe that the final word vectors are updated based on the combined prediction errors of all the words in the vocabulary. Every output prediction attributes a part of the final update: if the probability of a word w_j in the output layer is overestimated ($y_j >$ target$_j$), the weights between the input word, hidden layer and w_j are decreased, whereas if it is underestimated ($y_j <$ target$_j$) the weights are increased. With an accurate prediction, the weights remain the same. After a number of iterations the relative values of the weights stabilize, resulting in similar weights for words that often occur in similar contexts, which means that some of their semantic properties have been captured by their word embeddings.

2.3.2 Continuous Skip-Gram Model

The skip-gram architecture (SG), shown in figure 2.2b, does the reverse of CBOW: it predicts a number of context words, given an input word. As the name suggests, it makes use of skip-grams. Skip-grams were

introduced to capture dependencies across longer contexts in an efficient manner [46, 85, 95, 30] and have been applied in natural language models successfully in several occasions [101, 90, 88]. A k-skip-n-gram for a sentence $w_1, ..., w_t$ can be defined as the set

$$\{w_{i_1}, w_{i_2}, ..., w_{i_n} | \sum_{j=1}^{n} i_j - i_{j-1} < k\} \tag{2.21}$$

where i is the position in the sentence and the skip distance k indicates the maximum number of words that can be skipped for constructing the n-gram. For example, the sentence "The acting president called for unity." yields the following 2-skip-tri-grams: the acting president, the acting called, the acting for, the president called, the president for, etc.

SG can thus predict skip-grams of a certain k and n. The calculations for the different values in the network do not differ much from the CBOW-architecture. The activation values are calculated in the same manner (equations 2.13 and 2.14). This time however, the softmax function is applied C times, for every position c in context C:

$$y_{c,j} = P(w_{c,j} = w_{O,c} | w_I) = \frac{e^{u_{c,j}}}{\sum_{j'=1}^{V} e^{u_{j'}}} \tag{2.22}$$

The loss function is defined using the product rule for probabilities:

$$\begin{aligned} E &= -\log p(w_{O,1}, w_{O,2}, ..., w_{O,C} | w_I) \\ &= -\log \prod_{c=1}^{C} \frac{e^{u_{c,j_c^*}}}{\sum_{j'=1}^{V} e^{u_{j'}}} \\ &= -\sum_{c=1}^{C} u_{j_c^*} + C \cdot \log \sum_{j'=1}^{V} e^{u_{j'}} \end{aligned} \tag{2.23}$$

Similarly, to update the weights, the prediction error of every word in the vocabulary for every position c is calculated. Since all the weights are shared by the c different output panels, the summed prediction errors of the different panels are substituted in the update function for W' (equation 2.16):

$$w_{ij}'^{(new)} = w_{ij}'^{(old)} - \eta \cdot \sum_{c=1}^{C} (y_{c,j} - t_{c,j}) \cdot h_i \tag{2.24}$$

Finally the update of the weight matrix W remains the same (equation 2.20).

2.3.3 Optimizations

The Word2Vec architectures proved a significant improvement of the NNLMs as proposed by Bengio et al. [5], especially with regard to the computational efficiency. Dropping the computational intensive non-linear hidden layer provided a huge complexity reduction, while performing similarly or even better.

However, still a lot of computations are required in its original form. Especially the output layer is computationally expensive, since every training iteration for every word w_j the net output u_j, probability prediction y_j (y_{c_j} for SG), prediction error $y_j - t_j$ and weight update has to be calculated. Therefore a number of optimizations have been proposed to speed up the process of calculating the output [75].

Hierarchical Softmax

Hierarchical softmax was developed by Morin and Bengio for the sake of speeding up NNLMs [81]. It has been applied successfully since [78, 72], and is therefore an interesting addition to the Word2Vec architecture. Hierarchical softmax as used in Word2Vec uses a binary Huffman tree representation of weights between the hidden and output layer instead of a matrix, as is shown in figure 2.3. This tree has V leaf nodes, containing the relative probability estimations for the words, and $V - 1$ inner nodes, each containing a weight vector $v'_{n(w,j)}$. The probability $p(w = w_O)$ is defined as follows:

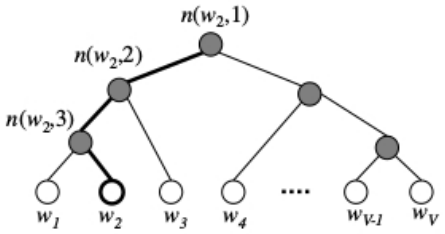


Figure 2.3: A binary tree as used in hierarchical softmax (image taken from[94]).

$$p(w = w_O) = \prod_{j=1}^{L(w)-1} \sigma([\![n(w, j+1) = \text{ch}(n(w, j))]\!] \cdot {v'_{n(w,j)}}^T h) \tag{2.25}$$

Here σ is a standard sigmoid function $1/(1 + e^{-x})$, ch(n) is the left child of unit n and $[\![x]\!]$ is a function that returns 1 if x is true and -1 otherwise. This function is used to assign probabilities to either going left or right, with:

$$p(n, \text{left}) = \sigma(v_n'^T \cdot h) \tag{2.26}$$

and

$$p(n, \text{right}) = 1 - \sigma(v_n'^T \cdot h) = \sigma(-v_n'^T \cdot h) \tag{2.27}$$

Now given a set of training words or contexts the inner node output vectors are trained to predict whether the correct path is down their left or right child. As with the standard versions of CBOW and SG, the vectors are updated based on the difference between the path they choose and the actual path that should be taken. If the prediction of an inner node is close to the actual path, its output vector is not changed much and vice versa. The vectors are updated using stochastic gradient descent and backpropagation as explained in the previous sections

It is clear that this method yields a lower computational complexity than standard softmax, going from $O(V)$ per training iteration per context word to $O(log(V))$. It is not said however, that hierarchical softmax performs equally well as standard softmax. Updating less parameters every cycle might also mean that less information is leveraged.

Negative Sampling

Negative sampling is a simplification of a technique called *noise contrastive estimation* (NCE) [31, 79] and relies on the idea that the loss function E does not have to be based on the prediction of all words in the vocabulary to provide a good measure of the accuracy of the predictions. Instead, only the prediction of the positive sample and a fixed number of negative training samples are used every training instance. This way less output has to be generated every iteration and less weights updated, resulting in a reduction of computations. With negative subsampling, the error then becomes [94]:

$$E = -\log \sigma(\mathbf{v}'^T_{w_O} \mathbf{h}) - \sum_{w_j \in W_{neg}} \log \sigma(-\mathbf{v}'^T_{w_j} \mathbf{h}) \tag{2.28}$$

Here W_{neg} is the set of negative sample words that is taken from a noise distribution. This distribution is empirically determined by Mikolov et al. [75] to be a unigram distribution raised to the power $3/4$.

Subsampling of Frequent Words

Often the most frequent words in a corpus ('the', 'and', 'a' etc.) do not contain much semantic content. These words contribute greatly to the number of computations while not adding to the informativeness or accuracy of the other word vectors. To counter this, subsampling can be applied to decrease the number of frequently occurring words in the dataset. In [75] the following heuristically chosen subsampling function is proposed, where the probability $P(w_i)$ determines whether a word w_i is discarded from the training set:

$$P(w_i) = 1 - \sqrt{\frac{t}{f(w_i)}} \tag{2.29}$$

where t is a threshold (typically around 10^{-5}) and $f(w_i)$ is the relative frequency of word w_i. This function aggressively subsamples words with a frequency greater than t and keeps all the words that occur less, while mostly preserving the ranking of the frequencies.

2.4 GloVe

With *Global Vectors*, Pennington et al. [89] aim to combine the strengths of the two major model families that were mentioned in the introduction to this chapter: *matrix factorization methods* (such as Symmetric Patterns, Singular, PSDVec) and *shallow window methods* (such as Word2Vec). According to Pennington et al. both of these families suffer from drawbacks. The former would not capture the dependency relationships between words optimally, causing a sub-optimal vector space structure with a poor performance on word analogy tasks. The latter perform well on such tasks, but would poorly utilize the statistics of a corpus, ignoring repetition in the data. GloVe on the other hand, would leverage the statistical information of the training corpus in an efficient manner, while producing a vector space with meaningful substructure, shown by its good performance in word analogy tasks.

The core of GloVe rests in the idea that the starting point for word vector learning should be in ratios of co-occurrence probabilities instead of the probabilities themselves. This is expressed in the most general form of their model:

$$F(w_i, w_j, \bar{w}) = \frac{P_{ik}}{P_{jk}} \tag{2.30}$$

Here $w \in \mathbb{R}^d$ are word vectors, $\bar{w} \in \mathbb{R}^d$ are separate context word vectors and P_{ik} is the probability of word k appearing in the context of target word i, which is calculated as X_{ik}/X_i, where X is the co-occurrence count matrix, X_{ik} the count of k appearing in the context of i and X_i is the count of i.

Let us look into an example of why this ratio would be more informative than the probabilities themselves. Suppose we want to learn about the relationship between the words *i = capitalism* and *j = communism* in the Dutch Wikipedia dump from March 2017.[1] Rather than just looking at their raw co-occurrence probability $P(j|i) \approx 0.06$ (given a weightless context of 10 preceding and 10 succeeding words), which is rather limited informatively, we look at ratios of co-occurrence given various probe words $k \in V$. For example, let *k = market*. Since k is often mentioned in the phrase 'free market' in the context of capitalism and less so in that of communism, we expect the ratio to yield a value significantly larger than 1, which is indeed the case: $\frac{P_{ik}}{P_{jk}} \approx 6.17$. Similarly, when *k = Stalin*, the ratio is close to 0: $\frac{P_{ik}}{P_{jk}} \approx 0.11$. Finally, if we let k be a more neutral word that we would expect to occur in both contexts equally, we get a value close to 1. For example, for *k = history*: $\frac{P_{ik}}{P_{jk}} \approx 1.04$, or *k = country*: $\frac{P_{ik}}{P_{jk}} \approx 1.41$. The results are shown in table 2.1.

Table 2.1: Co-occurrence ratios for words *capitalism* and *communism* and context words k.

Probability and Ratio	*k = market*	*k = Stalin*	*k = history*	*k = country*
$P(k\|\textit{capitalism})$	$1.09 * 10^{-2}$	$1.37 * 10^{-3}$	$6.83 * 10^{-3}$	$2.60 * 10^{-2}$
$P(k\|\textit{communism})$	$1.79 * 10^{-3}$	$1.25 * 10^{-2}$	$6.55 * 10^{-3}$	$1.84 * 10^{-2}$
$P(k\|\textit{capitalism})/P(k\|\textit{communism})$	**6.17**	**0.11**	**1.04**	**1.41**

Using the ratios, we can observe the difference in contexts that these words appear in and conclude more about their relative semantic properties than when we would just use their co-occurrence. Therefore Pennington et al. use them as starting point and aim to find the function F that optimally approximates these ratios (equation 2.30).

The first restriction they put on F is that it only deals with the difference between the two target words, since vector spaces are inherently linear structures:

[1] This contains ca. 219 million words, https://dumps.wikimedia.org/nlwiki/. The Dutch translations were used here.

$$F(w_i - w_j, \bar{w}) = \frac{P_{ik}}{P_{jk}} \tag{2.31}$$

The right-hand side of equation 2.31 is a scalar, while the arguments of F are vectors. This could be resolved if F were chosen to be a complicated function parametrized by, for example, neural networks. However, this might mix the vector dimensions in undesirable ways and with GloVe, Pennington et al. aim to preserve the simplicity of a linear structure. Therefore they take the dot product of the arguments to solve this:

$$F((w_i - w_j)^T \bar{w}) = \frac{P_{ik}}{P_{jk}} \tag{2.32}$$

The next thing to resolve is the distinction between w and $\bar{w}$, as it is arbitrary which word is the context and which is the target. Therefore $w \leftrightarrow \bar{w}$ and $X \leftrightarrow X^T$ have to be exchangeable. To fit this into the model, F is required to be a homomorphism, i.e. a map between two algebraic structures of the same type that preserves their operations, between the groups $(\mathbb{R}, +)$ and $(\mathbb{R}_{>0}, \times)$:

$$F((w_i - w_j)^T \bar{w}) = \frac{F(w_i^T \bar{w})}{F(w_j^T \bar{w})} \tag{2.33}$$

Solving this by $F(w_i^T \bar{w}) = P_{ik} = \frac{X_{ik}}{X_i}$ yields solution $F = \exp$, giving:

$$w_i^T \bar{w} = \log(P_{ik}) = \log(X_{ik}) - \log(X_i) \tag{2.34}$$

Since the term $\log(X_i)$ is independent of k it is absorbed into a bias b_i for w_i and to restore the symmetry an additional bias $\bar{b}_k$ for w_k is added:

$$w_i^T \bar{w} + b_i + \bar{b}_k = \log(X_{ik}) \tag{2.35}$$

To account for the log argument being 0 a shift can be added, such as $\log(1 + X_{ik})$.

The idea of factorizing the log of the co-occurring matrix, proposed in equation 2.35 is closely related to the more traditional LSA technique [23]. Pennington et al. state that a downside of this method is the fact that all co-occurrences in X contribute equally, while they assume that rare occurrences are noisy and carry less information than more frequent ones. To counter this, they cast equation 2.35 as a least squares regression problem and introduce a weighting function $f(X_{ij})$:

$$J = \sum_{i,j=1}^{V} f(X_{ij})(w_i^T \bar{w} + b_i + \bar{b}_k - \log(X_{ij}))^2 \tag{2.36}$$

Here f can be defined in many different ways, as long as $f(0) = 0$ and it does not overweight the rare or the frequent co-occurrences. They found one class of functions to work well:

$$f(x) = \begin{cases} (x/x_{\max})^a, & \text{if } x < x_{max}. \\ 1 & \text{otherwise.} \end{cases} \tag{2.37}$$

They empirically found a to be optimal at $3/4$ (interestingly similar to the unigram distribution power used for negative sampling by Mikolov et al., see section 2.3.3) and the cutoff $x_{\max}$ at 100.

As can be observed from equation 2.36, the complexity of the model depends on the number of non-zero entries in X, which is maximally $O(V^2)$. Instead, window based approaches such as Word2Vec scale linearly with the corpus size. Therefore, if the vocabulary is small compared to the corpus size, window based approaches are relatively expensive. On the other hand, vocabularies of large corpora can contain hundreds of thousands of words, causing V^2 to be in the tens of billions. If these vocabularies are relatively big, matrix factorization methods such as GloVe scale worse.

2.5 Symmetric Patterns

Symmetric pattern-based word embeddings were proposed by Schwartz et al. [98, 99] as alternative for purely context-based solutions. While the latter solutions perform impressively on associative tasks, such as analogy tests, they tend to be much weaker when it comes to word similarity. This is the case because they purely use context related information, making them adept at associating words that occur in the same context, but less so in determining the similarity between words occurring in the same context.

Schwartz et al. make use of *symmetric patterns*, which are specific sequences of words containing two words that, indicated by the specific pattern, hold a symmetry in semantical value. For example, a symmetrical pattern can be "*X and Y*", where *X* and *Y* are likely to be semantically similar and can therefore take on either position; *"chair and seat"* and *"seat and chair"* are semantically plausible alternatives.

Davidov et al. [22] introduced an algorithm to extract symmetric patterns. This algorithm first scans through a corpus looking for all frequent patterns within a specified window. It then traverses through all candidate patterns and selects those patterns for which the pair words w_i, w_j co-occur approximately equally in both forms $X = w_i, Y = w_j$ and $X = w_j, Y = w_i$.

To generate word embeddings using these patterns, Schwartz et al. create a $V \times V$ symmetric matrix M, V being the size of the vocabulary, where they store all the occurrences of patterns instantiated by w_i, w_j in both forms. They then use *Positive Pointwise Mutual Information* [15, 20] to weight the entries in M:

$$PPMI(w_i, w_j) = \max(\log_2 \frac{P(w_i, w_j)}{P(w_i)P(w_j)}, 0) \tag{2.38}$$

The intuition here is that if two words co-occur more often than we would expect them to given that their co-occurrences would be independent, the resulting value becomes higher than 0, indicating that their co-occurrence is informative. Effectively this designates co-occurrence combinations with highly frequent words such as *and* as less informative, while deeming co-occurrences with low occurring words more relevant. Here Schwartz et al. apply this principle to symmetric pattern co-occurrences, yielding PPMI-matrix $M*$, with every i-th row being the vector representation v_i of the word w_i.

Finally they smooth $M*$ to decrease its sparsity, by adding to each word embedding v_i a portion of the cosine distance of the top n word vectors closest to it:

$$v_i' = v_i + \alpha \cdot \sum_{v \in W_i^n} v \tag{2.39}$$

where α is a smoothing factor and W_i^n denotes the top n vectors with the smallest cosine-distance from v_i.

Following this algorithm we can immediately see one of the weak aspects of the technique: the smoothing step has a high computational cost, needing the calculation of the cosine distance between every combination of embeddings, which are V-dimensional. Not only does this cause the calculation of the embeddings to be memory and time consuming, the embeddings are less suitable for applicative use as well. For example, the search software created for the UMCG (chapter 6) cannot make use of these embeddings, because the memory used for loading them is too high and the time to calculate distances between them is too long for a fluent user experience. It would therefore require some form of dimensionality reduction which would cost extra computation as well as resulting in the loss of information.

2.6 Singular

The Singular architecture designed by Stratos et al. [107, 106] is based on *canonical correlation analysis* (CCA) [43]. CCA is a technique that seeks to maximize the Pearson correlation coefficient (PCC) ρ between two variables $X, Y \in \mathbb{R}^m$:

$$\rho(X, Y) = \frac{cov(X, Y)}{\sigma_X \sigma_Y} \tag{2.40}$$

where $cov(X, Y)$ is the covariance between X and Y and σ is the standard deviation. The PCC is a value between -1 and $+1$, where -1 describes a negative linear correlation, 0 no linear correlation, and $+1$ a positive one.

Let X and Y be two random real-valued vectors with a mean of zero, which can be achieved through *centering*, a preprocessing technique. The task in CCA is then to find a set of projection vectors a, b that maximizes ρ:

$$a_i, b_i = \arg\max_{a_i, b_i} \rho(a^T X, b^T Y) \tag{2.41}$$

where i is an iteration in which the vectors a, b are simultaneously optimized to get the maximum PCC. The dimensionality of a and b is equal to or smaller than the smallest vector of X, Y. To find a solution for equation 2.41, Stratos et al. apply a method based on singular value decomposition (SVD), proposed by Hardoon et al. [33]. SVD is a technique that aims to approximate an n-dimensional matrix using a linear combination of k unique vectors. So effectively, CCA finds a low-dimensional representation that provides the maximum correlation between two variables.

To apply CCA for the goal of obtaining word embeddings, X and Y are set to be one-hot encodings representing a target word and its context word respectively. After a number of simplifications (see [106]), each entry in matrix Ω, describing the linear relations between word w and context c, can then be calculated as:

$$\Omega_{w,c} = \frac{P(X_w = 1, Y_c = 1)}{\sqrt{P(X_w = 1)P(Y_c = 1)}} \tag{2.42}$$

the idea of which is similar to PPMI (equation 2.38). The sample estimate of $\Omega^{(a)}$ given a corpus is then given by:

$$\hat{\Omega}^{(a)}_{w,c} = \frac{\#(w, c)^a}{\sqrt{\#(w)^a \#(c)^a}} \tag{2.43}$$

where $\#(w, c)$ denotes the co-occurrence count of word w and context c in the corpus, $\#(w)$ the count of w over all possible co-occurrences, and a is an arbitrary power (for an extensive proof that this power can be added while the representational meaning of the word embeddings is preserved, see [106]). To stabilize the variance of the terms, Stratos et al. use $a = 1/2$. Finally, CCA is used to project X and Y to a low-dimensional space in which they are maximally correlated. The resulting projection of X serves as word embedding.

