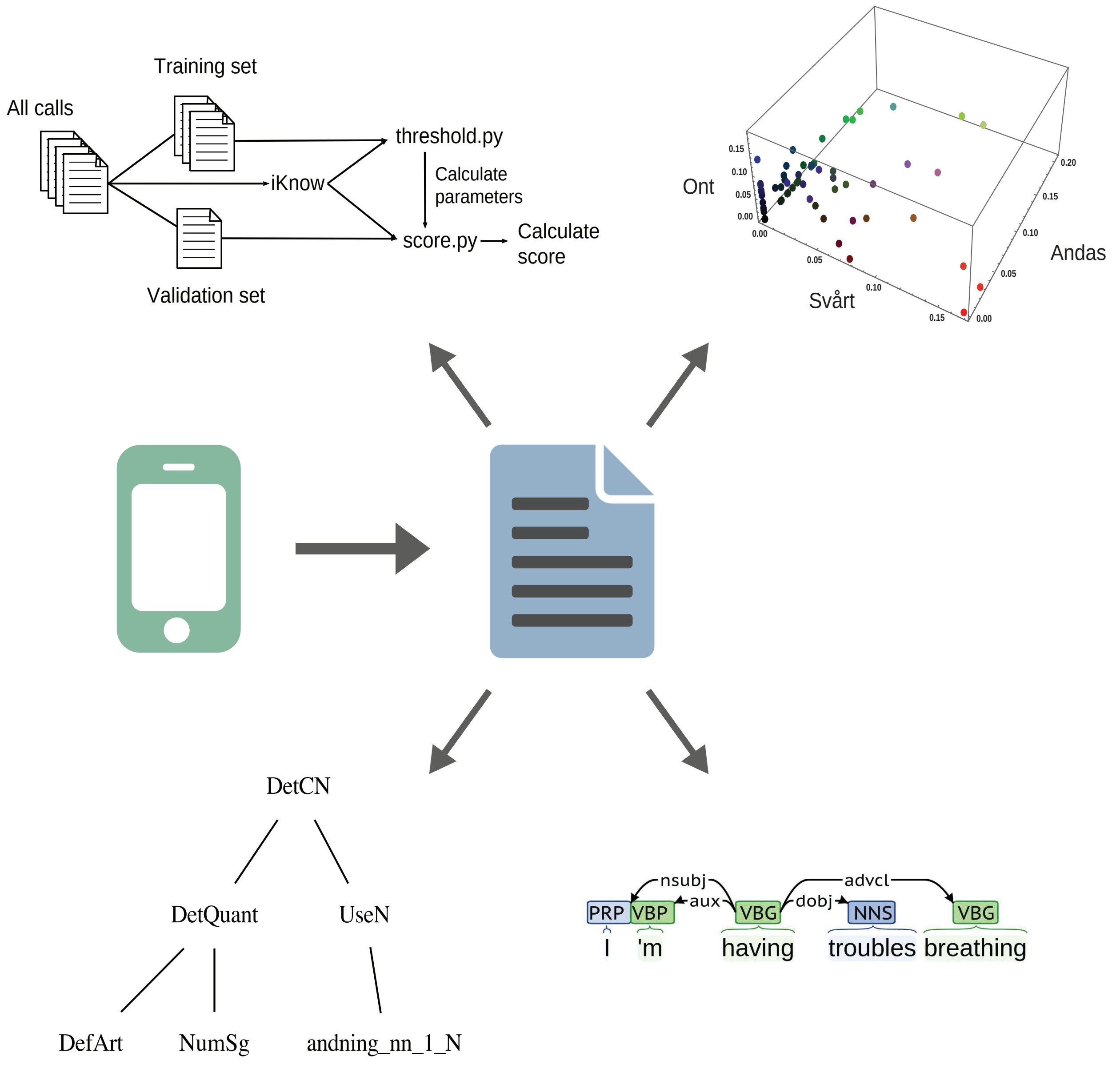
CHALMERS | UNIVERSITY OF GOTHENBURG



Automatised analysis of emergency calls using Natural Language Processing

Bachelor of Science Thesis in Computer Science and Engineering

Emarin Andersson, Benjamin Eriksson, Sofia Holmberg, Hossein Hussain, Lovisa Jäberg, Erik Thorsell

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
Gothenburg, Sweden 2016

Bachelor of Science thesis

Automatised analysis of emergency calls using Natural Language Processing

Emarin Andersson, Benjamin Eriksson, Sofia Holmberg,
Hossein Hussain, Lovisa Jäberg, Erik Thorsell

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg

Gothenburg, Sweden 2016

Automatised analysis of emergency calls using Natural Language Processing

Emarin Andersson
Benjamin Eriksson
Sofia Holmberg
Hossein Hussain
Lovisa Jäberg
Erik Thorsell

Supervisors: Magnus Almgren, Dep. of Computer Science and Engineering
Marina Papatriantafilou, Dep. of Computer Science and Engineering

Examiner: Arne Linde, Dep. of Computer Science and Engineering

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover:
An overview of the methods utilised to analyse the emergency calls.

Department of Computer Science and Engineering
Gothenburg, Sweden 2016

Automatised analysis of emergency calls using Natural Languague Processing

Emarin Andersson
Benjamin Eriksson
Sofia Holmberg
Hossein Hussain
Lovisa Jäberg
Erik Thorsell

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg

Bachelor of Science Thesis

Abstract

The operators at SOS Alarm receives thousands of calls each day at the different emergency medical communication centres, owned by SOS Alarm, all over Sweden. A subset of these calls contain room for improvement and the operators could learn to improve from these calls. The work of finding – and analysing – these calls is however too tedious to be done by a human. This thesis presents four automatised solutions to this issue. The human factor is removed and the job of finding and analysing the calls is done by a computer.

It is shown that it is possible to partly automatise the analysis, but the methods used have different strengths and weaknesses. Word frequency analysis is proven adequate at key word look-up. Similarity comparisons of various aspects of the calls are proven good at classifying calls, but less good at answering specific questions. Comparing parse trees seems promising, but the technology needs more work before it is ready to be used.

The solutions presented show that it could be possible to automatise the analysis of the calls given that the right questions are asked and the results from these are used appropriately.

Keywords: emergency medical dispatcher, EMD, emergency medical communication centre, EMCC, SOS Alarm, natural language processing, iKnow, gensim, CoreNLP, Grammatical Framework

Sammanfattning

Det inkommer tusentals samtal varje dag till operatörerna på SOS Alarms nödcentraler. I en delmängd av dessa samtal finns det rum för förbättringar och operatörerna kan lära av dessa samtal. Att hitta och analysera dessa samtal är dock allt för tidskrävande för att göras manuellt. Det här arbetet presenterar fyra tekniker för att automatisera processen med utgallring av samtal. Den mänskliga faktorn tas bort och arbetet görs istället av en dator.

Det är möjligt att till viss del automatisera analysprocessen, med hjälp av “natural language processing” (processande av naturliga språk). De olika metoderna som används har olika för- och nackdelar. Att räkna frekvensen av vissa ord visar sig lyckat för att besvara frågor med relevanta nyckelord. Att jämföra hela samtal mer matematiskt visar sig bra för att kategorisera hela samtal, men inte lika bra för att svara på specifika frågor om ett samtal. Sist används grammatiska träd för att jämföra samtal och även om tekniken verkar lovande behövs det mycket arbete innan den går att använda.

Lösningen som presenteras visar att det kan vara möjligt att automatisera analysen av samtalen, givet att rätt frågor ställs och att resultaten från dessa används på rätt sätt.

Nyckelord: SOS-operatör, larmcentral, processande av naturliga språk

Acknowledgements

The authors of this bachelor thesis would like to thank their supervisors Magnus Almgren and Marina Papatriantafilou. Magnus, for your dedication and willingness to go beyond your obligations to help us perform our best. Marina, for your input on our presentation and advice with respect to our group's dynamics. We would also like to thank professor Aarne Ranta, for taking his time to talk to us and show us his work, and our examiner Arne Linde, for the feedback given during the course.

Emarin Andersson, Gothenburg, May 2016
Benjamin Eriksson
Sofia Holmberg
Hossein Hussain
Lovisa Jäberg
Erik Thorsell

Glossary

AMI	Acute Medical Index (Guide to what questions the EMD are to ask the caller.)
CC	Concept-Concept is an iKnow data structure that consists of two related concepts.
Concept	A concept is an iKnow data structure that represents a word with meaning, i.e. not fillers.
CRC	Concept-Relation-Concept is an iKnow data structure that consists of two concepts and a relation between them.
EMCC	Emergency Medical Communication Centre
EMD	Emergency Medical Dispatcher, also called SOS operator.
Fillers	A word that does not add information to a sentence. Examples are: “Öh”, “Ehrm” and “Eh”.
iKnow	A tool for Natural Language Processing
NLP	Natural Language Processing
Regex	Regular expression, a way to describe sequences of characters often used to find sequences of certain interest.
RN	Registered Nurse
SOS Alarm	The company responsible for Sweden’s emergency medical communication centres.

Contents

1 Introduction 1
- 1.1 Background 1
- 1.2 Purpose 2
- 1.3 Scope 2
- 1.4 Problem specification 3
  - 1.4.1 How to define the quality of a call 3
  - 1.4.2 How the transcription affects the analysis' quality 3

2 Methodical Considerations for Analysing Emergency Calls 5
- 2.1 The ethical aspect of the project 5
- 2.2 The origin of the assessment protocol 5
- 2.3 The criteria for the choice of software 6
  - 2.3.1 Software chosen for text analysis 6
  - 2.3.2 Software chosen for speech recognition 7

3 Theory 9
- 3.1 Natural Language Processing 9
- 3.2 Regular Expressions 10
- 3.3 Representing words and phrases using vectors 10
- 3.4 Statistical validation 10
  - 3.4.1 K-fold cross validation 11
- 3.5 Graph distance algorithms 11
  - 3.5.1 Average distance in graphs 11
  - 3.5.2 Distance to centre in graphs 11
  - 3.5.3 K-nearest neighbour 12
- 3.6 Applications used for speech recognition 12
  - 3.6.1 iSpeechWriter 12
  - 3.6.2 Speech Recognition on Docs 12
- 3.7 Applications used to analyse the transcribed calls 13
  - 3.7.1 iKnow 13
  - 3.7.2 Gensim 14
  - 3.7.3 CoreNLP 14
  - 3.7.4 Grammatical Framework 14

4 Implementation 15
- 4.1 Analysis of the assessment protocol 15
- 4.2 Manipulation of the transcribed calls 16
  - 4.2.1 Manipulation for transcription consistency 17
  - 4.2.2 Removing the caller from the calls 17
- 4.3 The functionality of iKnow 17
  - 4.3.1 The iKnow Similarity function 17
- 4.4 Classification of calls using Score.py 18
- 4.5 Validation and optimisation using Threshold.py 18

4.6 Using gensim to vectorise calls . . . 19
4.7 Developing Swedish modules for CoreNLP . . . 19
4.7.1 Part of Speech-tagging . . . 19
4.7.2 Neural network dependency parsing . . . 20
4.8 Answering the KI-Protocol using Solve.py . . . 21
4.8.1 Solvers using the output of CoreNLP . . . 21
4.8.2 Solvers using word frequency analysis . . . 21
4.8.3 Categorising calls as high or low quality using Solve.py . . . 22
4.9 Implementing Grammatical Framework . . . 23
4.9.1 Python program to solve question S 2.12 . . . 23
4.10 Testing of speech recognition . . . 24

5 Results 25
5.1 Results using iKnow's Similarity function . . . 25
5.1.1 iKnow Similarity Concept Search . . . 25
5.1.2 iKnow Similarity CRC Search . . . 26
5.2 Gensim . . . 27
5.2.1 Gensim similarity search . . . 27
5.3 Solve.py . . . 28
5.3.1 Answer questions in the KI-protocol . . . 28
5.3.2 Classify calls as low or high quality . . . 29
5.4 Grammatical Framework . . . 29
5.5 Speech recognition . . . 30
5.5.1 iSpeechWriter . . . 30
5.5.2 Speech Recognition on Docs . . . 30
5.5.3 Analysing calls transcribed with a speech recognition software . . . 30

6 Discussion 33
6.1 Discussion of the results produced by iKnow . . . 33
6.2 Discussion of how manipulation of calls affect iKnow . . . 33
6.3 Classifying calls using gensim . . . 34
6.4 Solve.py's ability to answer questions . . . 34
6.5 Solve.py's ability to classify calls . . . 34
6.6 Discussion of the results produced by GF . . . 35
6.7 The possibility to classify speech recognition calls . . . 35
6.8 Sources of error and future recommendations . . . 35
6.8.1 Increasing the number of calls . . . 35
6.8.2 Quantifying quality correctly . . . 36
6.8.3 The importance of a good corpus . . . 36
6.8.4 Using the concept of similarity to answer questions . . . 36
6.8.5 Social, economic and environmental impacts . . . 36

7 Conclusion 37

Bibliography 39

A Appendix I
A.1 SOS Alarm quality assessment protocol . . . II
A.2 KI-protocol . . . III

1 Introduction

Natural Language Processing (NLP) is the process of analysing and deriving meaning from a text. It is one of the most difficult tasks in the field of computer science [1]. This thesis discusses the possibility to utilise NLP to digitally analyse emergency calls received at the Swedish Emergency Medical Communication Centres (EMCCs) operated by SOS Alarm.

