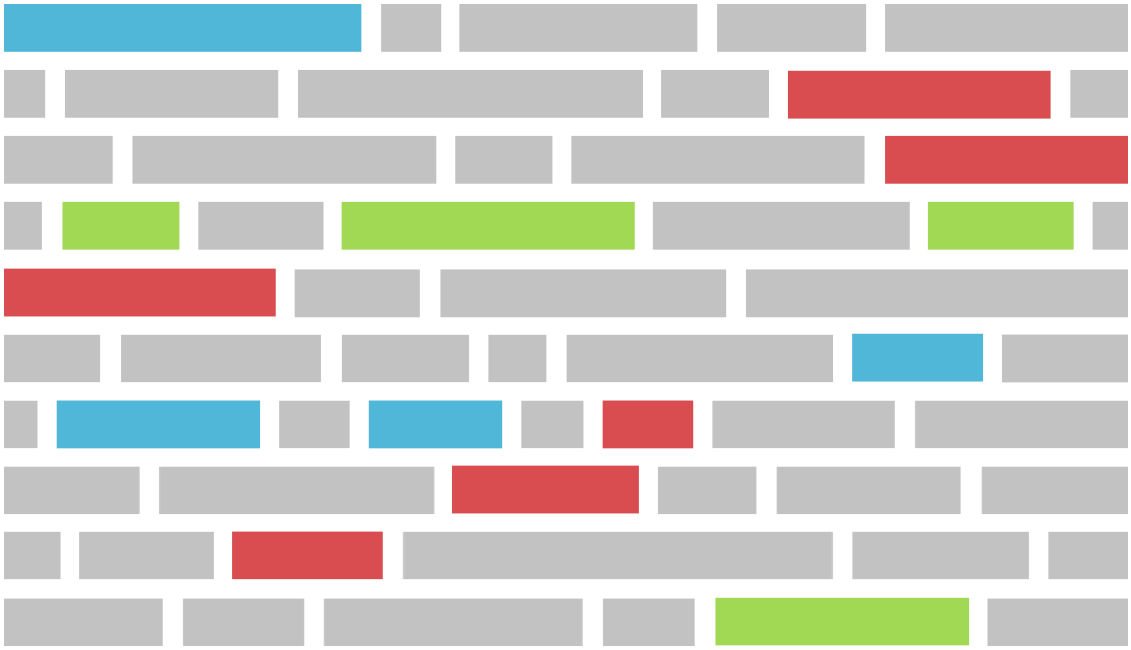


Semi-Supervised Named Entity Recognition of Medical Entities in Swedish

SIMON ALMGREN
SEAN PAVLOV

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2016

Master's thesis 2016

Semi-Supervised Named Entity Recognition of Medical Entities in Swedish

SIMON ALMGREN
SEAN PAVLOV

CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF
GOTHENBURG

Department of Computer Science and Engineering
CSE LAB Research Group
Chalmers University of Technology
Gothenburg, Sweden 2016

Semi-Supervised Named Entity Recognition of Medical Entities in Swedish
SIMON ALMGREN
SEAN PAVLOV

Supervisor: Olof Mogren, Department of Computer Science and Engineering
Examiner: Peter Damaschke, Department of Computer Science and Engineering

Master's Thesis 2016
Department of Computer Science Engineering
CSE LAB Research Group
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Visualization of named entity recognition showing categorization of entities in a document into pre-defined color-coded categories.

Typeset in LaTeX
Gothenburg, Sweden 2015

Semi-Supervised Named Entity Recognition of Medical Entities in Swedish
SIMON ALMGREN
SEAN PAVLOV
Department of Computer Science Engineering
Chalmers University of Technology

Abstract

A big opportunity within today's society is the vast amounts of data generated each day. Especially within the health-care sector where a lot of journals are written daily and needs to be processed in some way to properly identify the content within. Enter the field of Named Entity Recognition (NER), where text is analyzed to locate and classify entities into predefined classes; in our case *Disorder & Finding*, *Pharmaceutical Drug* and *Body Structure*. With a model that can do this with a great accuracy, analyzing medical texts could be automated and strain could be removed from people having to read through them manually. Since journals and other medical text often are very sensitive and should be handled with care due to privacy, a method for constructing these models without the need for real annotated journals would be a big step in the right direction.

During this thesis we have implemented two models for solving the problem of NER for medical texts in Swedish. Both models were created from lists of seed-terms, which consist of words and phrases found in medical taxonomies which we assume belong to one of the three categories. Training data were extracted from the health-care magazine Läkartidningen as well as a subset of Swedish Wikipedia. The first model implemented is based on the work of Zhang and Elhadad [23] where a vector representation is calculated for the possible words and compared against vectors calculated the same way for the different categories. The results of our implementation is on par with the results given by Zhang and Elhadad which suggests that this method works as well for Swedish as it does for English.

The second model implemented is based on recurrent neural networks and is built from the same seed-terms as the first model but instead of using only vector-calculations for classification the network is trained to automatically classify words on character-basis, reading the text both forwards and backwards at the same time.

Solving the problem of NER using only unsupervised methods is inherently hard and techniques for solving the problem are not quite there yet. However, by just improving them bit by bit will in the end lead to great results.

Keywords: named entity recognition, ner, unsupervised, semi-supervised, natural language processing, nlp, swedish, medical.

Acknowledgements

We would like to thank our supervisors Olof Mogren and Fredrik Axelsson for constructive criticism as well as insightful discussions during the meetings we had. We would also like to thank our examiner Peter Damaschke for giving us feedback when we needed it and for his enthusiasm for the subject we chose.

Furthermore, we would like to thank Henrik Alburg for always giving his opinions and challenging us in our methods as well as giving us tips and clarifications. We would also like to thank our family and friends for continuous support during this challenging time.

We would also like to thank Hercules Dalianis for providing evaluation data and support to run the models on the data.

Simon Almgren & Sean Pavlov, Gothenburg, June 2016

Contents

List of Figures

List of Tables

1

Introduction

In today's society the use of digital information is increasing rapidly. According to Statistiska centralbyrån (SCB), which is a Swedish administrative agency providing various statistics, the number of people in Sweden who have access to a computer in their household were 25.6% in 1994-95. In 2007, the number was 83.2%. As businesses store their information digitally, it becomes easier to access and more securely saved with backups stored remotely. The number of Internet users has increased from over 35 million in 1995 to 2.8 billion users in 2014 [11]. This has led to a great increase in accessibility of information retrieval, which in turn have increased the amount of digital information. Information has increased to such extent that one person cannot go through all information, not even within their own business or field. However, all information may not be useful for one person, but some information is useful for everyone. It is important that there are efficient ways to find the information that one searches for in order to keep up with the increasing information flow.

Health care in Europe is facing a situation where the waiting time is steadily increasing [20]. This is because they are short on staff and the number of patients with complex diagnoses increases, but also because the daily tasks they are faced with takes a long time to do. Such tasks could be to search for the right electronic health care record, find similar cases of the same symptoms or to find related published medical literature. If all data could be accessed from a single system and one could search for multiple symptoms or illnesses at once, the health care providers would be able to find the right information quickly and the probability to give a correct diagnose may increase. In order to have all information in one system, there must be a connection between the different kinds of information. Electronic health care records and published medical literature consist of text, using medical terms. Therefore, it is a good approach to enrich these words in the documents with information received from medical taxonomy. Natural Language Processing (NLP) is a field within Computer Science in which natural language written by humans are processed by a computer for data-extraction or other purposes. One task within the field is called Named Entity Recognition (NER) which is to find the entities, in this case the medical terms, in the documents and to categorize them into a predefined category. The task of Named Entity Recognition is what we are going to tackle in this thesis.

1.1 Purpose

The aim of the thesis is to explore the capabilities of two different methods both focusing on the same task, Named Entity Recognition for medical documents in Swedish. This task will be to take medical documents in Swedish as input and find and classify all entities within them into one of the predefined categories. The categories that are used in this thesis are:

- Disorder & Finding
- Pharmaceutical Drug
- Body Structure

1.2 Research questions

Two models are going to be explored: a Word-Vector model and a Recurrent Neural Network model (RNN model) which will both be explained in detail later on. The first model was originally proposed by Zhang and Elhadad [23] for the medical domain in English. The question that we asked was if an implementation in Swedish for the medical domain has the same capabilities as the original implementation. The second model aims to test if a supervised model could perform well by generating the required training data from the same external source as the first model. The scientific questions that we ask are thus:

- Is it possible to use a Word-Vector model for Named Entity Recognition of medical entities in Swedish?
- How well does a supervised machine-learning method work with automatically generated training data?

As mentioned above information flow is increasing and unlabeled and unstructured data are abundant. Especially in the medical domain where large amounts text are created every day some automatic way to make sense of all the data is needed. Named Entity Recognition will not solve the problem in its entirety, but a big part of the process.

1.3 Scope

Within the scope of this thesis we will investigate two different models, one built upon word-vector calculations and one based on a bi-directional recurrent neural network. Implement them, test them and tweak parameters as well as evaluate them in the end.

1.4 Outline

The thesis aim to investigate two very different models to handle the task of Named Entity Recognition. To be able to understand how the models work, prior knowledge about the methods is required. Chapter 2 describes how the used methods work to be able to understand how they are applied in the models. The chapter also touch

on how previous work on Named Entity Recognition has been approached. The thesis will go further on to describe in detail how the models work in Chapter 3. Beginning by describing the Word-Vector model, continuing with the Recurrent Neural Network model, then describing how measurement of the result is done and lastly about what limitations are taken in the thesis. Chapter 4 is going into detail about how the experiments are carried out. The chapter explains what kind of technique is used to perform each step and what all hyper parameters were set to. Chapter 5 presents the overall results of the models, individual steps and the differences when adding certain features to the model. It also shows a comparison of the results between the different models. Continuing to Chapter 6 with discussion about the result and other thoughts that needed to be addressed. Chapter 7 wraps up the thesis with a conclusion. Lastly we address what could be done in the future to improve the models.

2
Background

Named Entity Recognition is a task within the field of Natural Language Processing. The task is to recognize entities in a given context and classifying them into predefined classes. The context is often a document and entities consists of one or multiple words that fall within the definitions of one of the predefined classes. The most common categories used in NER-tasks are *person*, *location* and *organization*. There are several approaches available for recognizing and classifying entities; supervised, semi-supervised and unsupervised methods. Supervised methods rely on input with the correct output attached to it in contrast to unsupervised methods which have no information of the underlying distribution of the input data. Semi-supervised methods are somewhere in between, with the algorithm usually depending on a small set of labeled data or other knowledge specific to the domain. Supervised methods for the NER task takes words annotated with entity classes and tries to classify new entities extrapolating from the information that it has learned from the annotated data. Semi-supervised methods often use its limited resources of data to deduce information from unlabeled data that could further be used to solve the task.

2.1 Artificial Neural Networks

Artificial Neural Networks (ANNs) is a set of models inspired by the anatomy of the human brain with its neurons and synaptic connections. These networks are used for computational tasks where each neuron or node in the network performs very simple computations, but the network in its entirety can manage very complex tasks. The exact computations differs depending on the implementation, but in the simplest case each neuron only performs a weighted sum of all its inputs and puts it through an activation function, for example the hyperbolic tangent function. One important property of the first neural networks was that the neurons were arranged in layers with no connections within a layer and only "forward" in the network, hence they were called *feed-forward neural networks*.

2.1.1 Recurrent Neural Networks

The forward-propagating networks were later challenged by Jeff Elman in 1990 when he laid the ground for *Recurrent Neural Networks (RNNs)*. With RNNs, backward connections are allowed which gives the network a sense of time and memory. The

learning algorithms had to be modified slightly, but with modifications these networks could learn temporal information from the input.

2.1.2 Long Short-Term Memory

In 1997, Hochreiter and Schmidhuber revolutionized the field of Recurrent Neural Networks when they introduced a concept called Long Short-Term Memory (LSTM) [7]. They added a sense of memory to the network which not only lasted a few iterations but instead the network could remember information even further back in the training cycle. They kept all the features of ordinary RNNs with recurrent connections, but added some additional features making the nodes able to just pass along the old value unchanged and therefore increasing the memory-capacity of the network. LSTM-networks are very powerful and are used for a large variety of purposes today including; speech recognition, robot control, time-series predictions and music composition just to name a few.

Furthermore, LSTM-networks has also proven to be useful within the field of NLP. Words written in a text are often referred to much later in the text and for a network to be able to remember that and make that prediction long-term memory is required. A good example is a letter from Alice to Bob. In the beginning of the letter Alice writes about her vacation in France and a lot further along she writes that she also visited the capital city but does not mention France at all. For a human this connection is obvious due to the context but for a computer to make that connection it needs a good representation of the text in its entirety or memory to be able to remember the very beginning of the message.