2.7 PSDVec

The final architecture that will be compared is the PSDVec architecture, developed by Li et al. [65, 64, 66]. They proposed their method to answer to two distinct weaknesses they recognize in techniques such as Symmetric Patterns and Singular. First, these techniques reduce the dimensionality of their obtained vectors by methods such as SVD. This means that information is lost and embeddings might lose some of the semantic properties of the words they represent. Second, none of the discussed techniques offer generative properties. Therefore Li et al. propose to unify word embeddings and context words using three components:

- Capturing linear correlations of two words through the interaction of two embeddings
- A residual that captures nonlinear or noisy correlations
- The unigram priors

The first component is realized by assuming that different types of semantic and syntactic regularities between two words w_i, w_j are encoded in different dimensions of their embeddings. By applying the concept of the *product of experts* [40] - capturing the output of different 'experts' by multiplying their probability distributions offers a way of leveraging all their information while maintaining a workable dimensionality - Li et al. arrive at the following function to obtain embeddings:

$$\begin{aligned} P(w_i, w_j) &= \exp\{v_{w_j}^T v_{w_i} + a_{w_i w_j}\} P(w_i) P(w_j) \\ \log \frac{P(w_j|w_i)}{P(w_j)} &= v_{w_j}^T v_{w_i} + a_{w_i w_j} \\ \text{PMI}(w_i, w_j) &= v_{w_j}^T v_{w_i} + a_{w_i w_j} \end{aligned} \tag{2.44}$$

where $a_{w_i w_j}$ is a bigram residual and PMI is *pointwise mutual information* as in equation 2.38, but without the $\max$ term. Here the factor $v_{w_j}^T v_{w_i}$ captures linear interactions between w_i and w_j for every dimension, and $a_{w_i w_j}$ captures nonlinear or noisy interactions. To agree with the asymmetry of bigrams in natural languages, they do not assume $a_{w_i w_j} = a_{w_j w_i}$ as opposed to other matrix factorization methods.

For larger context windows Li et al. make the assumption that *pointwise redundant information* - where knowing the value of only one of the conditioning variables is necessary to reduce uncertainty - and *pointwise synergistic information* - where knowing the values of all conditioning variables is necessary - cancel each other out [117]. Given this assumption we get $\text{PMI}(w_2|w_0, w_1) \approx \text{PMI}(w_2|w_0) + \text{PMI}(w_2|w_1)$. Applying this assumption to word sequences of size t and substituting equation 2.44, we obtain:

$$P(w_0^t) \approx \exp\Big\{ \sum_{\substack{i,j=0 \\ i \neq j}}^{t} (v_{w_i}^T v_{w_j} + a_{w_i w_j}) + \sum_{i=0}^{t} \log P(w_i) \Big\} \tag{2.45}$$

yielding the following conditional distribution of w_c given its context $w_0, ..., w_{c-1}$:

$$\begin{aligned} P(w_c|w_0^{t-1}) &= \frac{P(w_0^t)}{P(w_0^{t-1})} \\ &\approx P(w_t) \exp\Big\{ v_{w_t}^T \sum_{i=0}^{t-1} v_{w_i} + \sum_{i=0}^{t-1} a_{w_i w_t} \Big\} \end{aligned} \tag{2.46}$$

To compensate for missing data and reduce overfitting of infrequent words, a spherical Gaussian prior is assigned to the embeddings. Furthermore, the residuals a are penalized, since $v_{w_j}^T v_{w_i}$ should capture as much correlations between w_i and w_j as possible. Moreover, since more frequent bigrams in the corpus mean that there is less noise in their empirical distribution, the a should be more heavily penalized. Therefore a weighting function similar to that used in GloVe (equation 2.37) is applied to the residuals based on their frequency.

As a final measure of compensating for missing data, Li et al. apply Jelinek-Mercer smoothing [122], where the interpolation between unigram and bigram distribution is used. This accounts for the situation in which two bigrams do not occur in the corpus, but one bigram contains often occurring unigrams and the other does not. It would be unlikely that both bigrams would occur equally often. Interpolation with the unigram distribution therefore makes sure that the bigram containing the often occurring unigrams gets a higher estimated probability. For the exact smoothing functions and parameter settings, see [65].

The generative aspect of PSDVec is realized by assuming the text is generated from a *Markov chain*, where a word only depends on words within its context. In the generative process (see figure 2.4), for each word s_i an embedding v_{s_i} is drawn from its Gaussian prior distribution, for each bigram s_i, s_j a residual $a_{s_i s_j}$ is drawn from its distribution, and finally, for each document d_i a word w_{ij} is drawn from the vocabulary with probability $P(w_{ij}|w_{i,j-t}^{i,j-1})$. Given this process and the embeddings V and residuals A, the probability of a document d_i can be derived as:

$$P(d_i|V, A) = \prod_{j=1}^{L_i} P(w_{ij}) \exp\Big\{ v_{w_{ij}}^T \sum_{k=j-c}^{j-1} v_{w_{ik}} + \sum_{k=j-c}^{j-1} a_{w_{ik} w_{ij}} \Big\} \tag{2.47}$$

The log-likelihood of the complete corpus D including the smoothing terms is then given as:

$$\log p(D, V, A) = C_0 - \log Z(H, \mu) - ||A||_{f(H)}^2 - \sum_{i=1}^{W} \mu_i ||v||^2 + \sum_{i,j=1}^{M, L_i} \Big\{ v_{w_{ij}}^T \sum_{k=j-c}^{j-1} v_{w_{ik}} + \sum_{k=j-c}^{j-1} a_{w_{ik} w_{ij}} \Big\} \tag{2.48}$$

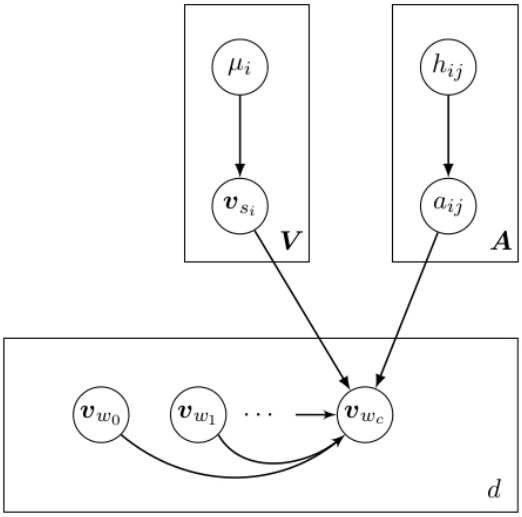


Figure 2.4: The generative process of PSDVec (image taken from [65]). Here μ_i describes a hyperparameter that increases with the frequency of w_i, h_{ij} represents the bigram empirical probability, $\mathbf{V}$ is the embedding matrix, $\mathbf{A}$ the residual matrix and d is a document.

where $Z(H, \mu)$ is a normalizing constant and $C_0 = \sum_{i,j=1}^{M,L_i} \log P(w_{ij})$ is constant.

As with all language models, the learning objective is to maximize the corpus log-likelihood. To achieve this, Li et al. make a number of simplifications and apply a weighted low-rank approximation technique proposed in [105]. To scale down the complexity of their training algorithm, they use online blockwise regression, where they divide the vocabulary into a number of blocks based on their frequency. The embeddings of the most frequent words, the *core words*, are learned first, disregarding the other words. Then the embeddings of these words are fixed, and other blocks are learned in order of frequency. Finally all the embeddings are combined to get the embeddings of the whole vocabulary. Training the embeddings in this way ultimately results in loss of information (word embeddings of core words are learned while disregarding non-core words), which is exactly what Li et al. aimed to prevent as opposed to the methods using dimensionality reduction techniques. Thus, the results will have to show whether the information loss in this technique is less severe.

Chapter 3

Paragraph Embedding

After finding adequate word representations, another challenge arises when entire sentences, paragraphs, or in our case reports, are to be classified according to the categories specified in the database. Bag-of-words methods have traditionally been used for such tasks, but their drawbacks are that they ignore both the semantics and the ordering of words. As an alternative, there exist several techniques in which single word embeddings are used. One group of such techniques focuses on first creating single distributed representations for paragraphs to perform classification on [8, 29, 120, 57, 21, 62, 53, 38, 64]. Other techniques use word embeddings directly to classify a document [49, 52, 35, 127, 126, 121]. This chapter discusses the first group. The second group will be discussed in the next chapter, where other classification techniques will be treated as well.

Several comparative studies have been performed toward distributed paragraph embeddings [38, 64, 109, 87]. These have reported impressive performances, but the results diverge on several occasions. Furthermore, similarly to the word embedding techniques, it is not yet clear which technique yields the most adequate report representations for applicational purposes given our highly specialized database, if they are applicable in the first place. A number of these methods is inapplicable because they require training data in specific formats or have other constraints which we cannot meet.

One method uses parse trees of sentences to combine the word embeddings in the order given by the parsing process [102, 103]. This is not applicable here, since grammar parsing can only be performed on single complete sentences. We want to be able to represent smaller phrases or larger paragraphs. Moreover, many incident reports are not always formulated according to proper grammar, but more in telegram style. In [53, 38] techniques are discussed that are similar to the skip-gram architecture of Mikolov et al., but applied to complete sentences. We cannot make use of this technique, because training it requires one running text where all sentences are sequentially built up. Our database consists of small reports that are usually not related to each other, nor are there large single texts (that we know of) that use a considerate amount of the Dutch jargon used in the incident reports. Recurrent types of neural networks, such as LSTMs, have been successfully applied to get paragraph representations [109, 25], but these mostly focus on the prediction of subsequent words or sentences given a sequence, which is less interesting for our purposes. Finally, other methods require more structured datasets, such as dictionaries [39], sentence-aligned translated texts [14] or 'click-through' data provided by a commercial web search engine [87].

Therefore we will focus on three techniques that are applicable in our case and have been shown to be either simple, effective or both: averaging, *distributed memory model of paragraph vectors* (Par2Vec) [57] and *TopicVec* [64]. The following sections explain these techniques in detail.

3.1 Averaging

Probably the most trivial way of representing a paragraph in vector space, given its separate word embeddings, is to average all of these word embeddings. This has been done as baseline to compare other methods with [103, 21, 64], often performing on a competitive level, but also as global context to support a word embeddings generation process [45] or in combination with otherwise acquired paragraph vectors [64].

The simplest form of averaging the word embeddings is given as $\frac{1}{N}\sum_{i=1}^{N} w_i$ where N is the number of words in the sequence and w_i is a word embedding. Not all words are deemed as influential for the meaning

of a paragraph though. For example, the word *the* occurs often while not representing a distinct semantic value. Therefore we might not want to value it as much as the less frequent words in the paragraph. To realize this, a weighted average can be used:

$$c = \frac{\sum_{i=1}^{N} f_w(t_i) w_i}{\sum_{i=1}^{N} f_w(t_i)} \tag{3.1}$$

Here c is the paragraph vector and f_w is a weighting function. An often used weighting function is *inverse document frequency* (IDF) [93], where the relevance of a term in the word sequence is inversely proportional to the number of documents in the corpus it appears in. In other words: if many different documents (paragraphs, reports) contain the word, it is deemed less relevant for distinguishing the content of the current document from that of the others. Often a logarithmic IDF function is used, which is what we will do as well:

$$f(t) = \log \frac{n_{total}}{1 + n_t} \tag{3.2}$$

Here n_{total} is the total number of documents in the corpus and n_t the number of documents containing word t. The 1 in the denominator is added to prevent zero division. This function yields 0 if the count is the same as the number of documents, i.e. if the word occurs in all documents and is therefore not relevant for distinguishing them from each other. The lower the count, the higher value the function returns.

Averaging is easily implemented and computationally cheap, but it suffers from the same drawback as traditional bag-of-words methods: it ignores word order. Therefore averaging is usually outperformed by more complex methods. Still, it offers a computationally cheap and intuitive method that provides meaningful paragraph representations nonetheless.

3.2 Distributed Memory Model of Paragraph Vectors

Not long after the introduction of Word2Vec the architecture was extended by Le and Mikolov [57] to generate paragraph embeddings. Just as the word vectors in Word2Vec, in the Par2Vec method the paragraph vectors are also trained to contribute to a word prediction task given a context sampled from the paragraph.

In the new model, depicted in figure 3.1, the input consists of the input vector **x** indicating the context words, as well as a paragraph id, indicating the row in matrix D containing all paragraph vectors. As a result, not just the word embeddings in matrix W are updated during training, but also the paragraph embedding in D. In this sense the paragraph vector can be thought of as another word vector, but since the paragraph vector is shared across all contexts generated from the same paragraph, it acts as a memory of what is missing from a given context. As such, this model can be called a *Distributed Memory Model of Paragraph Vectors* (PV-DM), although we will refer to it as Par2Vec.

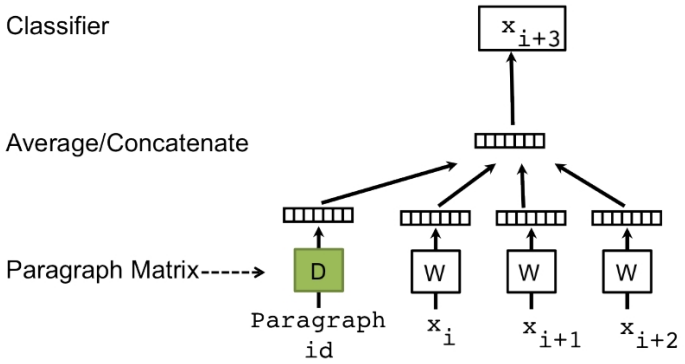


Figure 3.1: The Paragraph Vector architecture proposed in [57] (image taken from [21]).

Training Par2Vec is similar to the original Word2Vec model: the goal is to minimize a loss function that is determined by the error between the prediction yielded by a (hierarchical) softmax function and the actual

target word. This is done by updating the word and paragraph vectors via stochastic gradient descent and backpropagation. To acquire an embedding of a paragraph that was not present in the training set, all of the other parameters of the network are held fixed, and gradient descent is performed on the new paragraph vector. This way new paragraph embeddings can be generated without having to retrain all the weights in the network.

An advantage of the model compared to traditional bag-of-words models is that semantic properties of the words are inherited from the word vectors, while bag-of-words models only use word counts as features. Moreover, the paragraph vector takes word order into account, while a standard bag-of-words model would need high-dimensional n-grams to model this, which usually generalizes poorly.

Dai et al. [21] propose a variation that is very similar to the skip-gram model as discussed in section 2.3.2: instead of predicting a target word using context words, the context is predicted. This time however, the input is not a target word but the paragraph vector, as can be observed in figure 3.2. In this model backpropagation is only used to tune the paragraph vectors, making it more efficient.

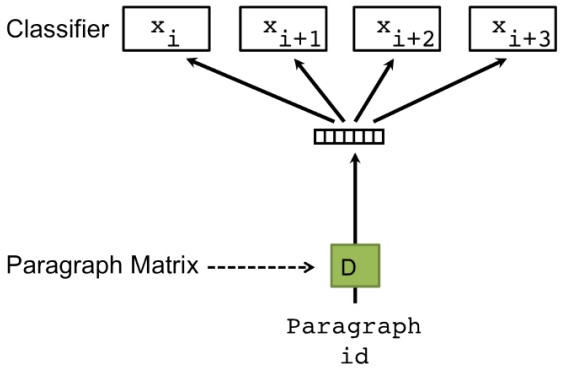


Figure 3.2: The Paragraph Vector adaption proposed by Dai et al. (image taken from [21]).

The DM-PV model has shown state-of-the-art results for sentiment classification [57] and topic classification of Wikipedia articles and arXiv arcticles [21], outperforming LDA and bag of words models.

3.3 TopicVec

TopicVec was introduced by Li et al. [64] as an extension to PSDVec (section 2.7). The latter uses the context words to determine the conditional distribution of a word. TopicVec builds on this and determines the distribution of both the context words and a topic in the form of an embedding. These topic embeddings are of the same dimensionality as the acquired word embeddings and every word in a document is assumed to be semantically similar to a topic. The topics are arranged in a topic matrix T and each document has K candidate topics. Each word in a document is assigned a topic and each topic has a document-specific prior probability ϕ that is assigned to a word. The vector $\phi_i = (\phi_{i1}, ..., \phi_{iK})$ is referred to by Li et al. as the *mixing proportions* of these topics in document d.

The conditional probability function for target word w_c that was introduced in [65] (equation 2.46) is extended here to incorporate the topic vector t and residual r:

$$P(w_c|w_0 : w_{c-1}, z_c, d_i) \approx P(w_c) \exp\left\{v_{w_c}^T \Big(\sum_{l=0}^{c-1} v_{w_l} + t_{z_c} \Big) + \sum_{l=0}^{c-1} a_{w_l w_c} + r_{z_c} \right\} \tag{3.3}$$

where z_c represents the topic assignment of the c-th word in document d_i, and the topic vector t_z may be specific to d_i. The impact of topics is restricted by constraining the magnitudes of all topic embeddings within a hyperball of radius γ. The value of the topic residual is approximated by setting the context size to 0.

The new generative process, depicted in figure 3.3, performs the following additional steps, aside from the steps already described under PSDVec:

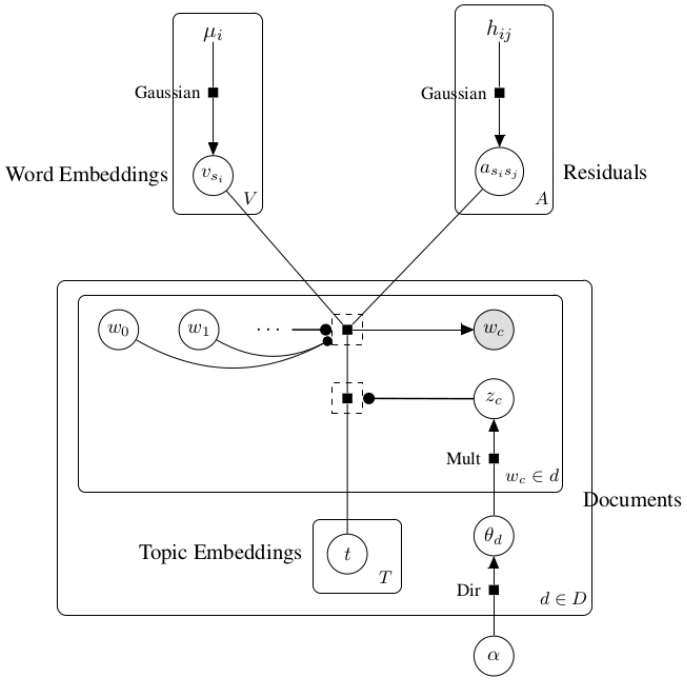


Figure 3.3: The generative process of TopicVec. Similar to the PSDVec architecture with the addition of topic assignment (image taken from [64]).

- For the k-th topic, uniformly draw a topic embedding from a hyperball of radius γ, which constrains the magnitudes of the topic embeddings to restrict the impact of topics and avoid overfitting.
- For each document d_i:
  1. Draw the mixing proportions ϕ_i from the Dirichlet prior $\text{DIR}(\alpha)$
  2. For every j-th word draw a topic assignment z_{ij} from the categorical topic distribution $\text{CAT}(\phi_i)$ and draw word w_{ij} from the vocabulary $\mathbf{S}$ according to equation 3.3.

The likelihood function is changed in a similar way, incorporating topic assignment and embeddings in function 2.48 defined for PSDVec (for details see [64]). The learning objective is then to find the word embeddings $\mathbf{V}$ and topic embeddings $\mathbf{T}$ that maximize the log-likelihood.