1.1 Background

Every year, more than 3 000 000 calls to the Swedish emergency number 112 are received at the Emergency Medical Communication Centres (EMCCs) run by SOS Alarm, and the number is increasing [2]. SOS Alarm consists of fifteen call centres all around the country. A call is answered by any SOS operator at one of the call centres regardless of the position of the caller. It is the SOS operator's job to determine what has happened, where it has happened, what help is needed and, in case of a medical emergency, set an ambulance priority [3]. A second SOS operator can connect to the call as a listener and help out with notifying the necessary emergency services such as ambulance, fire department or police. To aid the SOS operator, a Registered Nurse (RN) can connect to an ongoing call, and answer more specific medical questions. Despite this, the SOS operator occasionally fails to identify acute medical situations. For instance, Lindström et al. found that an SOS operator only identifies about 50-70 % of situations involving patients suffering from cardiac arrest, acute myocardial infarction, or stroke [4].

The act of sorting and prioritising patients is called triaging. To perform triage when one cannot see the patient is a notoriously difficult task [5, 6, 7]. Add to this finite resources in the form of ambulances and for example a panic stricken or intoxicated caller and it is easy to see that the position as an SOS operator is demanding [8].

Figure 1.1: A SOS operator at work. © SOS Alarm

SOS Alarm has high staff turnover [9]. The recruitment process and education of a new SOS operator can take over 6 months [10], which is why SOS Alarm is keen to keep their experienced staff for as long as possible.

The importance of the SOS operator's work cannot be overstated and it is of great interest to SOS Alarm, and the Swedish population, to minimise the number of incorrect assessments. An incorrect assessment costs money [11] and if an ambulance is sent when not really needed it is possible that a patient in actual need of help is unable to get help in time. In order to decrease the number of incorrect assessments, all SOS operator's use the Acute Medical Index (AMI). This is a guide to what questions to ask the caller, and what priority to assign. There are four different priorities where prio 1 is the most urgent, and prio 4 is the least urgent, priority [3].

SOS Alarm also analyse calls retrospectively to find issues in old calls that can be used to improve future calls. In a low quality – or "bad" – call it is important to the SOS operator and SOS Alarm to pinpoint what went wrong and how to improve. Likewise a high quality – or "good" – call can serve as an example. For the time being, the analysis of incoming calls is something of an evaluation of each person working at SOS Alarm, with the purpose of improving the way the operators approach their callers. This is currently done by manually analysing a random set of five calls received by each operator, in a group. The discussion concentrates on a set of predefined questions that can be found in appendix A.1. This means only a small subset of the total number of calls are analysed. An automated process would be beneficial to SOS Alarm as more calls could be analysed and problematic calls could be caught and further studied.

Karolinska Institutet (KI) has an ongoing research project involving SOS Alarm, Lindholmen Science Park and InterSystems, dedicated to exploring the possibilities for digitally analysing emergency calls received at SOS Alarm. Katarina Bohm at KI [12] has developed an assessment protocol (appendix A.2) – from now on referred to as the KI-protocol in this thesis. The KI-protocol consists of 25 questions designed to measure the quality of a medical emergency call.

1.2 Purpose

The purpose of this thesis is to offer an in-depth computer science angle concerning the possibility of digitally analysing incoming calls at SOS Alarm. I.e. classifying the calls in accordance to their quality, as is determined by the KI-protocol. This thesis will further evaluate whether it is possible to automate the analytical process. This includes, but is not limited to, evaluating the following questions:

- If the KI-protocol contains questions answerable by a computer, or if it is possible to improve the questions to better be suited for digital analysis?
- Available software to be used for digital analysis?
- If it is possible to digitally transcribe an emergency call using a speech recognition software?

1.3 Scope

Of the 3 000 000 calls received at SOS Alarm every year, approximately 30% are calls categorised as medical calls [13]. This amounts to approximately 2 500 medical calls per day. SOS Alarm has provided 67 manually transcribed, authentic, emergency calls for assessment. These include 50 high quality and 17 low quality calls, as classified by a team led by Katarina Bohm using the KI-protocol. The thesis will focus on these calls, but it is important that the result generalises well to a larger set of calls.

The KI-protocol will be used to evaluate all calls. Should any software be able to give further quality indicators those will be noted and, if time permits, used.

To test the medical accuracy of the KI-protocol is beyond the scope of this thesis. The testing will therefore focus on finding out *if the questions in the KI-protocol can be answered by a computer*, not whether or not the questions are the right questions to ask from a medical perspective.

Implementing any system will cost money. The thesis will not take any financial aspects into consideration.

1.4 Problem specification

For the purpose of structure and manageability, the project has been divided into two sub tasks. The first task focuses on how to define the actual quality of a call. The second task focuses on how the transcription quality affects the analysis.

1.4.1 How to define the quality of a call

In order to verify the correctness of a future program's ability to answer the questions, a solutions manual will be used. This manual was provided by Bohm and contains the answer to each question in the KI-protocol, for each call.

It is important to note that the quality of a call is independent of the outcome of said call. It is solemnly the SOS operator's ability to assess a call that is to be classified.

1.4.2 How the transcription affects the analysis' quality

How a call is transcribed will most likely affect a program's ability to "interpretate" the information in the text. A human can "read between the lines" and "fill in the blanks", but this might prove more difficult for a computer. A developer focused on NLP has to take this into consideration. Testing will have to be done in order to find out what makes a transcription good and where the limit of a software's ability to understand text is drawn.

The calls provided are formatted in similar, but not identical, ways. Tests will be performed to determine if this affects the analysis.

2

Methodical Considerations for Analysing Emergency Calls

Chapter 1 introduced the intricate problem of analysing emergency calls. This chapter will take a look at the ethical aspect of the project and the KI-protocol will be introduced. Further the criteria for, and the final choice of, software is presented.

2.1 The ethical aspect of the project

This thesis required handling of sensitive data which is why the ethical aspect of the work was of utmost importance. The transcribed calls entrusted to the project were anonymised, but it was important to consider unexpected scenarios. The project guaranteed complete confidentiality and that no person could trace any transcribed call back to its origin.

To ensure confidentiality the data was stored on USB flash drives, and these were kept in a safe when not in use by anyone in the research group. The safe also held a log book where each member wrote the time and date they used a particular drive. The drives were all AES encrypted [14]. This ensured that if a drive was lost, no one, without the right tools and passwords, would be able to acquire the data.

Additional precautions had to be taken in order to ensure the privacy of the callers. Throughout the project the mindset of everyone involved had to be: “If I knew this person, would I be able to tell whether he or she made this call, given this information?”. Could information in a table give someone away? A quote? An appendix note? No information that fit the previous descriptions could be allowed into the upcoming thesis.

2.2 The origin of the assessment protocol

The KI-protocol in appendix A.2 was developed by Katarina Bohm and her team at KI and is based on the article “Barriers and opportunities in assessing calls to emergency medical communication centre, a qualitative study” [4]. The protocol consist of a set of questions suitable for determining the quality of a given call.

Bohm has given the questions weights to stress positive and negative effects on a call by a particular circumstance. E.g. a positive answer to question number E 1.4 – “Language barriers of the caller?” has a negative weight since language barriers will prevent the assessment of the situation done by the SOS operator. On the other hand, if the operator asks questions about what the caller talks about, if question Z 3.7 –“Does the SOS operator ask questions about what the caller has told?” is answered with “Yes”, then the quality of the call increases. The question has therefore been given a positive weight. Some questions can be considered neutral, i.e. they neither lower or raise the quality of a call. E.g. question B 1.1 – “Where is the call made from?”.

This project will not focus on the correctness of the KI-protocol, as mentioned in section 1.3. Despite this, it is important to keep in mind that the KI-protocol can be a possible source of error. A change of either a question or a weight might result in a dramatically different outcome in the accuracy of the used NLP algorithms.

2.3 The criteria for the choice of software

In an early stage of the project, a large portion of time was spent to research what software to be used for the analysis. It was clear that the software had to be able to analyse text and present some sort of result suitable for answering our questions. Besides, several aspects had to be taken into consideration; these included (but were not limited to) the following:

- Does the software support analysing Swedish texts?
  If not, is it possible to implement such functionality?
- Is the software free of charge?
  If not, are there resources to be spent on additional software?
- Is the software available for the desired platforms?
  I.e. can it run on the hardware available? This project made use of various Linux distributions, Mac OS X and Windows.
- How much time seems to be required in order to learn how to use the software?

2.3.1 Software chosen for text analysis

Since the beginning of the project, InterSystems' iKnow had been set as one of the applications to be evaluated. iKnow is able to analyse Swedish texts and industry experts, as well as the InterSystems' developer community, were at hand to help with any issues that could arise. iKnow is, however, not free of charge.

When looking for additional software, problems arose. Several free and open source [15] applications for natural language processing could be found online but it proved difficult to find applications supporting analysis of text written in Swedish. There were, however, several tools for analysing English. OpenNLP [16], Grammatical Framework (GF) [17] and CoreNLP [18] were all candidates for alternative tools. Initial research on each of the applications resulted in CoreNLP and GF as the alternatives to iKnow.

CoreNLP and GF use different approaches to text analysis than iKnow. In order to compare iKnow to a more similar software, gensim [19] was added to the software list. Gensim is an open source Python library that uses a statistical approach to analysing text, in comparison to iKnow's predefined-knowledge-about-language approach.

CoreNLP is a free of charge, well documented, cross-platform software and seemed to be the easiest one to adapt to the needs of the project. GF has the advantage of being developed by a professor at Chalmers and the University of Gothenburg, Aarne Ranta [20]. Along with Inari Listenmaa [21], PhD student, Aarne could give hands-on advice and in depth information about the software.

2.3.2 Software chosen for speech recognition

Speech recognition was not the major focus of the project. Despite this, it was given a lot of consideration. Several applications were tested according to the questions in section 2.3. A lot of different software was available in English but finding a software suitable for Swedish proved to be more difficult.

Recommendations and reviews on the Internet [22] was useful as a guideline when sorting out candidate software. In the end, two programs were deemed good enough for the speech recognition tests to come; iSpeechWriter [23] and Speech Recognition on Docs [24]. The testing for accuracy was done by dictating calls using the different applications. The results of these tests can be found in section 5.5.

3

Theory

The previous chapter presented some of the considerations acquainted with the earlier stages of the project. Chapter 3 gives a brief introduction to a handful of topics, all important to grasp in order to understand what makes a software feasible to answer questions. The beginning of the chapter introduces NLP as a concept, along with the difficulties that comes with analysing texts and a common method to extract information from a text (regex). Then follows a concise explanation of word vectorisation, statistical validation and graph distance algorithms. Finally, each software that was presented in Chapter 2 is described.

3.1 Natural Language Processing

Time flies like an arrow; fruit flies like a banana.

Speaking and understanding a language is something that comes naturally to humans and it is not uncommon that people speak more than one language. Despite this, transferring the concept and notion of language to a computer seems to be difficult [1]. The difficulties can be found both in spoken language and written text. How is a computer supposed to be able to differentiate between "I scream" and "ice cream"? Or understand who has the telescope in the sentence: "Bob saw the man on the mountain with a telescope."

Even when the phonetic [25] – the study of speech sounds – part of a language is stripped away, scientists are faced with difficulties in forms of morphology [26] – the study and description of how words are formed in language. As an example one can consider the word "unlockable" that can refer to a door that can either be unlocked or not locked at all.

Further issues arise when syntactical ambiguity is present, that is when a sentence can be interpreted in more than one way [27]. "Flying planes can be dangerous.", could mean that it is either dangerous to fly a plane or that planes - which are flying - are dangerous. It could also be the case that the same word have different meanings [28]. "Wood" could refer to either a piece of a tree or a collection of trees (a forest) depending on the context.

Even though the concept of language might be a difficult matter for a computer, there exist software today which is able to analyse, and act on, natural language input. Apple's Siri [29] and Google Now [30] are two examples of software which takes voice input and acts on the information given. Telling Google Now: "Remind me to write a Python script that parses natural languages at 6 pm.", will create a reminder for said thing at said time. Asking Siri: "Which is the best pizza in town?", will return a list of the top rated restaurants in town that serves pizza.

Knowledge about phonetics, morphology, syntax, semantics, pragmatics and discourse are all needed to analyse and process a language [31]. Using state machines, formal rule systems, logic and probability theory, along with other machine learning tools, the problems mentioned earlier can be seen as sub tasks solvable with a few algorithms.

3.2 Regular Expressions

A fundamental part of natural language processing is the regular expression [31]. Regular expressions are used to match expressions in a text that follows a certain pattern, as shown in example 3.1.

Example 3.1 *Phone numbers in the United States are commonly expressed in the formats (111)123-4567 or 111-123-4567. Both formats can be matched using the following regular expression:*
`\(?(\d{3})\)?-?(\d{3})-(\d{4})`.

In regular expressions backslash, `\`, is used either to “escape” a character or as the start of a “function”. For instance `\(` denotes a left parentheses whilst `(...)` are used for grouping. Likewise, `\d` is used to denote “any digit” whilst `d` matches the character `d`. The quantifier `{3}` matches exactly three of the character class specified (in this case `\d`, any digit) and `?` matches 0 or 1 of the previous group.

Describing the regular expression in example 3.1 in words could be done like this: *Match any sequence of characters that consists of three groups of digits where the first two groups have three digits each and the last one has four digits. The last group of digits should be separated from the second by a hyphen. The second group could be separated from the first by a hyphen. The first group could be surrounded by parentheses.*

Note that this regular expression is not perfect. It would for instance match the strings: (111)-123-4567, (111-123-4567 and, 111123-4567 which are not common phone number formats. Using regular expressions the text matching process can be automatised and done by a computer. A computer is able to process a lot more data than a human and the loss of accuracy is often seen as small in comparison to the increase in speed.

3.3 Representing words and phrases using vectors

A vector is very useful for representing how well certain words and phrases relate to each other. There are different techniques used to yield such vectors, but the end goal is the same: provide a numerical representation of how words and phrases relate.