2.2 Related Work

There have been various attempts to solve the task of Named Entity Recognition, using supervised and semi-supervised models, but also some unsupervised models. The result of them have been varying, but generally the supervised implementations are performing better as can be seen below.

2.2.1 Supervised Named Entity Recognition

Supervised NER has been thoroughly explored in the past and a lot of different methods have been used. *Conditional Random Fields* has been proven very successful. Finkel et al. used this concept together with Gibbs sampling and got good results [5] on two NER corpora used in shared tasks, CoNLL and CMU Seminar Announcements corpus. The predefined categories in CoNLL is person, location, organization and miscellaneous and in CMU Seminar Announcements it is speaker, location, start time and end time. State-of-the-art in the medical domain have been achieved by Wang & Patrick [21]. They used a combination of CRF, Support Vector Machines (SVM) and Maximum Entropy (ME) to recognize and classify entities. The state-of-the-art in Swedish for the medical domain also used CRF and is done

by Skeppstedt, Kvist, Nilsson and Dalianis. [16]. There is not much annotated data for the NER task, especially in the medical domain. There is also the ethical and privacy aspect regarding medical journals. In most countries it is regarded as classified information and can therefore not be obtained so easily. The journals must be anonymized and even after that there are strict ethical rules to be followed about how to handle them.

2.2.2 Semi-Supervised Named Entity Recognition

Semi-supervised methods often rely on a small set data to be able to infer from after training. The data can, in the case of NER, for example be words that is known to belong to one of the categories, assured by some expert within the field. Liao and Veeramachaneni [10] start off with a small set of manually annotated examples and in each iteration tries to enlarge this set with new entities which have high enough confidence of being correctly classified. With sufficient amount of time, running the algorithm the set would have become large enough to include most of the entities wanted from the input text.

Zhang and Elhadad takes another approach in which they use lists of seed-terms for each of the different categories and from those lists try to apply the information to new examples. To be able to do that they use word-vectors and calculate a form of average vector for each of the categories and use that to compare against with vectors for the encountered words. The vectors are calculated from a bag-of-words model. Since the medical domain was the target, seed-term lists for *Disorder & Finding*, *Pharmaceutical Drug* and *Body Structure* were extracted from extensive medical taxonomies using the fact that they have hierarchical structure and that high level concepts for the categories exist.

3

Methodology

This section describes our two approaches to the NER task for medical entities in Swedish. The first method is heavily influenced by Zhang and Elhadad [23] and uses word-vectors calculated by a bag-of-words model to compare each word-vector to a generalized category vector. The second method is based on a bi-directional implementation of a recurrent neural network and using the same lists of seed-terms used by the first method to collect training examples, then trained to correctly classify all entities in some input document.

3.1 Word-Vector Model

This model is inspired by a model originally implemented by Zhang and Elhadad [23]. They developed a system for semi-supervised NER with clinical and biological texts in English as input documents. Since we want to annotate medical journals in Swedish, clinical texts matches our domain best. The method consists of a number of steps and a preprocessing stage. The preprocessing needed is tokenization, sentence splitting, part-of-speech tagging followed by noun phrase chunking which all will be described in more detail below. The idea behind the method is that noun phrases are likely to be entities and chunking is a more efficient way of finding candidates of entities than doing a full parsing on the input documents. After pre-processing, there are three steps in the main algorithm: seed-term collection, boundary detection and entity classification.

3.1.1 Tokenization and Sentence Splitting

The incoming documents are pre-processed with a tokenizer and sentence splitter, which will first tag tokens and then the sentences. Tokens are anything from ordinary words, special characters to spaces. This stage serves as input to the next stage, which is part-of-speech tagging.

3.1.2 Part-Of-Speech Tagging

After sentence splitting and tokenization of the input documents, they are tagged with Part-Of-Speech tags (POS-tags). POS-tagging is to let each token correspond to a particular part-of-speech, which is a tagging of their grammatical meaning.

3.1.3 Noun Phrase Chunking

Given the document with POS-tags, noun phrase chunking is required before the main part of the algorithm can take place. Noun Phrase chunking (NP chunking) is the task to divide the text into chunks and find noun-phrases amongst them. A noun phrase is a noun in the text including all of its modifiers. In the sentence *"This is a blue car"*, the word car is a noun while blue is describing what the car is and therefore "a blue car" is the noun phrase.

3.1.3.1 Inverse Document Frequency

For every noun phrase *np*, we calculate the average Inverse Document Frequency (IDF) over each word w within it. If the resulting average is below a certain threshold T, we then disregard it as a possible entity candidate. IDF is calculated as:

$$\text{IDF(w, D)} = \log \frac{|\text{sentences}|}{|\{\text{s} : \text{w} \in s, w \in D\}|} \tag{3.1}$$

where $|sentences|$ is the number of sentences in the input document and $|\{\text{s} : \text{w} \in s, w \in D\}|$ is the number of sentences the given word is contained in in the input document D. The IDF filter will filter out noun phrases that are common in the document since they are mostly general phrases that does not belong to any of the categories. A good example would for example be "the patient" which probably is very common in the texts and therefore will have a very low value for its IDF and thus will be filtered out.

IDF for a noun phrase is calculated as:

$$\textit{NP-IDF(np)} = \frac{\sum_{i=1}^{|\text{np}|} IDF(np(i))}{|\text{np}|} \tag{3.2}$$

and the filtering is done as following:

$$\text{IDF-filter(np)} = \begin{cases} \text{included}, & \text{if NP-IDF(np)} > T \\ \text{dicarded}, & \text{otherwise} \end{cases} \tag{3.3}$$

3.1.4 Seed Knowledge

The algorithm first collects *seed knowledge* which are terms assumed to belong to one of the per-defined categories. This can be done in several different ways by either consulting an expert in the domain or using lists of terms collected in some way. The main concern is to be somewhat sure that the terms really do belong to the specified category.

3.1.5 Entity Classification

Having a set of examples for each of the different categories, classifying the encountered entities in the document will be done with a measurement called Cosine

similarity which measures how similar two vectors are. To be able to compute this, every term needs to be represented as a vector. The categories will have a vector representation calculated in the same way as the terms. If the vocabulary consists of V words, all possible unigrams, each vector s^t will be V-dimensional, that is $s^t =< s_1^t, s_2^t, s_3^t, ..., s_V^t >$. All the values in the vectors are calculated according to this equation:

$$s_i^t = w_i * f(v_i, t) * IDF(v_i, D), i = 1..V \tag{3.4}$$

In the above equation w_i is a weight parameter which is different depending on if the word occurs in the context or as an internal word. A context word is a neighboring word to the seed-term and an internal word is a word in the seed-term. The weight parameter is different because it is assumed that internal words are more important than context words. The functions f(t,d) and IDF(v,D) together is know as *term frequency-inverse document frequency* which is a measure of how important a word is to a document in a corpus. The more common a word is in the document, the lower the IDF value will be. The value is then multiplied by how many times it occurs in total in the context or internal words depending on which one is calculated. The t refers to the internal words of the term which is important because terms can consist of multiple words.

The cosine-similarity mentioned earlier is calculated by dividing the dot-product of two vectors v and w by the product of both of their magnitudes as follows:

$$cosinesimilarity(v, w) = \frac{v * w}{||v|| * ||w||} = \frac{\sum_{i=1}^{V} v_i * w_i}{\sqrt{\sum_{i=1}^{V} v_i^2} * \sqrt{\sum_{i=1}^{V} w_i^2}} \tag{3.5}$$

This gives a value between 0 and 1 with 0 meaning completely different and 1 being exactly the same. Worth noting here is that this calculation differs from the method implemented by Zhang and Elhadad where they used a modified version of cosine similarity.

The vectors for each category will just be the average of all vectors calculated from all the seed-terms corresponding to that category. This means that the vector for each of the different categories will be the middle-point of the entire cluster of seed-term vectors. Classification will therefore consist of calculating the similarity between the term in question and the different vectors for the categories and choosing the one with the value closest to 1 with respect to the following thresholds:

- Classification Threshold
- Similarity Threshold

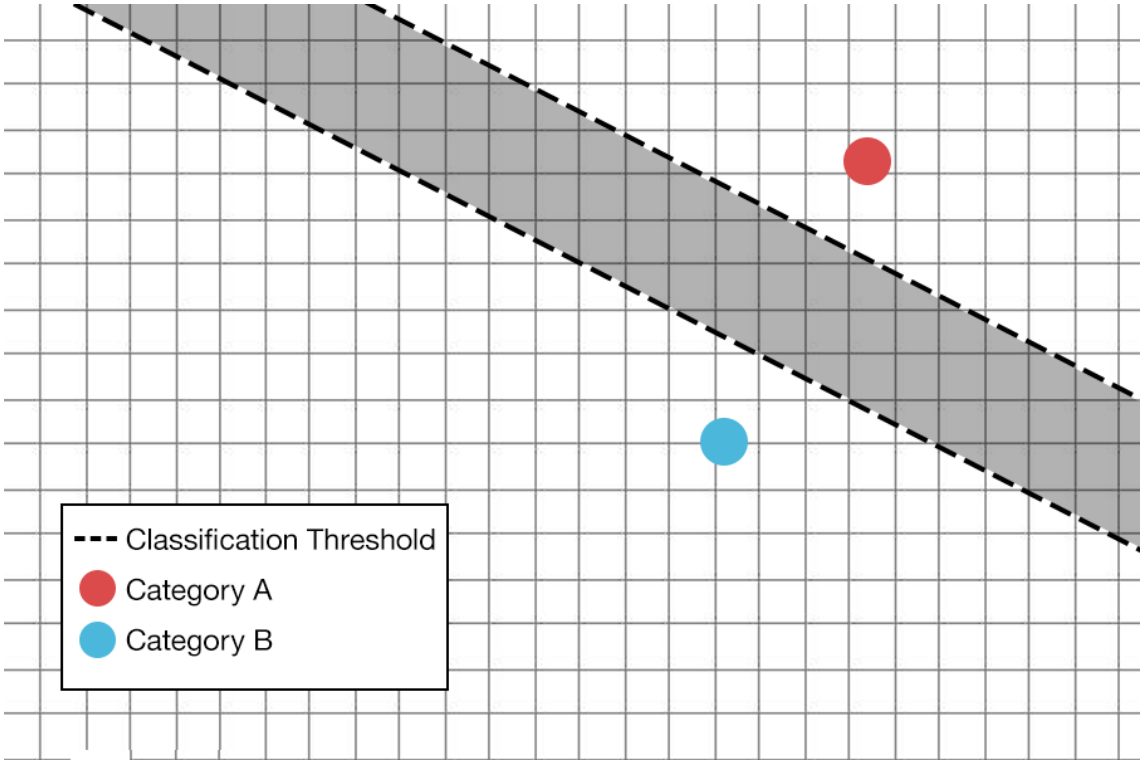
Classification Threshold will make sure that entities that have too similar similarity values for each of the three categories are discarded. That is, given an entity vector V_e, S is the set of similarity values between vector V_e and every category $c \in C$ where C is the set of all categories. The constraint is formulated as following:

$$\forall c \in C, c \neq C_{MAX} : \frac{S_c}{S_{MAX}} < T \tag{3.6}$$

where S_{MAX} is the maximum similarity value in S, C_{MAX} is the category for S_{MAX}, S_c is the similarity value for category c and T is the classification threshold. In

Figure 3.1, the classification threshold between two categories in a 2D plane is illustrated.

Figure 3.1: A visualization of the classification threshold between two categories on a 2D plane. An entity in the grey area will remain unclassified.



Similarity Threshold will make sure that any entity e that have a maximum similarity value lower than a certain threshold is discarded. This is to filter out entities that in general are not similar to any of the categories.

3.2 Recurrent Neural Network Model

This method loosely base its information on the same concept as the Word-Vector model, by using inherent knowledge already in taxonomies to solve the problem. Using the knowledge, it will be able to train the model to recognize named entities from the categories *Disorder & Finding*, *Pharmaceutical Drug* and *Body Structure*. Instead of using vectors for terms and comparing them, a recurrent neural network will be trained to read through text and will with the output predict the category that the different words or phrase in the text-stream falls into. While it is a supervised method, the training data will be generated with the information from the taxonomies which will make it a semi-supervised approach.

3.2.1 Preprocessing

To be able to train the network, a stream of characters with their respective label is needed. Given a large set of documents in the domain, the documents are scanned for occurrences of the seed-terms in the seed-term list to automatically generate an annotated data set. Training examples will then be extracted from this annotated text with examples of a predetermined sequence length.

3.2.2 Training

Neural networks are usually trained with an algorithm called *backpropagation* which literally means backwards propagation of errors. It is often used together with gradient decent to train a neural network in the most efficient way. Gradient decent is a method to solve optimization problems with respect to an objective function. What the algorithm does is to compute the gradient of the objective function with respect to all of the weights in the network and updates them in the direction of the steepest gradient. Two things makes this possible: With the chain-rule it is possible to compute the derivatives with respect to the weights in the beginning of the network and with help of the already computed derivatives in the later stages of the network. The second important fact is that each neuron in the network uses an activation-function that is differentiable to be able to calculate the derivatives.