Figure 3.4: A topic cloud produced by TopicVec learned from one New York Times article. Topic slices are proportional to the topic proportions, and the font size indicates the relevance to the topics (image taken from [64]).

The process of learning the embeddings is divided into two stages. The first stage consists of the original PSDVec under the assumption that the topics have a relatively small impact to word distributions and that this impact is evened out over all documents. In the second stage, the word vectors V and bigram residuals A are treated as constants and plugged into the likelihood function to find the corresponding optimal topic embeddings T and the conditional probability $p(Z, \phi|D, A, V, T)$ of the full model using a *generalized expectation-maximization* algorithm with gradient descent.

After learning, documents can be represented by their topic proportion vectors. One way of grouping them could be to use one set of topics across the whole corpus. Another is to use different topics for different subsets of documents. For example, if document categories are available, documents in each category could share their respective sets of topics. However, similar redundant topics could be learned in different documents simultaneously but projected in different dimensions, resulting in the possibility of similar documents being projected into very different topic proportion vectors. Therefore pairwise similarities between topic embeddings could be used as similarity measure between two projected topic proportion vectors, where two similar documents will still receive a high similarity score.

Li et al. found their method to perform better than other state-of-the-art methods. They also test the topic proportion vectors concatenated with the mean word vectors of the documents, which in some test cases yield improved results compared to the single topic proportion vectors. Figure 3.4 shows an example of topics learned by TopicVec from a single document.

Chapter 4

Classification

A classifier is a function that assigns class labels to a given observation represented by a number of features in feature space. Usually classifiers are trained on training data which are labeled according to the given classes. From these data the classifier learns to associate new samples to one of the possible classes. The UMCG database contains many categorical variables that can provide such labels, such as department of occurrence, type of incident, degree of risk etc.

The first four sections of this chapter discuss traditional classification methods that have proven their worth over time: a naive Bayes classifier, k-nearest neighbors [19], support vector machines [113, 10] and multilayer perceptrons. The final sections expand on a number of recently proposed architectures that make direct use of the single word embeddings to classify a paragraph [49, 52, 121, 126]. In these methods word embeddings are concatenated, creating a sentence or paragraph matrix, which is subsequently used as input to a convolutional neural network (CNN) [58]. While their architectures are similar, they each offer their own variant claiming improvements over the other.

4.1 Naive Bayes Classifier

In a naive Bayes classifier the abstract goal is to find the conditional probabilities of the possible classes C_k for $k \in K$ given an input feature vector, which in our case is a sequence of words W. This conditional probability is then given as:

$$P(C_k|W) = \frac{P(C_k)P(W|C_k)}{P(W)} \tag{4.1}$$

where in some contexts $P(C_k|W)$ is called the posterior probability, $P(C_K)$ the prior probability, $P(W|C_k)$ the likelihood and $P(W)$ the evidence. As the evidence is independent of the class it is irrelevant for classification purposes and therefore disregarded in the classifier.

Applying the chain rule for probabilities in combination with the naive Bayes assumption that features (words or n-grams) are conditionally independent of each other given the class C_k, we obtain the following function for the posterior class probability given a sequence W of N words or n-grams:

$$P(C_k|W) = \frac{1}{\alpha} P(C_k) \prod_{i=1}^{N} P(w_i|C_k) \tag{4.2}$$

Here the evidence is replaced by a constant scaling factor α. Finally, to arrive at the final classification, a class label $\hat{y}$ is assigned to the input sample based on an $\arg\max$ decision rule over all possible classes:

$$\hat{y} = \underset{k \in \{1,\ldots,K\}}{\arg\max}\ P(C_k) \prod_{i=1}^{N} P(w_i|C_k) \tag{4.3}$$

Interestingly, even though the naive Bayes assumption is unrealistic, the classifier has achieved remarkable successes and can still be competitive with newer techniques [115, 116, 52, 121].

4.2 k-Nearest Neighbors

The k-nearest neighbors classifier [19] is an intuitive technique that has been widely used in many different areas of research and applications. It is based on the idea that feature vectors that are located close to each other in feature space are more likely to share the same class than vectors that are further apart.

The algorithm trains by simply storing all the training feature vectors (report embeddings) with their respective class labels. To classify a new sample a distance metric, such as Manhattan or Euclidean distance, is used to determine the nearest k training vectors in feature space. The most frequent occurring class label among these neighbors is then usually chosen as the designated class for the new sample.

The simplicity and intuitiveness of the technique together with instance based training and often adequate performances make it a popular classifier, even today. It has some drawbacks however. First, there is the choice of k, which has to be predefined. Using a small k makes it susceptible to noise and outliers, while using a large k can result in less distinct classes. Another drawback can be the discreteness of majority voting. This completely disregards the actual distance between the vectors, which can be severely different for the first and second neighbor. Several techniques have been developed to counter this, a popular one being weighing the votes of neighbors based on their distances to the input vector [51].

4.3 Support Vector Machines

A support vector machine (SVM) [113, 10] tries to find an n-dimensional hyperplane such that two classes containing training examples are separated and the distance from that plane to the nearest example on each side is maximized, indicating an optimal margin (see figure 4.1). The training samples that are closest to the hyperplane are called *support vectors*. In a linear SVM, if the data are not linearly separable, no *hard-margin* exists and a loss function is minimized to find a *soft-margin*, i.e. the best separation 'compromise'.

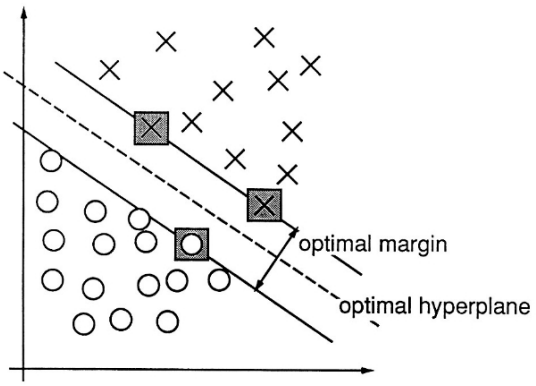


Figure 4.1: A two dimensional separable problem, where the support vectors define the margin of largest separation between the two classes (image taken from [18]).

Nonlinear classification is also possible with SVMs by using *kernel* functions. Kernel functions map pairs of input data onto their inner products yielding one dimension, which can be interpreted as their similarity. As such, by using a kernel function the dimensionality of the feature space is moved from the input data dimensions to a space with one dimension for every training example. This makes it possible to calculate the optimal nonlinear margin in input feature space by performing calculations in the reduced *implicit* feature space.

Finally, to create a multiclass SVM classifier, the problem can be separated into multiple binary classification problems. Multiple binary SVMs are then combined, where every SVM separates either one class from all other classes (*one-versus-all*) or one class from one other class (*one-versus-one*).

4.4 Multilayer Perceptrons

A multilayer perceptron is a type of feedforward artificial neural network of which the neural network language model proposed by Bengio et al. [5] (section 2.2) is an instance. It consists of at least three fully

connected layers: an input layer, a hidden layer and an output layer. The hidden and output layer apply activation functions, which traditionally were typically sigmoid functions, whereas lately the rectified linear unit [84] has gained a lot of traction. The input layer consists of a feature vector that represents a training sample, such as a paragraph embedding or bag of words. In multi-class classification, the output layer represents the output for every one of the class labels, applying e.g. a softmax function (equation 2.10). The weights between the layers are updated using backpropagation and a solver method, such as stochastic gradient descent, as explained in sections 2.2 and 2.3.

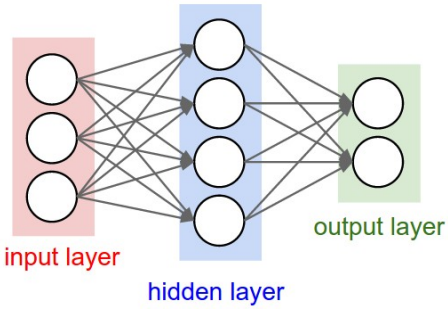


Figure 4.2: A simple MLP (image taken from http://cs231n.github.io/neural-networks-1/).

4.5 Convolutional Neural Networks

As stated in chapter 3, several new techniques have been developed where convolutional neural networks are used on word embeddings [17, 44, 49, 52, 121, 127, 126] or on character level [124, 125] to classify documents. This section briefly explains convolutional neural networks in general and the most promising of these techniques that will be competing with the report embedding/classifier combinations.

A *convolutional neural network* (CNN) [58] is a variant of a feed-forward multilayer perceptron (MLP) which is used in Word2Vec (section 2.3). The CNN is inspired by the visual perception of animals and is therefore often applied for image recognition purposes.

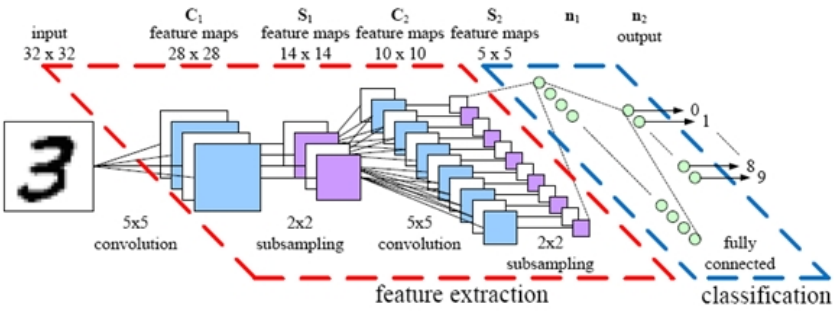


Figure 4.3: A typical CNN architecture (image taken from http://parse.ele.tue.nl/cluster/2).

A CNN revolves around the process of extracting features from an input matrix using convolution. In this process a *filter* or *kernel* is convolved across the input volume, where for every location in the input volume, called a receptive field, the dot product between the entries of the filter and the receptive field is calculated. This results in an activation or *feature map* of the input volume for that specific filter, indicating the extent to which the filter matches its receptive fields. The goal of the CNN is then to learn the filters that extract the most relevant features for the final classification purposes.

Aside from these convolutional layers, many CNNs contain *pooling* or *subsampling* layers which are usually located after convolutional layers. Here the feature maps acquired from the convolutional layers are subsampled to reduce the dimensionality of the network, thereby bringing back the complexity of the

problem and the risk of overfitting. This subsampling is done by partitioning a feature map into non-overlapping areas which are mapped to single values using a filter. For example, a common filter is *max pooling*, where every area is mapped to its maximum value. A common filter size is 2×2, which reduces a feature map to a quarter of its size (see figure 4.3).

The final layers of a CNN are fully connected layers, which perform the classification. These are similar to layers in standard MLPs, as discussed in sections 2.2 and 2.3. They receive their input from the final convolution or pooling layer and calculate their activations and output using matrix multiplications. A softmax function is used to arrive at classification results and a loss function in combination with backpropagation can be used to learn the weights (which are the filters in the convolutional and pooling layers).

4.5.1 C-CNN

The first discussed CNN-model is a simple architecture proposed by Kim et al. [52], referred to as *concatenation-CNN*. The model is a slight variant on a CNN architecture proposed by Collobert et al. [17] and contains just one convolutional layer that extracts features from word sequences. It can start out with random word embeddings, training them in the process using backpropagation, or use pre-trained word embeddings which can be kept static or fine-tuned during training.

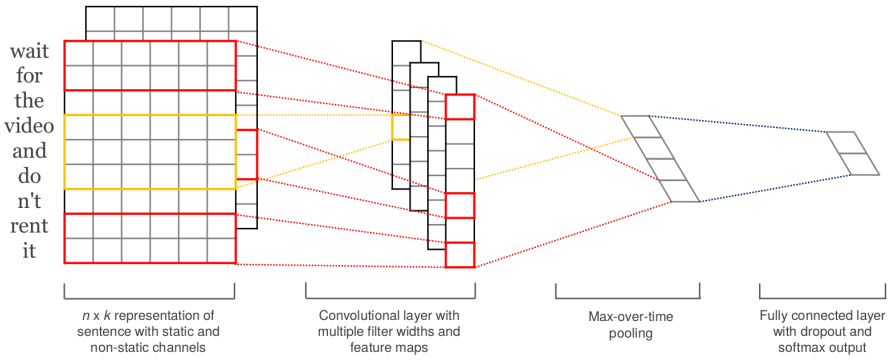


Figure 4.4: The CNN-architecture proposed by Kim et al. (image taken from [52]).

Given an input sequence $v_i, ..., v_j$ (padded where necessary) where v is a k-dimensional word embedding, the first operation consists of concatenating the embeddings, forming $v_{i:i+j}$ (see figure 4.4). A convolution layer performs a convolution on $v_{i:i+j}$ to extract features:

$$c_i = f(w^T v_{i:i+h-1} + b) \tag{4.4}$$

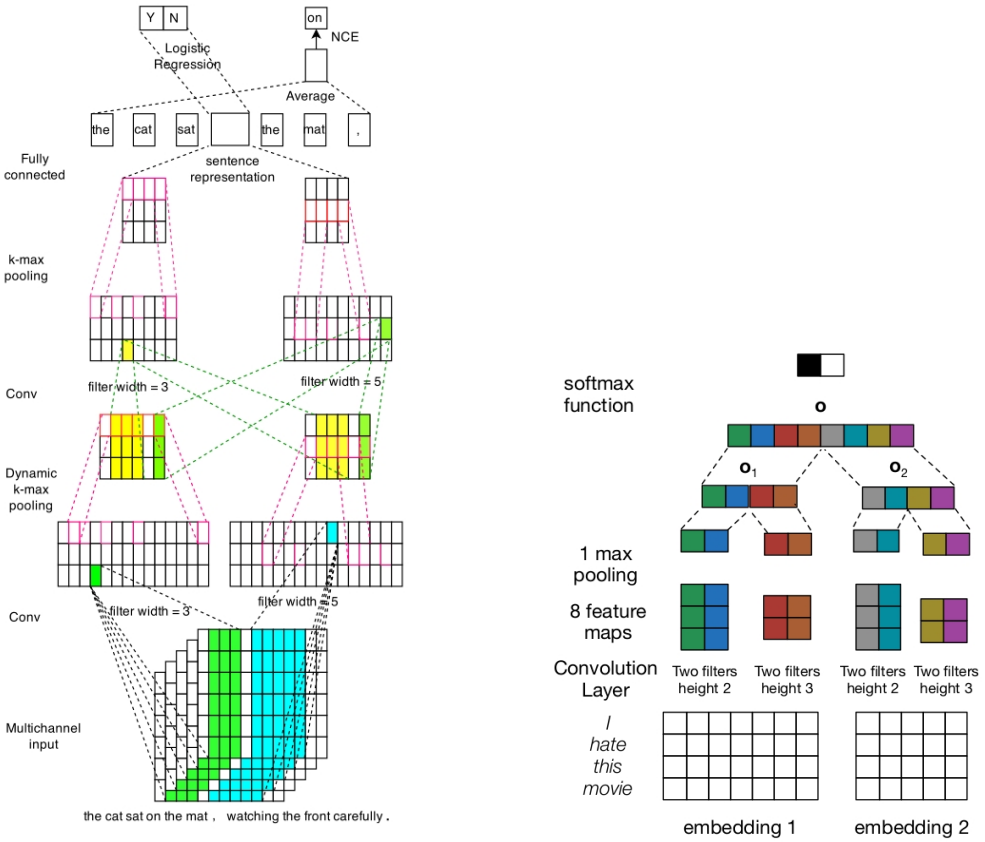
Here c_i is an extracted feature, f is a non-linear function such as a hyperbolic tangent, $w \in \mathbb{R}^{hk}$ is a filter applied to a window of h words, and $b \in \mathbb{R}$ is a bias term. Applying this filter to each h-sized window of words in the sequence produces the feature map $\mathbf{c} = [c_1, ..., c_{n-h+1}]$.

The pooling layer applies a max-over-time pooling operation, which takes the maximum value $\hat{c} = \max(c)$ of the entire feature map, considered to be the most relevant feature. As such, this pooling scheme can easily deal with variable sized input sequences. This way, one feature is extracted using one filter. To obtain multiple features, several filters are applied with varying window sizes, which together are passed on to a fully connected softmax layer. This layer yields the probability distribution over the labels. Using backpropagation they train the filters and optionally the word embeddings.

As is shown in figure 4.4, Kim et al. also experimented with dual-channel CNNs, where two channels of word embeddings are being used. They kept one static throughout training, whereas the other channel was updated via backpropagation. Filters were applied to both of the channels and added to arrive at feature map $\mathbf{c}$. Their models provided competitive performances to other state-of-the-art models, with the fine-tuned pre-trained embeddings and the multi-channel architectures slightly ahead of the randomly initialized and static ones.

4.5.2 MVCNN

Recently, Yin et al [121] proposed the *multi-channel variable-size convolutional neural network* architecture, in which separate input channels are initialized by different embedding versions, acquired from different techniques [1]. They argue that using multiple channels has at least two advantages: more information per word is leveraged and a word missing in one type of embedding can be compensated by another ('partially known words').



(a) The multi-channel CNN-architecture proposed by Yin et al. (image taken from [121]).

(b) MGNC-CNN proposed by Zhang et al. (image taken from [126]).

Figure 4.5: Different CNN architectures for paragraph classification.

Figure 4.5a shows the MVCNN architecture, which is built from a combination of components found in the models proposed in [49] and [52]. The input consists of a three-dimensional array of size $z \times k \times n$, where z is the number of embedding types, k is the dimensionality of the word embeddings and n is the number of words in a sequence. The filters in the convolutional layers differ both in weights and in size and are applied to all channels (their outputs are summed). This yields multiple different feature maps.

Subsampling is done using *dynamic k-max pooling*, introduced by Kalchbrenner et al. [49]. Here the k most active features are pooled instead of just the one max as with Kim et al. The advantage of this is that different convolutional layers can be connected to extract higher level abstract features. The pooling parameter k is determined as follows:

$$k_i = \max\left(k_{top}, \frac{L-i}{L}n\right) \tag{4.5}$$

Here i is the number of the convolutional layer to which the pooling is currently applied, L the total number of convolutional layers in the network, and k_{top} a fixed pooling parameter for the topmost convolutional layer. This pooling process makes sure that different sized feature maps are reduced to the same size. The

[1] Yin et al. use Word2Vec and GloVe, as well as techniques that are not discussed in this thesis.

advantage of this is that features from multiple sized phrases within the input sequence can be taken into account simultaneously. Finally, logistic regression is applied to arrive at a classification from the output of the fully connected layer.

4.5.3 MGNC-CNN

Multi-group norm constraint CNN (MGNC-CNN) proposed by Zhang et al. [126] is also derived from the models introduced in [49] and [52], but with it they claim to provide the following advantages as opposed to MVCNN: MGNC-CNN allows for different dimensionalities of input embeddings, which is not allowed in MVCNN, and MGNC-CNN is comparatively simple and efficient.

The augmentations of their model lies mainly in the fact that they treat each word embedding independently (see figure 4.5b). As such, the input to the CNN comprises multiple sentence matrices $A_1, A_2, ..., A_z$, where each $A_l \in \mathbb{R}^{n \times k_l}$ has its own width k_l. Different groups of filters are then applied independently to each A_l, where each group can contain filters of various weights and size. At the classification layer the different feature vectors obtained from the independent convolution and pooling layers are concatenated and softmax is applied.

So where MVCNN used the same filters on all channels simultaneously and generates a scalar from all channels in every layer, MGNC-CNN uses different filters on different embeddings independently and in the end combines the found features.

Finally, as another augmentation they propose grouped regularization constraints for the features extracted from each embedding set independently. These constraints penalize larger weight estimates for features derived from less discriminative embeddings, as to capitalize on features that capture the most discriminative properties of the input sequence for the given classification task.