The first program used in the project for the purpose of vectorising text was GloVe [32]. It was used to generate a word embedding that was later used when developing the Swedish modules for CoreNLP (section 4.7). GloVe was developed by the Stanford NLP Group in 2014. It uses a weighted least square model which is trained on global word-to-word co-occurrence counts. This ensures that GloVe makes good use of statistics. GloVe produces a word vector space that can be used to see the relation between words and phrases in the corpus.

Additionally, gensim [19] was used as a standalone application to answer the questions in the KI-protocol as well as classifying the calls. Gensim is a Python library developed by Radim Řehůřek and Petr Sojka. Short for “generate similar”, gensim is used to generate vectors that represents the similarity of documents in a given corpus.

3.4 Statistical validation

When developing a program dedicated to data analysis it is not suitable to test the program on the actual data it will later analyse. Because of this, a subset of the prospective data is used to train the program and test parameters. However, it is not enough for the program to return good results when analysing the subset. Unless the program generalises to an independent data set it is not good enough to be put into production.

3.4.1 K-fold cross validation

In order to verify that a program generalises well to new data, it is common to use some sort of cross validation. Cross validation lets the developer feed the program with a subset of data and test it against the remainder of the set in different ways [33].

When using k-fold cross validation the data set is divided into k subsets. The program is then given one of the k subsets as a validation set while the other $k-1$ sets are used as training sets. The program is scored each run and the average score overall k runs is considered the over all score.

3.5 Graph distance algorithms

The graph-distance algorithms can calculate distances between groups of vertices in a graph. However, the distance between a vertex and a group is not well defined. The question “Is Sweden closer to Asia or America?” could be answered in multiple ways. One approach would be to calculate the average distance to every country in Asia and America. Another method is to find the most central country in Asia and America, then compare the distance to them. The question can also be answered by looking at Sweden’s neighbours, are most of them Asian or American? The following sections will give a formal explanation of how these methods can be implemented.

3.5.1 Average distance in graphs

Consider a complete graph G, where each vertex is directly connected to every other vertex. The distance from a vertex v_s to a group of vertices V can be calculated as follows. Start by calculating the total distance T to all the vertices in V.

$$T = \sum_{v_i \in V} |v_i - v_s|$$

Then calculate the average distance d over all the vertices in that set. This will result in the average distance from vertex v_s to the vertices V.

$$d = \frac{T}{|V|}$$

3.5.2 Distance to centre in graphs

Instead of comparing the distance to all the vertices in a graph, the *centre* algorithm only consider the distance to the most central vertex in a graph. The most central vertex in a graph can be defined as the vertex with minimum eccentricity [34]. The eccentricity is defined as “The maximum distance between a vertex to all other vertices” [35]. For a directed complete graph the distance from a vertex to all other vertices is simply the weight of the edge connecting them. Thus, the eccentricity of a vertex v in a directed complete graph $G = (V, E)$ can be calculated as:

$$ecc(v) = max\{e_1, e_2, ..., e_n\}$$

Where e_i defines the weight of the edge between the vertex v and a connected vertex v_i. The central vertex v_c is then calculated as:

$$v_c = min\{ecc(v_1), ecc(v_2), ..., ecc(v_n)\}$$

3.5.3 K-nearest neighbour

Nearest neighbour algorithms classify elements in a set by examining their closest neighbours. The number of neighbours, k, being examined can be tuned to yield different results. It is not straight forward which k will give the best result for a certain data set. Cross-validation is usually a good method for finding good values for k (section 3.4.1).

The K-Nearest neighbour algorithm can also be applied to complete graphs. Having a complete graph G, and a vertex to be classified v_s, find the k closest vertices to v_s and add them to the set V. This can then be divided into subsets $V_{0...i}$, where each subset contains vertices of the same class. The unclassified vertex v_s is then classified in the same way as the vertices in the subset with the highest cardinality. The equation below shows the formal definition.

$$v_s \in max\{|V_0|, |V_1|, ..., |V_i|\}$$

Example 3.2 *Imagine an image of an animal. The goal is to classify it as either a cat or a dog by using the K-nearest neighbour. By using $k = 3$ the 3 most similar images in our training set are returned. $V = \{cat_1, dog_1, cat_2\}$. V is further divided into $V_{cat} = \{cat_1, cat_2\}$ and $V_{dog} = \{dog_1\}$. Since $max\{|V_{cat}|, |V_{dog}|\} = |V_{cat}|$, the new image is classified as a cat.*

3.6 Applications used for speech recognition

Speech recognition has become more popular as of late. Good examples are the widely popular virtual voice assistants on the market, for instance Cortana by Microsoft [36], Siri by Apple [29] and Google Now (Voice) by Google [30].

There are two different kinds of speech recognition systems available, speaker dependent and speaker independent speech recognition. Speaker dependent speech recognition requires training to become more accurate at speech to text conversation. Speech to text conversion samples need to be run through the system several times. This will teach the system to understand a specific accent and voice. Speaker independent speech recognition does not require training and is the opposite of speaker dependent speech recognition.

As mentioned in section 2.3.2 two speech recognition applications were singled out for testing: iSpeechWriter and Speech Recognition on Docs. Both of these applications are speaker independent voice recognition applications and are further described in section 3.6.1 and section 3.6.2.

3.6.1 iSpeechWriter

iSpeechWriter [23] is a speech recognition and text translation software for English, Swedish and several additional languages. The software is free of charge and can be used as a browser plugin, desktop application or as a website application.

iSpeechWriter can recognise most of the words which can be found in a Swedish dictionary, and register most words spoken in a dialogue. Words that do not exist in a dictionary, like foreign words or fillers, are not transcribed.

3.6.2 Speech Recognition on Docs

Speech Recognition on Docs [37] is an add-on developed by EFV-Solutions which uses the Google Web Speech API. The software is capable of transcribing from speech to text in multiple languages including Swedish. The add-on is speaker independent and integrated with the text editor Google Docs.

As this software is speaker independent some words become more difficult to transcribe than others. This unfortunately generates erroneous words and incorrect sentences. For some languages the user is able to select a particular dialect of said language. E.g. English speakers have the option of British, American or Indian dialects amongst others.

With unhurried and articulated speech, almost all words are transcribed correctly. The software handles a higher speech rate fairly well and registers a majority of the words. Background noise is a problem, especially when paired with a high speech rate. This may cause the software to omit words – or even whole sentences – if the background noise is too loud.

3.7 Applications used to analyse the transcribed calls

A good understanding of each application's capability and limits are important in order to utilise it to its full potential. As described in section 2.3, a handful of applications were deemed good enough for further testing and a brief description of each software is given below.

3.7.1 iKnow

iKnow is a tool for natural language processing that can structure text by using predefined knowledge of the language. Contrary to other NLP software iKnow uses a bottom-up approach instead of the more traditional top-down. The bottom-up approach uses knowledge about the structure of a language to extract information about the text. More specifically iKnow finds concept pairs and relations that binds them together.

Here is an example of how iKnow would parse the sentence: "The patient is having problems breathing". First of all it finds the `concepts`, these are words that carry information. In the sentence above the concepts would be: "patient","problems" and "breathing". The next step would be to find relations between the concepts. The combinations of concepts and relations in iKnow are called Concept-Relation-Concepts (`CRC`). In this example they would be: "patient having problems" and "problems breathing". More specifically, "problems breathing" is a Concept-Concept (`CC`) which is a subclass to `CRC`.

3.7.2 Gensim

Gensim is a Python library that can be used to find how similar two documents are based on all the words in a corpus. Gensim uses latent semantic analysis (LSA) to calculate these similarities. LSA is a method for finding relations between documents and words. The method works by extracting every word from the corpus and then map the documents to a high dimensional vector space. The idea of mapping the documents to a vector space is that they can easily be compared using cosine similarity [38].

Consider the three strings: A = "Natural Language Processing", B = "Statistical analysis" and C = "Analysis of natural language". For a two-dimensional vector space the strings would be vectorised as: $\vec{a} = (1.38, -0.87)$, $\vec{b} = (0.51, 1.19)$ and $\vec{c} = (1.89, 0.32)$. The table below shows the similarities, calculated using dot products, of the vectors. As table 3.1 shows, A and B are not similar while A and C are very similar, B and C are moderately similar.

Table 3.1: Similarities of vectors using gensim dot products

	$\vec{a}$	$\vec{b}$	$\vec{c}$
$\vec{a}$	1	-0.16	0.75
$\vec{b}$	-0.16	1	0.54
$\vec{c}$	0.75	0.54	1

3.7.3 CoreNLP

CoreNLP is a set of natural language analysis tools [39], developed at Stanford University and licensed under the GNU General Public License (GPL v3) [40]. Using 14 different annotators [41], CoreNLP can – amongst other things – give the base form of words (lemmas), their part of speech and describe how words in a sentence depend on each other. A more in depth explanation of the annotators used will be given in section 4.7.

When working with CoreNLP the user specifies which annotators and language models to use. This enables for a large variety of configurations and because of the licensing of the software, users are able to provide their own implementations of the various parts of the program. Given an arbitrary text as input, CoreNLP is able to process the text and return an output depending on the configuration of the software [42]. As default, an XML-file is printed with all the information asked for by the specified annotators.

3.7.4 Grammatical Framework

Grammatical Framework [17] (GF) is a programming language intended for multilingual grammar applications. It is developed at Chalmers University of Technology. Different parts of GF are licensed under different licenses (GPL [40], LGPL [43] and BSD [44]). As stated on the projects website, it is: "a special-purpose language for grammars, a functional programming language, a development platform for natural language grammars" and much more.

GF requires expert knowledge about a specific language in order to program the grammar to be used. GF comes with libraries for 37 languages (Swedish included) that are more or less ready to use. However the user must also learn how to use the GF interpreter. GF can be embedded in various applications, and it is possible to use pre-compiled GF code in both Haskell, Java and Python.

4 Implementation

This chapter builds on the theory presented in the previous chapter and explains how the different parts of the project were implemented. The beginning of the chapter focuses on the assessment protocol and the calls provided by Katarina Bohm. Thereafter the implementation of each application is described.

4.1 Analysis of the assessment protocol

In accordance to what was written in section 1.4.1, the only asset available for grading the calls was the KI-protocol (appendix A.2). During the early stages of the implementation of the program `Solve.py` (section 4.8), it became clear that the KI-protocol might not be suitable to classify the calls. It was expected that the individual questions in the KI-protocol would be well-suited to find a consistent mapping to judged high-quality versus low-quality calls, but this was not the case.

In order to verify that answering the questions in the KI-protocol gave the same results as the classification done for the complete calls by Bohm, a more thorough analysis of the assessment protocol and the solutions manual was conducted. This analysis gave unexpected results. It was expected that a high quality call would get a high score. And likewise a low quality call a low score.

In table 4.1 it is shown that the points are widely distributed over the calls. Of the 67 calls provided, two calls got 0 points but both were considered as "high quality calls" according to the information provided by Bohm in the beginning of the project.

Table 4.1: The result from the analysis of the solutions manual.

Score	# High Q	# Low Q
0	2	0
1	0	0
2	0	2
3	0	1
4	0	1
5	1	3
6	2	2
7	2	4
8	8	4
9	9	0
10	13	0
11	12	0
12	1	0
Total	50	17

When extrapolated from the result seen in table 4.1, it seemed reasonable that if a call received a score of 9 or higher in the testing it was to be considered to be of good quality. A call with a score below 9 could however neither be considered a good or a bad call since low scored calls were not as clearly polarised.

It was concluded that answering the questions in the KI-protocol was not enough to determine the quality of a call. Instead there must have been some other aspects to the classifications of the calls Bohm provided the project in the beginning. The project progressed by assuming that the classifications received was correct, but that these were not solemnly determined by answering the questions in the KI-protocol. The purpose of the project did however still remain the same, as described in section 1.2. A recommendation to the medical experts is thus to further investigate the individual questions in order to find a more consistent correlation between the judged quality of a call and the quality given for a call by letting a computer answer the questions.

Because of the poor correlation between the score of a call and the predefined quality classification, the analysis process was revised. iKnow and gensim were easily adapted to analyse the similarity between calls and was therefore used solemnly to analyse the quality of the calls without respect to the KI-protocol. The same approach was not possible for Solve.py and Grammatical Framework, which were still used to answer the questions in the KI-protocol in accordance to the original problem specification.

4.2 Manipulation of the transcribed calls

All the transcribed calls received from Katarina Bohm were not transcribed in exactly the same way. The majority of the calls did however follow the template seen in Fig. 4.1. Note that “XX” is added by whoever transcribed the call in order to protect the privacy of the caller.

```
Call ID - Call length: 3 min 42 s

Index
Dispatch (D)
Caller (C)
Nurse (N)
Not applicable [n/a]
[action/behavior/description]
(writer note)

D: SOS 112, what has happened?
C: Hello, my name is XX...
...
(Nurse joins the call.)
N: What has happened?
...
N: Bye
C: Bye
```

Figure 4.1: Example structure for an emergency call.

Manipulations of the calls was divided into two parts: manipulations for consistency and manipulations to tweak the results. Both are described below. The code for manipulations was written i Python and made use of regular expressions (section 3.2).

4.2.1 Manipulation for transcription consistency

As was mentioned in section 4.2, all the calls were not transcribed in the same way and, as a consequence thereof, lack in consistency. The following manipulations sought to make the transcriptions more consistent.

The transcriptions start with a header (Fig. 4.1 in section 4.2), but the header differ somewhat between the calls. The header can include call-id, call length and explanation of abbreviations. This information was removed before any tests to prevent it from interfering with the result.

Some transcriptions have comments surrounded by parentheses and square brackets. E.g. "(Nurse added)" and "[short break]". This information and the header has been added by the transcriber and is not considered part of the original call. As was described in section 3.2, regular expressions are useful to filter out such text. The following regular expressions were used for removal of transcriber comments: `\[.*\]` for square brackets and `\(.*\)` for parentheses.