The objective function is needed in order to know how much error the network does. The most common one to use is cross-entropy error, sometimes referred to as the negative log-likelihood, which essentially is a measure of how similar two probability distributions are. With f(x) as the probability-distribution over classes computed by the neural network, y as the correct class and summing over all possible classes c, the objective function takes the form:

$$L(f(x), y) = -\sum_{c} 1_{(y=c)} \log f(x)_c = -\log f(x)_y \tag{3.7}$$

Optimizing the network is then to minimize the negative logarithm of the output of the network corresponding to the correct label y. In distribution terms the similarity between the distribution output by our network is compared with the distribution of 1 for the correct label and 0 for the incorrect ones.

3.2.2.1 Backpropagation

Backpropagation is usually used for feed-forward neural networks. It will however need to be modified when adding recurrent connections. Since the only major difference is the recurrent connections, the network can be unrolled and thought of as a very deep feed-forward network with each time step represented as a layer. The error-signal for a character in the sequence then propagates backwards through all the previous unrolled steps to update the weights as you would with a normal feed-forward neural network. This means that backpropagation through time is just normal backpropagation but since connections are shared between layers, or in this case unrolled networks, it needs to sum up the gradient for each previous time step.

So when the weights are updated they are updated according to the sum of all gradients in the previous steps.

3.2.2.2 Vanishing gradient

Vanishing gradient is occuring to very deep feed-forward neural networks, but is primarily a problem for recurrent networks due to their depth. RNNs can be thought of as a very deep feed-forward neural network with a layer corresponding to each time-step. The weights in the network are updated proportional to the gradient of the cost-function with respect to the weight in question. Also most activation functions produce gradients within the range -1,1 or [0,1), meaning that they often can be close to 0. Computing the update for a weight very early in the network means compute a lot of gradients by applying the chain-rule. These n gradients will then be multiplied, where n is the depth of a feed-forward neural network or the amount of time-steps in a recurrent one. With deep networks, this can often result in gradients very close to 0. The most common way to prevent this problem is to use LSTM-modules in which the activation function simply is the identity function, which always has a gradient of 1.0 and allows the error to propagate backwards unchanged.

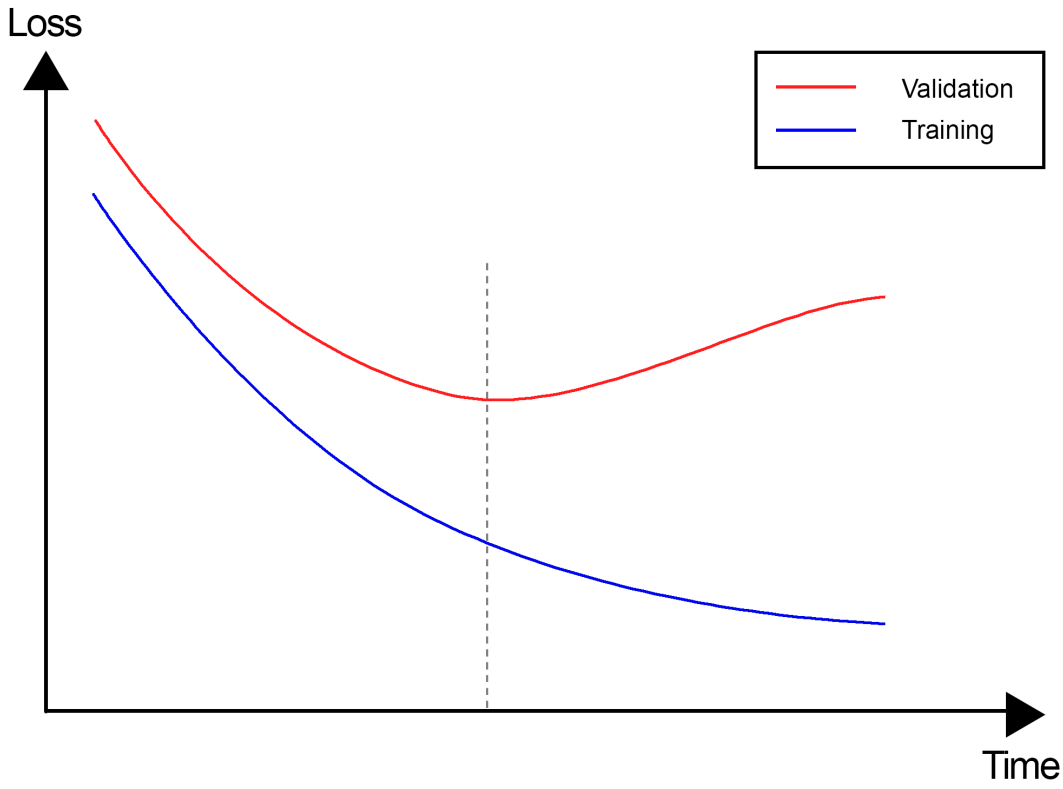
3.2.2.3 Exploding gradient

A problem closely related to the vanishing gradient problem is the exploding gradient problem. This is the opposite to the vanishing gradient where the gradient instead grows very large and the updates to the weights become enormous. There exists several ways to solve this problem, but the most common one is to apply gradient clipping. Gradient clipping can be performed in different ways but they all involve reducing the gradient if it is above a certain clipping value.

3.2.2.4 Validation

The goal with training a network is to get as low training loss as possible. The loss measures the amount of error made by the network. After training the model for a given amount of time, the training loss will become very small. It will however perform worse than before. This is called overfitting and occurs when the model learns the training data "by heart" and performs well on the training data, but poorly on text not seen before in the training data. It is therefore good to have a validation set which is hidden from the model when performing the training. Between certain intervals in the training, the loss of the validation set is calculated without performing any training. Keeping track of the validation loss during training is crucial to know when overfitting happens. Therefore, if keeping track of the validation loss as it changes from decreasing to increasing, one will know when it is time to stop training. Further training will only make the model perform worse. Figure 3.2 shows an example of how the loss of training and validation can change over time.

Figure 3.2: Example of a graph of training and validation loss over time.



3.2.3 Character-by-Character

The network will process the input-stream one character at the time. For each of the characters, it will output whether it belongs to a category or not and if so, which category. Another approach which may seem more intuitive at first would be to input one word at a time instead, we have however chosen to go with character by character because this would probably mean that the grammatical version of the word will not matter and also that lower and uppercase will contribute as well. Furthermore, the algorithm might be able to generalize to character-patterns instead of just learning the words. As an example, most diseases and pharmaceutical drugs often have names with an uppercase letter in the beginning which is a feature that the network should pick up on after sufficient training. Analyzing the text, character-by-character, has other benefits as well, for example doing it one word at a time needs a large vocabulary, containing all the words, whilst characters only need a very small vocabulary which should improve the speed of the training.

3.2.4 Bi-Directionality

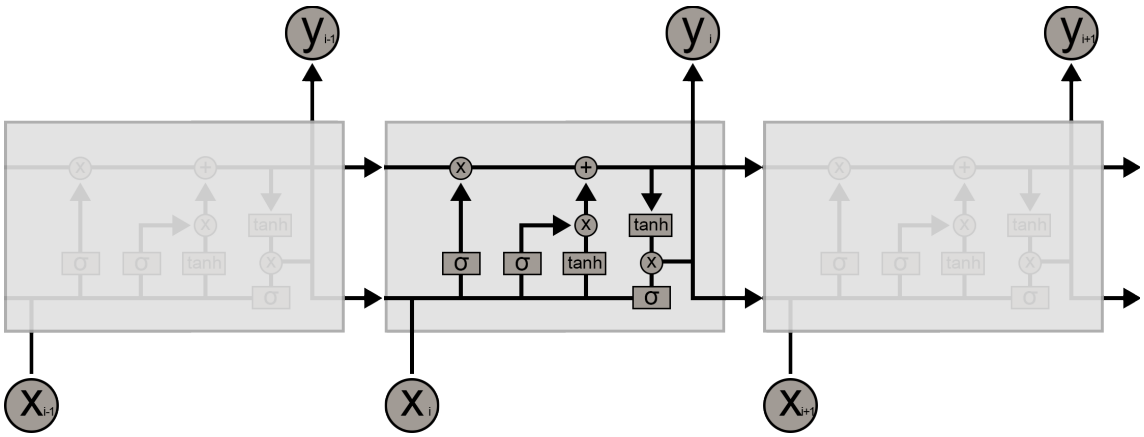
Recurrent neural networks are very good when the task at hand includes some temporal information in the input. Language is a good application since one often needs context from the beginning of the sentence to deduce what the latter part is really about. However, only having knowledge about what has been written

previously is not always enough and that is why bi-directionality should improve the performance. Bi-directionality in the context of RNNs is that we essentially have two identical networks, one processing the text from start to end and the other one in reversed order. When classification occurs not only do we have information up to the current character but also every character after the current one. This should give us more information for the classification step and thus give us a better chance of classifying correctly.

3.2.5 Long Short-Term Memory

Long Short-Term Memory is an RNN model that enables learning of long-term dependencies [7]. LSTMs are based on a concept called cell state. By having a stream of data feeding information from one time-step in the network to the next, information from the past can be preserved. The cell state will be modified in various ways depending on what the input is. In the horizontal line with an x and a plus-sign in Figure 3.3 there are several gates that decides whether the network wants to add or remove information. The first gate is called *Forget Gate Layer* which will

Figure 3.3: The inner workings of an LSTM model.



decide how much of the previous state to keep. Colah comes with an example in his blog [13] that if the text was talking about a woman and later on talking about another person who is a man, then the network needs to forget information about the gender of the person since the focus has shifted from a female to a male. The forget gate layer output a value between 0 or 1 for each number in the cell state, where 0 is to completely forget and 1 is to keep it as is. The second state is called the *Input Gate Layer* which decides what kind of and how much new information to add to the cell state. This is done by first having a sigmoid layer which decides which values in the cell state that should be updated followed by a hyperbolic tangent function to generate the new values for the cell state. The sigmoid layer and the hyperbolic tangent are then multiplied to only update the values that it decided to update. In the third and the final step, the decision of what to output is done. This will depend on the cell state, but filtered with the help of a sigmoid function in order to only output some of the values form the cell state. It is finally put through a hyperbolic tangent function to produce the output to the next step.

3.3 Evaluation

The most common way to measure the performance of a model created for Named Entity Recognition is to use a measurement called F_1 score. It is widely used in statistical analysis of binary classification and can easily be extended to a multi-class classification to suite this problem better. F_1 score is based on two concepts called precision and recall. On a class basis, precision is the number of entities correctly classified as belonging to the class divided by the total number of entities classified to the class. Recall is the number of entities correctly classified as belonging to the class divided by the total number of entities that actually belong to the class. The F_1 score is then the harmonic mean of precision and recall:

$$precision = \frac{|true\ positives|}{|true\ positives + false\ positives|} \tag{3.8}$$

$$recall = \frac{|true\ positives|}{|true\ positives + false\ negatives|} \tag{3.9}$$

$$F_1 = 2 * \frac{precision * recall}{precision + recall} \tag{3.10}$$

The idea behind F_1 score is to measure the performance by balancing the number of correctly classified entities with the number of wrongly classified entities.

3.4 Limitations

With the fact that we are implementing two different models in mind, the time was already very limited. This means that a lot of aspects for both of the models had to be omitted. There was just enough time to create the models, run the experiments and observe the results. Below is a collection of things that we wanted to try out and experiment with but due to time-constraints was not included in this thesis.

3.4.1 Lemmatization & Stemming

Both of the models could possibly benefit from using lemmatization and/or stemming. Lemmatization is a way of turning any word into its lemma or base-form while stemming is just cutting away the ending of a word. Both of these techniques would make it easier to distinguish between words of different grammatical form since they will probably be turned into the same lemma or stemmed to the same word. Lemmatization could help with the fact that the seed-terms most often only contain one version of a word and we want to find it in the text regardless of what tense or form the word has. However, since both models should be able to infer from the seed-terms the models should be able to capture the entities even if they have some form that does not exist in the seed-term list. This is due to the fact that the goal from the start is for the models to be able to extrapolate from that data and draw conclusions about entities that it has never seen before.

3.4.2 Hyperparameter Search

The Word-Vector model has a lot of parameters that affect different parts of the model. Finding the best set of parameters is an extensive task and is nearly impossible to do in a reasonable time frame. As can be seen in Section 5.2.6, a search for the best parameter value for each individual parameter was done by changing the value for each parameter and let the other parameters be a default value. With enough time, there would be other search strategies that would be better at finding the best parameter values.