Part II

Model Implementation and Application

Chapter 5

Methods and Experiments

To address the research questions posed in the introductory chapter, we applied several pipelines executing the techniques discussed in part 1. Figure 5.1 illustrates the final model, indicating the different stages of these pipelines and the combinations of techniques that were used.

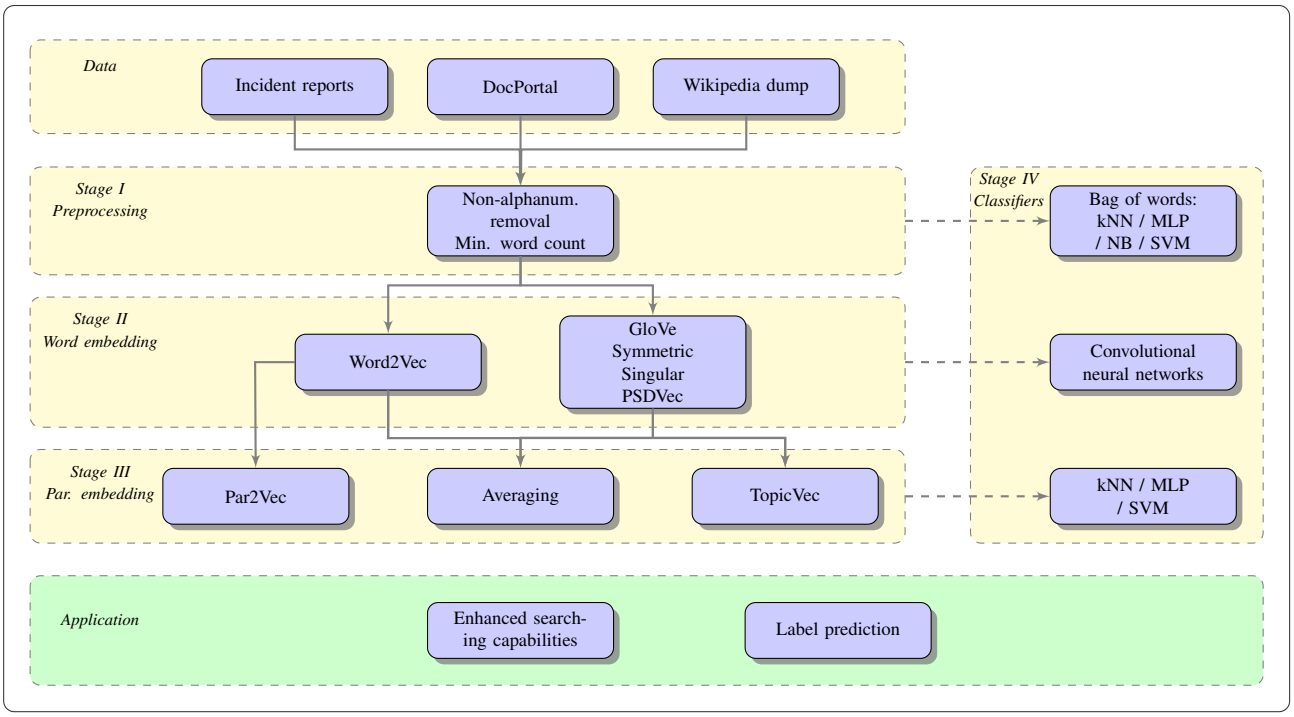


Figure 5.1: The final model.

This chapter sheds light on the training data, the configurations that were used in each stage of the model and the methods that were used to evaluate the different techniques.

5.1 Training Data

Three datasets were utilized for training purposes: the incident report database, DocPortal and Wikipedia dumps. The most relevant is the incident report database, since ultimately the goal is to enhance search capabilities within this database and to predict labels for it. The dataset consists of 39,164 reports up to April 2017, containing approximately 2.4 million words in total. It is split into two sets, distinguished by the software with which they were entered into the database. The older set, 'TPSC', contains all reports up to April, 2014, amounting to 25,994 documents, roughly two thirds of the entire set. The second set,

'Infoland', contains all reports from there onward. There is no reason to assume that the descriptions of the incidents themselves differ significantly between the two sets. However, the categories and labels were subjected to a revision going from TPSC to Infoland, which poses a problem for classification. To make use of all the data, we had to map certain categories and labels to each other, ultimately resulting in loss of accuracy, since most of these cannot be mapped precisely one on one. In chapter 7 the choices we made in this respect are explained in detail.

The size of the report dataset is relatively small when compared to corpora and datasets that are typically used for NLP purposes, often being in the order of hundreds of millions or even billions of words. However, since the relevant vocabulary is greatly limited by the fact that we are only concerned with the medical field and the dataset is mostly comprised of such jargon, the relative size of the dataset is not that small (with on average ca. 161 words for every word in its vocabulary, compared to e.g. 211 in the Wikipedia dump). The typical giant corpora provide a lot of overhead for this specific application, containing many words related to irrelevant topics such as politics or sports.

Fortunately, another dataset containing Dutch, medical terminology was available: DocPortal. DocPortal is a collection of all the protocols that are used within the UMCG.[1] Using the Python library *textract*[2], we extracted the raw text from the PDF-files that the collection contains. The extracted set of texts consists of approximately 26 million words that are mostly related to medical topics. Finally, we made use of the Dutch Wikipedia dumps of March 2017, incorporating approximately 220 million words, involving a wide spectrum of topics.[3]

We took a number of steps to preprocess these corpora for use in embeddings and classification. In order to prevent words combined with punctuation marks to be entered as unique entries in the vocabulary, all the non-alphanumerical characters were removed. We decided to keep numerical characters, because these could be insightful, for example in the case of regularly occurring medicinal doses. We then attempted to improve the results using a number of preprocessing techniques that are well known within natural language processing.

First, a procedure called stop-word removal was applied, where frequently used words with little semantic value are removed from the texts [100]. Akin to stop-word removal is subsampling, where words more frequent than some threshold are removed with a certain probability. Finally, we applied a procedure called stemming, in which all words with the same stem are reduced to a common form [67]. This procedure was only applied on the bag of words classifiers, since it would counteract the goal of providing helpful search suggestions with the search embeddings.

Interestingly, none of these three preprocessing techniques resulted in either better word embeddings or higher classification rates. It seems that the techniques work best when given the data raw instead of tempered with. We therefore decided to adhere to the raw texts, aside from the removal of non-alphanumerical characters. Finally, for a word to be included in the vocabulary of the word embedding techniques and the bag of words classifiers, its minimum number of occurrences in the corpus was set to 5.

5.2 Configurations

5.2.1 Word Embedding

We performed a preliminary evaluation in the same manner as our final evaluation, explained in section 5.3, to determine the configurations of the embedding techniques.[4] Following this evaluation we set the window size to 15 and created vectors with dimensionalities 50, 100, 200 and 300. For technique-specific parameters the settings as proposed in their respective papers were used, except for the negative samples and the subsampling limit in Word2Vec which were set at 25 and 10^{-3} respectively.

On the basis of this preliminary evaluation and visual inspection, we decided to discard the Symmetric technique (see for example figure 7.3f). It was clear that the results of Symmetric do not represent the semantic properties of the target words properly when trained on the incident reports alone, regardless of

[1] https://protocollen.umcg.nl/

[2] https://textract.readthedocs.io/en/stable/

[3] https://dumps.wikimedia.org/nlwiki/

[4] With the difference that we evaluated the search suggestions ourselves instead of experts, as well as taking only the best three suggestions per configuration instead of the best five. We had to make these concessions, because with the 183 different evaluated parameter configurations, on average 80 unique suggestions per search term were presented, using the five closest words per configuration. This would have required too much time of the available experts, so we had to make a preliminary selection.

whether the symmetric patterns are extracted from solely the reports or a combination of the reports and the Wikipedia dumps. The patterns it extracts are *X and Y*, *X or Y*, *X and a Y*, *X and the Y*, *they X Y* and *X Y and*. It seems that these patterns either do not occur often enough in the incident reports or that the patterns do not represent word similarity or analogy in the way that they are supposed to. At least the former is probable, as the incident reports are rarely written out in full grammatically correct sentences. Furthermore, to test this method on the corpus incorporating the Wikipedia dumps would be highly memory and time consuming. As an indication: training merely on the reports with the configuration as described above, the training took approximately 20 hours, using 8 threads of an Intel i7 6700K processor. The reason for this is that the smoothing step of the method requires the calculation of the cosine distance between every possible pair of word embeddings in the vocabulary, amounting to $\frac{|V|^2-|V|}{2}$ cosine calculations between $|V|$ sized vectors. With a vocabulary size of approximately 15,000 (following a minimum word count of 5), this amounts to about 112 million distance calculations of maximally 225 million floating point operations each (in practice it is less due to sparsity). If trained on a larger dataset, providing e.g. a vocabulary size of 100,000, approximately 5 billion distances of maximally 10 billion operations each - totaling $5\text{e}19$ operations - would be required, making it infeasible to pursue, given the preliminary results.

5.2.2 Paragraph Embedding

For training paragraph embeddings with Par2Vec the same configurations were used as for Word2Vec. For average embeddings, the inverse document frequency is used as weighting function (equation 3.2). Lastly, for TopicVec the parameter settings as suggested in [64] were used.

5.2.3 Classification

We implemented several classifiers for every type of input. The bag of words is used as input for naive Bayes (NB), k-nearest neighbors (kNN), support vector machines (SVM) and multilayer perceptrons (MLP). The convolutional neural networks all get (combinations of) single word embeddings as input, and the paragraph embeddings will be used as input for k-NN, SVM and MLP. We performed parameter sweeps to arrive at the optimal configurations for the different classifiers. Table 5.1 shows the unique parameter settings of the final configurations used. All CNNs apply 10 filters of 3 and of 4 'words' long. Furthermore, similar to the MLPs, they use stochastic gradient descent, a softmax classifier, batch sizes of 32 and stop training when the decrease of loss is less than 0.01 per epoch (with a maximum of 15 epochs). Finally, they apply strides of 2 and use dropout rates of 0.5.

Table 5.1: The configurations of the different classifiers.

Classifier	Configuration
NB	zero probability $= 1/$total training words
kNN	$k = 20$; metric: Euclidean; weights: uniform
SVM	linear; 'one-vs-rest'; $C = 1.0$
MLP	1 hidden layer; 200 hidden rectified linear units
C-CNN	conv/pooling layers: 1; channels: 1 / 3
MVCNN	conv/pooling layers: 3; channels: 3
MGNC-CNN	conv/pooling layers: 3; channels: 3

5.3 Evaluation

Evaluating the word-embedding techniques is not straightforward, as there are no complete criteria by which to measure the extent to which they represent words. Generally, for evaluating word embeddings, analogy or similarity questions regarding manually created lists of word pairs are used, such as in [71]. Word similarity tasks focus on the semantic content of word embeddings as a whole and whether these coincide with each other: synonym embeddings should be highly similar in vector space, whereas antonyms should be highly dissimilar. In analogy tasks the various dimensions of the word embeddings are examined and not just their scalar distance. This is done by evaluating analogies such as "king is to queen as man is to woman" through

the equation $v_{king} - v_{queen} = v_{man} - v_{woman}$, where v is a word embedding. Word embeddings with meaningful dimensions tend to score well on these tests.

However, with regard to the purposes of this thesis, we do not necessarily require those embeddings that are good at analogy or similarity tests. We need those embeddings that provide the best search suggestions and indirectly the best classification results. Naturally these largely coincide with the most similar words, but not necessarily. For example, when searching for all incident reports that involve depression, it might be helpful to not only get words similar to 'depression', such as its synonyms or derivations, but also antidepressant brands that are often prescribed or symptoms that usually occur with depression. In other words, we need to assess the extent to which words are associated with each other.

Therefore, to evaluate the word vectors we presented several experts with possible target search terms paired with search suggestions, in order to acquire their assessment of the possible relevance of the suggestions. The list of target search terms was generated by extracting those terms from the vocabulary of the word embeddings that also occur in the medical dictionary found at http://www.medisch-woordenboek.nl/. From this list 100 terms were randomly selected, of which 7 were removed, because they resembled other terms already present in the list, or were unlikely to be used as search terms. The lists of presented search suggestions were then created by taking all unique words from the combined top five search suggestions of every embedding technique (using the cosine distance), amounting to an average of 28 search suggestions per target search term. The experts were asked to evaluate these suggestions based on their relevance, given three options: (1) irrelevant, (2) possibly relevant and (3) relevant, valued -1, 0 and 1 respectively. These valuations were then weighted by the relative positions of the suggestions to the target search term, multiplying them by $1, 1/2, 1/3, 1/4$ or $1/5$ accordingly, and summed to get a score between -2.283 and 2.283 per search term. Finally, the average of all 93 scores was taken to arrive at the score of an embedding technique. An example of the rating form presented to the experts is shown in figure 5.2.

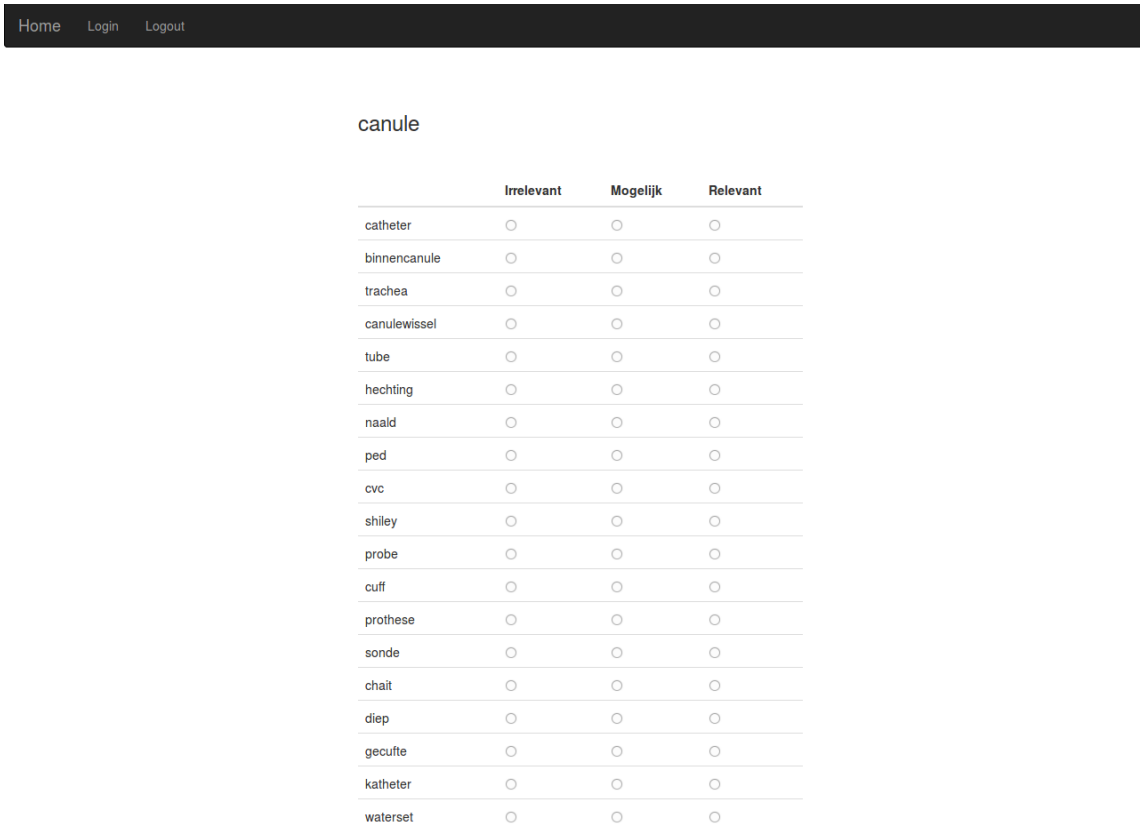


Figure 5.2: The rating form, asking to rate the provided suggestions for the word 'canule' (cannula). Note that the suggestions are presented in a random order.

For evaluating paragraph embeddings, usually classification tasks are performed. However, this requires labels that are to a large extent representative of the content of the paragraphs, such as in sentiment classification of reviews [21, 52, 57, 121, 126, 127]. The content of the incident reports is not easily captured in

one or two labels. The database contains over a hundred different categories of labels, all representing either a piece of meta data or a specific part of the semantic content of the reports. Therefore, we will evaluate the embeddings based on a combination of classification results for different labels. Another method, proposed in [21], makes use of manually composed triplets of articles, where two are semantically similar to each other and the third is not. Based on the distance in vector space between the report representations a model would have to select the two semantically similar ones. With the UMCG dataset such a method might be more informative. However, on top of the word embedding evaluation, this would have required too much time of the available experts and was therefore omitted.

Finally, evaluating the classifiers is straightforward, as the classification results can be directly translated into accuracies. One side note here is that not all classifiers receive the same kind of input, meaning that we cannot compare these classifiers directly, as it is not a 'level playing field'. For example, we cannot conclude that a CNN is a better classifier than naive Bayes, since they both leverage different kinds of information. Nevertheless, we can compare the combinations of techniques as there will ultimately be a combination that provides the best classification results. After all, in an application that makes use of label prediction, you simply require the system that makes the most accurate predictions.

Chapter 6

Application: iSearch

In the introductory chapter of this thesis we specified two ways in which the Central Incident report Commision (CIM) of the UMCG would benefit from this research project: (1) improving the effectiveness of search queries by incorporating content-related words in the search results and (2) providing a means of knowing whether new reports contain content that has not been encountered in the database before. To meet these requests, we applied the investigated methods in what could be described as a 'smart' search tool for the incident report database (but also extensible toward other databases). This tool is implemented using the open Python library PyQt4[1]. This chapter discusses the implemented features of what we called *iSearch*.