4.2.2 Removing the caller from the calls

The KI-protocol states that repetition of the caller's statement by the dispatcher increases the quality of a call. Hence, in an ideal call all information present should be conserved if the caller is removed. To test this hypothesis a way to filter out the caller was implemented. The filter loops through a call, line by line. Each line that matched the regex: `[nNdD]:.*` was kept while the lines that did not match were discarded.

4.3 The functionality of iKnow

The iKnow software is rich in functionality and can be used in many different ways. The most prominent function for this project was `GetSimilar` [45]. To parse and further improve the results from this function two other programs were developed, `Score.py` and `Threshold.py`. These programs are explained further in section 4.4 and 4.5 respectively.

4.3.1 The iKnow Similarity function

The similarity function compares a specific document with all other documents in the same domain. The function returns an ordered list of all the other documents and their similarity score, a real number between 0 and 1.

Since the iKnow software is proprietary, the exact algorithms used are not public but based on the documentation some insight can be gained. When computing the similarity, `GetSimilar` searches for common properties in the documents. By default it searches for concepts, e.g. "Ambulance" or "Breathing". The function can also search for Concept-Relation-Concepts (CRC) which, contrary to searching for only concepts, would differentiate between "patient is breathing" and "patient is not breathing". However, "he is breathing" and "she is breathing" would also be considered different CRCs. In conclusion, searching for only concepts might make the documents seem more similar than what they really are, while searching for CRCs might make them seem more different. The configurations used in this project were:

1. `Simsrcsimple`, searching for concepts
2. `Simsrcsimple`, searching for concept-relation-concepts

To get the results from `GetSimilar` all calls were added to an iKnow domain. The function then produced a directed, weighted graph where each node represented a call and each edge represented the similarity between two calls. After the graph had been generated the calls were divided into training and validation sets according to the 10-fold cross validation method (section 3.4.1). The validation set was then sent to `Score.py` (section 4.4), where the calls were classified as of either high or low quality. To improve the classification the program `Threshold.py` (section 4.5) was used. This process is described in Fig. 4.2, and more thoroughly in section 4.5 and section 4.4.

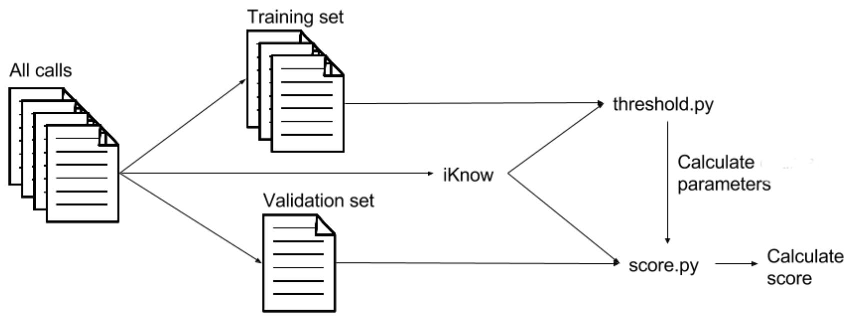


Figure 4.2: Flow chart showing how the calls were divided and processed by iKnow.

4.4 Classification of calls using Score.py

`Score.py` is a Python program that analyses a similarity graph and classifies the calls. Both iKnow and gensim produce these similarity graphs. `Score.py` uses different graph distance algorithms, explained in section 3.5, to classify the calls as of either high or low quality. The distance algorithms calculates the distance from an unclassified call to all the other calls. The methods are based on the idea that if a call is closer to the high quality calls than the low quality calls it should be considered a high quality call.

Moreover, `Score.py` is able to handle skewness in the data. By adding a threshold parameter, T, the accuracy of the classification can be improved. If the distance algorithm is biased towards good calls, the threshold value will counter-weigh that bias. Formally `Score.py` is calculating the following:

$$D_G(c) - D_B(c) > T \implies \text{High quality call}$$

Where $D_G(c)$ and $D_B(c)$ define the distance from the unclassified call to the high and low quality calls.

4.5 Validation and optimisation using Threshold.py

`Threshold.py` was used to analyse the training set of calls and calculate a threshold value for `Score.py` (section 4.4). `Threshold.py` uses the same method as `Score.py` to classify the calls. The difference between the programs is that `Threshold.py` tries multiple threshold values instead of just one. To do this it calculates an interval for the threshold to sweep over. The program then checks how many correct classifications were made, comparing the result to the solutions manual. The threshold value resulting in the maximum number of correct classifications is then chosen as the best threshold for the training set.

To make sure that the threshold is representative of all calls, the calculation is done multiple times on different training sets. 10 training sets are generated using the 10-fold method, as was described in section 3.4.1. Each of these sets will produce a best fit threshold value. These values are then averaged to return the final threshold value later used in `Score.py`.

4.6 Using gensim to vectorise calls

In order to make use of gensim it is necessary to create a program using this library. In section 3.3 a short description of gensim was given and below follows a description of how gensim is used in `gensim.py`.

Before the analysis could begin, a frequency dictionary had to be created. This was done using a `defaultdict(int)` where each word in a corpus was related to the number of times it occurred. The dictionary was then filtered by removing all words that only occurred once in the corpus.

The code in Fig. 4.3 shows the next four i in the process. `corpora.Dictionary()` gives each word in the corpus `texts` an id. `dictionary.doc2bow()` takes a call in the corpus as an argument and returns a list of the words in the call replaced by their ids. `models.LsiModel()` takes a corpus, a dictionary and a number as arguments. The function then returns an LSI object which is an object that represents the corpus as a vector with as many dimensions as was asked for in the call of the function. The fourth line generates a `MatrixSimilarity` object that is used for extracting similarities from the vector. Using this object it is possible to pairwise compare each call and generate a similarity graph just like iKnow does. The same method as shown in Fig. 4.2 can then be used to classify the calls.

```
dictionary = corpora.Dictionary(texts)
corpus = [dictionary.doc2bow(text) for text in texts]
lsi = models.LsiModel(corpus, id2word=dictionary, num_topics=67)
index = similarities.MatrixSimilarity(lsi[corpus])

```

Figure 4.3: Code used to generate the gensim word embedding.

The actual comparison of an unclassified call is done by adding a new vector to the vector space created earlier. Depending on which other vectors are closer to the new vector a similarity score is given. Adding a new vector is called querying and the query could be either a whole call or a sentence suitable to answer a specific question in the KI-protocol.

4.7 Developing Swedish modules for CoreNLP

CoreNLP worked as expected when analysing English texts, as well as texts written in any of the languages having already developed modules. Swedish was not amongst these and therefore the first task – to make CoreNLP a useful tool – was to develop the required modules. The annotators used in analysing the emergency calls are “tokenize”, “ssplit”, “pos” and “depparse” [41]; out of these “pos” and “depparse” required new language modules and the description of the development of these can be found in section 4.7.1 and 4.7.2.

4.7.1 Part of Speech-tagging

A software dedicated to part of speech-tagging parses a text in some language and assigns parts of speech to each word or token [46]. This is exemplified in Fig. 4.4.

On the website of CoreNLP there were several extensions listed [47] and one of the extensions was a Swedish POS-tagger developed by Andreas Klintberg [48]. By following the guide Klintberg posted on Medium [49], it was possible to recreate his work and it is this POS-tagger that is used in the analyses.

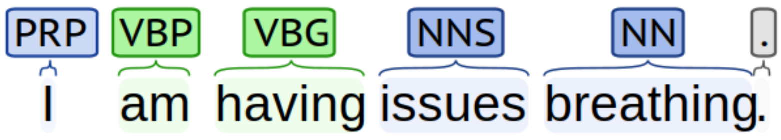


Figure 4.4: An example of part of speech-tagging.

Briefly, the development of the tagger consisted of training the CoreNLP software to recognise already tagged Swedish words (the list of words used in the training could be found at the Swedish Treebank [50]). The training class for the CoreNLP POS-tagger [51] was then used to train the module; the training resulted in a binary file that could be used by the software in order to tag an arbitrary – Swedish – text. Unfortunately the visualisation tool used in Fig. 4.4 does not work for texts in other languages than English, why a visual demo of the tagger is not present in the thesis.

4.7.2 Neural network dependency parsing

A dependency parser analyses a given sentence – and the sentence's grammatical structure – after which it returns the relationship between the words in the sentence [52]. If some words play a larger role in the sentence (which is often the case) the parser will also return how the other words in the sentence influence these. The example in Fig. 4.4 is used again in Fig. 4.5 but now also tagged using the default CoreNLP dependency parser.

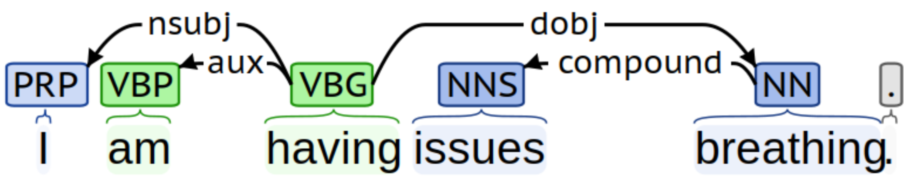


Figure 4.5: An example of dependency parsing, on top of POS-tagging.

Developing the dependency parser consisted of several steps. Unlike the POS-tagger described in section 4.7.1 there was no guide to follow, the training of the Swedish language module had to be done from scratch. Fortunately the documentation of the CoreNLP annotators was comprehensive and proved very useful.

Training a parser for an arbitrary language could be done using a dependency treebank [53], a word embedding model [54] and a CoreNLP specific "TreebankLanguagePack" [52]. Fortunately there was no need to recreate the language package used by default – for English – as it proved well suited for Swedish.

The dependency treebank was easily downloaded from the "Universal Dependencies" website. What proved more difficult was the creation of a word embedding model to represent the distribution of Swedish words in comparison to each other. This was finally achieved using GloVe [32] (section 3.3). Provided with a corpus, GloVe calculates how often words co-occur with one another in the corpus. The best results are achieved if the corpus is large and discusses the topic to be analysed.

GloVe included a demo used to create a word embedding model for English. When given a text file containing the first 100 000 000 characters from the English version of Wikipedia [55], GloVe returned a matrix – a word embedding model – representing the relations between the words. The same approach was taken when a word embedding model for Swedish was created. A corpus containing almost 27 000 000 characters was created by downloading the first 5 000 articles from the

Swedish version of Wikipedia using the Wikipedia API [56]. After processing the text (removing unwanted tags and assuring that only plain text remained in the file), GloVe was used in the same way as described earlier. This yielded a matrix representing how Swedish words relate to each other.

In combination with the files from the dependency treebank and the treebank language pack, the word embedding model was given as arguments to the CoreNLP dependency parser. This created and trained a model to be used in forthcoming tests.

4.8 Answering the KI-Protocol using Solve.py

`Solve.py` is a Python program able to answer each question in the KI-protocol (making use of a so called "solver" for each question) as well as to calculate the quality of a call. Some of the solvers in `Solve.py` make use of the output from CoreNLP while others perform various kinds of frequency analyses, of key words or phrases, on the transcribed calls.

4.8.1 Solvers using the output of CoreNLP

An example of when the output from CoreNLP was used in `Solve.py` can be seen in the code for solver `QS` (Fig. 4.6). Solver `QS` was used to answer question S 2.12 in the KI-protocol: "Are there problems with breathing, circulation or consciousness described in the call within 30 seconds?"

`QS` starts by generating a soup. A soup is a parsable data structure created by the Python library Beautiful Soup [57]. For every key word in the list `terms`, `QS` will iterate the soup and look for an occurrence of the key. If the key is found, `QS` will make use of the `character offset` tag, given by CoreNLP, and check how many characters precedes the key. If the offset is larger than a set value (in this case 130) more than 30 seconds of the call has most likely passed and the question is answered with a 0 (no). If the offset is less than the limit value the solver returns a 1 (yes). If none of the keys are present in the call, 0 is returned.

```python
def QS(document):
    soup = parseXML(document)
    terms = ["andas","andning","luft","andades","andningen",
             "medveten","medvetande"]
    for key in terms:
        for sentence in soup("sentence"):
            for w in sentence("word", string=key):
                charOff = w.parent.characteroffsetbegin.text
                wordOff = int(charOff)/5
                if(wordOff > 130):
                    return 0
                else:
                    return 1
    return 0
```

Figure 4.6: Code for solver QS in Solve.py.

4.8.2 Solvers using word frequency analysis

Some questions could be solved using word frequency analysis. Solvers using this approach counts the number of occurrences for specific words or phrases in a given call. Depending on which words

```
def QB(document):
    home = ["hemma","hemmifrån","lägenhet","hus","villa"]
    matchHome = []
    hospital = ["sjukvård","sjukhus","hemtjänst","hemtjänsten"]
    matchHospital = []
    public = ["jobb", "park","utomhus","tåg","buss","tunnelbana",
              "spårvagn"]
    matchPublic = []

    with open(document, 'r') as file:
        for line in file:
            cLine = callLine(line)
            cWords = re.sub("[^\wåäöÅÄÖ]", " ", cLine[1]).split()

            for w in home:
                if w in cWords:
                    matchHome.append( [w, 1, nLines] )

            for w in hospital:
                if w in cWords:
                    matchHospital.append( [w, 2, nLines] )

            for w in public:
                if w in cWords:
                    matchPublic.append( [w, 3, nLines] )

        if( len(matchHome) == 0 and len(matchHospital) == 0 and
            len(matchPublic) == 0):
            return 4

        maxMatch = max([matchHome, matchHospital, matchPublic],
                       key=lambda x: len(x))
        return maxMatch[0][1]
```

Figure 4.7: Code for solver QB in Solve.py.

or phrases occur the most in the text, the solvers would return different answers. The technique proved useful for answering questions having multiple answers. E.g. solver `QB` answers the question B 1.1 (“Where is the call made from?”). The code for the solver can be found in Fig. 4.7.