3.4.2.1 Grid Search

In a perfect scenario, grid search would be used to determine what parameter values works best. Grid search would run the model through all possible combinations of the hyperparameters within predetermined intervals in order to find the parameter values that will maximize the result of the model. It is however time-consuming, considering that adding more parameters to evaluate would increase the number of evaluations exponentially.

3.4.2.2 Random Search

Another way to approach hyperparameter searching is to use random search. It is done by determining a distribution for each parameter so that they have a maximum and a minimum value. In each run, a random parameter is chosen that will change its value randomly within the given distribution. Since new category vectors or vocabulary need to be generated depending on the parameters, random search would be time-consuming as well and therefore discarded as a choice of finding the best parameter values.

3.4.3 Sequence length

Running the experiments for the RNN model, a sequence length is set. The sequence length needs to be long enough to include the surrounding context of the seed-terms, but a too long sequence length would include category-words that are not in the seed-term lists. It is therefore important to have a sequence length that is balanced. The parameter should be chosen carefully, but due to the fact that the model needs to be retrained every time it is changed there was not enough time.

4
Experiments

In this section we will talk about the experimental setup for each of the two models we have implemented. The first part will be about the data sets that were used to evaluate the model. Next section is about the Word-Vector model, followed by the RNN model which we developed ourselves.

4.1 Data sets

To be able to compare the performance of our model to other existing solution it is important to have a good data set to evaluate the model on to make the comparison as fair as possible. The best way would naturally be to use the exact same data but that is not possible to do when working with a smaller language. Furthermore, since journals contain sensitive information getting access to some of those data set could possibly be very hard or impossible. During the evaluation of our models we have used two very different data sets: one where text from 1177.se was annotated by ourselves as well as Stockholm EPR Corpus which contain real-world anonymized patient journals.

4.1.1 1177 Vårdguiden

This data set was created from the list of documents from *1177.se* that can be found in Appendix A. It is a Swedish site containing information, counseling and services regarding health-care. We got a total of 15 annotated documents and 2740 annotations to measure the results with. The documents were carefully chosen to not favor any of the three categories, but contains about an equal number of entities from all of them. Since we do not consider ourselves medical professionals in any sense the quality of the annotations can be questioned. On the other hand 1177.se's target audience is not only people within the medical field but anyone should be able to read and understand the text. This should make it easier to annotate correctly and anything that were unclear was researched in order to find the correct meaning.

The reason we created this data set was because during the course of this thesis we did not know that we would be able to evaluate our models on Stockholm EPR and needed something that we could test our models on by ourselves. Therefore several of the evaluations, especially for the Word-Vector model, has only been done using this data set and not Stockholm EPR.

4.1.2 Stockholm EPR Clinical Entity Corpus

Stockholm EPR (Electronic Patient Record) Corpus [3] is a set of 512 clinical units from Karolinska University Hospital in Stockholm encompassing the years 2006 - 2014 and over two million patients. It consists of 7946 documents containing real-world anonymized patient journals with annotations in the 4 categories: *Disorder*, *Finding*, *Drug* and *Body structure*. Since we have a category where Disorder and Finding are bundled together we merge them into one. Due to the fact that the goal was to perform NER on patients journals from the beginning, evaluation on this data-set should give a better representation of its performance than evaluation on text from 1177 Vårdguiden.

4.1.3 Seed-terms

The seed-terms are collected from the medical taxonomies SweMeSH [8] and Snomed CT [17]. Due to their hierarchical structure, one can extract all children from a category. After extracting terms for all the categories, most of the terms will be used as an input-seed to the algorithm. Some of the terms were left out because of ambiguities in the Swedish language. An example is the Swedish word *"hand"* (*hand*) which was under the category body structure. The word hand is also used as in other contexts in Swedish. For instance, "på egen hand" means "at your own" in Swedish. This means that "hand" may occur in other contexts and will cause a too much noise in our model. The seed-term lists were also compared against a list of common grammatical words in order to filter out the more ambiguous words that would cause noise. This filtering were done manually by us just going though the list of seed-terms.

4.2 Word-Vector Model

The experiment was performed on documents manually annotated by us. Since we did not have any annotated data at the time when the evaluation was done on the model, we decided to annotate a number of documents ourselves. The documents were taken from *1177 Vårdguiden* [1]. In order to pick the best hyperparameter values, experiment where the model was run with different hyperparameter values was performed. The values were picked based on the values that yielded the best result when running the model. The experiment and its results can be read in detail in Section 5.2.6.

4.2.1 Sentence Splitting & Tokenization

The input documents are first preprocessed by doing sentence splitting. It is done with GATE Embedded [12] which is a language processing framework in Java which can connect different processing tasks into a pipeline and output the enriched documents in XML-format. We use a Swedish sentence splitting module from OpenNLP [6] to be able to split the sentences. Similarly we use a Swedish tokenizer from OpenNLP afterwards to be able to tag all tokens.

4.2.2 Part-Of-Speech Tagging

Part-Of-Speech tagging is the next step in the GATE pipeline. There is a POS-module from OpenNLP which is trained on the Stockholm Umeå Corpus [19] (SUC) which is a corpus annotated with POS-tags. There exist several different standards for POS-tags. The POS plugin from OpenNLP uses SUC-tags, which is a standard based on Stockholm Umeå Corpus [19]. These tags are then converted with a simple one-to-one mapping to a format called Parole-tags which is required in the next step, Noun Phrase chunking.

4.2.3 Noun Phrase Chunking

This step is performed with Swe-SPARK [2], which is an implementation of a NP chunker trained on the Swedish language. Swe-SPARK uses POS-tags to identify the noun phrases. Swe-SPARK uses the given POS-tags to chunk the input text into different chunks where the NP-chunks are the ones important to us. The noun phrases are then filtered based on an IDF-threshold. Phrases that got an IDF value lower than the threshold will be filtered out. An IDF-threshold of 5 was chosen since that gave the best results in the experiments with parameters in Section 5.2.6.

4.2.4 Seed-Term Collection

Seed-terms are collected from two taxonomies, SweMeSH [8] and Snomed CT [17]. SweMeSH is a taxonomy of Swedish medical terms and Snomed CT consists of Swedish medical concept terms. Since they have a hierarchical structure, we were able to extract all the subordinate terms for each of our predefined categories by traversing the tree downwards.

We use the following predefined categories to classify the entities into: Disorder & Finding (Swedish: *sjukdom & symtom*), Pharmaceutical drug (Swedish: *läkemedel*) and Body structure (Swedish: *kroppsdel*). There have been previous work done on NER for the medical domain which uses these mentioned categories [16] which means comparing our results with them will become much easier for us and for future research of NER in the medical domain.

4.2.5 Signature Calculation

The vocabulary takes words from all available releases of Läkartidningen (1996-2005) [18]. To keep the vocabulary from being too large, only words with a term frequency greater than a certain threshold are added to vocabulary. This threshold is called *Frequency Threshold*. The results from the experiment in Section 5.2.6 shows that 40 is a good threshold value. When the vocabulary is created, it also filters out words beginning with a number or any special characters since they are most often words that are not relevant and since every year and every combination of numbers would result in a word in the vocabulary they were removed to leave space for more impactful words. Signature generation of the category vectors are done by looking for the seed-terms in Läkartidningen and using the average vector

for each seed-term list. When calculating the signature for each word, weights for the context words as well as the internal words are needed. From hyper-parameter searching we found that internal weight of 20 and context weight of 1 worked best for this task. The weight for a word in the context will be 1/k where k is the distance in number of words that the word is from the term and 1 is the chosen context weight. This means that the two words right next to the noun phrase will have weight 1/1 and the next two will have 1/2 and so on.

Zhang and Elhadad used different vectors for internal words and context words and then concatenated them together resulting in a vector which is twice the size of their vocabulary [23]. Since the vectors already are sparse to begin with, we added them together instead and made it possible to increase the size of our vocabulary even further.

For each possible entity from the input document, a signature vector is generated and compared with all category vectors using standard cosine-similarity introduced in Section 3.

4.2.6 Entity Classification

Entity classification can be done in several different ways. The idea, following Zhang and Elhadad [23], is to compare the vector for each NP with the three different category vectors and choose one of them depending on the results. Note that a threshold is needed. In case the word-vector is far away from all category vectors, it should not be classified to the category of the closest vector. This threshold is named *Similarity Threshold* and was set to be 0.005, which gave the best results in the experiments in Section 5.2.6. Furthermore, an additional threshold was introduced. The magnitude of the similarity compared against the other two similarities is needed as well so that entities where the algorithm is not sure will not be classified as one of the categories. This threshold is called *Classification Threshold* and was set to 0.7 based on the results from the experiment mentioned earlier.

4.2.7 Evaluation

Result is measured with F_1 score, which was explained in Section 3.3. The model has several steps that depends on each other. F_1 score is measured on three parts to be able to measure all parts separately:

- Boundary Detection
- Entity Classification
- Total

Boundary Detection is the part of the model where it finds the entities in the input document. Measuring the F_1 score of the Boundary Detection gives a figure of how many of the entities it finds. For Word-Vector model, the entities that it should classify is directly dependent on the detection of the boundaries, this will determine how many entities that it will even have potential to classify right. It will also show

how good the detection is. One important thing to note is that boundary detection is not only dependent on the preprocessing for the Word-Vector model, but Classification Threshold as well as Similarity Threshold will influence it.
Entity Classification is the part of the model where the entity candidates are classified into one of the predefined categories.. For the Word-Vector model, the entities are determined before hand while the RNN model recognizes and classifies them as it traverses the document. To measure the F_1 score for Entity Classification, the right entities are given to the algorithms and only classification is performed. This will give back a measure of how well it classifies.
Total is the whole model. The F_1 score is measured after doing both Boundary Detection and Entity Classification.

4.3 Recurrent Neural Network Model

The second implemented model is based on a recurrent neural network. This is a supervised model and acts accordingly. However, the data is not annotated by humans, but rather generated from the seed-terms. Comparing with the Word-Vector model, instead of vector calculations, a recurrent neural network is trained to correctly classify the entities on a character-by-character basis. The network is implemented with the framework TensorFlow, which is a framework used to create various machine-learning related models. It provides a lot of already implemented functionality with small amounts of coding.

4.3.1 Preprocessing

The network need training and validation data in form of list of sequences. To be able to get a good spread between the training set and the validation set, we generate the sequences from different sources.

4.3.1.1 Training

Since all of Läkartidningen 1996 - 2005 was used to train the Word-Vector model described in Section 4.2, we wanted to use the same data to train the RNN model. In the preprocessing, all of Läkartidningen is scanned to find occurrences of all seed-terms in the lists. For each occurrence, a sequence of 60 characters containing the seed-term positioned randomly in the span are used as an input. Since the seed-terms are known, the targets can be automatically generated. The targets also consists of a 60 character sequences, containing zeroes, ones, twos and threes representing each category and zero for unclassified. The window need to be small enough in order to minimize the risk of including words that are not in the seed-term list, but belongs to a category. On the other hand it should be large enough to contain a context for the seed-term for the algorithm to take advantage of. The sequence length of 60 characters was chosen by intuition after inspecting examples of different lengths. There was unfortunately not enough time to try out different sequence lengths. The inputs and targets together makes up the training set and will be fed to the network during the training steps with all characters from the seed-term classified correctly

and the rest of the characters without any class. A problem with this strategy is that there is a possibility that other medical terms which is not in our seed-term list will be caught within the span and incorrect examples will make their way into the training data. This can be managed roughly by keeping the span small enough. In addition, negative training examples are generated in order to prevent the network from learning that classified entities always occur in every sequence. They contain only non-classified characters and this should improve the performance since in real world examples there is not always an entity in every 60-character span. Including negative examples mimics the real world in a more accurate way. To have a large amount of training data is good. To increase the number of training data, each occurrence of seed-term that was found were generated together with three variant where the window was shifted a little bit into one direction. This would increase the training set by four times with sequences that are slightly different.

4.3.1.2 Validation

A validation set is generated from the Swedish Wikipedia. Wikipedia was chosen since it is a large free resource and contains many relevant articles about the medical domain. By crawling selected articles within the medical domain, we collect links to other articles. The articles from the links are then used to generate the validation set in the same way that we generated the training set. Counting the number of seed-terms that occurred in the validation set, but not the training set, we saw that it was fewer than 5% that were unique to the validation set. One kind of overfitting for Named Entity Recognition could be that it finds the seed-terms only. Since this would be bad, we had to increase the number of articles we crawled from. While increasing the number of articles would find more unique seed-terms, it would also find more non-unique seed-terms as well. To be able to increase the ratio of sequences containing unique seed-terms, we generated 5 variants of each sequence containing unique seed-terms and no variants for the sequences containing non-unique seed-terms. This generated 48720 sequences of which 62192 occurrences of seed-terms were found and 11022 occurrences of seed-terms that are not in the training set. This means that almost 18% of the occurrences of the seed-terms are unique to the validation set, which gives us a better validation. The articles we crawled from are listed in Appendix B. They have been chosen manually to be balanced between the categories.

