Bachelor Thesis

Computer Science and Engineering, 300 credits

Chatbot for Information Retrieval from Unstructured Natural Language Documents

Computer Science and Engineering, 15 credits

Halmstad 2019-06-07

Joakim Fredriksson, Falk Höppner

HALMSTAD UNIVERSITY

Chatbot for Information Retrieval from Unstructured Natural Language Documents

Bachelor Thesis for the Computer Science and Engineering Programme

Falk Höppner & Joakim Fredriksson

Halmstad University

Supervisor: Eric Järpe

Examiner: Struan Gray

June 7, 2019

Abstract

This thesis brings forward the development of a chatbot which retrieves information from a data source consisting of unstructured natural language text. This project was made in collaboration with the company Jayway in Halmstad. Elasticsearch was used to create the search function and the service Dialog ow was used to process the natural language input from the user. A Python script was created to retrieve the information from the data source, and a request handler was written which connected the tools together to create a working chatbot. The chatbot correctly answers questions with a accuracy of 72% according to testing with a sample of $n = 25$. The testing consisted of asking the chatbot questions and determining if the answer is correct. Possible further research could be done to explore how chatbots might help the elderly or people with disabilities use the web with a natural dialogue instead of a traditional user interface.

Sammanfattning

Denna rapport presenterar utveckling av en chatbot som hamtar information fran en datakalla som bestar av ostrukturerad naturlig text. Detta projekt utfordes i sammarbete med foretaget Jayway i Halmstad. Elasticsearch anvandes for att skapa sokfunktionen, och tjansten Dialog ow anvandes for att behandla det naturliga spraket fran anvandaren. Ett programm skrevs i Python som hamtar information fran datakallan, och en request handler skrevs som sammankopplar de olika delarna for att skapa en fungerande chatbot. Chatboten svarar korrekt med en nogrannhet pa 72% enligt testning med $n = 25$. Testningen bestod av att stalla fragor till chatboten och avgora om svaret ar korrekt. Mojlig vidare forskning skulle kunna berora hur chatbotar kan hjalpa aldre och personer med funktionsnedsattningar anvanda webben genom att anvanda naturlig dialog istallet for ett traditionellt anvandargranssnitt.

Contents

1. Introduction

Traditional visual websites and user interfaces may soon be on their way out. Chatbots look to be a prime candidate for replacing or enhancing the way a user interacts with their devices.

More and more data and information is available on the world wide web. However, a lot of this information is in the form of free-form text, which can make it di cult to extract the desired information. Currently, search engines are commonly used to nd the information users are interested in. One disadvantage of this method is that the user has to be very speci c with their search query. Machine learning makes it possible to design a more intuitive method for searching, and to present the results in a personalised way.

This method of searching uses a dialogue instead of a traditional search query. The input from the user does not need to consist of a couple of keywords, but may be comprised of complete sentences. The input is then interpreted by a chatbot, in order to search for the desired information. Nowadays the majority of searches are conducted via text. A chatbot that holds a natural conversation is also more suitable for using a voice based interface, as the bot's responses more naturally t in a conversation than results from a traditional search engine.

A study about why people would like to use chatbots was performed in 2017 by Petter Bae Brandtzaeg and Asbj rn F lstad. In this study 146 participants were asked why they would want to use a chatbot. 100 of the 146 participants believed it would help them with their productivity. The most reported reason was ease of use in information and assistance retrieval [1].

This kind of search feature has already started being used by a number of companies. One of them is an insurance group from Switzerland who use a chatbot which makes it possible to report a bike theft using a messaging application [2].

1.1. Aim

The purpose of the project is to develop a chatbot for the company Jayway, which can be used to answer questions from employees or guests. The bot is intended to be an alternative to the search engine available on Jayway's internal

wiki. The users' questions can be anything from what the password to the Wi-Fi network is, or how to go about completing certain work tasks. Jayway's internal wiki will be used as the source of information for the chatbot. As the wiki is only available in English, the chatbot will only be able to communicate using English.

One possible extension of the project is to make it possible for the bot to retrieve answers from other sources, such as Wikipedia, 1177.se, or the Halmstad Municipality website, for example. This may take considerable effort, and will only be attempted if there is time for it.

Google Home integration is a further goal. As the bot will already work with free-form, natural text as a user interface, it ought to be relatively simple to extend it to also use speech as a source of input, and to integrate it with Google Home. As with the previous extension, this will only be done if there is time left over.

1.2. Problem definition

When creating systems with some sort of intelligence it is quite difficult to determine the quality of the system. In the case of the development of a chatbot that will answer questions, the result will have to meet some criteria. Is the response made in such a way that it feels human? Do the results have too much information or too little? These are some problems that have to be taken into consideration when developing the system.

To complete the project the following milestones have been determined.

Scraper | Can a program be built to retrieve the data from the internal wiki and build a database?

Search engine | Is it possible to build a functional traditional search engine that works with the database?

Natural language processing | Can natural language processing be used as a user interface to this search engine?

Chatbot | Is it possible to connect the milestones listed above to make a functional chatbot?

2. Background

A natural dialogue has become very desired in today's development in interaction with di erent platforms. Much research has been done on how people converse, and how human language can be used as a form of input [3]. Nowadays it is not unusual that companies develop chatbots for di erent purposes. For example, there are Google Assistant, Siri, and Amazon Alexa, which can all answer general questions and hold a conversation with the user. They can also complete simple tasks, for example reminding the user about something, or adding an appointment to the user's calendar. Other companies use chatbots to solve more speci c problems, for instance the front end of customer support or as a method for their customers to place orders.