Solver `QB` scans through a call and counts occurrences of key words, much like solver `QS` in section 4.8.1. But since none of the meta data provided by CoreNLP is needed, scanning the document is faster done without first having to parse it using CoreNLP. The number of lists and the key words will vary, depending on the question. In solver `QB` there are several lists and they represent key words related to places where a call to SOS Alarm is likely to be made from. If a word in one of the key word lists occurs in the text it is appended to a designated list. At the end of the solver the length of the lists are checked. If all lists are empty the solver returns 4 (“not in proximity to the patient”), otherwise it returns the number associated with the longest list, i.e. the list that got the most hits when scanning the document.

4.8.3 Categorising calls as high or low quality using Solve.py

When all the questions have been answered using designated solvers, the result is summed up according to the weighting of each question as described in section 2.2. Depending on what score the call receives it is categorised with high or low quality. The threshold for whether a call’s score is high enough to be classified as a high quality call or not is set to 9. This is in according to what was discovered when the analysis of the KI-protocol was conducted (section 4.1).

4.9 Implementing Grammatical Framework

As mentioned in section 3.7.4, Grammatical Framework (GF) requires expert knowledge in its own specific programming language in order to be used efficiently. In order to *fully* utilise GF, a good knowledge of proper grammar in the natural language studied is also crucial.

Early in the testing stage of GF, it was clear that the program could not read the transcribed calls if they were not first edited. To begin with, the program had to recognise all words in a call or it would not be able to output any results that could further be used. This was very problematic because there were many words that did not appear in the GF's dictionary that was used, but almost always appeared in a typical SOS call. Fillers, "SOS", names of streets and addresses, along with the XXs used to anonymise the calls, all proved problematic to handle. A separate word list had to be written, and all the unknown phrases added. The words also had to be grammatically categorised. The lack of linguistic knowledge in the project group made this even more tedious and only one call was edited and used for testing.

In addition to the lack of words in the predefined word list, GF was unable to parse any punctuation or capital letters. This also meant that GF had to be fed with a single sentence at a time. When manually editing the transcribed calls, it was therefore necessary to add a newline after each sentence. The two columns below shows how this adaptation had to be done. The left column shows the original text and the right shows the input sentences that could be fed to GF, one at a time.

```
D: Ja, vad är det som har hänt dårå?      - vad har hänt där
C: Ehm, det har inte hänt så mycket       - det har inte hänt mycket
vi har en man som har kommit hit på       - vi har en man som har kommit hit på
morgonen och har eh, ont över             morgonen
bröstkorgen, ont över bröstet.            - har ont över bröstkorgen
                                          - ont över bröstet
```

To test GF, a program in Python was written. The program was used to answer two of the questions in the assessment protocol: J 2.3 ("Is there a description of how the patient looks in the call?") and S 2.12 ("Are there problems with breathing, circulation or consciousness described in the call within 30 seconds?").

Question S 2.12 was chosen because it was deemed one of the more important questions in the KI-protocol. Question J 2.3 was chosen because it tests the grammatical abilities of Grammatical Framework. A description of how question S 2.12 was solved is described in section 4.9.1, question J 2.3 was solved in a similar way.

4.9.1 Python program to solve question S 2.12

In Fig. 4.8, the phrase: "mannen andas" ("the man is breathing") is shown as a parse tree, generated by GF. The root `PhrUtt` denotes a phrase or sentence being read. The function `PredVP` searched for the relations between nouns (NP), verbs (VP) and declarative clauses (CL). The function `PredVp` found that `UseN = man_N` was the noun, `UseV = breathe_V` the verb, and `UseCl = PPos` that it was a positive phrase. Question S 2.12 will be answered with "No", since the man does not have a problem with his breathing.

The parse tree for the phrase: "mannen kan inte andas" ("the man cannot breathe") is shown in Fig. 4.9. The function `PredVP` searched for the relations between nouns, verbs and declarative clauses, as described above. In this figure the function `PredVp` found that `UseN = man_N` was the noun, `UseV = breathe_V` the verb, and `UseCl = PNeg` that it was a negative phrase. Question S 2.12 will be answered with "Yes", because the man has a problem with his breathing.

In the phrase "hur andas mannen" ("how is the man's breathing"), in Fig. 4.10, `PredVp` found that `UseN = man_N` was the noun, `UseV = breathe_V` the verb, and `UseQCl = PPos` that is was a positive phrase in a question. Since no answer was given to question S 2.12, the function will return "No".

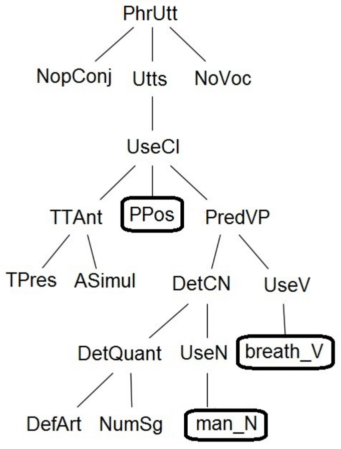


Figure 4.8:
"mannen andas".

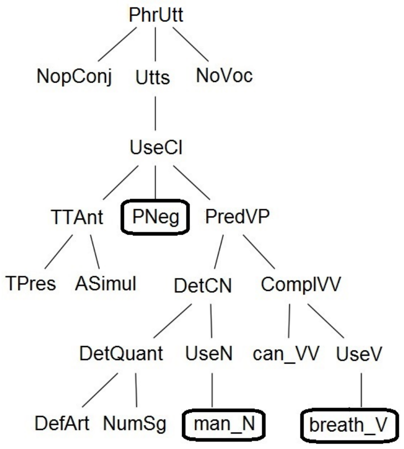


Figure 4.9:
"mannen kan inte andas".

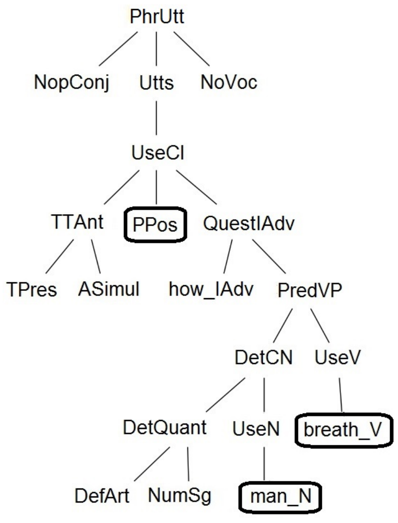


Figure 4.10:
"hur andas mannen".

4.10 Testing of speech recognition

Before any testing could be done, it was important to determine what tests were appropriate. As mentioned previously (section 3.6), there were two speech to text candidates: iSpeechWriter and Speech Recognition on Docs.

The tests were performed with one person reading either the dispatcher's or the caller's part in the conversation in the transcribed calls. A conversation between two persons was not done because it is not possible to tell who said what in the conversation after the tests were made. At the same time this was recorded by either iSpeechWriter or Speech Recognition on Docs. In order to protect the sensitive text material, tests were done by two people in the thesis group, in a secluded room. The software then provided the results in form of a text, which could be compared with the actual text of the dialogue. The tests were done with: clear speech, unclear speech and speech with background noise.

5
Results

As was mentioned in section 4.2.1 and section 4.2.2, the calls at hand were manipulated. At first to achieve consistency over all calls and furthermore the caller was removed from the calls. Table 5.1 show all tests performed, denoted with a label, which sets of calls were used, what type of result the tests give and where they can be found.

Table 5.1: Table showing all tests performed.

Label	Consistency	No caller	Type of result	Section
iKnow-C	✓	✓	Classification	5.1.1
iKnow-CRC	✓	✓	Classification	5.1.2
gensim	✓	✓	Classification	5.2.1
Solve.py-Q	✓	✓	Answers to questions	5.3.1
Solve.py-C	✓	✓	Classification	5.3.2
Grammatical Framework			Answers to questions	5.4
iSpeechWriter	✓		Transcribed Call	5.5.1
SRD	✓		Transcribed Call	5.5.2
SRD iKnow	✓*		Classification	5.5.3
SRD gensim	✓*		Classification	5.5.3
SRD Solve.py-C	✓*		Classification	5.5.3
SRD Solve.py-Q	✓*		Answers to questions	5.5.3

* Denotes that the call was transcribed using Speech Recognition on Docs (SRD).

5.1 Results using iKnow's Similarity function

The similarity test uses iKnow to determine how similar two calls are, a more thorough explanation of this mechanism can be found in section 4.3.1. The following sections will present results achieved by using different combinations of configurations for the iKnow similarity function, graph distance algorithms and modifications of the calls. During testing it was concluded that iKnow was not suitable to answer specific questions in the KI-protocol why these specific results are omitted from the report, a short discussion on this is given in section 6.8.4

5.1.1 iKnow Similarity Concept Search

In the first experiment iKnow's similarity function was configured to search for entities. The calls were modified using both the consistency method (section 4.2.1) and the caller removal method (section 4.2.2). Table 5.2 shows the number of calls that were correctly classified. A call is classified in accordance to Bohm's classifications (section 4.1). Table 5.3 shows the thresholds used to acquire the scores in table 5.2. The columns *Average*, *Centre* and *KNN* refer to the different graph distance algorithms mentioned in section 3.5.

Table 5.2: Number of correct classifications, out of 67

Modification	Average	Centre	KNN
Consistency	67	56	54
No Caller	60	53	52

Table 5.3: Thresholds used to generate table 5.2

Modification	Average	Centre	KNN
Consistency	0.987	0.0748	2
No Caller	3.90	-0.0210	2

As table 5.2 shows, the "average distance" method gives the highest score. The table also shows that calls modified for consistency score better than those without the caller. The confusion matrices below, table 5.4 and table 5.5, show the difference between the two modifications, both using the average distance algorithm.

Table 5.4: Confusion matrix for consistency manipulated calls.

		Predictions	
		High	Low
Actual Quality	High	50	0
	Low	0	17

Table 5.5: Confusion matrix for calls without caller.

		Predictions	
		High	Low
Actual Quality	High	46	4
	Low	3	14

The effectiveness of the *KNN* algorithm depends on the value of k. Fig. 5.1 shows how the *KNN* algorithm is affected by different values of k. The y-axis shows how many calls where classified correctly and the x-axis shows the value of k.

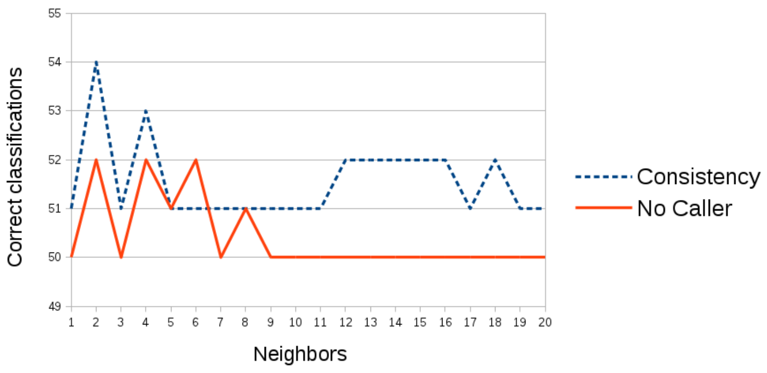


Figure 5.1: Plot showing how the score from *KNN* varies depending on k.

5.1.2 iKnow Similarity CRC Search

For this experiment iKnow was configured to search for CRCs, instead of concepts as in the previous experiment. Similarly table 5.6 shows the number of calls correctly classified and table 5.7 shows the corresponding thresholds.

Table 5.6: Number of correct classifications, out of 67

Modification	Average	Centre	KNN
Consistency	62	52	61
No Caller	54	48	57

Table 5.7: Thresholds used to generate table 5.6

Modification	Average	Centre	KNN
Consistency	0.0663	-0.0126	4
No Caller	-0.116	-0.0257	1

Fig. 5.2 shows how the *KNN* algorithm is affected by different values of k. In comparison to Fig. 5.1, the calls modified for consistency got a higher score when iKnow searched for CRCs. It is also worth noticing that the difference between the curves is greater for lower values of K, in Fig. 5.2 than in Fig. 5.1.

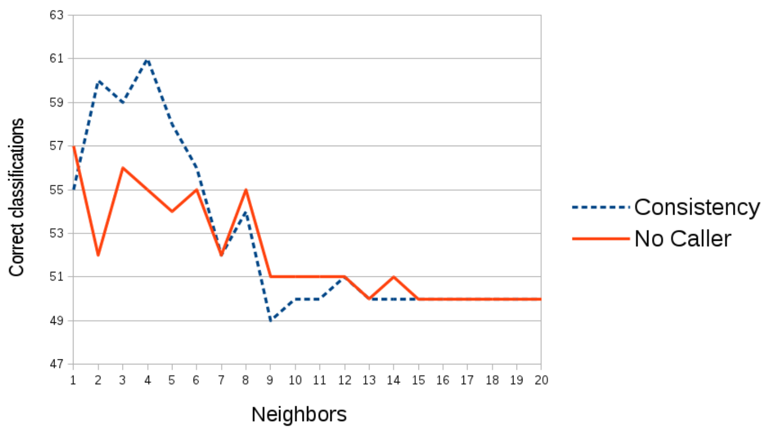


Figure 5.2: Plot showing how the score from KNN varies depending on K.

5.2 Gensim

By analysing the vectors generated by gensim it is possible to tell how similar a call is to a given query. These queries can either be full calls, in which case the vectors represent how similar two calls are, or they can be sentences related to a question in the KI-protocol, as was mentioned in section 4.6. Querying gensim with answers suitable to answer the questions in the KI-protocol did however not yield any usable results. These results are therefore omitted from the report, but a short discussion on this is given in section 6.8.4.