4.3.2 Network Setup

The network is built with Google's machine learning framework TensorFlow. The framework has out-of-the-box implemented models for the most common types of neural networks from which we use their LSTM-cells to build our network. In order to get bi-directionality, we take two LSTM-models, one that reads the input from left-to-right and another from right-to-left. Both these models are connected to the output layer which takes both of them into account when making the classification. The model is implemented with 3 hidden layers and 128 neurons in each layer and the vocabulary is built up from all characters encountered in the training examples. We chose the size of the network by trail and error as well as some guidance from

our supervisors. All characters in the input is mapped to an entry in the vocabulary. Characters that are not in the vocabulary will be mapped to the last entry in the vocabulary, called the misc ID, so that they are covered as well. The network will output a probability distribution over the different classes, which sums to 1. In our case, the we have 4 probabilities, one for each class and one for unclassified. From this distribution the maximum value is chosen and the character is classified accordingly. With a classification of each character, a word is classified to the class which the majority of characters has been classified to. Furthermore, if the token between two words belonging to some specific class is classified to that class as well, the two words are combined into one classified term.

4.3.3 Training

The training data is a total of 775,000 of sequences with 60 characters each. 10% of the data is negative data, containing only unclassified characters. We have chosen a learning rate of 0.002 and a decay rate of 0.975. These numbers were chosen carefully after extensive testing and running the models with different parameters. Usually with RNN or LSTM models, it can be a good idea to pre-train the model as a language model. A language model is, given an input character, predicting the next character in the sequence. This could help the model to understand the fundamental features of the language which would reduce the training time. Since our model contains bi-directionality, it is hard to pre-train the model as a language model since the layers goes in opposite direction to each other. Furthermore, since this only speeds up the convergence of the model and would not affect the performance, we did not traing a language model.

In each iteration, the input to each hidden layer is subject to *dropout*. This is a concept regularly used when training neural networks to reduce over-fitting and to make the network converge faster [14]. Dropout essentially depends on a single parameter p which is a probability to keep each value in the input vector. The probability p is set to 0.5 which means that each input to a certain layer has a 50% probability of being set to 0. This is universally known to be the best setting for the parameter and should only be increased slightly if one has a huge network, 1000+ units in each layer [22]. Since our network has 128 neurons in each layer, on average only 64 of these values are kept and used for training. The reason why this does works is that having dropout keeps the network from relying heavily on some parameters and teaches it to classify correctly with any subset of inputs. On the other hand the expressively of the network decreases since on average only half of the signals in the network are being used which can be counteracted by simply increasing the number of neurons.

During the training a readily available optimizer was used called Adamoptimizer [9]. Adamoptimizer implements the Adam algorithm for solving gradient-based optimization problems on stochastic objective functions. What algorithm to use depends on the problem at hand, but Adam combines two optimizer that works well on machine-learning related problems [9], RMSprop and AdaGrad. RMSprop deals

well with a moving objectives and AdaGrad deals with very sparse gradients in a good way.

Below the different settings we tried are explained in greater detail:

Deeper	For this model we left most of the settings to default but increased the depth of the network to check what impact that had on the performance. A network with 128 neurons in each layer with a depth of 4 layers were trained with a learning rate of 0.05 as well as a decay rate of 0.975.
Low learning rate	This model also set most settings to default. A network with 128 neurons in each layer, 3 layers deep but this time with a learning rate of 0.002. The decay rate was set to 0.975. Lowering the learning rate proved useful and 0.002 became the default setting for learning rate.
Smaller network	This model had default settings, but a smaller network with 64 neurons in each layer, 3 layers deep. Learning rate set to 0.002 and decay rate to 0.975.
No dropout	This model left all the settings as default but removed dropout entirely. This means a network with 128 neurons in each layer, 3 layers deep and a learning rate of 0.002 as well as a decay rate of 0.975. This setting proved to be the best, which meant that the default settings subsequently never used dropout.
Even lower learning rate	Here we checked if lowering the learning rate even more would further improve the result. All settings to default apart from lowering the learning rate to 0.0002. Default settings was 128 neurons per layer and 3 layers deep. Decay rate set to 0.975 and no dropout used.

4.3.4 Evaluation

Result is measured with F_1 score, which is explained in Section 3.3. Classification with a character-based network is classifying entities one character at a time. This means that it will determine the boundary as it classifies. There is however a point in measuring the performance of both boundary detection and the overall run. If the boundary detection has poor performance, that would suggest that any system that uses this information will also do poorly. F_1 score is measured on two parts:

- Entity Classification
- Total

Measuring the Entity Classification and the Total will be done the same way as described in Section 4.2.7.

5
Result

Below is the result for both the Word-Vector model and the Recurrent Neural Network model as well as a simple baseline.

5.1 Baseline

The baseline simply consist of dictionary look-ups of the encountered words in the list of seed-terms.

Table 5.1 shows the results of running the baseline model on Stockholm EPR Corpus [3]. The baseline model achieves a precision of over 70% on Disorder & Finding and Body Structure, but is significantly lower for Pharmaceutical Drug. It has a higher precision than recall in general due to the fact that if a match is found it is probably correct. The algorithm got a precision of 0.6693, a recall of 0.1177 and an F_1 score of 0.2001 in total.

Table 5.1: Result of the run on baseline algorithm on Stockholm EPR Corpus.

Category	**Precision**	**Recall**	F_1 **Score**
Disorder & Finding	0.7648	0.1152	0.2002
Pharmaceutical Drug	0.2465	0.0380	0.0659
Body Structure	0.7041	0.2924	0.4132
Total	0.6693	0.1177	0.2001

5.2 Word-Vector Model

In this section the results of the Word-Vector model are presented. Each part and kind of experiment that we have conducted are divided into separate sub sections. All parts have been measured on the manually annotated documents based on articles from *1177.se*.

5.2.1 Model Result

In Table 5.2 we can see the F_1 score and the parameter settings for the two best runs, two runs in the middle and the two worst runs. The best run gave an F_1 score

of 21.84%. This could be compared to the run with the parameter settings that yielded the worst result which got an F_1 score of 17.02%.

Table 5.2: Result of F_1 score for different runs. # notes on what position the score was amongs all runs, *CW* is the context weight, *IW* is the internal weight, *FT* is frequency threshold, *CT* is classification threshold, *ST* is similarity threshold, *IDF-T* is IDF threshold, *#C* is context windows size and F_1 *Score* is the performance score.

#	CW	IW	FT	CT	ST	IDF-T	#C	F_1 Score
1	1	20	10	0.7	0.005	5	5	0.2184
2	1	20	40	0.7	0.005	2	5	0.2146
12	1	20	10	0.75	0.005	2	5	0.2029
13	1	20	10	0.85	0.005	2	5	0.2027
25	1	20	10	0.7	0.01	2	5	0.1780
26	5	20	10	0.7	0.005	2	5	0.1702

Table 5.3: Precision score for different runs. # notes on what position the score was amongs all runs, *CW* is the context weight, *IW* is the internal weight, *FT* is frequency threshold, *CT* is classification threshold, *ST* is similarity threshold, *IDF-T* is IDF threshold, *#C* is context windows size and *Precision* is precision score.

#	CW	IW	FT	CT	ST	IDF-T	#C	Precision
2	1	20	10	0.7	0.005	5	5	0.3204
15	1	20	40	0.7	0.005	2	5	0.2376
12	1	20	10	0.75	0.005	2	5	0.2424
13	1	20	10	0.85	0.005	2	5	0.2406
8	1	20	10	0.7	0.01	2	5	0.2711
26	5	20	10	0.7	0.005	2	5	0.1303

5.2.2 Boundary Detection Result

In Table 5.5 below, results for evaluating only the boundary detection are presented. This only depends on one parameter, the IDF-threshold, and is essentially just an evaluation of Swe-SPARK when capturing medical entities. The best F_1 score is 0.2727 with a precision of 0.2060 and a recall of 0.4033.

5.2.3 Entity Classification Result

Classification is the second part that the algorithm is relying on. In order to measure the performance of the entity classification in a proper way, the algorithm was given the right entity boundaries and set to always classify the entities into one of three categories. In Table 5.6 we see the best run got an F_1 score of 62.51%. The range of scores are spanning from 60.47% to 62.51%. Precision and recall are well-balanced

Table 5.4: Recall score for different runs. # notes on what position the score was amongs all runs, *CW* is the context weight, *IW* is the internal weight, *FT* is frequency threshold, *CT* is classification threshold, *ST* is similarity threshold, *IDF-T* is IDF threshold, *#C* is context windows size and *Recall* is recall score.

#	CW	IW	FT	CT	ST	IDF-T	#C	Recall
23	1	20	10	0.7	0.005	5	5	0.1657
8	1	20	40	0.7	0.005	2	5	0.1956
15	1	20	10	0.75	0.005	2	5	0.1745
14	1	20	10	0.85	0.005	2	5	0.1752
26	1	20	10	0.7	0.01	2	5	0.1325
4	5	20	10	0.7	0.005	2	5	0.2453

Table 5.5: Results for only boundary-detection using Swe-SPARK

IDF Threshold	Precision	Recall	F_1 score
2	0.1228	0.4960	0.1969
3	0.1407	0.4869	0.2183
4	0.1488	0.4763	0.2267
5	0.1689	0.4460	0.2450
6	0.2060	0.4033	0.2727

and matching the placement of the runs very well. The run with best F_1 score has the best precision and recall as well. A random classifier would statistically get 33% correct since we have three categories which means that our model almost classifies almost twice as many.

Table 5.6: Entity classification results of the Word-Vector model. All runs were made on the manually annotated data with different parameter values in each runs.

CW	FT	C	Precision	Recall	F_1 score
5	10	5	0.6258	0.6245	0.6251
1	10	3	0.6236	0.6223	0.6229
1	10	2	0.6206	0.6190	0.6198
10	10	5	0.6167	0.6153	0.6160
1	10	5	0.6163	0.6150	0.6156
1	10	7	0.6152	0.6139	0.6145
1	20	5	0.6145	0.6131	0.6138
2	10	5	0.6127	0.6113	0.6120
1	10	10	0.6090	0.6077	0.6083
1	40	5	0.6061	0.6047	0.6054
1	50	5	0.6053	0.6040	0.6047

5.2.4 Part-of-speech performance

We wanted to assure that the POS-tagger worked properly, so we used Läkartidningen [18] corpus that is annotated with POS tags in order to compare how well GATE Embedded [12] manages to do POS-tagging. Part-of-speech tagging was done on a subset of Läkartidningen 2003 with the 163,731 first words. Tokenization and sentence splitting is proven to work almost perfectly. It only failed at very few special cases. Mail addresses can in particular be hard if it contains *dot*, which is a stop word. It also fails at words ending with an apostrophe, for instance *Jovanovic'*. These words were discarded when comparing POS tags. Out of the 163,656 that were compared, 11,083 words were different from the manually annotated POS-tags. With the corpus as a reference, it managed to have 93.2% similarity with the manually annotated POS-tags. One of the more frequent errors were due to special characters such as double left/right pointing angle ». Other errors were that it did not manage to differentiate if a noun is of singular article or plural article. This is often hard to see without knowledge about the word. With the result of the comparison, we can conclude that the POS-tagger is working well. It would be interesting to see if the NP chunker did a similarly good job, but since we do not have a corpus tagged with noun phrases, we were not able to measure the performance of the NP chunker.

5.2.5 Global Inverse Document Frequency

Inverse Document Frequency is a measure of how common a word is on a document basis. The more common, the lower value it gets. In our implementation of the Semi-Supervised Word-Vector model, we count words on a sentence basis instead. We do the assumption that users are inputting one document at the time and it is therefore more suitable to do it on sentence basis. IDF is used to filter out common noun phrases, when considering entity candidates. We had a theory that the filtering, based on IDF, could affect the results negatively. The reason behind this is that IDF, based on the input document, is simply not enough to make an assumption on irrelevant candidates. A document usually consist of a topic, which means that the subject of the topic will have a high frequency and have a high risk of being filtered out. We propose to use a global IDF. By including Läkartidningen 1996-2005, the text we use to create our category vectors, we hope to be able to reduce the number positive candidates that are filtered out. To see if that is the case, we have run the algorithm with the parameters that worked well in the experiments with different hyperparamater values in Section 5.2.6 and with global IDF. Then we run with the same parameters, but without global IDF. Since the number of noun phrases that are filtered out is different with global IDF, we run with different IDF threshold values in order to see that the hypothesis is true for most of the cases.

Table 5.7 shows the result of boundary detection with global IDF and local IDF with different values for IDF-threshold. Furthermore, Table 5.8 show the same runs but instead shows F_1 score for both boundary detection as well as the score running the entire model.