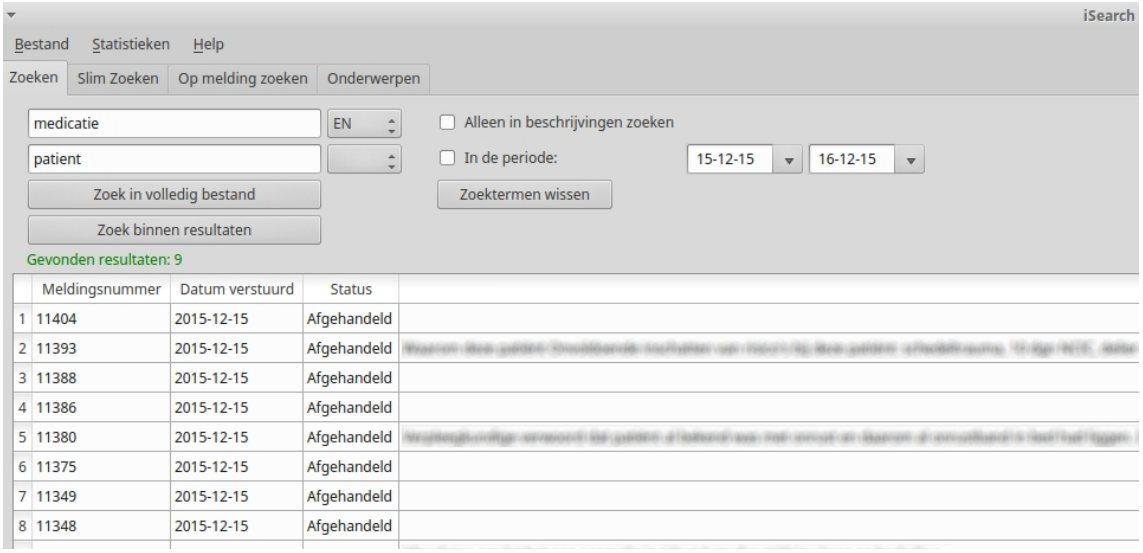


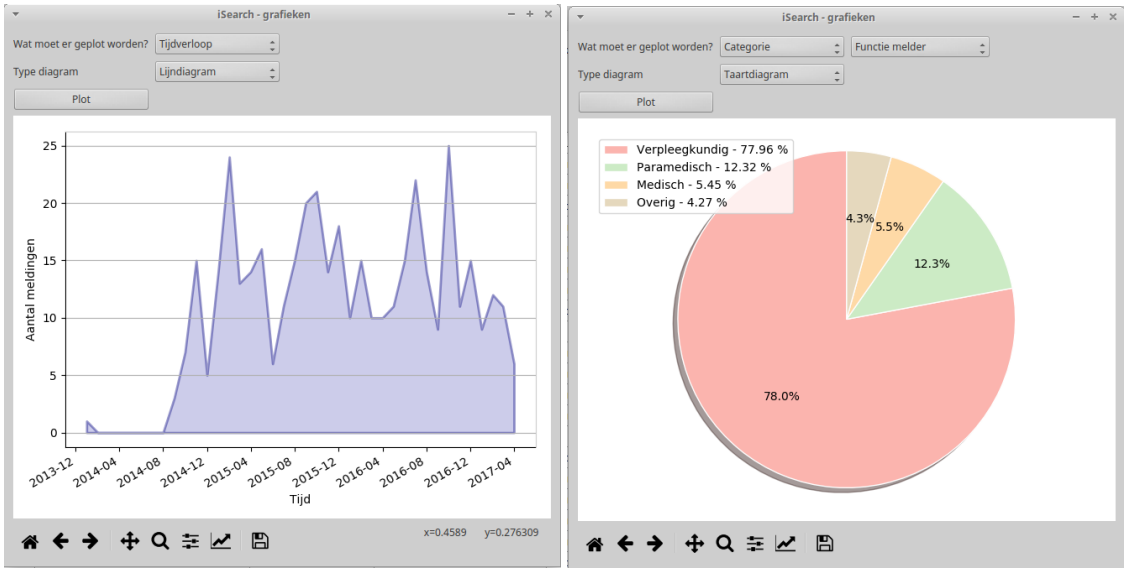
Figure 6.1: Part of the main screen of iSearch with the 'search' tab open, showing the basic functionality. Possibly sensitive text has been blurred (as is the case in figure 6.3, 6.4 and 6.5).

6.1 Basic Functions

Figure 6.1 shows the main screen of iSearch. The current open tab offers 'standard' boolean searching functionality, with the possibility of searching only within the descriptions of the incident reports (excluding categorical data) or to search within a certain time period. After searching, the results can be exported to an xls document or a new document can be opened. Another basic function the software offers is to plot the search results, making use of the Python library Matplotlib[2]. For example, the frequency of search hits can

[1] http://pyqt.sourceforge.net/Docs/PyQt4/index.html
[2] https://matplotlib.org/

be plotted over time or per department, function, risk, consequence or type incident, as is shown in figure 6.2.



(a) Line diagram of the number of incidents over time, given a specific search query.

(b) Pie diagram showing the function of employees that were involved in the incidents found with the search query.

Figure 6.2: Examples of plotting within iSearch.

6.2 'Smart' Search

While the basic search functions already provide a way to significantly speed up the process of searching within the database compared to the previous situation (where just one search term was possible, without the option to search within the descriptions only), we applied word embeddings in a way that actually helps the user with the content of his or her search query. Figure 6.3 shows an example of how the smart search function works. To use the function, a document containing a vocabulary of word embeddings has to be loaded. Using this vocabulary, every search query is then extended with the x closest words to the target term (using the cosine distance as measure), meaning that both the original search term and the suggested terms are sought for in an 'OR' fashion. For example, the figure shows a search for 'medication' and 'patient'. These terms are extended with a number of related terms such as 'medicine' (both correctly and incorrectly spelled) or 'patiënt', as well as some less relevant terms, shown in the right panel of figure 6.3. The user can then remove the irrelevant terms to optimize the query.

There are three advantages to this approach compared to the basic search functionality. First, it saves effort and time, since the user does not have to manually add every possible derived form of the search term or other terms he/she can come up with. Second, the function can suggest relevant search terms that the user would not have thought of, making the search more complete. Finally, it can bring to light patterns in the data that the user was not aware of initially. For example, the user might search for a certain condition and find a specific type of medication in the suggestion list. While no hard conclusions can be drawn from this, it can be a reason for the user to investigate the co-occurrence of the condition and the medicine further.

A second 'smart' search feature is shown in figure 6.4. Here the paragraph vectors are used to search on the basis of a complete report description, using the report numbers. The number of documents closest to the target report can be specified, which is then presented in an ordered list, showing the most equal document on top. The advantage of this type of search is that the user does not need to think of any search terms. Moreover, the program can find resembling documents, even if no term in the documents overlap. This can be especially useful when an initial standard search query yields few or no results. The user can then use this function to find out whether there are any reports at all that come close to the content of the target report, before spending too much time fruitlessly searching for what could turn out to be non-existent.

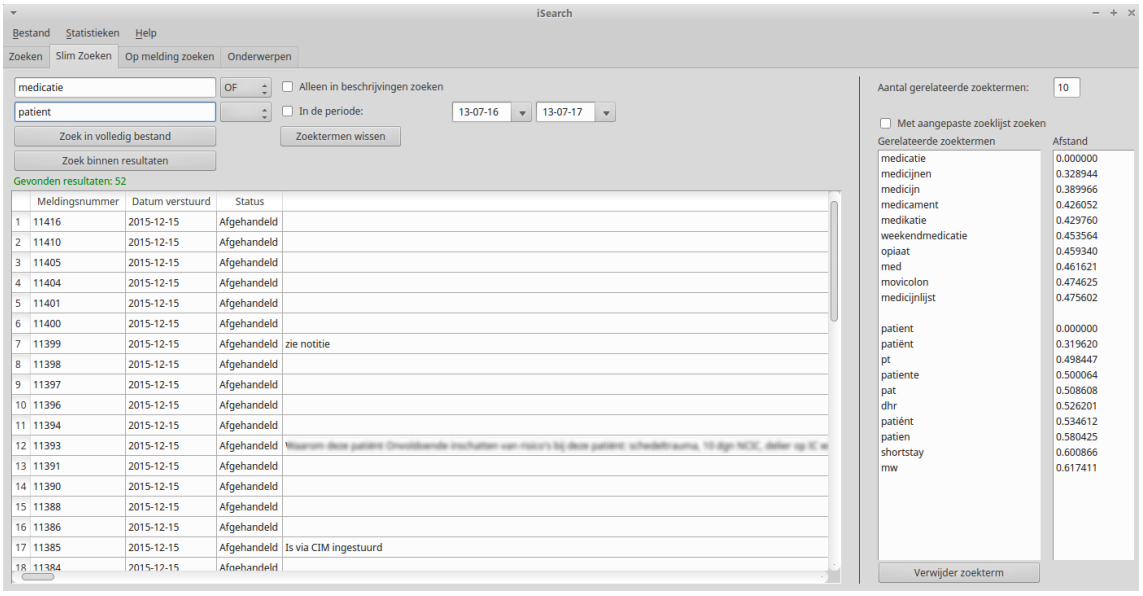


Figure 6.3: The smart search tab of iSearch showing an extended search query.

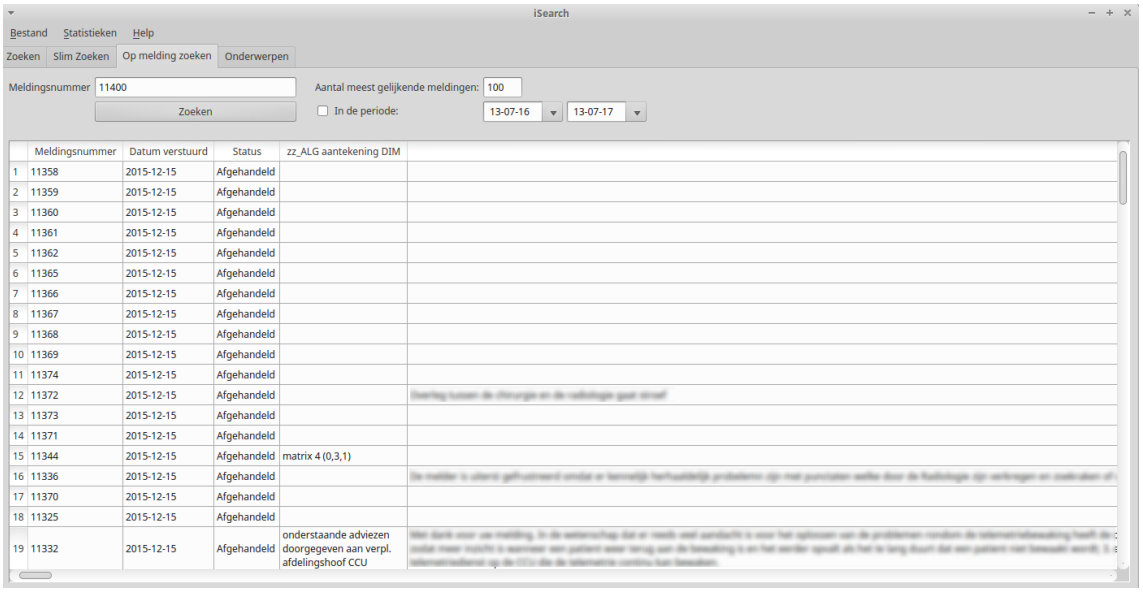


Figure 6.4: The report search tab.

6.3 Topic Extraction

Finally, we added a function involving topic extraction, which is something the paragraph embedding technique TopicVec uses (see section 3.3). In the application a simpler, well known method is used for topic extraction, called latent Dirichlet allocation (LDA) [91, 9]. This is a generative model that assumes documents are generated through a mixture of topic distributions. These distributions can be found by first randomly assigning words to topics given a specified number of topics and a Dirichlet prior distribution, and then iteratively reassigning them according to the proportions $P(t|d)$ and $P(w|t)$, with t being a topic, d a document and w a word.

Figure 6.5 shows the topic extraction tab of iSearch. Here a date can be entered to extract the n topics most prevalent in the incident reports that day, of which the k best describing words are shown in the lists on the right. Pressing the button 'Woordenwolk' (word cloud) in the bottom of the screen then generates a word cloud from these topics according to the method proposed in [63], which is also used in [64] (see

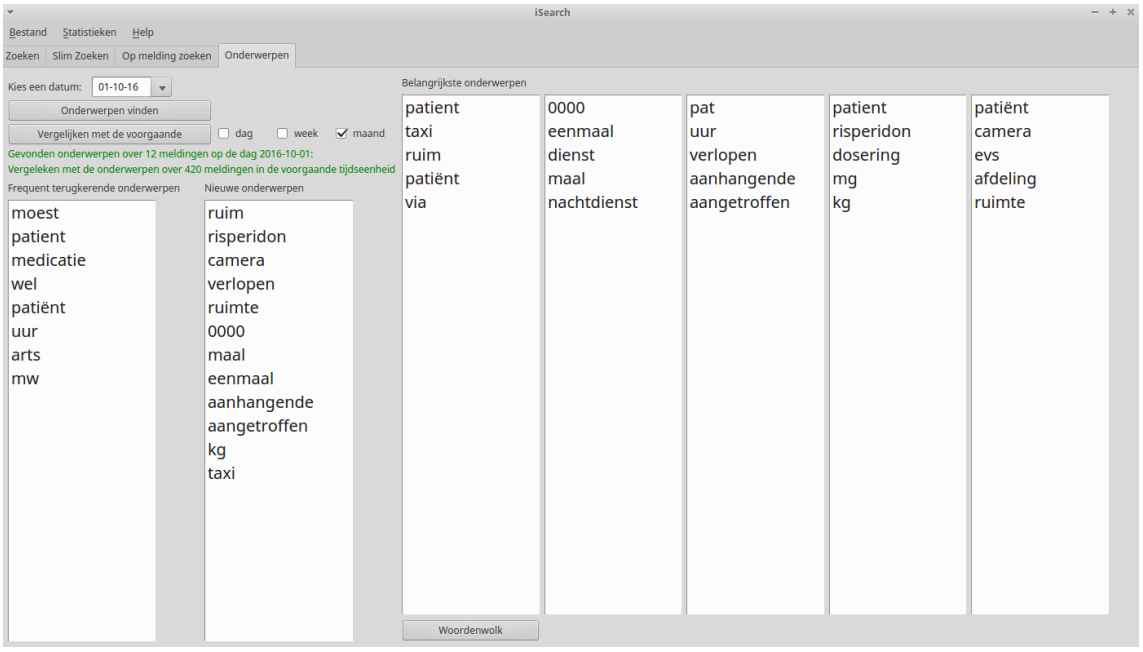


Figure 6.5: The topic extraction tab.

figure 3.4). Figure 6.6 shows two examples of such topic clouds. This representation can be useful for getting a quick idea of the types of incidents that occurred on the specified date, without having to scan through all of the corresponding reports.

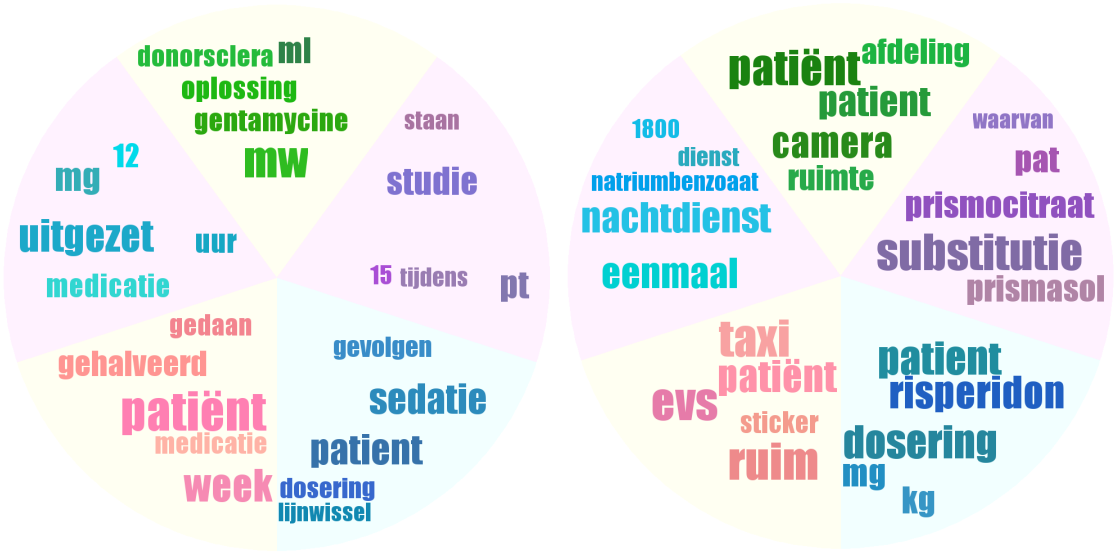


Figure 6.6: Word clouds representing five topics extracted from single days.

Another feature the topic tab offers, is to compare the topics found for the specific date with the topics of the days in a specified period of time prior to it (day, week or month). Given this comparison, the left most panel in figure 6.5 shows the topic words that frequently occur within the time period, including the inserted date. This is done using a hard margin of $1/4$, meaning that if a topic word appears in over $1/4$ of the specified days, it is marked as frequently appearing. The second panel from the left shows all topic words that did not occur in the topics of the period prior to the specified date. This feature could help in detecting trends of newly occurring types of incidents in an early stage.

Part III

Evaluation

Chapter 7

Results

The following sections elaborate on the results of the experiments that were discussed in chapter 5. The first section provides the results of the expert rating experiment, offering a measure of the extent to which word embeddings capture the relatedness between words. The second section treats the classification results, assessing the influence of both the input data and the used technique on the classification accuracy.

Table 7.1: Evaluation scores per technique and dimensionality.

	Dimensionality			
Method	50	100	200	300
CBOW	0.367	0.395	**0.418**	**0.403**
SG	**0.399**	**0.407**	0.267	-0.169
PSDVec	0.307	0.373	0.326	0.221
GloVe	-0.197	-0.081	-0.062	-0.057
Singular	-0.732	-0.570	-0.379	-0.282

7.1 Expert Rating Experiment

Table 7.1 and figure 7.1 show the scores of the different word embedding techniques, averaged over the number of search terms that were presented to the experts, and weighted according to the position in which search suggestions were proposed by the techniques. These are the results for the embeddings trained on the incident report database ('UMCG-trained').

For a dimensionality of 50 and 100, CBOW, SG and PSDVec perform similarly, while the CBOW architecture performs significantly better with 200 and 300 dimensions. The performance of CBOW also seems less dependent on the dimensionality, with a mean absolute deviation of ca. 0.015, compared to 0.198, 0.043, 0.049 and 0.16 for SG, PSDVec, GloVe and Singular respectively. The latter two clearly perform worse, with Singular especially reaching relatively low scores for low dimensionalities. Interestingly, both these techniques seem te perform better with increased dimensionality, whereas the opposite is true for SG and PSDVec.

In line with these results, the scores for the embeddings trained on the larger corpora, illustrated in figure 7.2, also show that CBOW outperforms the other techniques, followed by SG, GloVe and Singular in the same order as in figure 7.1.[1] Another phenomenon we can observe in 7.2, is that the scores are higher overall, compared to the UMCG-trained embeddings. This indicates that the rated association between the target words and their suggestions increases when the embeddings are trained on larger corpora.

We can visualize this association by plotting the relative distances between the embeddings using t-SNE, a method, proposed in [69], that projects groups of high-dimensional vectors on two dimensions. Next to the statistical data we acquired from the expert rating experiment, such visualizations can aid in assessing the techniques in a qualitative manner.

[1] PSDVec is not included, because training with the larger corpora and vocabularies required more memory ($>$ 16GB) than our machine had available.

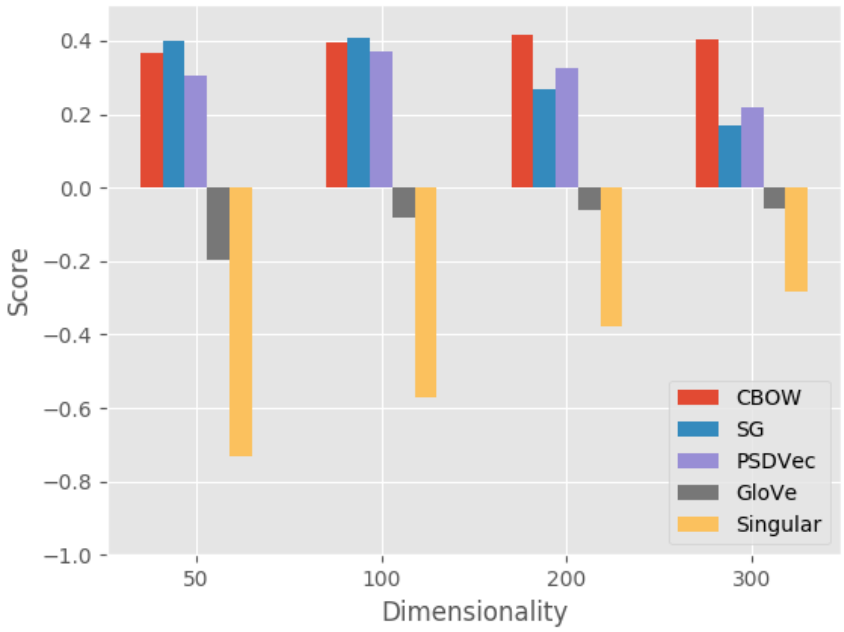


Figure 7.1: Evaluation scores of the different word embedding techniques, based on expert ratings of the search suggestions provided by the word embeddings. The embeddings were UMCG-trained with dimensionalities 50, 100, 200 and 300.