5.2.1 Gensim similarity search

This method closely resembles the iKnow similarity search mentioned in the previous chapter. Table 5.8 and 5.9 shows the scores and threshold acquired from gensim. It is worth noting that for gensim, in contrast to iKnow, *KNN* is the best performing graph algorithm.

Although both the *average* algorithm and the *centre* algorithm got the same score, their classifications differed immensely, as can be seen in table 5.10 and 5.11 below. The *centre* algorithm was

Table 5.8: Number of correct classifications, out of 67

Modification	Average	Centre	KNN
Consistency	57	52	64
No Caller	54	54	62

Table 5.9: Thresholds used to generate table 5.2

Modification	Average	Centre	KNN
Consistency	-4.392	-0.1901	4
No Caller	-14.51	-0.1273	2

better at classifying high quality calls while the *average* method was better at classifying the low quality calls. The plot in Fig. 5.3 shows the result of the *KNN* algorithm. It can be seen that the *KNN* algorithm performs best for lower values of k, just like the case with iKnow. On the other hand this plot is smoother than the iKnow plots which is further discussed in section 6.3.

Table 5.10: Classifications using *Average* algorithm.

		Predictions	
		High	Low
Actual Quality	High	38	12
	Low	1	16

Table 5.11: Classifications using *Centre* algorithm.

		Predictions	
		High	Low
Actual Quality	High	41	9
	Low	4	13

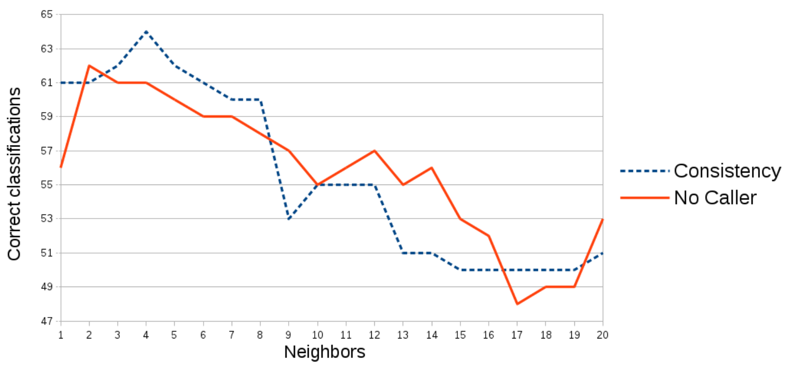


Figure 5.3: *KNN* algorithm using gensim data

5.3 Solve.py

Due to the nature of `Solve.py` it is suitable to not only classify calls as low or high quality calls, but also answer individual questions in the KI-protocol (section 4.8).

5.3.1 Answer questions in the KI-protocol

76% of the answers matches the answers in the solutions manual provided by Bohm, when running `Solve.py` with the calls manipulated for consistency (as described in section 4.2.1). Running the program with the calls where the caller is not represented (as described in section 4.2.2), 68% of the answers matches the answers in the solutions manual. The result for each solver, when run with the calls manipulated for consistency, can be seen in Fig. 5.4. In Fig. 5.4 the orange bars are results for solvers making use of the output from CoreNLP, the blue bars are results for solvers using other techniques.

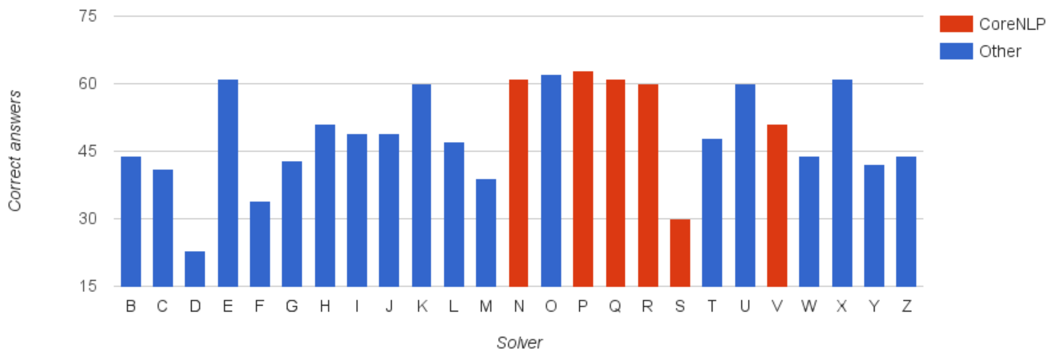


Figure 5.4: Results for Solve.py when run with calls manipulated for consistency.

In Fig. 5.4, two results stand out: `D` and `S`. Question D 1.3 asks if the caller is able to see or hear the patient. Question S 2.12 asks if the operator ensures the patients ability to breathe and if the patient is conscious or not. This is discussed more in section 6.4.

5.3.2 Classify calls as low or high quality

Looking at the classification of the calls as low or high quality calls, `Solve.py` classifies 49 calls as low quality calls and 18 calls as high quality calls when the calls are manipulated for consistency. When the caller is removed, `Solve.py` classifies 66 call as low quality calls and 1 call as high quality. As mentioned in section 1.3, there are actually 50 high quality calls and 17 low quality calls available. This means that the accuracy of `Solve.py` is 49% and 27% respectively. Also recall that the threshold for whether a call is of high quality or not is set to a score of 9 or above (section 4.8.3).

How the classification of the calls compares to the classification made by Bohm can be seen in the confusion matrices in table 5.12 and 5.13.

Table 5.12: Confusion matrix for consistency manipulated calls.

		Predictions	
		High	Low
Actual Quality	High	17	33
	Low	1	16

Table 5.13: Confusion matrix for calls without caller.

		Predictions	
		High	Low
Actual Quality	High	1	49
	Low	0	17

5.4 Grammatical Framework

Grammatical Framework was able to answer the two questions asked, according to section 4.9. However – as also discussed in section 4.9 – the program was only used to look for answers in one call, and the call had to be manually rewritten for GF to work at all. A brief discussion on the matter can be found in section 6.6.

5.5 Speech recognition

Two speech recognition applications were tested, in accordance to what was discussed in section 2.3. When the tests had been done it was concluded that Speech Recognition on Docs (section 3.6.2) was the best suited application, of those two. Two complete calls were then transcribed and analysed using the add-on. The result of these tests are shown in section 5.5.3.

5.5.1 iSpeechWriter

There were certain words, like foreign words or fillers, which were not recognised by the iSpeech-Writer. As seen in table 5.14 the software was unable to recognise certain words and therefore they was not transcribed. Sometimes the remaining sentence after an "invalid" word would not be registered at all.

Table 5.14: A test run of a dialogue between an SOS operator and a caller, using iSpeechWriter.

Original call	**Transcribed call**
SOS 112, vad har inträfat?	Sos 112 vad har inträffat
Mm-	N/A
Är ni på väg nu eller?	på väg nu eller
Vet du vilket spår ni kommer in på?	vet du vilken ordning kommer in på
Nä, okej.	N/A

5.5.2 Speech Recognition on Docs

Speech Recognition on Docs is able to transcribe almost all words correctly if the speaker articulates well. Talking fast works decently as the program registers the majority of the words. When it comes to noise in the background, the program misses a lot of the words. Sometimes it will not register a single word if the background noise is too loud. As can be seen in table 5.15, the program is not transcribing any fillers but is successfully transcribing the majority of the sentences.

Table 5.15: A test run of a dialogue between an SOS operator and a caller, using Speech Recognition on Docs.

Original call	**Transcribed call**
SOS 112, vad har inträffat?	Sos 112 vad har inträffat
Mm-	N/A
Är ni på väg nu eller?	är ni på väg nu eller
Vet du vilket spår ni kommer in på?	vilket spår ni kommer in på
Nä, okej.	okej

5.5.3 Analysing calls transcribed with a speech recognition software

As was mentioned in section 5.5, a brief analysis was conducted where two transcribed calls with known quality (both high) were classified. The calls were transcribed using Speech Recognition on Docs and the only modification made to the transcription was adding newlines after each phrase.

iKnow was able to classify both calls correctly. The `simsrcsimple` function in iKnow was set to search for concepts (section 4.3.1). iKnow was given the calls modified for consistency (section 4.2), along with the two calls transcribed by the speech recognition software as domain. iKnow was able to classify both calls correctly, along with classifying all the other calls in the similarity graph.

Because these settings yielded the most accurate results in the previous test cases, only this test was run.

Gensim was also able to classify the calls correctly. The transcribed calls were added to the calls manipulated for consistency (section 4.2) and the algorithm used was the KNN algorithm (section 3.4.1). Several values for k were used. The graph in Fig. 5.5 shows the result for the different values of k. At the vertical line, where $k = 7$, both the transcribed calls were predicted as high quality calls by gensim.

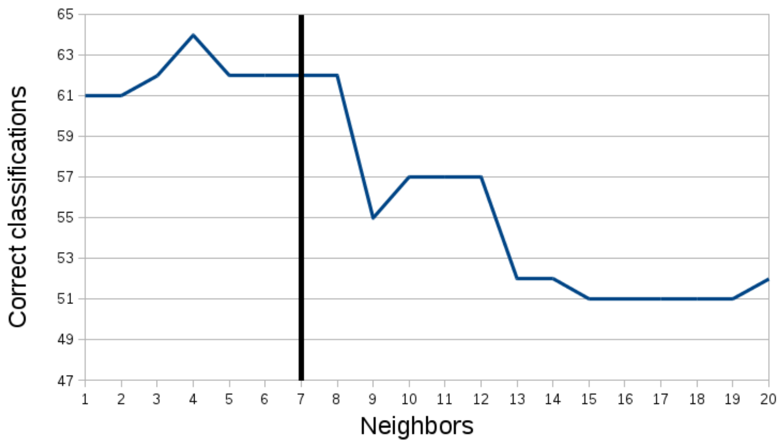


Figure 5.5: Results for gensim, run with the KNN algorithm.

`Solve.py` answered 72% of the questions in the KI-protocol correctly but `Solve.py` classified both calls as low quality calls. The calls got 5 and 6 points respectively and in accordance to section 4.1 a score of 9 are needed for a call to be considered of high quality.

6 Discussion

The programs at hand have proved successful at identifying calls of low or high quality according to the classification made by Bohm to various degrees. Below follows a discussion of each program's strengths and weaknesses.

6.1 Discussion of the results produced by iKnow

The most impressive results produced by iKnow is its ability to correctly classify 100% of the calls using the *Average* algorithm. This is quite surprising since iKnow only compares low-level concepts, how words relates to each other. Whereas, the KI-protocol uses very high level concepts to determine if a call has high quality. An example is question X 3.5 "Is the SOS operator focusing on things not related to the description of the disease?".

It is interesting that iKnow can draw the correct conclusions without understanding the high-level concepts. There are two possible reasons for this. The first reason is that there is a low-level systematic difference between the high quality and the low quality calls. The second reason is that classification of the calls might not been done solely using the KI-protocol. When reading the calls it is quite clear which calls are low and high quality calls. It seems plausible that whoever classified the calls used this notion more than the KI-protocol. Section 4.1 acknowledges this by mentioning that there is no clear correlation between a low score from the protocol and a low quality call. This would imply that the quality of the call was determined by some other, perhaps more subjective, factor that iKnow was able to find.

The results from iKnow also says a lot about how the calls are related to each other. It seems like the calls of high quality are more similar to each other than the calls of low quality are. This could imply that reason for a call having low quality is because it is unstructured and seemingly random. Thus, what the low quality calls all have in common is that they do not have much in common.

6.2 Discussion of how manipulation of calls affect iKnow

As table 5.2 and 5.6 show, every result was higher for the calls modified for consistency than the calls without the caller. This is reasonable since the call without the caller contains less information, it is however noteworthy. If the result was the opposite – removing the caller increased the accuracy of the analysis – this could be due to the caller obstructing information from the SOS operator.

It is interesting to note that the threshold for the *Average* algorithm is increased for the calls without the caller. This means that when the information from the caller was removed, iKnow gave a higher similarity between low quality calls and high quality calls. The opposite behaviour was observed when iKnow searched for CRCs, see table 5.7, instead of just concepts.

In both cases the absolute value of the threshold was greater for the calls without the caller. This implies that, as information is removed the similarities becomes more biased. Some of the increase in bias might be from random fluctuations.

6.3 Classifying calls using gensim

As was discussed in section 5.2, the approach used to classify calls using gensim is similar to the approach used with iKnow. The results are however not as good. The best result is obtained when the calls optimised for consistency are analysed with the *KNN* algorithm; the accuracy was 96%. Without more knowledge about the underlying algorithms used by iKnow it is hard to give good explanation to why this is the case.

It is also interesting to note that the *KNN* graph, Fig. 5.3, is “smoother” than the iKnow *KNN* graph, Fig. 5.2. This implies that gensim is able to make more distinct groups, with less noise, than iKnow. This could prove very useful if gensim is properly trained on a bigger corpus.

6.4 Solve.py’s ability to answer questions

The results for answering specific questions using `Solve.py` varies a lot between questions. It is clear that some questions are more suitable for a computer to answer than others. `Solve.py` performs very well when the question is answerable by looking for key words. When answering a question that requires knowledge about non-contextual matters, especially in combination with temporal aspects, it performs worse.

Questions D 1.3 and S 2.12 was mentioned in section 5.3.1 as problematic questions. Question D 1.3 asks if the caller is able to see or hear the patient, and is difficult to answer because there are many unspoken elements in a conversation that answers the actual question. The caller might talk to the patient and therefore the operator will know that the patient is both present and conscious. Question S 2.12 is difficult in the same way.