Table 5.7: Runs of the Word-Vector model with global IDF (G-IDF) and local IDF (L-IDF). Local IDF was run with different IDF threshold values (denoted with t). Result shows true positives, false positives and false negatives for each run.

	True Positive	**False Positive**	**False Negative**
G-IDF t=5	1223	6014	1517
L-IDF t=5	631	3056	2109
L-IDF t=4	743	4356	1997
L-IDF t=3	833	5405	1907
L-IDF t=2	936	6329	1804
L-IDF t=1	993	9274	1747

Table 5.8: Runs of the Word-Vector model with global IDF (G-IDF) and local IDF (L-IDF). Local IDF was run with different IDF threshold values (denoted with t). Result shows F_1 score of boundary detection and on the whole model for each run.

	F_1 **Score Boundary Detection**	F_1 **Score Total**
G-IDF t=5	0.2452	0.2160
L-IDF t=5	0.1964	0.1318
L-IDF t=4	0.1896	0.1314
L-IDF t=3	0.1856	0.1445
L-IDF t=2	0.1871	0.1634
L-IDF t=1	0.1527	0.1628

5.2.6 Behaviour of Model Depending on Parameters

The Word-Vector model has a set of hyperparameters that can be adjusted in various grades. The parameters are explained throughout the methodology in Section 3. Parameters with adjustable values are the following:

- Context Weight
- Context Window Size
- Frequency Threshold
- Classification Threshold
- Similarity Threshold
- IDF Threshold

To be able to evaluate how the model is behaving depending on the parameters, a hyperparameter search was performed. By choosing one parameter at the time and adjusting its value between each run, it gives a result on how the model will behave depending on the value of each specific parameter. The experiment is done on the documents from 1177.se [1] that were annotated by us. The articles that were annotated can be found in Appendix A. For each parameter, a number of suitable values have been chosen to be tested with. The parameter values were chosen by doing number of observations by manually adjusting parameters. While adjusting one parameter, the other parameters are set to default values that have been chosen on intuition after observing some runs. When the runs are done, one can

see how different parameter values affects precision, recall and F_1 score. Not all of the parameters affects the entire model. Some only affects the boundary detection, other parameters affects the entity classification and other affects both. The F_1 score measurement is done in three ways: only on boundary detection, only on entity classification and on the whole model. Boundary detection is the part where the model finds the boundaries of the entities. Entity classification is giving the model the right boundaries and letting the model classify them. The whole model is measured from boundary detection to the entity classification. In the results below, only overall performance as well as performance on entity classification are shown. For a complete list of runs and the settings used, see Appendix C. A complete list of results with Precision, Recall and F_1 Score can be read in Appendix D.
Default values for the parameters were set to following:

- Context Weight: 1
- Context Window Size: 5
- Frequency Threshold: 10
- Classification Threshold: 0.7
- Similarity Threshold: 0.005
- IDF Threshold: 2

5.2.6.1 Context Weight

Context Weight determines how important context words are set in relation to the internal words. Internal weights are always 20 which means that the value v for context weight will be in ratio *v:20* against internal weights. The best result for

Table 5.9: Result of runs with different parameter values for context weight on Word-Vector model. Result is measured using F_1 score. It is measured on classification of entities with given boundaries and also on the whole model.

Context Weight	**F_1-Classification**	**F_1-Total**
1	0.6156	**0.2033**
2	0.6120	0.1954
5	**0.6251**	0.1702
10	0.6160	0.1796

entity classification was given when context weight was set to 5, which can be seen in Table 5.9. With context weight set to 5, the ratio between context and internal weight is 1 : 4. This results in an improvement of 0.009 over the second best run. Overall result gave the best score when context weight was set to 1, which gives a ratio of 1 : 20. It has only an increase of 0.005 over the second best, due to the generally low score. It has however an improvement of 0.03 over the run with lowest F_1 score. The improvement is hardly noticeable for classification, while the performance of the whole model is over a large span. It is hard to draw an absolute conclusion about why the parameter behave differently between classification and the whole model. One possible explanation is that if the context is more important to the model, the better it is able to classify the given entities. On the whole run, a

high context weight can cause candidates that are not entities to be classified since the context words might occur often in the category vectors.

5.2.6.2 Context Window Size

Context Window Size sets the window of the context words. It is the number of neighbouring words on each side of the internal words. Result in Table 5.10 is

Table 5.10: Result of runs with different parameter values for context window size on Word-Vector model. Result is measured using F_1 score. It is measured on classification of entities with given boundaries and also on the whole model.

Context Window Size	**F_1-Classification**	**F_1-Total**
2	0.6198	**0.2131**
3	**0.6229**	0.2102
5	0.6156	0.2033
7	0.6145	0.2006
10	0.6083	0.1978

showing that it is generally bad to have a too large context window. Even with a weighted context, where a word gets less value the further away from the internal words. Classification result is better with a size of 3, suggesting that having a context is relevant in some aspect. The performance of the whole model is better with a size of 2.

5.2.6.3 Frequency Threshold

Frequency Threshold is a threshold that decides how large the vocabulary is going to be. The vocabulary is built by counting all the words in Läkartidningen. Words that occur less times than a certain threshold value will be filtered out from the vocabulary. This would naturally mean that the lower the frequency threshold is, the larger the vocabulary. For classification of entities, we see in Table 5.11 that it

Table 5.11: Result of runs with different parameter values for frequency threshold on Word-Vector model. Result is measured using F_1 score. It is measured on classification of entities with given boundaries and also on the whole model.

Frequency Threshold	**F_1-Classification**	**F_1-Total**
10	**0.6156**	0.2033
20	0.6138	0.2088
40	0.6054	**0.2146**
50	0.6047	0.2124

is generally better to have as low frequency threshold as possible. With frequency threshold of 10, the F_1 score is 61.56% and with a value of 20 it decreases to 61.38%. This suggests that the more details that can be captured in the model, the better it gets. The same reasoning does not apply when running on the whole model however.

It is better with a higher frequency threshold of 40. Running the whole model also means that it has to be able to discard candidates that are not correct. To be able to do that, it discards entities that are not similar enough to a category. A larger vocabulary will make the vector more sparse and could potentially increase the possibility of discarding an entity. That could be a possible reason why it works better with higher frequency threshold.

5.2.6.4 Classification Threshold

Classification Threshold is based around the idea that if a candidate is almost equally similar to all categories, it does not make sense to classify it into any of them. The threshold value determines how many times more similar the candidate must be to the most similar category than the other categories. The parameter do not affect the result of entity classification, since it will always classify the entities. There is no consistent behavior of how the model's performance changes as the value is changing. Threshold values between 0.7 and 1.0 gives different F_1 score between 20.22% to 20.32%. The one exception is for classification threshold of 0.5, which works particularly well with an F_1 score of 20.98%. A threshold value of 0.5 means that the maximum similarity must be more than two times greater than the similarity of the other categories. This suggests that a more strict classification manage to filter out false candidates better while keeping the true candidates. Since the best result is given on the smallest given parameter value in our span, we see that we misjudged the span of parameter values to test. An even lower value would be interesting to examine, but nothing we were able to test because of the time constraint.

Table 5.12: Result of runs with different parameter values for classification threshold on Word-Vector model. Result is measured using F_1 score. It is measured on the whole model.

Classification Threshold	F_1**-Total**
0.5	**0.2098**
0.7	0.2033
0.75	0.2029
0.8	0.2022
0.85	0.2027
0.9	0.2024
1.0	0.2025

5.2.6.5 Similarity Threshold

Similarity Threshold is a constraint to only classify entities if they have a similarity larger than the threshold value. In Table 5.13 it is clear where it performs best. When similarity threshold is set to 0.003 it is better than both lower and higher values. The higher the value is, the more similar the entity must be to the category.

Table 5.13: Result of runs with different parameter values for similarity threshold on Word-Vector model. Result is measured using F_1 score. It is measured on the whole model.

Similarity Threshold	F_1**-Total**
0.01	0.1780
0.008	0.1983
0.005	0.2033
0.003	**0.2070**
0.001	0.1880
0.0009	0.1881

In other words, 0.003 must be the point where it filters out most false positives while keeping the true positives.

5.2.6.6 IDF Threshold

IDF Threshold is a threshold that determines which of the noun phrases found by Swe-Spark that are filtered out. Noun phrases with an IDF lower than the threshold value will be discarded. A threshold value of 5 gives the best F_1-score as one can see

Table 5.14: Result of runs with different parameter values for IDF threshold on Word-Vector model. Result is measured using F_1 score. It is measured on the whole model.

IDF Threshold	F_1**-Total**
2	0.2033
3	0.2086
4	0.2117
5	**0.2184**
6	0.2024

in Table 5.14. In comparison, it is about 0.007 higher than 4 and 0.016% higher than 6. A low value could make the model keep many false candidates that are classified in a later stage while a high value could potentially filter out true candidates.

5.3 Recurrent Neural Network Model

Below are the results from the evaluation of the RNN-based model. The evaluation was performed on 733 real-world examples of medical journals from Stockholm EPR Corpus [3]. Since the data is very sensitive the evaluation was not performed by ourselves but instead let the holder of the data perform the evaluations.

5.3.1 Model Result

The results in Table 5.15 shows the run of the Recurrent Neural Network model with *No dropout*-settings. Both Disorder & Finding and Body Structure have a much higher precision than recall, while Pharmaceutical Drug is better balanced. In total, the Recurrent Neural Network model got 0.6709 in precision, 0.2359 in recall and 0.3491 in F_1 score.

Table 5.15: Results of run on Recurrent Neural Network model on Stockholm EPR Corpus.

Category	**Precision**	**Recall**	F_1 **Score**
Disorder & Findings	0.7241	0.1817	0.2905
Pharmaceutical Drugs	0.6922	0.4321	0.5321
Body Structure	0.4564	0.2775	0.3452
Total	0.6709	0.2359	0.3491

5.3.2 Classification Result

Given the boundaries for the entities in Stockholm EPR Corpus, the Recurrent Neural Network model performed according to Table 5.16. The table shows promising results for both the category Disorder & Finding and Pharmaceutical Drug which has an F_1 score of 0.814 and 0.736 respectively. Body Structure shows a weaker score of 0.471 in F_1 score. In total the model got an F_1 score of 0.751.

Table 5.16: Entity classification results of run on Recurrent Neural Network model on Stockholm EPR Corpus.

Category	**Precision**	**Recall**	F_1 **Score**
Disorder & Findings	0.917	0.732	0.814
Pharmaceutical Drugs	0.639	0.866	0.736
Body Structure	0.361	0.678	0.471
Total	0.751	0.751	0.751

5.4 Comparison

Since there are two models and a baseline that tries to solve the same task, a comparison between the Baseline model, Word-Vector model and Recurrent Neural Network model will show how they compare against each other. Since we did not have the opportunity to run the Word-Vector model on Stockholm EPR Corpus due to time constraints, the comparison was made on the results from the manually annotated data set which all were run on. Since the Word-Vector model has a boundary detection where it annotates a wider span, a run on the manually annotated data with an adjusted, wider boundary was added.

Comparing the results of the models in Table 5.17, we see that the Word-Vector model do not perform well due to the wider boundaries that it detects. This can clearly be seen in the adjusted run, where it performs around 47% better. All models have around 50% in precision, except for the adjusted Word-Vector model which comes in at 32%. Recall is much lower where they are in the span of 8.5% to 16.7%, except for the Recurrent Neural Network model which has a recall value of 20.7%. This is mostly the reason why the Recurrent Neural Network model has the highest F_1 score with 28.9%. At the second place comes the adjusted Word-Vector model at 21.84%, followed by the Baseline model at 21.80% and lastly the Word-Vector model with 14.8%. Even though the Baseline model and the adjusted Word-Vector model have a similar performance scores, we can see that their precision and recall are vastly different. Baseline model has a high precision and a low recall, while the adjusted Word-Vector is more balanced between precision and recall.

Table 5.17: Comparison of the results between each model on the manually annotated data set.

Model	**Precision**	**Recall**	F_1 **Score**
Baseline	0.5406	0.1365	0.2180
Word-Vector	**0.5538**	0.0856	0.1482
Word Vector (adj.)	0.3204	0.1657	0.2184
Recurrent Neural Network	0.4788	**0.2072**	**0.2892**

6
Discussion

In this chapter discussion about the results as well as other relevant observations will be presented.

6.1 Language

The structure of the Swedish language is different from English in a numerous ways from a grammatical perspective. These differences can affect the performance of the model and must be taken into account when designing it.