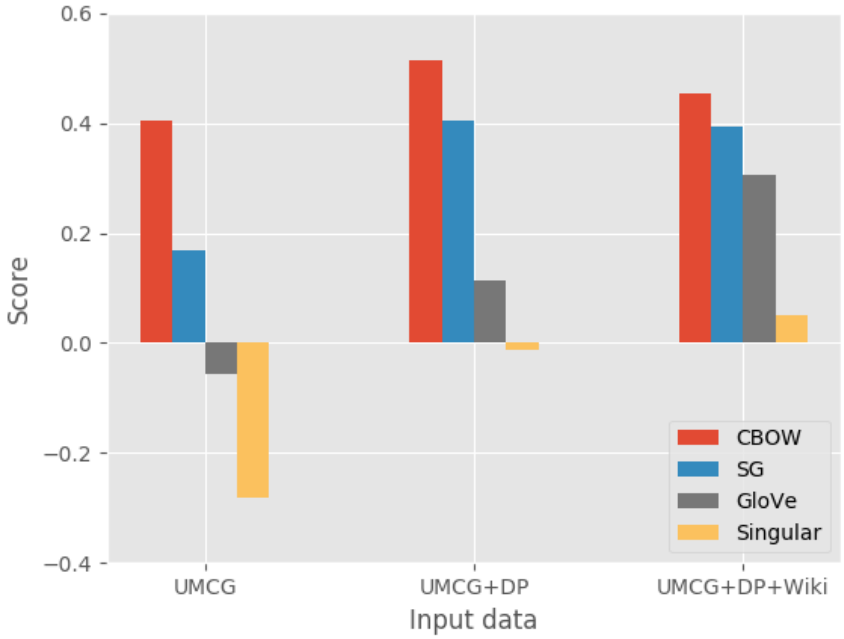
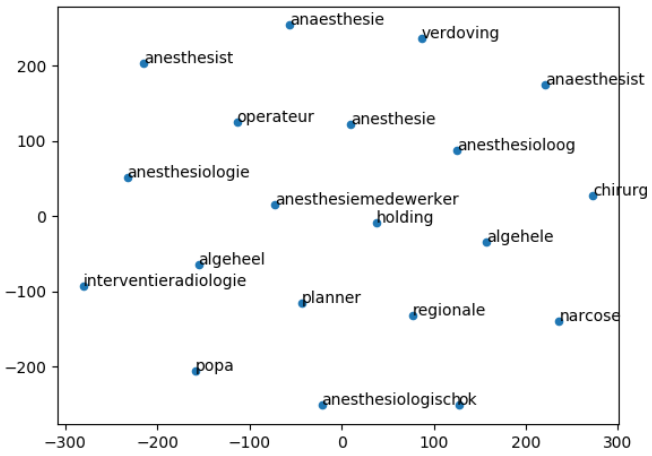
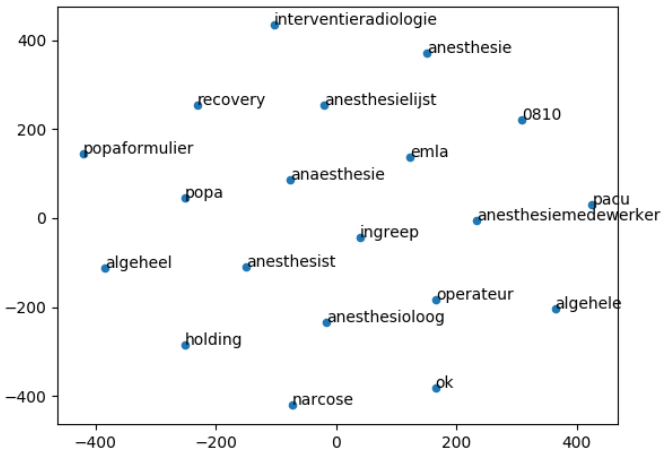


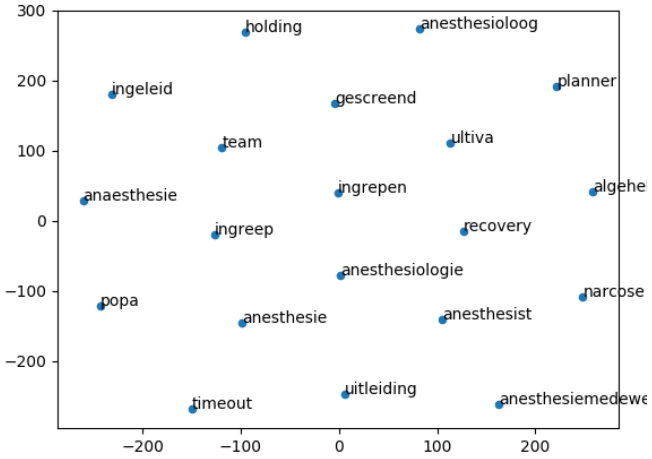
Figure 7.2: Expert evaluation scores of the different word embedding techniques per training database. Here 'UMCG' denotes the collection of incident reports, DP is the DocPortal database containing medical protocols and Wiki is the Wikipedia dump.



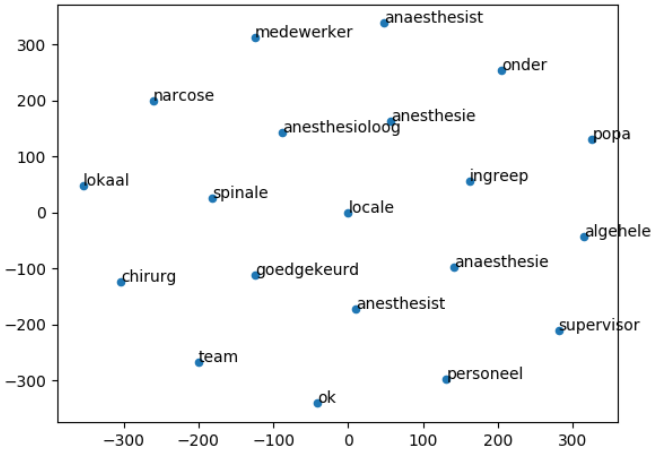
(a) CBOW



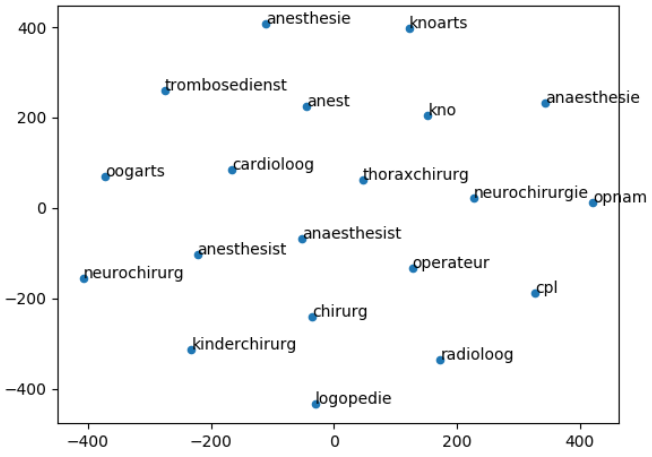
(b) SG



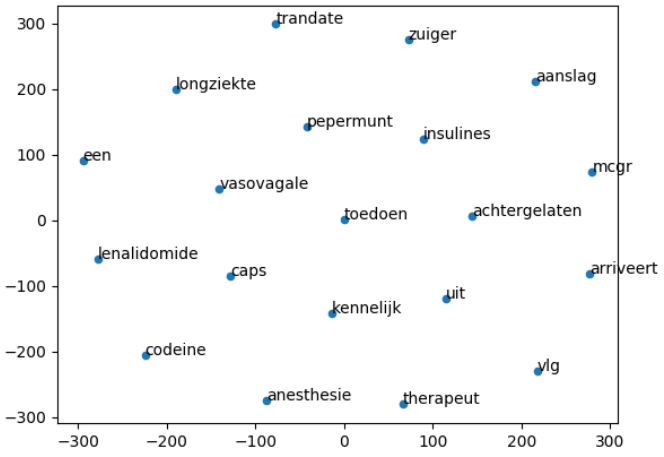
(c) PSDVec



(d) GloVe



(e) Singular



(f) Symmetric

Figure 7.3: Visualization in two dimensional space of the twenty words closest to ‘anesthesie’ (anesthesia) for each word embedding technique.

Figure 7.2 shows such a visualization of the twenty words closest to 'anesthesie' (anesthesia) for every word embedding technique trained on the UMCG corpus, using 50-dimensional embeddings. The first observation we can make (as mentioned in section 5.2.1), is that no coherence can be found in the cluster of Symmetric-trained word embeddings, which is why we dropped it in the other tests. The other clusters do show words that are related to anesthesia, although we can observe a difference in how directly related these words are. For example, Singular provides a few words that are directly related such as 'anaesthesie' (same word, different spelling) or 'anesthesist' (anesthetist), but it mostly finds indirectly related words, such as types of surgeons or departments where surgeries take place. While these terms are not unlikely to be found in the context of anesthesia, they are not related directly. The other four techniques seem to find more directly related terms in the proximity of the target term, with CBOW clearly superior in this respect. As such, these visualizations support the results found in the expert rating experiment.

7.2 Classification Results

As is mentioned in section 5.1, the dataset consists of 39,164 reports. These reports are divided into two parts: the newest set, Infoland, amounting to one third of the data, and the older set, TPSC, amounting the two thirds. Both of these parts contain over 100 categorical variables, of which a portion needs to be filled in by the reporter and a portion by the analyzer. Table 7.2 shows five of these categories that we think represent a significant and important portion of the content of the reports. The final two columns show the number of labels that were available for these categories in both datasets and the number of labels that we mapped them to.

The first category, 'consequence', describes how severe the consequences of the incident were for the patient in terms of discomfort, injury, prolonged hospitalization period and media attention. Both sets use slightly different terms and measures, which we mapped to 'unknown', 'small', 'average' and 'large'. The second category describes the risk score of the incident, which was mapped similarly, with the exclusion of the 'unknown' label, which did not occur in either dataset.

The third category describes the type of incident as a whole, which likely represents a large portion of the topics discussed in the reports. This category contains labels representing specific types of incidents, such as 'medication' and 'falling', and labels representing groups of incident types such as 'research / treatment' and 'organization / communication / documentation / patient transport'. The labels of the two datasets mostly coincide. The category 'function' requires the reporter to enter his or her function description. While TPSC uses a very specific function description (25 labels), Infoland requests the entry of a more general one (6 labels). We mapped the TPSC classes to the 6 Infoland classes, with the exclusion of the 'other' label, which is present in TPSC, but not in Infoland. Finally, 'department' describes the hospital department the incident took place in. This category is not present in TPSC, meaning that we could only make use of the Infoland set. Moreover, 47 possible labels are used for the category, making the number of available reports per label relatively small. Still, since many incidents are department-specific, we expected that classification should be very well possible within this category.

Table 7.2: The different categories that were used for classification

Category	Datasets	Original number of labels	Mapped number of labels
Consequence	Infoland, TPSC	Infoland: 8, TPSC: 6	4
Risk	Infoland, TPSC	Infoland: 5, TPSC: 5	3
Incident type	Infoland, TPSC	Infoland: 7, TPSC: 10	7
Function	Infoland, TPSC	Infoland: 6, TPSC: 25	7
Department	Infoland	Infoland: 47	47

7.2.1 Classifiers using Paragraph Embeddings

For our experiments we ran k-fold cross validation for $k = 10$, for every k pre-training the word and paragraph embeddings on the 90% of the reports that is used for classifier training, optionally combined with the other corpora. To assess the strength of the paragraph embeddings and to find the best performing combination of paragraph embedding and classifier, we first tested all of these combinations on UMCG-trained embeddings. This resulted in the accuracies given in table 7.3. For comparison we added a baseline, based

on the probability of the most occurring label in the training data. Since we trained 300-dimensional word embeddings on DocPortal and the Wiki dumps, we used this dimensionality for the paragraph embeddings as well.

Table 7.3: Classification results per type of paragraph embedding and classifier. All embeddings are 300 dimensional and UMCG-trained. The baseline is set by always choosing the most occurring class label. The highest accuracies are boldfaced and the accuracies within 1% of these are italic.

		Category				
	Classifier	Consequence	Risk	Incident type	Function	Department
Input	Baseline	**80.49**	**57.87**	33.81	61.78	12.69
Avg. (CBOW)	MLP	*80.45*	56.64	**64.49**	**75.45**	42.60
	kNN	*80.25*	54.60	60.85	73.01	40.96
	SVM	*80.47*	56.52	62.40	73.74	42.06
Avg. (SG)	MLP	**80.49**	*56.75*	61.09	72.83	27.16
	kNN	*80.38*	54.20	62.36	72.44	38.34
	SVM	*80.46*	56.83	62.47	73.37	**45.32**
Avg. (PSDVec)	MLP	**80.49**	**57.87**	60.32	72.70	23.16
	kNN	*80.22*	52.42	61.60	73.05	35.46
	SVM	*80.47*	56.71	62.76	73.82	44.02
Avg. (GloVe)	MLP	**80.49**	*57.71*	58.38	71.04	23.22
	kNN	*80.35*	54.12	61.02	71.38	29.45
	SVM	*80.48*	*56.93*	60.16	72.11	35.33
Avg. (Singular)	MLP	**80.49**	**57.87**	55.01	69.21	12.76
	kNN	*80.31*	53.50	61.35	72.00	29.23
	SVM	*80.48*	*56.89*	60.88	71.23	32.22
TopicVec (CBOW)	MLP	79.37	55.27	51.01	67.62	23.39
	kNN	**80.49**	54.71	49.44	65.04	21.81
	SVM	*80.34*	56.72	54.81	68.21	27.30
TopicVec (SG)	MLP	78.44	52.18	49.83	65.12	17.85
	kNN	*80.48*	54.07	51.34	65.85	21.35
	SVM	*80.36*	56.78	49.98	66.26	21.11
TopicVec (PSDVec)	MLP	76.64	52.92	33.00	63.61	15.25
	kNN	*80.49*	54.06	46.95	63.49	16.74
	SVM	*80.35*	56.85	47.03	64.48	17.16
TopicVec (GloVe)	MLP	79.48	*57.57*	49.44	63.01	16.49
	kNN	**80.49**	53.85	48.06	63.71	17.38
	SVM	*80.39*	56.65	50.32	65.37	19.78
TopicVec (Singular)	MLP	79.73	51.88	36.24	62.95	14.42
	kNN	*80.43*	48.24	46.84	63.48	16.16
	SVM	*80.40*	*56.95*	45.98	64.18	16.74
Par2Vec	MLP	*80.33*	56.78	50.96	69.60	27.77
	kNN	**80.49**	55.40	39.78	63.03	16.64
	SVM	*80.42*	56.65	48.16	68.78	32.53

A number of patterns catch the eye when analyzing these results. First, no classifier supersedes the baseline accuracies of the ‘consequence’ and ‘risk’ categories, indicating that the highest accuracies are obtained within these categories by always choosing the most frequent label. Second, the average word embeddings perform significantly better than both the TopicVec and the Par2Vec based embeddings, with the latter two performing somewhat similarly, with slight differences per category. Third, the performance distinction we saw in the expert rating experiment between the different word embedding architectures is not as clear in these results. This is illustrated by the fact that the classification results are relatively closer to each other than the word embedding results are.

Table 7.4: The mean and mean absolute deviation of the classification accuracies achieved by the three classifiers MLP, kNN and SVM for all different input embeddings.

	Category		
Classifier	Incident type	Function	Department
MLP	51.80 / 7.33	68.47 / 3.64	22.19 / 6.21
kNN	53.60 / 7.12	67.86 / 4.10	25.77 / 8.10
SVM	55.00 / 6.13	69.23 / 3.29	30.32 / 9.01

Next to these patterns, the results show that the highest accuracies of the final three categories are obtained by MLPs using CBOW-averaged embeddings in the 'incident type' and 'function' categories (64.49% and 75.01%) and an SVM using SG-averaged embeddings in the 'department' category (45.32%). Furthermore, we observe that CBOW-averaged embeddings as input result in the highest accuracies averaged over the three classifiers. As such, our experiments show CBOW as the most solid choice for generating word embeddings.

Concerning the classifier type, table 7.4 shows that while the MLP achieves the best results in two out of three categories (disregarding the first two categories), averaged over all input embedding types its accuracies are actually lower than those of kNN and SVM, with SVM performing best on average in all categories. This is especially the case for the 'department' category, with an average of 22.19% opposed to 25.77% and 30.32% of kNN and SVM respectively.

7.2.2 Overall Comparison

Table 7.5 displays the classification results for every classifier and type of input. The input for the convolutional neural networks consists of CBOW- and SG-trained word embeddings, since these performed best in the expert rating experiment. The paragraph embeddings used are CBOW-averaged embeddings, since these provided the overall best classification results (see table 7.3). Again all embeddings are 300-dimensional. Finally, to leverage the information of all three sources of input, we implemented a voter that applies $\arg\max$ on the summed estimates of the three top-performing classifiers for each input type: naive Bayes, MGNC-CNN and the MLP using paragraph embeddings, the latter two using UMCG-trained embeddings.

Similarly to table 7.3, table 7.5 shows that the MLP utilizing UMCG-trained paragraph embeddings is the top performing classifier for the 'incident type' and 'function' categories with accuracies of 65.49% and 75.45% respectively. Naive Bayes follows closely, scoring not much lower on these categories and performing best in the department' category, with most classifiers performing poorly, at an average of 32.99% (see table 7.6). Again we observe that none of the combinations of input data type and classifier achieve higher accuracies than the baseline for the 'consequence' and 'risk' categories. In the discussion we will look into the possible causes for this deficiency.

The results for the 'consequence' and 'risk' categories contain little variance within the group, with most classifiers scoring within one percent of the highest result, and the mean accuracies being just 0.44% and 1.78% lower than the maximum accuracies, as is shown in table 7.6. For 'incident type' we see a significant difference between the CNNs and the classifiers using paragraph embeddings, with the latter almost exclusively scoring over 60% and the former below it. This diversity is less present in the 'function' category, where the mean absolute deviation is small at 1.69%, and the maximum score, obtained by the MLP using paragraph embeddings, is just over 3% higher than the mean accuracy, as opposed to ca. 9% for incident type. The most diversity, we find in the final category, 'department', where the mean absolute deviation is high (5.08) and the highest accuracy of 45.95%, obtained by naive Bayes, is almost 30% higher than the lowest, obtained by kNN with a bag of words.

To see if we can improve upon the results of the first two categories, we incorporated a number of other, more 'objective', categories in the input of the classifiers utilizing a bag of words and paragraph embeddings. To achieve this, we concatenated the input vectors with one-hot encoded vectors representing labels of other categories. We decided on the categories empirically, resulting in 'incident type' and 'cause' being added as input, the latter being the cause of the incident, as indicated by the reporter. The 'cause' category was an optional one for the reporter to fill in, so we labeled the missing entries as 'None', so

Table 7.5: Classification results per classifier and input data type. For the CNNs the CBOW- and SG-trained word embeddings were used. The paragraph embeddings are averaged 300-dimensional CBOW-embeddings, which achieved the highest average scores (see table 7.3). The baseline is set by always choosing the most occurring class label and the voter applies $\arg\max$ on the sum of the outputs of three classifiers: naive Bayes, MGNC-CNN and the MLP using paragraph vectors, both using UMCG-based embeddings. The highest accuracies are boldfaced and the accuracies within 1% of these are italic.