Another example where `Solve.py` struggles is when answering question X 3.5 “Is the SOS operator focusing on things not related to the description of the disease?”. Since the SOS operators seldom talks about unrelated topics the result from solver `QX` is good (94% correct for both sets of calls) but this is not because of the solver being good – but the operators doing well. As a matter of fact, solver `QX` returns 0 on each call. The question is hard to answer because the computer must first confirm what the description of the symptoms is and after that, it has to look through the call and find out if the operator is focusing on something else than the specified description.

6.5 Solve.py’s ability to classify calls

When it comes to classifying calls according to their quality, the method is straight forward. The results do however stand out. As was explained in section 4.1 a call is considered to have low quality if it is assigned a score of 8 or lower. In table 5.12 we see the results when `Solve.py` is run with calls modified for consistency and the result is clear: only 18 calls are predicted as high quality calls, one of which is actually a low quality call, according to Bohm (section 1.3). Looking at table 5.13 – showing the result when `Solve.py` is run with caller removed from the calls – the result is even more extreme. While all bad calls are classified correctly, only 1 call is considered to have high quality.

The over all performance of `Solve.py`’s ability to classify the calls could therefore be seen as poor. `Solve.py` correctly classifies 49% of the calls modified for consistency and 27% of the calls where the caller is removed. As was discussed in section 6.4, `Solve.py`'s ability to answer the questions in the KI-protocol is 76% and 68% respectively. This is a lot better than what is seen when the calls are to be classified.

It seems plausible that classifying the calls is more difficult than answering 25 preset questions and assigning each question a weight, as was discussed in section 4.1.

6.6 Discussion of the results produced by GF

As for now, GF answers 100% of the questions asked correctly. However, there were only two questions asked and only one call was used as the test set.

Grammatical Framework seems to have potential. A substantial amount of additional testing is however required in order to verify the true potential of the software. It has been concluded (see section 5.4) that GF is excellent at finding information in a sentence, but if every sentence requires human intervention before it is passed through the program the automation is lost.

6.7 The possibility to classify speech recognition calls

In section 5.5.3 it was concluded that out of iKnow, gensim and `Solve.py`, iKnow and gensim were able to correctly classify the calls transcribed using the speech to text program Speech Recognition on Docs. It is worth noticing that iKnow was able to also classify all the other calls in the similarity graph, but gensim (using the KNN algorithm) needed a value for *k* that skewed the program towards classifying too many calls as good. This lowered the overall result for gensim. Therefore, even though gensim *did* classify the calls correctly, the overall result is not as good as the result for iKnow. It is also worth noticing that gensim was able to classify these calls in their "transcribed version", as was provided by Bohm. The speech-to-text transcription made the classification harder for gensim.

As a disclaimer, it is important to understand that the transcriptions were made under good circumstances. The room was quiet and the speaker spoke clearly. In an emergency situation this might not be the case. Regardless, it is still very interesting to see the results and it seems plausible to transcribe a call, by running it through a speech to text software, and analyse it – without human intervention.

6.8 Sources of error and future recommendations

Looking back at the project there are aspects of the work that could have been done differently. Some decisions that were made might have influenced the results in unpredictable ways and would the work described be recreated there is reason to change some aspects of the implementation and how the tests were done.

6.8.1 Increasing the number of calls

67 calls are not a lot of calls. It would be interesting to test how the programs generalises to a larger sample. Unfortunately the project is dependant on calls from SOS Alarm and is therefore unable to produce additional calls.

6.8.2 Quantifying quality correctly

As has been discussed throughout the report, the KI-protocol did not prove as useful to classify the calls as was expected. The results presented in chapter 5 clearly shows that it is possible to classify calls as high or low quality call, given a well defined training set. It is also shown that it is possible to answer the questions in the KI-protocol. The problem seems to be that these are not easily related.

For future studies it is important to make sure that there is a clear correlation between a call's quality and the answers to any questions meant to define the quality of said call.

6.8.3 The importance of a good corpus

After a discussion with professor Aarne Ranta, it was clear that the choice of corpus to train a program was much more exquisite than what was first thought. Instead of using the Swedish Wikipedia to train the parsing module for CoreNLP (section 4.7.2), it might have been more suitable to use another corpus. Gensim does also depend on what corpus it is trained on (section 4.6) and maybe it would be advantageous to make use of another corpus here as well. Ranta suggested using Swedish subtitles, which is more similar to a talking person, than factual texts written on Wikipedia. An even better corpus could be a large sample of actual emergency calls.

6.8.4 Using the concept of similarity to answer questions

Neither iKnow, nor gensim, was used to answer specific questions in the KI-protocol. Some attempts were made to utilise both programs to achieve this, but the results were not useful and it was decided that the remaining time was better spent researching Grammatical Framework to give the project a broader perspective. Given more time, and a more suitable corpus, future studies should look into this possibility.

6.8.5 Social, economic and environmental impacts

As was discussed in section 1.1, SOS Alarm has a high staff turnover. Enhancing the evaluative process with respect to the operator's ability to assess their patients will only do good. As a consequence of this, it seems reasonable to assume that the operators will be more prone to stay at their jobs. The suggested method of analysing calls will result in a cost for SOS Alarm, but if the result is both a more accurate assessment and more satisfied employees it will most likely result in a net win for SOS Alarm.

It is also possible that Sweden as a whole will benefit from implementing a system like the one described in the thesis. Fewer faulty assessments leads to more people getting the help they need, in time. The decrease in waste of resources is beneficial both with respect to social, economical and environmental aspects. Social, because the population of Sweden will gain an increased amount of trust knowing that an ambulance will most likely be there to help them in case of emergency. Economic, for the aforementioned reasons and environmental, because a more efficient use of ambulances implies less driving to unnecessary locations and use of unnecessary materials.

7 Conclusion

The purpose of this thesis was to offer an in-depth computer science perspective concerning the possibility to digitally analyse incoming calls to SOS Alarm. This section will discuss the possibility to analyse already transcribed calls. As well as, the possibility to transcribe the incoming calls digitally, and later analyse these transcriptions.

It has been concluded that it is possible to classify a given call as a low or high quality call. The calls provided by the project's medical expert Katarina Bohm were mostly formatted as described in section 4.2. The programs used to analyse the calls are built to either take advantage of this formatting, or to discard it. IKnow was able to classify 100% of the 67 calls successfully (section 5.1), gensim classified 96% of the calls successfully (section 5.2), `Solve.py` was able to answer 76% of the questions in the KI-protocol (section 5.3) and GF showed potential for future work (section 5.4).

It is worth noticing that the success of classifying the calls seems unrelated to a program's ability to answer the questions in the KI-protocol (sections 4.1, 5.2 and 5.3). As was discussed in section 6.1 there must be an underlying structure in the texts which the similarity concerned programs are able to utilise.

The potential of digitally transcribing calls seems high. The results from the transcription tests are good (section 5.5) and, when mixed with the transcribed calls from SOS Alarm, iKnow was able to correctly classify these calls amongst the others.

Bibliography

[1] C. Cardie, “Cs674: Natural language processing,” *CS674 Natural Language Processing*, 2003, [Accessed on 2016-02-12]. [Online]. Available: http://www.cs.cornell.edu/courses/cs674/2003sp/history-4up.pdf

[2] SOS Alarm, “SOS Alarm föreläsning,” [Accessed on 2016-02-01]. [Online]. Available: https://www.sosalarm.se/Global/Bibliotek/SOS%20Alarm%20om%20alarmeringstjanstutredningen_ver_3.pdf

[3] ——, “Ambulansbeställning,” 2016, Accessed on 2016-03-15. [Online]. Available: https://www.sosalarm.se/Vara-tjanster/Vard/Ambulansbestallning/

[4] V. Lindström, K. Heikkilä, K. Bohm, M. Castrèn, and A.-C. Falk, “Barriers and opportunities in assessing calls to emergency medical communication centre - a qualitative study,” *Scandinavian Journal of Trauma, Resuscitation and Emergency Medicine*, vol. 22, pp. 1–9, 2014, [Accessed on 2016-01-28]. [Online]. Available: http://www.sjtrem.com/content/22/1/61

[5] B. Ek, P. Edström, A. Toutin, M. Svedlund, A. för omvårdnad, F. för humanvetenskap, and Mittuniversitetet, “Reliability of a Swedish pre-hospital dispatch system in prioritizing patients,” *International Emergency Nursing*, vol. 21, no. 2, pp. 143–149, 2013, [Accessed on 2016-02-12]. [Online]. Available: http://search.proquest.com/docview/851945799?pq-origsite=summon

[6] A. Khorram-Manesh, K. Lennquist Montán, A. Hedelin, M. Kihlgren, and P. Örtenwall, “Prehospital triage, discrepancy in priority-setting between emergency medical dispatch centre and ambulance crews,” *European Journal of Trauma and Emergency Surgery*, vol. 37, no. 1, pp. 73–78, 2011, [Accessed on 2016-02-12]. [Online]. Available: http://onlinelibrary.wiley.com/doi/10.1111/jocn.2015.24.issue-7pt8/issuetoc

[7] L. Hjälte, B.-O. Suserud, J. Herlitz, I. Karlberg, I. för vårdvetenskap, S. of Health Sciences, U. of Borås, and H. i Borås, “Why are people without medical needs transported by ambulance? A study of indications for pre-hospital care,” *European Journal of Emergency Medicine*, vol. 14, no. 3, pp. 151–156, 2007, [Accessed on 2016-02-12]. [Online]. Available: http://onlinelibrary.wiley.com/doi/10.1111/jocn.2015.24.issue-7pt8/issuetoc

[8] B. Ek, M. Svedlund, A. för omvårdnad, F. för humanvetenskap, and Mittuniversitetet, “Registered nurses’ experiences of their decision-making at an Emergency Medical Dispatch Centre,” *Journal of Clinical Nursing*, vol. 24, no. 7-8, pp. 1122–1131, 2015, [Accessed on 2016-02-12]. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1755599X11001200

[9] SOS Alarm. (2014) SOS Alarms årsberättelse 2014. (Accessed on 2016-03-11). [Online]. Available: https://www.sosalarm.se/Global/Om%20oss/Finansiell_information/2014/SOS%20Alarms%20%c3%a5rsber%c3%a4ttelse%202014.pdf

[10] ——. (2016) Jobba hos oss - Frågor och svar. (Accessed on 2016-03-11). [Online]. Available: https://www.sosalarm.se/Jobba-hos-oss/Fragor-och-svar/

[11] Region Halland. (2013) Vad får du för dina skattepengar? (Accessed on 2016-02-29). [Online]. Available: http://www.regionhalland.se/om-region-halland/invanartidningen-halland-basta-livsplatsen/halland-basta-livsplatsen-juni-2012/vad-far-du-for-dina-skattepengar/

[12] Karolinska Institutet. Katarina bohm. Accessed on 2016-03-11. [Online]. Available: http://ki.se/en/people/kabohm

[13] SOS Alarm. (2014) Anrop till 112 under 2014. (Accessed on 2016-02-12). [Online]. Available: http://2014.sosalarm.se/statistik-112-tjansten/

[14] (2016) AES Crypt. [Accessed on 2016-02-12]. [Online]. Available: https://www.aescrypt.com/

[15] Free Software Foundation. What is free software? Accessed on 2016-03-11. [Online]. Available: https://www.fsf.org/about/what-is-free-software

[16] OpenNLP. OpenNLP. Accessed on 2016-03-11. [Online]. Available: http://opennlp.apache.org/

[17] Grammatical Framework. Grammatical Framework. Accessed on 2016-03-11. [Online]. Available: http://www.grammaticalframework.org/

[18] Stanford. Stanford CoreNLP. Accessed on 2016-03-10. [Online]. Available: http://stanfordnlp.github.io/CoreNLP/index.html

[19] R. Řehůřek and P. Sojka, "Software Framework for Topic Modelling with Large Corpora," in *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*. Valletta, Malta: ELRA, May 2010, pp. 45–50, http://is.muni.cz/publication/884893/en.

[20] Chalmers, University of Gothenburg. Aarne Ranta. Accessed on 2016-05-04. [Online]. Available: http://www.cse.chalmers.se/~aarne/

[21] ——. Inari Listenmaa. Accessed on 2016-05-04. [Online]. Available: http://www.chalmers.se/en/staff/Pages/inari.aspx

[22] PCWorld. (2016) Control your pc with these 5 speech recognition programs. (Accessed on 2016-03-15). [Online]. Available: http://www.pcworld.com/article/2055599/control-your-pc-with-these-5-speech-recognition-programs.html

[23] AMG. iSpeechWriter. Accessed on 2016-05-04. [Online]. Available: http://i-speechwriter.com/index.html

[24] EFV-Solutions. Speech recognition on Google Docs. Accessed on 2016-05-04. [Online]. Available: http://efv-solutions.com/

[25] Merriam-Webster. Phonetics. Accessed on 2016-03-23. [Online]. Available: http://www.merriam-webster.com/dictionary/phonetics

[26] ——. Morphology. Accessed on 2016-03-03. [Online]. Available: http://www.merriam-webster.com/dictionary/morphology

[27] ——. Amphibology. Accessed on 2016-03-03. [Online]. Available: http://www.merriam-webster.com/dictionary/amphibology

[28] ——. Polysemous. Accessed on 2016-03-03. [Online]. Available: http://www.merriam-webster.com/dictionary/polysemous

[29] Apple. Siri. Accessed on 2016-03-03. [Online]. Available: https://www.apple.com/ios/siri/

[30] Google. Google Now. Accessed on 2016-03-03. [Online]. Available: https://www.google.com/landing/now/

[31] D. Jurafsky and J. H. Martin, “Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition,” *Speech and Language Processing An Introduction to Natural Language Processing Computational Linguistics and Speech Recognition*, vol. 21, pp. 0–934, 2009. [Online]. Available: http://www.mitpressjournals.org/doi/pdf/10.1162/089120100750105975

[32] J. Pennington, R. Socher, and C. D. Manning, “Glove: Global vectors for word representation,” in *Empirical Methods in Natural Language Processing (EMNLP)*, 2014, pp. 1532–1543, Accessed on 2016-04-21. [Online]. Available: http://www.aclweb.org/anthology/D14-1162

[33] J. Schneider. Cross validation. Accessed on 2016-04-16. [Online]. Available: http://www.cs.cmu.edu/~schneide/tut5/node42.html

[34] Wolfram Research, Inc. Mathematica. Accessed on 2016-04-27. [Online]. Available: http://reference.wolfram.com/language/ref/GraphCenter.html

[35] Tutorialspoint. Graph theory - basic properties. Accessed on 2016-04-27. [Online]. Available: http://www.tutorialspoint.com/graph_theory/graph_theory_basic_properties.htm

[36] Microsoft. What is Cortana? Accessed on 2016-05-05. [Online]. Available: http://windows.microsoft.com/en-us/windows-10/getstarted-what-is-cortana

[37] EFV-Solutions. EFV-Solutions. Accessed on 2016-05-05. [Online]. Available: http://efv-solutions.com/

[38] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval.* New York, NY, USA: Cambridge University Press, 2008.