6.1.1 Compound Words

The Swedish language consists of a lot of compound words. A compound word is when two or more words are joined together to form a longer word. An example is the phrase *"a very strong medication"* would be *"en jättestark medicinering"*, where *jätte* and *stark* is translated to *very strong*. The Word-Vector model relies on the vocabulary that consists of words. This would likely lead to losing some context since there are many compound words and not all are included in the vocabulary if it is of resonable size. The RNN model has a character based vocabulary, which means that it has the possibility to learn compound words. One way to tackle compound words is to implement a compound splitter that splits the compound word into the original words. However, this practise may affect the context of the text. For instance if we take the phrase *"rökfritt på stationen"* which translates to *"no smoking allowed at the station"*, we have the compound word *rökfritt*. It means *no smoking allowed* or *free of smoke* and consists of the words smoke (*rök*) and free (*fritt*). If compound splitting were to be performed, the phrase would now be *"rök fritt på stationen"* and translates to *"smoke freely on the station"*. The phrase has now completely changed its meaning because the compound words is now split up into two separate words. Whether compound splitting is performed or not, context may be lost in either way.

6.1.2 Ambiguous Words

Ambiguity is occurring in many languages. We could however conclude that many of the seed-terms, especially in the Body Structure category, were ambiguous. They often have a second meaning that is part of Swedish sayings, which makes them occur frequently in another context. The Swedish word huvud (*head*) often occurs as *"över*

huvud taget", which is translated to *"at all"*. Another example is hand (*hand*), which occurs as *"på egen hand"* and translates to *"on your own"*. The Swedish language also has words such as vad (*calf*) which also means *what*, led (*joint*) which also means *to have suffered*, lever (*liver*) which also means *is alive* and so on. This is a flaw in the semi-supervised approach. A supervised approach would learn from manually annotated data where the terms with correct context being annotated while the terms in another context would be left out. This is however impossible to do with a semi-supervised approach with seed-term lists, since one cannot distinguish same terms with different contexts in the learning process. One cannot simply go through all the text and find the terms with wrong context. Removing ambiguous words or terms would very likely reduce the number of wrongly classified entities, but also decrease the number of correctly classified entities. In other words: it would increase the precision, but would decrease the recall.

6.2 Named Entity Recognition

The problem of Named Entity Recognition for the medical domain is inherently hard. One big problem is the similarity between body structures and disorders where they often are closely connected. Many disorders contain body structures within them which leads to ambiguity.

6.3 Word-Vector Model

The first major flaw with the Word-Vector models was that we had little to no control over two of the main parts of the model. Since we used external frameworks for both NP-chunking and POS-tagging the end-results could not possibly be better than the NP-chunker due to the errors it was making. Furthermore, since the NP-chunker depends on the POS-tagger even more errors were introduced since the POS-tagger also cannot be perfect.

6.3.1 Results

The best result got an F_1 score of 21.84%, which is measured on both boundary detection and entity classification together. It is run with IDF threshold of 5 and the other parameters have pre-determined default values based on earlier runs with manual tweaking of parameters. The difference between the best performed run and worst performing run is roughly 4%.

In Table 5.3, we see the precision score for the runs from Table 5.2. Here we see that they differ widely in how well they performed. The run that had best F_1 score is the second best performing on precision. The run with second best F_1 score has the 15th best precision, putting it in the second half of the list over precision. Looking at the bottom, the run that got second worse F_1 score has the 8th best precision. The difference between the best and worst precision is also a little bit wider than the F_1 score, ranging from 13.03% to 33.17%.

Table 5.4 shows the recall score for the same runs. The performance of the runs are mostly the opposite of how they performed on the precision, meaning that the runs are in general unbalanced between precision and recall. The first run that got the second best precision have the 23rd, or the third worst recall value. The overall worst run that had the worst precision has the fourth best recall value. The span of recall values are between 13.25% to 27.04%.

The results presented in Section 5 suggest an in-balance in precision and recall for the different runs. It is generally good if precision and recall are balanced. However there are cases where one is more preferred over the other. If it is important for the use-case that there are not any wrong classifications or false positives, it may be a better idea to aim for a higher precision rather than good recall. Especially in the medical domain.

6.3.2 Boundary detection

Performance of the algorithm is highly dependent on two parts: to be able to detect the boundaries of the entities and to correctly classify them. Classification cannot be done unless the boundaries have been found. Boundary detection is done with external frameworks, combining POS-tagging, noun phrase chunking as well as an IDF-threshold. After running the experiment with different parameters, we realized that the boundary detection is not doing particularly well. In Table 5.5 we see that F_1 score ranges from 19.69% to 27.27%. The precision and recall are very unbalanced here where precision ranges from 12.28% to 20.60% and recall ranges from 40.32% to 49.60%. For the run where F_1 score is highest, it manages to find 1105 boundaries of 2740. It is not even half of the entities and on top of that, it finds 4258 false positives. Regardless of how the value of IDF Threshold changes, the precision will decrease when recall increases and vice versa. One problem with the boundary detection step is that we have not implemented it ourselves. This makes it hard to steer the way that it detects the boundaries. One way to annotate entities is to have the boundaries to be the least descriptive phrase. For instance in the phrase *"en svår tropisk sjukdom"* (*"a severe tropical disease"*), *tropisk sjukdom* is the least descriptive phrase. In comparison, the noun phrase chunker we use detects noun phrases so in this case it could be the whole phrase *"en svår tropisk sjukdom"*. It is hard to predict exactly how much details the noun phrase chunker will include. An example of this could be seen when the article is mentioning an example. When an article contains *"Till exempel morfin"* (*"For example morphine"*) it will misstakenly detect *"exempel morfin"* instead of *"morfin"*. Another example is for the phrase *"ont i magen"* ("pain in the stomach") where it detects *"ont"* and *"magen"*. It might be how one prefers to have the boundaries of the entities, but it could as well be that one prefer the whole phrase as the boundary instead. It is hard to modify the current boundary detection step without adding rule based layer on top. Another approach would be to build our own boundary detection model, but since that is a whole new subject in and of itself we did not have time to do that.

6.3.3 Comparison with Zhang and Elhadad

The results in the Zhang and Elhadad [23] paper are slightly higher than our results. They got a score of 69.5% compared to our 62.5%. On the overall performance, they got a score of 26.5% while our got 21.8%. They have no score available for boundary detection, which makes it hard to fully compare the performance. Since they got a higher overall performance, it suggests that the boundary detection is better than ours. They however got a better score on entity classification as well, which might suggest that their model is better on parts besides the boundary detection. Another factor might be the larger amount of data that they have. We extracted seed-terms from Snomed CT and SweMeSH, which gave us roughly 50,000 Swedish seed-terms in total. They had a total of over 610,000 which is more than 10 times the seed-terms we have. Given that they are able to cover so many more terms, it might not be so strange that their algorithm performs better.

6.3.4 Inverse Document Frequency

Based on the results in Section 5.2.5, local IDF never manages to find as many true candidates even after adjusting the IDF threshold. With the different values of IDF threshold, it finds at most 993 true candidates with threshold value 1. In the case where it finds 993 true candidates, it also find 3260 false candidates. For lower IDF thresholds, the filter will start to let false candidates pass through as well. The best case is to find a trade-off between increasing true positives and decreasing false positives if the use-case for the task is not sensitive for false positives. In Table 5.8 we see that so is the case. The performance of boundary detection with global IDF is 24.52% while the highest one with local IDF is achieved when IDF threshold is set to 5 with a score of 19.64%. Looking at F_1, we see that the best F_1-score for local IDF is achieved when IDF threshold is set to 2. The reason behind this is that while IDF threshold set to 5 performed better on boundary detection because it had fewer false positives as well as true positives. Lower IDF thresholds have more true positives and false positives, but will filter out some of them in the entity classification step. So there is a trade-off between how many true positives that is worth to have versus how many extra false positives that will be added. Regardless of the value of local IDF, global IDF has a better performance in all aspects.

6.4 RNN Model

When training the network we calculate a loss for each batch and backpropagate through the network. At first we thought that since most of the input data do not belong to any class, the network would play it safe and always yield good loss if everything was classified as the unclassified category. To counteract this, we implemented a different type of loss weights. When calculating the loss, we did not only count the miss-classifications but weighted them higher if the miss-classification was of an entity that belonged to any of the three categories. We later realized that this approach did not have an impact on the model since the network was weighting those classifications and only made the network use more classifications instead.

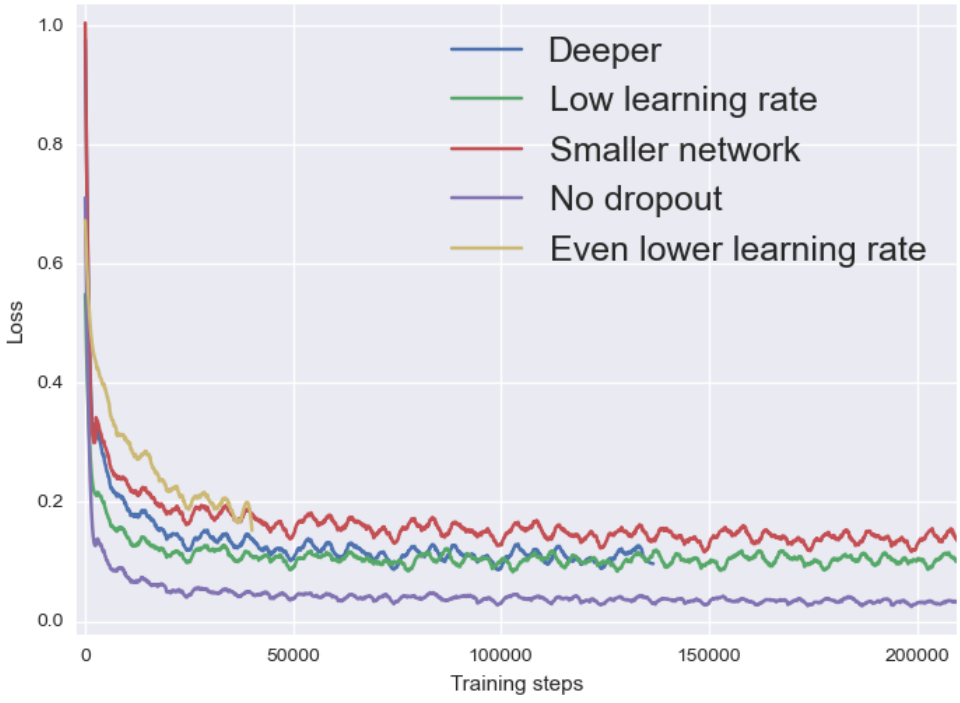
6.4.1 Network setup

How large and how deep a network is is directly correlated to how well it performs and therefore having a well-suited size of the network is very important. If it is too small, it will not have enough capacity to learn from the data and the network will never learn. On the other hand if it is too large, it has room to fit even more information than intended from the beginning and it will start to learn other correlations in the data which will lead to overfitting. In each layer we used 128 neurons but we also tried decreasing the amount but this only made the network perform worse. Regarding the depth of the network we used three hidden layers in addition to the inherent depth of the RNN structure.

6.4.2 Training

The results of all the training done are presented in Figure 6.1. As can be seen the error drops very fast and remain on that stage during the remaining time. Another observation is that most of the settings produced somewhat similar results with only slight variation in the end. However, the model with no dropout and learning rate set to 0.002 seemed to perform best and was the model we later used during the evaluation.

Figure 6.1: Training loss for RNN-models with different settings.



6.4.3 Classification & Boundary detection

Since the model essentially performs two tasks bundled up into one we also performed evaluation where the boundaries of the entities were known and only tried to classify them into the correct category. As can be seen from the results in Table 5.15, the model got an F_1 score of 0.3491 on both of the tasks combined. For only the classification part the model has an F_1 score of 0.751. This suggests that the task of boundary detection is inherently harder or not reflected well enough in the training data for the model to learn from it. It has however higher F_1 score than the Word-Vector model in both classification and the model as a whole. Since both models are basing their knowledge about the domain from the same source, this suggests that the Recurrent Neural Network model manages to learn features that the Word-Vector model did not learn.

6.5 Ethical Aspect

In the medical domain there are a lot of regulations on how to handle information and data about the employees, their work and most of all their patients. Patient journals contains private information and only a few people with authority have access right to it. There is a great privacy concern in the health care industry regarding handling and using sensitive information digitally. This is something we have witnessed in this thesis project as we were only allowed to use anonymized patient journals to train a supervised model if we were on site. There is a question about how well the anonymization is done. Anonymizing the names of the patient might not be enough. Given the context of the patient journal, one might figure out who the journal is about. Feeding anonymized patient journal to a Named Entity Recognition model will give the model a knowledge about the context of the journals and also the patients in some way. It is hard to predict exactly how it could affect the privacy, but there certainly is some risk involved. While there are several papers about anonymizing medical journals [4] [15], there is still much work left to be done. There needs to be a better relation between the governments, health care and the scientists so that these kinds of data could be used for purpose of improving our lives. However we, as scientists, need to be able to assure that the data is handled properly.