		Category				
	Classifier	Consequence	Risk	Incident type	Function	Department
Input	Baseline	**80.49**	**57.87**	33.81	61.78	12.69
Bag of Words	NB	71.55	52.77	*64.05*	72.54	**45.95**
	MLP	*80.32*	*56.21*	62.12	74.29	35.01
	kNN	*79.64*	50.90	44.26	63.54	17.59
	SVM	*80.45*	*57.25*	49.21	64.21	22.31
W. emb. (UMCG)	C-CNN (sc)	**80.49**	*57.83*	57.59	72.05	33.38
	C-CNN (mc)	**80.49**	56.08	54.42	72.15	34.06
	MVCNN	**80.49**	*56.90*	57.41	73.18	32.14
	MGNC-CNN	**80.49**	56.14	61.28	74.07	36.02
W. emb. (UMCG+DP)	C-CNN (sc)	*80.44*	56.24	55.89	71.78	30.71
	C-CNN (mc)	80.46	*57.01*	56.64	71.88	30.57
	MVCNN	**80.49**	*56.89*	58.06	73.02	34.15
	MGNC-CNN	80.45	56.68	59.85	73.43	29.23
W. emb. (UMCG+DP+Wi)	C-CNN (sc)	*80.48*	*57.24*	56.66	70.35	26.29
	C-CNN (mc)	*80.47*	*57.39*	55.12	70.03	26.14
	MVCNN	**80.49**	*57.65*	57.59	71.45	23.17
	MGNC-CNN	**80.49**	*56.88*	58.65	72.18	29.48
P. emb. (UMCG)	MLP	*80.45*	56.64	**64.49**	**75.45**	42.60
	kNN	*80.25*	54.60	60.85	73.01	40.96
	SVM	*80.47*	56.52	62.40	73.74	42.06
P. emb. (UMCG+DP)	MLP	**80.49**	56.06	60.60	74.25	36.46
	kNN	*80.27*	53.47	62.79	73.32	35.04
	SVM	*80.46*	56.60	63.06	73.47	41.59
P. emb. (UMCG+DP+Wi)	MLP	*80.48*	57.43	59.42	73.22	33.20
	kNN	*80.31*	53.51	60.21	71.91	29.39
	SVM	*80.47*	56.46	61.71	72.30	37.15
All	Voter	75.21	53.98	**64.80**	73.76	**46.53**

Table 7.6: The mean, mean absolute deviation (MAD), minimum and maximum of the classification results for the different categories, excluding the baseline and voter.

Category	**Mean**	**MAD**	**Minimum**	**Maximum**
Consequence	80.05	0.71	71.55	80.49
Risk	56.05	1.20	50.90	57.83
Incident type	55.57	3.32	44.26	64.49
Function	72.03	1.69	63.54	75.45
Department	32.99	5.27	17.59	45.95

that the number of incidents remained the same, as well as ensuring that any information contained in the presence or absence of the label is taken into account. Finally, for predicting the 'risk' label, we added the 'consequence' label to the input as well. Table 7.7 shows the results for the classifiers using the input labels in combination with either a bag of words or UMCG-trained averaged paragraph embeddings, trained with

CBOW, SG and PSDVec.

Table 7.7: Classification results using 300-dimensional CBOW-, SG- and PSDVec-based averaged paragraph embeddings trained on the report database, extended with one-hot encoded feature vectors representing other labels.

		Category	
	Classifier	Consequence	Risk
Input	Baseline	**80.49**	57.87
BoW + labels	NB	71.31	52.99
	MLP	*79.73*	*60.68*
	kNN	**80.49**	56.16
	SVM	80.45	57.83
Avg. (CBOW) + labels	MLP	*80.30*	**61.01**
	kNN	*80.31*	55.43
	SVM	*80.46*	58.63
Avg. (SG) + labels	MLP	**80.49**	*60.68*
	kNN	*79.72*	58.16
	SVM	*80.46*	*60.64*
Avg. (PSDVec) + labels	MLP	**80.49**	*60.02*
	kNN	*80.46*	56.65
	SVM	*79.89*	*60.58*

Adding these labels as input did not improve the classification results of the ‘consequence’ category, with all classifiers still performing on or below the baseline. However, for the ‘risk’ category this is not the case, as the performances of all classifiers have improved, most of which now show accuracies higher than the baseline. As with the categories ‘incident type’ and ‘function’ (table 7.5), the MLP with CBOW-based averaged paragraph embeddings as input performs best here, reaching an accuracy of 61.01%, 3.14% above the baseline.

Chapter 8

Discussion and Conclusion

The following sections provide a discussion on the results of the performed experiments and on the implemented application, in the context of the questions raised in the introductory chapter of this thesis. The last section concludes this discussion and puts forward suggestions for further research.

8.1 Discussion

8.1.1 Experimental Results

We have performed multiple experiments, all testing for different facets of our final model. Let us look into each of these separately and discuss the consequences of the results.

Word Embeddings

In the introductory chapter we raised the question what technique yields the word embeddings that best represent their semantic properties. While different conceptions of 'semantic properties' are possible, in section 5.3 we defined the semantic properties relevant to this research as the extent to which words are associated with each other, which we measured with the expert rating experiment.

The results of this experiment show the CBOW architecture producing the best word embeddings, compared to the other techniques. It obtained the best performance of 0.418 for a dimensionality of 200, but did not stray far from this score with its worst performance of 0.367 for 50-dimensional embeddings. This consistency is contrasted by the inconsistency of the other techniques: SG supersedes CBOW for embeddings of 50 and 100 dimensions, but obtains a score of -0.169 for 300 dimensions. PSDVec shows something similar, albeit less extreme. GloVe and Singular do the opposite, and score poorly with lower dimensional embeddings and better for higher dimensions, although they never reach positive scores. Overall CBOW clearly outperforms the other techniques, both in maximum performance and in consistency over dimensionality. This is supported by the obtained results for embeddings trained on larger corpora, as well as by the classification results showing higher accuracies for CBOW-based embeddings.

The fact that CBOW outperforms the other techniques so clearly is curious when we compare this with other research. Mikolov et al. themselves reported SG to perform significantly better than CBOW on most tests [71]. This is supported by the results of Schwartz et al. [99] and Stratos et al. [106], who also found their own respective architectures, Symmetric and Singular, to be competitive with Word2Vec and GloVe, which is not supported by our findings either. In contrast, Baroni et al. [4] reported a slight advantage for CBOW, similar to our results. Pennington et al. on the other hand [89] did not find large differences between them, but found GloVe to perform better than both, which is neither supported by our findings, nor by most other research [60, 106, 99]. Finally, Li et al. [65] found SG to be dominant in analogy tests (they did not test CBOW), while showing superior results for PSDVec in similarity tasks. As such, few papers reported results that are in line with either our findings or those of other research.

One possible cause for these discrepancies is that the test objectives, the test methods and the way in which test datasets are composed all have a severe influence on the relative performances of the tested techniques. This is clearly shown in [65], where not only testing for analogies and similarities resulted in completely different top performers, but among the five different similarity test datasets they used, the

results differed significantly as well. For example, according to their results, GloVe outperforms Word2Vec in the WS Sim, WS Rel and MEN datasets, while losing in the Turk and Simlex datasets, even though they are all supposed to test for the same thing: word similarity. Since we did not necessarily test the techniques for analogy or similarity, but for association, it is possible that other techniques that better represent this in their embeddings came out on top, even though analogy and similarity are undoubtedly represented by association to some degree. Moreover, the way in which we built our test set differed from the other test sets as well, as our testing focused specifically on the applicative purposes we hoped to accomplish.

Second, the papers discussing their newly found techniques usually extensively test different configurations of their own method to find the optimal settings for the datasets they use, while applying the 'default' settings that are used in the original papers for the methods they compare it with. Levy et al. [60] investigated the use of hyperparameter settings and its influence on results, in the context of the dispute with respect to predict- or count-based models. They found that with proper parameter tuning, there were no global advantages to one type of methods over the other, even when only using the PPMI matrix with every row being a word vector. Since we lacked both the time and resources to carefully tune every method, this is something that we have to take into consideration. Still, one might wonder whether 'lesser' configurations justify the performance difference in our results, which was especially significant between CBOW and GloVe/Singular. Moreover, both the fact that CBOW performed well for all dimensionalities and corpora, and the fact that it served as best input for classification, at least provide some merit to the assessment of it being the best method for the purposes of the intended applications.

Third, the corpora that the methods are trained on might influence the relative results of the methods. As opposed to a large, English corpus, our dataset is Dutch, contains mostly medical terminology and consists for a large part of sentences that are not constructed according to proper grammar, but in telegram style. As a result, the ordering of key words can differ, for example placing more informative words closer to each other. This could benefit methods that are less dependent on word order, such as CBOW, while disadvantaging methods that are more so, such as Symmetric. Moreover, the way in which different languages treat the relevance of context might differ, and as such it could be the case that the suggested parameters are mostly suited for the English language.

Taking these reservations into consideration, it would be unfitting to conclude that CBOW-based word embeddings best represent the semantical properties of words in general. However, following the criteria we defined with respect to the applicative purposes of this research and for this particular dataset, we can state that out of the six compared techniques CBOW obtains the most meaningful search suggestions and provides the best basis for classification.

Paragraph Embeddings

We did not specifically test the 'quality' of the paragraph embeddings, as explained in chapter 5.3. However, we did extensively test the classifiers using paragraph embeddings with the three different types of paragraph embedding for every type of word embedding (table 7.3). As we would expect given the word embedding results, the CBOW-based paragraph embeddings perform better than their counterparts, which follow approximately the same order of performance as the word embeddings.

Curiously, the averaged word embeddings outperform both TopicVec and Par2Vec by a large margin, especially for the categories 'incident type' and 'department'. Considering that Par2Vec is basically CBOW with an added input node representing the paragraph, this was an unexpected result, since CBOW performs so well as word embedding and input for averaged paragraph embeddings. One reason for the averaged embeddings outperforming Par2Vec might be the added influence of the inverse document frequency (IDF), the weighting function we applied to explicitly rate the relevance of single words. One possible reason for the relatively poor performance of TopicVec, is that the incident reports are too diverse to represent in 300 topics, which are used to construct the paragraph embeddings. Other than these possibilities, it seems the combination of all word embeddings present in a report already contains most of the information that is relevant for its classification. As such, for these incident reports the sum is perhaps not larger than its parts.

As for the similarity measure between the paragraphs, we found the cosine distance (or similarity) to work well for finding similar words as search suggestions. Since the paragraph embeddings we ultimately ended up using in the application were averaged word embeddings, we also used this metric for finding similar incident reports (see chapter 6.2). While the cosine distance is generally used for defining the similarity between word embeddings, its value as a similarity measure within natural language processing needs further research. Drozd et al. [24] argue that purely cosine distance based offset methods are sensitive

to the idiosyncrasies of individual words, and that more factors need to be taken into account. For example, it might pay off to look at the single dimensions of word embeddings in addition to the distances between the entire embeddings, as different aspects of their semantic content can be stored in separate dimensions, depending on the embedding technique.

Classifiers

Because of the significant disparities between the classification results for the different categories and corpora, it is not possible to select a single 'winner'. Naive Bayes performs well on the latter three categories and has the advantage that it is the least computationally expensive, training in the order of mere seconds. However, it performs poorly on the first two categories compared to the baseline and the other classifiers. The MLP-paragraph embedding combination using CBOW-based UMCG-trained paragraph vectors performs best in terms of accuracies. It performs well on the first four categories, scoring the highest accuracies on the third and fourth, while being second best in the final category, albeit over 3% less accurate than NB. The SVMs using paragraph embeddings perform fairly well on all categories and for all corpora, but never reach the highest accuracies, suggesting they are the suboptimal choice.

Apart from kNN and SVM using a bag of words, the CNNs perform worst, with MGNC-CNN obtaining the highest accuracies. While we performed a global parameter sweep of the number of layers and filters and the size of the filters, the performance of the CNNs might be improved by more carefully tuning these parameters. CNNs have been shown to be sensitive to parameter changes and should therefore be tuned precisely right for the used dataset to get optimal performances [127]. This is one of their disadvantages as opposed to classifiers such as naive Bayes or MLPs, which hardly contain parameters and are far less sensitive to changes in the parameters that they do have.

The voter leveraging the output of naive Bayes, the MGNC-CNN and the MLP using paragraph embeddings, performs best for the 'incident type' and 'department' categories. Not surprisingly it scores poorly on the first two categories, because of the poor results obtained by NB there. One might have expected it to perform better in the 'function' category, since all three input classifiers perform well there. It seems that NB, being the worst performer in that category, was more 'certain' of its responses than the other two, causing the sum of probabilities of the wrong class to be greater than that of the right one in a number of cases. For possibly better results, other classifier combinations could be tested. Although this could improve the accuracies, an obvious downside to it is that it requires more training than a single classifier.

Finally, we saw that for the categories 'consequence' and 'risk' no classifier was able to surpass the baseline, except when using other labels as input, resulting in slight improvements in the 'risk' category. Since the baseline was established by always selecting the most frequently occurring class label, it appears that the classifiers were not able to extract more useful information from the data than just to apply this strategy. One possible cause for this, is that the label mapping of the two parts of the dataset, 'Infoland' and 'TPSC', resulted in inaccuracies in the data, labeling dissimilar reports with the same label. Although this might explain a portion of the classifiers' inability to supersede the baseline, it is unlikely that this is the main factor, because when we classify the Infoland and TPSC reports separately (so without any mapping of the labels), the accuracies barely supersede their respective baselines either. We believe a more influential cause is the fact that these categories are of a subjective nature. While for example the department is a non-debatable issue, the potential risk of an incident is something that reporters or analyzers will regularly assess differently. As such, patterns within reports labeled similarly are harder to detect, simply because there are less.

Corpus

While this was not originally one of the topics we focused on in the research questions, one points of interest that emerged from the classification results and the expert rating experiment is the difference in performance given different input corpora. Figure 7.2 shows that the score of the word embeddings is higher when trained on UMCG and DocPortal, which was to be expected, since more text offers more training potential. However, whereas GloVe and Singular improve, we see a slight decrease in score for CBOW and SG when trained on the Wikipedia dump. In line with these observations, the classification results seem to be either similar or slightly worse when using word embeddings trained on the Wikipedia dump.

A possible cause for this, is that the word embeddings were scored with regard to specific medical

terminology, whereas the Wikipedia dump probably contains just a small portion of medical articles. The other, non-medical texts might prove more counterproductive than productive for our test, since they can cause words to become less associated with the medical field. For example, the word 'mobility' is a word that occurs regularly in the incident reports, usually in the context of patients with reduced mobility. In the Wikipedia dump however, it occurs in many different contexts, most of which are irrelevant to our use case. As such, Wikipedia trained word embeddings might supply search suggestions that are not associated with mobility in the context of medical incident reports, which are subsequently negatively scored.

8.1.2 Application

Next to using word embeddings for making predictions about medical incident reports, we developed an application that uses word embeddings for finding patterns within the incident reports, in the form of iSearch. The software has been in use by the Central Incident Commision (CIM) for several months now, and the preliminary evaluations are positive. In spite of the fact that we raised the question of whether we could *implicitly* expand search queries to incorporate similar words, we learned that doing this *explicitly* was preferable. The user prefers to compose their own list of search terms assisted by the word embeddings, instead of being kept in the dark as to what the extension of the query is exactly. While we have not subjected the software to extensive user reviews, the general experience so far has been that queries are usually significantly more effective using the smart search function of iSearch. Both the recall (the fraction of retrieved relevant reports over the total number of relevant reports) and the precision (the fraction of relevant reports over the total number of retrieved reports) have increased quite notably, leading to more complete search sessions.

Although the CBOW-based word embeddings have proven to perform well, it would be interesting to improve them in an active manner. One possible method for doing so would be to update the embeddings through user feedback. This could be done by moving the embeddings closer to each other when positive feedback is given (a search suggestion is found relevant) and farther away from each other when given negative feedback. Such a method would be similar to vector quantization techniques, such as *learning vector quantization* [55], where the search terms can be seen as the prototypes. Further research is required to measure the effects of such learning methods on the quality of word embeddings.

The second research question regarding the applicative goals of this project focused on detecting new words in the database, with the purpose of detecting new patterns of incidents. For example, if lately a new type of incident occurs regularly, which was not encountered before, the CIM would have to be notified of this. While the smart search function is of value in finding known patterns in the data, and even in sprouting new ideas in the user's mind (search terms they would not have thought of), it is limited in the sense that it can only give information on the words it has already learned from the data. As such, for this purpose we were not able to directly use word embeddings, since training new word embeddings for every new incident report would not be viable. However, we have seen statistical language models that can extract topics from new documents without any prior training, such as TopicVec and LDA, which we subsequently applied for the detection of new trends (section 6.3). The method has not been applied enough yet for an evaluation however, and more options could be explored to optimize it. For example, recently a combination of Word2Vec and LDA was introduced, which looks promising [80].

8.2 Conclusion

This thesis has shown how context-based word embeddings can be combined with machine learning techniques to make predictions about and find similarities between medical incident reports. For this purpose, several word embedding and classification techniques were subjected to a comparative analysis, and an application was developed that helps find patterns within the incident report database of the University Medical Center Groningen.

We developed a method to test the extent to which word embeddings that are similar in vector space represent words that are associated with each other. Using this method, we found that the CBOW architecture developed by Mikolov et al. performed best out of a selection of six different word embedding methods, both in this level of associativity and as a basis for classification. We compared different methods for using word embeddings to embed entire incident reports, and found that simply averaging them in combination with an inverse document frequency weighting function yields the best results.

For the prediction of class labels, the combination of such report embeddings with multilayer perceptrons proved promising, especially for predicting the type of incident, the function of the reporter and the department of occurrence. On the other hand, predicting the risk and consequence of an incident based on the reports proved difficult, as no classifier was able to surpass the strategy of simply selecting the most frequent label. The only exception to this was when other labels were used as input, yielding slight improvements for the risk prediction of an incident report.

We applied the concepts of word and paragraph embeddings to improve searching capabilities within the database, which proved successful. The Central Incident Commission experienced the smart function of the tool we designed to drastically increase the efficiency and effectiveness of search queries, both saving them time and helping them find more complete patterns within the data. While the tool does not immediately apply classification, applications using this for incident reports could be implemented in a later stage. For example, predicting labels might alleviate work load, as reporters or analyzers would not need to fill them in manually anymore (as mentioned before, the number of possible categories exceeds a hundred). This would allow the reporter to fully focus on writing a complete description of the incident.

A number of topics has been suggested for future research. Active learning of the word embeddings through user input would be an interesting function to implement in the search tool and investigate the effects of. Different similarity measures between word or paragraph embeddings need inspection, since it is not clear whether the cosine distance is a complete enough measure of the semantical properties of an embedding. Another possibly fruitful research direction would be the possible merits of topic extraction for detecting new patterns within the incident report database.

Throughout the course of this project, several parties within the UMCG have expressed their interest in the possibilities the discussed techniques and applications could offer them. We believe that this research project provides a firm basis for answering their needs, as well as carrying out further research.

Bibliography

[1] Z. Akata, S. Reed, D. Walter, H. Lee, and B. Schiele. Evaluation of output embeddings for fine-grained image classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2927–2936, 2015.

[2] S. C. AP, S. Lauly, H. Larochelle, M. Khapra, B. Ravindran, V. C. Raykar, and A. Saha. An autoencoder approach to learning bilingual word representations. In *Advances in Neural Information Processing Systems*, pages 1853–1861, 2014.

[3] A. Bagga and B. Baldwin. Entity-based cross-document coreferencing using the vector space model. In *Proceedings of the 17th international conference on Computational linguistics-Volume 1*, pages 79–85. Association for Computational Linguistics, 1998.

[4] M. Baroni, G. Dinu, and G. Kruszewski. Don't count, predict! a systematic comparison of context-counting vs. context-predicting semantic vectors. In *ACL (1)*, pages 238–247, 2014.

[5] Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin. A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137–1155, 2003.