[39] C. D. Manning, M. Surdeanu, J. Bauer, J. Finkel, S. J. Bethard, and D. McClosky, “The Stanford CoreNLP natural language processing toolkit,” in *Association for Computational Linguistics (ACL) System Demonstrations*, 2014, pp. 55–60, Accessed on 2016-03-10. [Online]. Available: http://www.aclweb.org/anthology/P/P14/P14-5010

[40] Free Software Foundation. Gnu general public license. Accessed on 2016-03-10. [Online]. Available: http://www.gnu.org/licenses/gpl.html

[41] Stanford. Stanford CoreNLP – Anotators. Accessed on 2016-03-10. [Online]. Available: http://stanfordnlp.github.io/CoreNLP/annotators.html

[42] ——. Stanford corenlp – output options. Accessed on 2016-03-10. [Online]. Available: http://stanfordnlp.github.io/CoreNLP/cmdline.html

[43] Free Software Foundation. Gnu lesser general public license. Accessed on 2016-04-21. [Online]. Available: https://www.gnu.org/copyleft/lesser.html

[44] The Linux Information Project. Bsd license definition. Accessed on 2016-04-21. [Online]. Available: http://www.linfo.org/bsdlicense.html

[45] Intersystems. iKnow.Queries.SourceAPI. Accessed on 2016-03-11. [Online]. Available: http://docs.intersystems.com/cache20152/csp/documatic/%25CSP.Documatic.cls?APP=1&LIBRARY=%25SYS&CLASSNAME=%25iKnow.Queries.SourceAPI

[46] Stanford. Stanford corenlp – log-linear part-of-speech tagger. Accessed on 2016-03-10. [Online]. Available: http://nlp.stanford.edu/software/tagger.shtml

[47] ——. Stanford corenlp – extensions – packages and models by others extending stanford corenlp. Accessed on 2016-03-10. [Online]. Available: https://stanfordnlp.github.io/CoreNLP/extensions.html

[48] A. Klintberg. POS tagger model for Swedish for Stanford CoreNLP. Accessed on 2016-03-10.

[Online]. Available: https://github.com/klintan/corenlp-swedish-pos-model

[49] ——. Training a swedish pos-tagger for stanford corenlp. Accessed on 2016-03-10. [Online]. Available: https://medium.com/@klintcho/training-a-swedish-pos-tagger-for-stanford-corenlp-546e954a8ee7#.gfl0x0391

[50] Swedish Treebank. Swedish Treebank. Accessed on 2016-03-10. [Online]. Available: http://stp.lingfil.uu.se/~nivre/swedish_treebank/

[51] Stanford. edu.stanford.nlp.tagger.maxent. Accessed on 2016-03-10. [Online]. Available: http://nlp.stanford.edu/nlp/javadoc/javanlp/edu/stanford/nlp/tagger/maxent/MaxentTagger.html

[52] ——. Stanford CoreNLP – Neural Network Dependency Parser. Accessed on 2016-03-10. [Online]. Available: http://nlp.stanford.edu/software/nndep.shtml

[53] Universal Dependencies. Universal Dependencies. Accessed on 2016-03-10. [Online]. Available: http://universaldependencies.org/

[54] C. Mellon. Non-Negative Sparse Embedding. Accessed on 2016-03-10. [Online]. Available: http://www.cs.cmu.edu/~bmurphy/NNSE/

[55] M. Mahoney. The 100M first characters of Wikipedia. Accessed on 2016-03-11. [Online]. Available: http://mattmahoney.net/dc/text8.zip

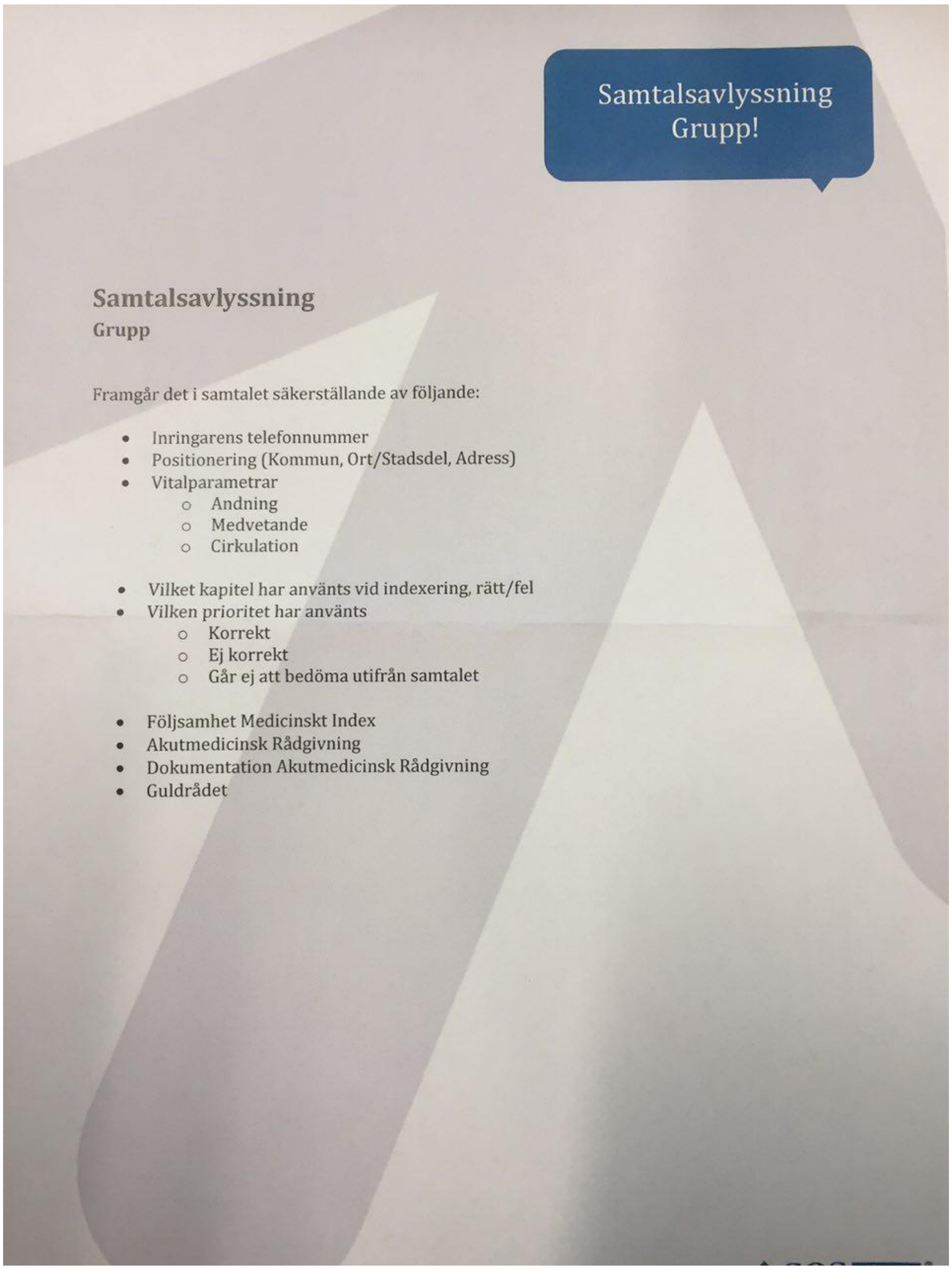
[56] MediaWiki. MediaWiki – API. Accessed on 2016-03-11. [Online]. Available: https://www.mediawiki.org/wiki/API:Main_page

[57] L. Richardson. Beautiful soup. Accessed on 2016-04-21. [Online]. Available: https://www.crummy.com/software/BeautifulSoup/

A

Appendix

A.1 SOS Alarm quality assessment protocol



Samtalsavlyssning
Grupp!

Samtalsavlyssning

Grupp

Framgår det i samtalet säkerställande av följande:

- Inringarens telefonnummer
- Positionering (Kommun, Ort/Stadsdel, Adress)
- Vitalparametrar
    - Andning
    - Medvetande
    - Cirkulation

- Vilket kapitel har använts vid indexering, rätt/fel
- Vilken prioritet har använts
    - Korrekt
    - Ej korrekt
    - Går ej att bedöma utifrån samtalet

- Följsamhet Medicinskt Index
- Akutmedicinsk Rådgivning
- Dokumentation Akutmedicinsk Rådgivning
- Guldrådet

A.2 KI-protocol

2012-07-15 **0=Nej** **1= Ja**

1. Bakgrundsinformation	+ = gynnar bedömning
B 1.1 Var ringer man ifrån?	- = hindrar bedömning
(ex 1. hemmet,2. sjukvårdsinrättning,3. allmän plats 4 ej där pat är)	Neutral
C 1.2 Vem ringer?	+
(1. närstående, 2. sjukvårdspersonal, 3. hemtjänstperson, 4. Patient, 5 annan)	(++ om sjukvårdspersonal?)
D 1.3 Har man sett patienten/den som är sjuk eller hörs patienten i bakgrunden?	+
E 1.4 Språkliga hinder hos inringaren?	-
(ex 1. annat modersmål, 2. kan inte prata)	
F 1.5 Språkliga hinder hos patient/den som är sjuk?	-
(ex1. annat modersmål, 2. kan inte prata, 3. Döv, annat)	
G 1.6 Uppskattad ålder på den som är sjuk	+
(barn – ungdom medelålder- gammal)	
2. Inringaren	
H 2.1 Uppger inringaren att det är en transport mellan olika vårdgivare?	neutral
(ex från vårdcentral till sjukhus eller från hemsjukvården till sjukhus)	
I 2.2 Finns det en beskrivning av vad patienten gör under samtalet?	+
(ex ligger på golvet, sitter uppe, benen bär inte kan inte stödja på benen)	
J 2.3 Finns beskrivning av hur patienten	+

ser ut?	
(ex patienten är blek, kallsvettig, grå, röd, blå-lila, kritvit, svullen om bena, svullen)	
K 2.4 Upprepar inringaren symtomen > 4 gånger?	-
L 2.5 Anges symtom utveckling över tid?	+
(patienten anger att det har gjort ont en längre tid >1 vecka och nu har det blivit mycket värre eller för 30 minuter sedan fick hon/han ont i bröstet)	
M2.6 Jämförs nuvarande tillstånd med hur det brukar vara?	+
(ex tidigare kunde han/hon stå, gå och prata, nu bara ligger han/hon ned och pratar inte, eller har kissat/bajsat ned sig, det brukar inte vara så)	
N 2.7 Uppges att patienten har >2 problem?	-
(ex han hon är yr, har feber och kan inte gå)	
O 2.8 Uppges motsägelser i beskrivningen av tillstånd?	-
(ex uppger att problemet är att hon/han blöder näsblod men sedan beskrivs även att det är jobbigt att andas eller ont i bröstet eller patienten säger att hon/han inte kan andas men patienten pratar utan att vara andfådd)	
P 2.9 Kan inringaren svara på larmoperatörens frågor?	+
(ex inringaren ser inte eller vet inget om patienten)	
Q 2.10 Svarar inringaren på larmoperatörens frågor?	+
(ex pratar om andra saker än vad larmoperatören frågar)	
R 2.11 Beskrivs fler än 3 olika tidigare sjukdomar i samtalet?	-
(ex har högt blodtryck, diabetes och demens)	

S 2.12 Beskrivs problem med Andning-Cirkulation-Medvetande problem direkt i samtalet (inom 30 sek)? *Se exempel.*	+
3. Larmoperatören	
T 3.1 Sammanfattar och eller upprepar larmoperatören vad inringaren berättar?	+
U 3.2 Använder larmoperatör stödfunktioner *(ex giftinformations central, barnmorska*)?	
V 3.3 Säkerställer larmoperatören med frågor och/eller kommentarer att patienten andas och är vid medvetande? Eller hörs det att patienten andas/pratar under samtalet?	+
W 3.4 Leder larmoperatören samtalet?	+
(*ex larmoperatören ställer frågor och inringaren svarar/gör vad den är uppmanad till att göra)*	
X 3.5 Larmoperatören fokuserar på annat än sjukdomsbeskrivning?	-
(ex pratar om annat än vad som är problemet som det finns ett behov av ambulans eller sociala eller alkohol problem)	
Y 3.6 Ställer larmoperatören flera frågor i samma mening?	-
(ex när började det göra ont och var och hur gör det ont?)	
Z 3.7 Frågar larmoperatören om det som inringaren berättat?	+
(ex inringaren berättar att det gör ont i bröstet och larmoperatören frågar på vilket sätt/var/när det började/om det strålar någonstans eller inringaren uppger att patienten är grå-blek och larmoperatören frågar om hur huden känns)	