While the data needs to be handled properly, it is also important to be accurate in the medical field. If medical equipment is not accurate or information is incorrect somewhere, it could possibly lead to disastrous consequences. Named entity recognition will most certainly not be used in the same way as patient monitoring equipments, but there are still risks with a NER model with a poor performance. For instance, the model might have missed to detect a patient's symptom in the journal that could have been the determining factor of figuring out what kind of treatment the patient should have gotten in order to become healthy. It could be that other patients have had the same symptoms and the connections between the patient could not be found because of the annotations were missing. Another issue is

if some entity is wrongly classified and end up in another category. This could cause confusion for the people who read the journals. It could also be time consuming to fix, which is not good in a field where there already is too little time.

7
Conclusion

Starting off with the Word-Vector model, we implemented the model with inspiration from Zhang and Elhadad which also is reflected in the result. This suggests that the method works almost equally well for Swedish as it does for English, although comparison is not entirely justified since we evaluated on different data sets. Furthermore, we realized how dependent the model is on boundary detection which is a straight-forward way to improve upon our results.

The RNN-model performs better than the Word-Vector model which is probably due to the fact that it is able to capture more features and can make better decisions based on those. With an F_1 score of almost 0.35 it is about 60% better than the Word-Vector model which is a huge improvement. Although the improvement is great the model is still has a long way to go before it can make accurate predictions. Cleaning up of the seed-term lists would probably increase the performance. Furthermore, counteracting the slicing of the words in the beginning and end of the sequences would also be very interesting to see how it would affect the results. Hyperparameter searching is very hard when dealing with machine learning methods which need to be trained for long periods of time before one can know how well they perform.

In general, two models for NER was implemented, trained using Läkartidningen and finally evaluated on real medical journals. Since we have not found any similar unsupervised approach we cannot compare our results with any other NER-attempt on medical entities in Swedish. Therefore, we are happy with our results and hope others can stand on our shoulders to develop this method even further and get even better results.

8
Future work

There are several options that we did not have time for or discovered to late in the process to be able to try but we think could have improved our result.

8.1 Seed-terms

Both of the algorithms depend heavily on the seed-terms. The quality of them are translated into the quality of the models. As the quality of our seed-terms turned out to be not that good, a better way of generating the seed-term lists would potentially increase the accuracy of the models. After running the models for some time, we discovered that the entity *depression* is classified as a Body Structure rather than a Disorder & Finding. Therefore our model has learned to almost always classify *depression* into Body Structure which is behaviour that we most definitely do not want. Other terms that belonged to another category were found as well.

For future work, it is better to put a much greater emphasis on making the seed-term lists of a much higher quality, with fewer errors. The total number of seed-terms in the lists are over 53,000 and takes a lot of time to go through manually. Another way to take control over the quality of the seed-term lists is to get a deeper knowledge of the structure of Snomed CT and SweMeSH, the taxonomies that we collect the seed-terms from, and be more restrictive on what entities to extract.

8.2 Word-Vector Model

In this section future work related to the Word-Vector model are presented.

8.2.1 Noun-phrase chunking

Where this model performed the poorest was clearly the boundary detection. It could be improved by either using a different method for finding the boundaries or by developing a better NP chunker more suited for this task.

8.2.2 Lemmatization and Stemming

Regarding mostly the first model lemmatization and stemming of the words would make it a lot easier to distinguish between the different grammatical forms of the words. Since most of the time only one form of each word is found in the seed-term

lists this approach would increase the effectiveness of each seed-term by making them capture all forms and tenses of the incoming words.

8.3 Recurrent Neural Network Model

In this section future work and opportunities to improve the RNN model are presented.

8.3.1 Traininig

The more obvious one is just more training for the RNN-based model. Since training is directly related to the performance of the model and due to the fact that we never got to a stage where the validation-loss, see Section 3.2.2.4, went up suggests that the model was not "saturated" with training and did not reach the best possible value.

8.3.2 Sequence length

In the experiments performed within the borders of this thesis, a sequence length of 60 characters was used. This could be explored further and different lengths of sequences could be tried until the best length for the task is found.

Bibliography

[1] 1177 - vårdguiden. http://www.1177.se/. Accessed: 2016-06-08.

[2] John Aycock. Swe-spark. http://stp.lingfil.uu.se/~bea/resources/spark/. Accessed: 2016-06-09.

[3] Hercules Dalianis, Martin Hassel, Aron Henriksson, and Maria Skeppstedt. Stockholm epr corpus: A clinical database used to improve health care. *Swedish Language Technology Conference*, pages 17–18, 2012.

[4] Khaled El Emam and Fida Kamal Dankar. Protecting privacy using k-anonymity. *Journal of the American Medical Informatics Association*, 15(5):627–637, 2008.

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

[6] The Apache Software Foundation. Apace opennlp. http://opennlp.sourceforge.net/models-1.5/. Accessed: 2016-06-09.

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

[8] Karolinska Institutet. Svensk mesh. http://mesh.kib.ki.se/swemesh/swemesh_se.cfm. Accessed: 2016-06-09.

[9] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[10] Wenhui Liao and Sriharsha Veeramachaneni. A simple semi-supervised algorithm for named entity recognition. In *Proceedings of the NAACL HLT 2009 Workshop on Semi-Supervised Learning for Natural Language Processing*, pages 58–65. Association for Computational Linguistics, 2009.

[11] Mary Meeker. Internet trends 2015-code conference. *Glokalde*, 1(3), 2015.

[12] The University of Sheffield. Gate - general architecture for text engineering. https://gate.ac.uk/family/embedded.html. Accessed: 2016-06-09.

[13] Christopher Olah. Understanding lstm networks – colah's blog. http://colah.github.io/posts/2015-08-Understanding-LSTMs/. Accessed: 2016-06-08.

[14] Vu Pham, Théodore Bluche, Christopher Kermorvant, and Jérôme Louradour. Dropout improves recurrent neural networks for handwriting recognition. In *Frontiers in Handwriting Recognition (ICFHR), 2014 14th International Conference on*, pages 285–290. IEEE, 2014.

[15] Mark A Rothstein. Is deidentification sufficient to protect health privacy in research? *The American Journal of Bioethics*, 10(9):3–11, 2010.

[16] Maria Skeppstedt, Maria Kvist, H. Nilsson Gunnar, and Dalianis Hercules. Automatic recognition of disorders, findings, pharmaceuticals and body structures from clinical text: An annotation and machine learning study. *Journal of Biomedical Informatics*, 49:148–158, 2014.

[17] Socialstyrelsen. Snomed ct. `http://www.socialstyrelsen.se/nationellehalsa/snomed-ct`. Accessed: 2016-06-09.

[18] Språkbanken. Läkartidningen 2003 corpus. `https://spraakbanken.gu.se/eng/resource/lakartidn-vof`. Accessed: 2016-06-09.

[19] Språkbanken. Stockholm-umeå corpus. `http://spraakbanken.gu.se/eng/resources/suc`. Accessed: 2016-06-09.

[20] Nina Viberg. Swedish waiting times for healthcare in an international perspective. *Sveriges Kommuner och Landsting*, 2011.

[21] Yefeng Wang and Jon Patrick. Cascading classifiers for named entity recognition in clinical notes. In *Proceedings of the workshop on biomedical information extraction*, pages 42–49. Association for Computational Linguistics, 2009.

[22] Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329*, 2014.

[23] Shaodian Zhang and Noémie Elhadad. Unsupervised biomedical named entity recognition: Experiments with clinical and biological texts. *Journal of biomedical informatics*, 46(6):1088–1098, 2013.

A

Appendix for: Annotation of 1177

To measure the accuracy of the Semi-Supervised Named Entity Recognition model, text from *1177.se* were manually annotated. Paragraphs with irrelevant information such as contact information were left out in some of the text. The annotated data contains 14 articles and 1 short question answer. There is a total of 2740 annotations with Disorder & Finding having 1574 annotations, Pharmaceutical Drug 546 and Body Structure 620. The annotations is done by surrounding the entities with different characters depending on the category. Parenthesis ( ) is mapped to Disorder Finding, hard-bracket [ ] is Pharmaceutical Drug and curly braces { } is Body Structure. For instance, the phrase *"Motion minskar också risken för (övervikt) och..."* has the word *övervikt* annotated as Disorder & Finding. The following articles were annotated:

- Demens - Alzheimers sjukdom
- Bedövning och smärtlindring vid tandvård
- Receptfria läkemedel vid tillfällig smärta
- Så åldras kroppen
- Läkemedel vid pollenallergi
- Vaccinationsprogram för barn
- Stroke – slaganfall
- Behöver jag testa mig för zikaviruset?
- Denguefeber
- KOL – kroniskt obstruktiv lungsjukdom
- Knöl i bröstet
- Läkemedel som tas upp genom huden
- Drick sundare
- Myggbett och knottbett
- Ibumetin

B

Appendix for: Validation Set from Wikipedia

Validation set for the machine learning model is generated by crawling Wikipedia. It is done by gathering the links from relevant articles to the domain and take the content from the articles in the links. The links were collected from the following articles:

- Infektionssjukdom
- Cancer
- Patogen
- Sjukdomar i blodet och blodbildande organ
- Endokrina sjukdomar
- Autoimmunitet
- Reumatism
- Medicinsk diagnostik
- Förkylning
- Depression
- Sjukdomar i nervsystemet
- Hjärntumör
- Allergi
- Astma
- Smitta
- Stress
- Människans anatomi
- Kroppsdel
- Muskel
- Anatomi
- Anatomisk variation
- Nervsystemet
- Skelett
- Perifera nervsystemet
- Ansikte
- Magsäck
- Ryggrad
- Tarm
- Hormon
- Läkemedel
- Antibiotikum

- Medicinalväxt
- Kirurgi
- Selektiva serontinåterupptagshämmare
- Ibuprofen
- Enzym
- Aminosyror
- Vitamin
- Näringsämne
- Fettvävnad
- Vaccination
- Farmakokinetik
- Insulin
- Pencillin

C

Appendix for: Settings of Runs on Word-Vector Model

Table C.1: List of runs on Word-Vector Model and their settings. Run No. shows the name of the run, CW is Context Weight, IW is Internal Weight, FT is Frequency Threshold, CT is Classification Threshold, ST is Similarity Threshold, IDF-T is IDF Threshold, #C is Context Window Size

Run No.	**CW**	**IW**	**FT**	**CT**	**ST**	**IDF-T**	**#C**
R1	1	20	10	0.7	0.005	5	5
R2	1	20	40	0.7	0.005	2	5
R3	1	20	10	0.7	0.005	2	2
R4	1	20	50	0.7	0.005	2	5
R5	1	20	10	0.7	0.005	4	5
R6	1	20	10	0.7	0.005	2	3
R7	1	20	10	0.5	0.005	2	5
R8	1	20	20	0.7	0.005	2	5
R9	1	20	10	0.7	0.005	3	5
R10	1	20	10	0.7	0.003	2	5
R11	1	20	10	0.7	0.005	2	5
R12	1	20	10	0.75	0.005	2	5
R13	1	20	10	0.85	0.005	2	5
R14	1	20	10	1.0	0.005	2	5
R15	1	20	10	0.7	0.005	6	5
R16	1	20	10	0.9	0.005	2	5
R17	1	20	10	0.8	0.005	2	5
R18	1	20	10	0.7	0.005	2	7
R19	1	20	10	0.7	0.008	2	5
R20	1	20	10	0.7	0.005	2	10
R21	2	20	10	0.7	0.005	2	5
R22	1	20	10	0.7	0.0009	2	5
R23	1	20	10	0.7	0.001	2	5
R24	10	20	10	0.7	0.005	2	5
R25	1	20	10	0.7	0.01	2	5
R26	5	20	10	0.7	0.005	2	5

D

Appendix for: Result of Runs on Word-Vector Model

Table D.1: List of runs on Word-Vector Model with results measured in Precision, Recall and F_1 Score.

Run No.	**Precision**	**Recall**	F_1 **Score**
R1	0.3204	0.1657	0.2184
R2	0.2376	0.1956	0.2146
R3	0.2929	0.1675	0.2131
R4	0.2307	0.1967	0.2124
R5	0.2822	0.1693	0.2117
R6	0.2742	0.1704	0.2102
R7	0.2766	0.1690	0.2098
R8	0.2446	0.1821	0.2088
R9	0.2602	0.1741	0.2086
R10	0.2100	0.2040	0.2070
R11	0.2442	0.1741	0.2033
R12	0.2424	0.1745	0.2029
R13	0.2406	0.1752	0.2027
R14	0.2365	0.1770	0.2025
R15	0.3317	0.1456	0.2024
R16	0.2369	0.1766	0.2024
R17	0.2406	0.1745	0.2022
R18	0.2320	0.1766	0.2006
R19	0.2809	0.1533	0.1983
R20	0.2202	0.1796	0.1978
R21	0.1878	0.2036	0.1954
R22	0.1453	0.2668	0.1881
R23	0.1476	0.2588	0.1880
R24	0.1345	0.2704	0.1796
R25	0.2711	0.1325	0.1780
R26	0.1303	0.2453	0.1702