[6] Y. Bengio and Y. LeCun. Scaling learning algorithms towards AI. *Large-scale kernel machines*, 34(5):1–41, 2007.

[7] C. M. Bishop. *Neural networks for pattern recognition*. Oxford university press, 1995.

[8] W. Blacoe and M. Lapata. A comparison of vector-based representations for semantic composition. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 546–556. Association for Computational Linguistics, 2012.

[9] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent dirichlet allocation. *Journal of machine Learning research*, 3(Jan):993–1022, 2003.

[10] B. E. Boser, I. M. Guyon, and V. N. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 144–152. ACM, 1992.

[11] T. Botsis, M. D. Nguyen, E. J. Woo, M. Markatou, and R. Ball. Text mining for the vaccine adverse event reporting system: medical text classification using informative feature selection. *Journal of the American Medical Informatics Association*, 18(5):631–638, 2011.

[12] P. F. Brown, P. V. Desouza, R. L. Mercer, V. J. D. Pietra, and J. C. Lai. Class-based n-gram models of natural language. *Computational linguistics*, 18(4):467–479, 1992.

[13] S. F. Chen and J. 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.

[14] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[15] K. W. Church and P. Hanks. Word association norms, mutual information, and lexicography. *Computational linguistics*, 16(1):22–29, 1990.

[16] R. Collobert and J. Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*, pages 160–167. ACM, 2008.

[17] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa. Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12(Aug):2493–2537, 2011.

[18] C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.

[19] T. Cover and P. Hart. Nearest neighbor pattern classification. *IEEE transactions on information theory*, 13(1):21–27, 1967.

[20] I. Dagan, S. Marcus, and S. Markovitch. Contextual word similarity and estimation from sparse data. In *Proceedings of the 31st annual meeting on Association for Computational Linguistics*, pages 164–171. Association for Computational Linguistics, 1993.

[21] A. M. Dai, C. Olah, and Q. V. Le. Document embedding with paragraph vectors. *CoRR*, abs/1507.07998, 2015.

[22] D. Davidov and A. Rappoport. Efficient unsupervised discovery of word categories using symmetric patterns and high frequency words. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, pages 297–304. Association for Computational Linguistics, 2006.

[23] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman. Indexing by latent semantic analysis. *Journal of the American society for information science*, 41(6):391, 1990.

[24] A. Drozd, A. Gladkova, and S. Matsuoka. Word embeddings, analogies, and machine learning: Beyond king-man+ woman= queen. In *COLING*, pages 3519–3530, 2016.

[25] S. Ghosh, O. Vinyals, B. Strope, S. Roy, T. Dean, and L. Heck. Contextual LSTM (CLSTM) models for large scale NLP tasks. *CoRR*, abs/1602.06291, 2016.

[26] G. H. Golub and C. Reinsch. Singular value decomposition and least squares solutions. *Numerische mathematik*, 14(5):403–420, 1970.

[27] I. J. Good. The population frequencies of species and the estimation of population parameters. *Biometrika*, pages 237–264, 1953.

[28] J. T. Goodman. A bit of progress in language modeling. *Computer Speech & Language*, 15(4):403–434, 2001.

[29] E. Grefenstette, G. Dinu, Y. Zhang, M. Sadrzadeh, and M. Baroni. Multi-step regression learning for compositional distributional semantics. *CoRR*, abs/1301.6939, 2013.

[30] D. Guthrie, B. Allison, W. Liu, L. Guthrie, and Y. Wilks. A closer look at skip-gram modelling. In *Proceedings of the 5th international Conference on Language Resources and Evaluation (LREC-2006)*, pages 1–4, 2006.

[31] M. U. Gutmann and A. Hyvärinen. Noise-contrastive estimation of unnormalized statistical models, with applications to natural image statistics. *Journal of Machine Learning Research*, 13(Feb):307–361, 2012.

[32] A. Haghighi, P. Liang, T. Berg-Kirkpatrick, and D. Klein. Learning bilingual lexicons from monolingual corpora. In *Proceedings of Association for Computational Linguistics 08*, pages 771–779, 2008.

[33] D. R. Hardoon, S. Szedmak, and J. Shawe-Taylor. Canonical correlation analysis: An overview with application to learning methods. *Neural computation*, 16(12):2639–2664, 2004.

[34] Z. S. Harris. Distributional structure. *Word*, 10(2-3):146–162, 1954.

[35] H. He, K. Gimpel, and J. J. Lin. Multi-perspective sentence similarity modeling with convolutional neural networks. In *EMNLP*, pages 1576–1586, 2015.

[36] K. M. Hermann and P. Blunsom. Multilingual distributed representations without word alignment. *arXiv preprint arXiv:1312.6173*, 2013.

[37] F. Hill, K. Cho, S. Jean, C. Devin, and Y. Bengio. Embedding word similarity with neural machine translation. *arXiv preprint arXiv:1412.6448*, 2014.

[38] F. Hill, K. Cho, and A. Korhonen. Learning distributed representations of sentences from unlabelled data. *arXiv preprint arXiv:1602.03483*, 2016.

[39] F. Hill, R. Reichart, and A. Korhonen. Simlex-999: Evaluating semantic models with (genuine) similarity estimation. *Computational Linguistics*, 41(4):665–695, 2015.

[40] G. E. Hinton. Training products of experts by minimizing contrastive divergence. *Neural computation*, 14(8):1771–1800, 2002.

[41] G. E. Hinton, J. L. Mcclelland, and D. E. Rumelhart. Distributed representations, parallel distributed processing: explorations in the microstructure of cognition, vol. 1: foundations, 1986.

[42] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[43] H. Hotelling. Relations between two sets of variates. *Biometrika*, 28(3/4):321–377, 1936.

[44] B. Hu, Z. Lu, H. Li, and Q. Chen. Convolutional neural network architectures for matching natural language sentences. In *Advances in neural information processing systems*, pages 2042–2050, 2014.

[45] E. H. Huang, R. Socher, C. D. Manning, and A. Y. Ng. Improving word representations via global context and multiple word prototypes. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers-Volume 1*, pages 873–882. Association for Computational Linguistics, 2012.

[46] X. Huang, F. Alleva, H.-W. Hon, M.-Y. Hwang, K.-F. Lee, and R. Rosenfeld. The SPHINX-II speech recognition system: an overview. *Computer Speech & Language*, 7(2):137–148, 1993.

[47] F. Jelinek and R. Mercer. Interpolated estimation of Markov source parameters from sparse data. In *Proc. Workshop on Pattern Recognition in Practice*, 1980.

[48] S. Ji, S. V. N. Vishwanathan, N. Satish, M. J. Anderson, and P. Dubey. Blackout: Speeding up recurrent neural network language models with very large vocabularies. *CoRR*, abs/1511.06909, 2015.

[49] N. Kalchbrenner, E. Grefenstette, and P. Blunsom. A convolutional neural network for modelling sentences. *CoRR*, abs/1404.2188, 2014.

[50] S. Katz. Estimation of probabilities from sparse data for the language model component of a speech recognizer. *IEEE transactions on acoustics, speech, and signal processing*, 35(3):400–401, 1987.

[51] J. M. Keller, M. R. Gray, and J. A. Givens. A fuzzy k-nearest neighbor algorithm. *IEEE transactions on systems, man, and cybernetics*, SMC-15(4):580–585, 1985.

[52] Y. Kim. Convolutional neural networks for sentence classification. *CoRR*, abs/1408.5882, 2014.

[53] R. Kiros, Y. Zhu, R. R. Salakhutdinov, R. Zemel, R. Urtasun, A. Torralba, and S. Fidler. Skip-thought vectors. In *Advances in neural information processing systems*, pages 3294–3302, 2015.

[54] H. C. Koh and G. Tan. Data mining applications in healthcare. *Journal of healthcare information management*, 19(2):65, 2011.

[55] T. Kohonen. Learning vector quantization. In *Self-Organizing Maps*, pages 175–189. Springer, 1995.

[56] V. Krishnaiah, D. G. Narsimha, and D. N. S. Chandra. Diagnosis of lung cancer prediction system using data mining classification techniques. *International Journal of Computer Science and Information Technologies*, 4(1):39–45, 2013.

[57] Q. V. Le and T. Mikolov. Distributed representations of sentences and documents. In *ICML*, volume 14, pages 1188–1196, 2014.

[58] Y. LeCun, Y. Bengio, et al. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995, 1995.

[59] D. L. Lee, H. Chuang, and K. Seamons. Document ranking and the vector-space model. *IEEE software*, 14(2):67–75, 1997.

[60] O. Levy and Y. Goldberg. Neural word embedding as implicit matrix factorization. In *Advances in neural information processing systems*, pages 2177–2185, 2014.

[61] O. Levy, Y. Goldberg, and I. Dagan. Improving distributional similarity with lessons learned from word embeddings. *Transactions of the Association for Computational Linguistics*, 3:211–225, 2015.

[62] J. Li, M. Luong, and D. Jurafsky. A hierarchical neural autoencoder for paragraphs and documents. *CoRR*, abs/1506.01057, 2015.

[63] S. Li and T. Chua. Document visualization using topic clouds. *CoRR*, abs/1702.01520, 2017.

[64] S. Li, T.-S. Chua, J. Zhu, and C. Miao. Generative topic embedding: a continuous representation of documents (extended version with proofs). *arXiv preprint arXiv:1606.02979*, 2016.

[65] S. Li, J. Zhu, and C. Miao. A generative word embedding model and its low rank positive semidefinite solution. *arXiv preprint arXiv:1508.03826*, 2015.

[66] S. Li, J. Zhu, and C. Miao. PSDVec: A toolbox for incremental and scalable word embedding. *Neurocomputing*, 237:405–409, 2017.

[67] J. B. Lovins. Development of a stemming algorithm. *Mechanical Translation and Computational Linguistics*, 11(1, 2), 1968.

[68] K. Lund and C. Burgess. Producing high-dimensional semantic spaces from lexical co-occurrence. *Behavior Research Methods, Instruments, & Computers*, 28(2):203–208, 1996.

[69] L. v. d. Maaten and G. Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9(Nov):2579–2605, 2008.

[70] T. Mikolov. Language modeling for speech recognition in czech. Master's thesis, Brno University of Technology, 2007.

[71] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781, 2013.

[72] T. Mikolov, A. Deoras, D. Povey, L. Burget, and J. Černockỳ. Strategies for training large scale neural network language models. In *Automatic Speech Recognition and Understanding (ASRU), 2011 IEEE Workshop on*, pages 196–201, 2011.

[73] T. Mikolov, M. Karafiát, L. Burget, J. Cernockỳ, and S. Khudanpur. Recurrent neural network based language model. In *Interspeech*, volume 2, page 3, 2010.

[74] T. Mikolov, J. Kopecky, L. Burget, and O. Glembek. Neural network based language models for highly inflective languages. In *Acoustics, Speech and Signal Processing, 2009. ICASSP 2009. IEEE International Conference on*, pages 4725–4728, 2009.

[75] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 3111–3119. Curran Associates, Inc., 2013.

[76] T. Mikolov, W.-t. Yih, and G. Zweig. Linguistic regularities in continuous space word representations. In *HLT-NAACL*, volume 13, pages 746–751, 2013.

[77] G. A. Miller and W. G. Charles. Contextual correlates of semantic similarity. *Language and cognitive processes*, 6(1):1–28, 1991.

[78] A. Mnih and G. E. Hinton. A scalable hierarchical distributed language model. In *Advances in neural information processing systems*, pages 1081–1088, 2009.

[79] A. Mnih and Y. W. Teh. A fast and simple algorithm for training neural probabilistic language models. *arXiv preprint arXiv:1206.6426*, 2012.

[80] C. E. Moody. Mixing dirichlet topic models and word embeddings to make LDA2Vec. *arXiv preprint arXiv:1605.02019*, 2016.

[81] F. Morin and Y. Bengio. Hierarchical probabilistic neural network language model. In *AISTATS*, volume 5, pages 246–252, 2005.

[82] T. Muneeb, S. K. Sahu, and A. Anand. Evaluating distributed word representations for capturing semantics of biomedical concepts. *Proceedings of ACL-IJCNLP*, pages 158–163, 2015.

[83] H. J. Murff, A. J. Forster, J. F. Peterson, J. M. Fiskio, H. L. Heiman, and D. W. Bates. Electronically screening discharge summaries for adverse medical events. *Journal of the American Medical Informatics Association*, 10(4):339–350, 2003.

[84] V. Nair and G. E. Hinton. Rectified linear units improve restricted Boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[85] H. Ney, U. Essen, and R. Kneser. On structuring probabilistic dependences in stochastic language modelling. *Computer Speech & Language*, 8(1):1–38, 1994.

[86] T. D. Nielsen and F. V. Jensen. *Bayesian networks and decision graphs*. Springer Science & Business Media, 2009.

[87] H. Palangi, L. Deng, Y. Shen, J. Gao, X. He, J. Chen, X. Song, and R. Ward. Deep sentence embedding using long short-term memory networks: Analysis and application to information retrieval. *IEEE/ACM Transactions on Audio, Speech and Language Processing (TASLP)*, 24(4):694–707, 2016.

[88] J. Pelemans, N. Shazeer, and C. Chelba. Sparse non-negative matrix language modeling. *Transactions of the Association for Computational Linguistics*, 4:329–342, 2016.

[89] J. Pennington, R. Socher, and C. D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.

[90] R. Pickhardt, T. Gottron, M. Körner, P. G. Wagner, T. Speicher, and S. Staab. A generalized language model as the combination of skipped n-grams and modified Kneser-Ney smoothing. *CoRR*, abs/1404.3377, 2014.

[91] J. K. Pritchard, M. Stephens, and P. Donnelly. Inference of population structure using multilocus genotype data. *Genetics*, 155(2):945–959, 2000.

[92] V. V. Raghavan and S. M. Wong. A critical analysis of vector space model for information retrieval. *Journal of the American Society for information Science*, 37(5):279, 1986.

[93] S. Robertson. Understanding inverse document frequency: on theoretical arguments for IDF. *Journal of documentation*, 60(5):503–520, 2004.

[94] X. Rong. Word2Vec parameter learning explained. *CoRR*, abs/1411.2738, 2014.

[95] R. Rosenfeld. *Adaptive statistical language modeling: A maximum entropy approach*. PhD thesis, Carnegie Mellon University, 1994.

[96] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.

[97] G. Salton, A. Wong, and C. S. Yang. A vector space model for automatic indexing. *Commun. ACM*, 18(11):613–620, Nov. 1975.

[98] R. Schwartz, R. Reichart, and A. Rappoport. Minimally supervised classification to semantic categories using automatically acquired symmetric patterns. In *COLING*, pages 1612–1623, 2014.

[99] R. Schwartz, R. Reichart, and A. Rappoport. Symmetric pattern based word embeddings for improved word similarity prediction. In *Proceedings of CoNLL*, 2015.

[100] C. Silva and B. Ribeiro. The importance of stop word removal on recall values in text categorization. In *Neural Networks, 2003. Proceedings of the International Joint Conference on*, volume 3, pages 1661–1666. IEEE, 2003.

[101] M. Singh and D. Klakow. Comparing RNNs and log-linear interpolation of improved skip-model on four Babel languages: Cantonese, Pashto, Tagalog, Turkish. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 8416–8420, 2013.

[102] R. Socher, C. C. Lin, C. Manning, and A. Y. Ng. Parsing natural scenes and natural language with recursive neural networks. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 129–136, 2011.

[103] R. Socher, A. Perelygin, J. Y. Wu, J. Chuang, C. D. Manning, A. Y. Ng, C. Potts, et al. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the conference on empirical methods in natural language processing (EMNLP)*, pages 1631–1642, 2013.

[104] R. Speer and J. Chin. An ensemble method to produce high-quality word embeddings. *CoRR*, abs/1604.01692, 2016.

[105] N. Srebro, T. Jaakkola, et al. Weighted low-rank approximations. In *ICML-03*, pages 720–727, 2003.

[106] K. Stratos, M. Collins, and D. J. Hsu. Model-based word embeddings from decompositions of count matrices. In *ACL (1)*, pages 1282–1291, 2015.

[107] K. Stratos, D.-k. Kim, M. Collins, and D. J. Hsu. A spectral algorithm for learning class-based n-gram models of natural language. In *UAI*, pages 762–771, 2014.

[108] M. Sundermeyer, R. Schlüter, and H. Ney. LSTM neural networks for language modeling. In *Interspeech*, pages 194–197, 2012.

[109] K. S. Tai, R. Socher, and C. D. Manning. Improved semantic representations from tree-structured long short-term memory networks. *arXiv preprint arXiv:1503.00075*, 2015.

[110] D. Tang, B. Qin, and T. Liu. Document modeling with gated recurrent neural network for sentiment classification. In *EMNLP*, pages 1422–1432, 2015.

[111] D. Tang, F. Wei, N. Yang, M. Zhou, T. Liu, and B. Qin. Learning sentiment-specific word embedding for twitter sentiment classification. In *ACL (1)*, pages 1555–1565, 2014.

[112] S. Tsumoto and S. Hirano. Risk mining in medicine: Application of data mining to medical risk management. *Fundamenta Informaticae*, 98(1):107–121, 2010.

[113] V. N. Vapnik and S. Kotz. *Estimation of dependences based on empirical data*, volume 40. Springer-Verlag New York, 1982.

[114] I. Vulić, W. De Smet, and M.-F. Moens. Identifying word translations from comparable corpora using latent topic models. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: short papers-Volume 2*, pages 479–484. Association for Computational Linguistics, 2011.

[115] S. Wang and C. D. Manning. Baselines and bigrams: Simple, good sentiment and topic classification. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Short Papers-Volume 2*, pages 90–94. Association for Computational Linguistics, 2012.

[116] S. I. Wang and C. D. Manning. Fast dropout training. In *ICML (2)*, pages 118–126, 2013.

[117] P. L. Williams and R. D. Beer. Nonnegative decomposition of multivariate information. *arXiv preprint arXiv:1004.2515*, 2010.

[118] L. Wolf, Y. Hanani, K. Bar, and N. Dershowitz. Joint Word2Vec networks for bilingual semantic representations. *International Journal of Computational Linguistics and Applications*, 5(1):27–44, 2014.

[119] B. Xue, C. Fu, and Z. Shaobin. A study on sentiment computing and classification of Sina Weibo with Word2Vec. In *Big Data (BigData Congress), 2014 IEEE International Congress on*, pages 358–363, 2014.

[120] W. Yin and H. Schütze. An exploration of embeddings for generalized phrases. In *ACL (Student Research Workshop)*, pages 41–47, 2014.

[121] W. Yin and H. Schütze. Multichannel variable-size convolution for sentence classification. *arXiv preprint arXiv:1603.04513*, 2016.

[122] C. Zhai and J. Lafferty. A study of smoothing methods for language models applied to information retrieval. *ACM Transactions on Information Systems (TOIS)*, 22(2):179–214, 2004.

[123] D. Zhang, H. Xu, Z. Su, and Y. Xu. Chinese comments sentiment classification based on Word2Vec and SVM^{perf}. *Expert Systems with Applications*, 42(4):1857–1863, 2015.

[124] X. Zhang and Y. LeCun. Text understanding from scratch. *arXiv preprint arXiv:1502.01710*, 2015.

[125] X. Zhang, J. J. Zhao, and Y. LeCun. Character-level convolutional networks for text classification. *CoRR*, abs/1509.01626, 2015.

[126] Y. Zhang, S. Roller, and B. Wallace. MGNC-CNN: A simple approach to exploiting multiple word embeddings for sentence classification. *arXiv preprint arXiv:1603.00968*, 2016.

[127] Y. Zhang and B. Wallace. A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification. *arXiv preprint arXiv:1510.03820*, 2016.