2.1. Natural language conversation frameworks

A number of di erent cloud based tools and frameworks for developing chatbots have also entered the market [4]. These services are made for the development of systems that use natural language as a way to communicate with the user. To simplify development the services come with a wide range of tools that facilitate the analysis of natural language built in. Some of the services have the possibility to integrate the system with a smart speaker that will be available to get input by voice communication. Each service has libraries for di erent programming languages, however a lot of the systems have languages such as Java, Python and Node.js in common.

2.2. Chatbot advantages

Because of an increase in the amount of people using smart phones and a decrease in desktop computers or laptops as seen in gure 2.1 companies have tried to make user interfaces as easy and intuitive as possible. As more modern product development processes like DPD (Dynamic product development) [5] put a larger focus on the user than the product, companies are trying di erent ways to communicate with the user to build a more personal relationship. Chatbots are very suitable for this kind of solution as the communication is

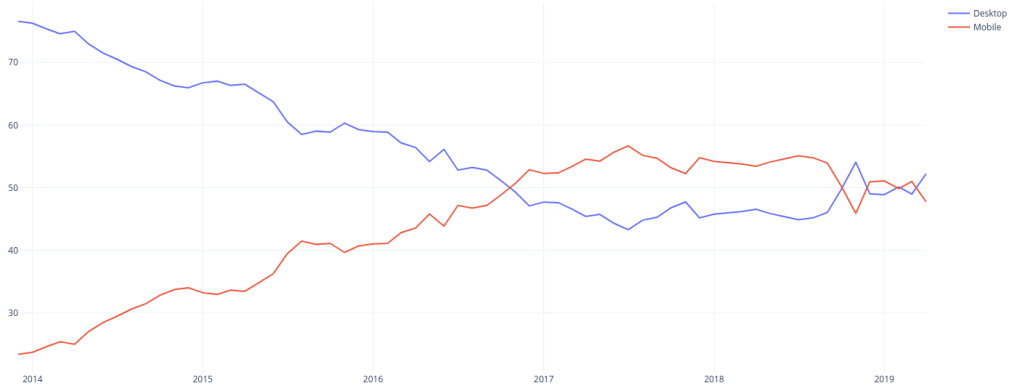

Figure 2.1.: Graph of Mobile and Desktop usage retrieved from `http://gs.statcounter.com`

very natural and can be presented in a messaging platform which the majority of people who have a smartphone are already familiar with. This leads to the user being more involved in the communication and leads to a more positive experience [1].

When searching for information, a traditional search engine will be more efficient, as it only requires the entry of keywords, while a conversation with a chatbot may contain extraneous information, such as greetings and filler words that are used to build sentences. Nevertheless, a traditional search engine may not be the most preferred way of receiving information, as it may be perceived as dull and impersonal. However, for some users this is still the preferred way of searching for information, but for companies that focus on building a positive relationship with the users a chatbot is the preferred solution.

2.3. Full-Text Search Engine

To be able to search and retrieve snippets from the wiki some sort of search engine has to be implemented. Today there are many different full-text search engine that can be adapted for this application, instead of having to build a custom solution. These search engines are made for retrieving information from unstructured natural language databases. Information retrieval from such text databases is quite difficult as unstructured data is hard to index. It is also difficult to determine which words should be considered more important in the search. In contrast, in a structured database every value has some sort of

unique identi er which makes information retrieval signi cantly more straightforward. To solve these problems these libraries make use of methods and algorithms to give each word some kind of value to make information retrieval possible. When it comes to full text search, most of the libraries are built around *term frequency inverse document frequency* or TF-IDF, as described in section 3.3, which searches using a query and tries to nd documents which match with every term in the query.

2.4. Related work

One of the very rst chatbots that was created was called ELIZA [6], and was completed in 1966. The program was developed at the Massachusets Institute of Technology, in their *Project on Mathematics and Computation* (MAC), could hold simple conversations with users, and answer with natural human language. ELIZA was mainly based on the identi cation of keywords in the user input, and prede ned transformations applied to the input. The analyser iterated over the text and retrieved a keyword. The keyword had a form of rank which was used for selecting the keyword that is most likely correct. Therefore, if a keyword with a higher rank was found, the previous keyword would be discarded. This keyword iteration continued until punctuation was found. If a keyword had been located before the punctuation, the remaining text would be discarded from the input and would therefore not be analysed. If a keyword was not located, the same procedure would begin after the punctuation. ELIZA was very advanced for the time, however, ELIZA was quite limited from today's point of view.

In 2006, IBM developed their chatbot Watson [7]. Watson is signi cantly more advanced than ELIZA, and can analyse sentences with deeper and more e ective methods. It uses deep natural language processing, which is a method that uses machine learning to analyse sentences. Watson uses context and a large database of knowledge to be able to answer with a very high degree of accuracy. Watson answers by rst dividing the question into smaller parts, and then retrieving similar questions from its knowledge database. After this, hundreds of di erent possible answers are retrieved and compared to similar questions. With this information a value is calculated, which indicates the probability that an answer is correct, and the answer with the highest score is presented.

Compared to the time when ELIZA was created, the average person now has access to a cellphone, which is able to communicate using some sort of messaging platform. The amount of people with cellphone access increases every year. Contemporary communication is often in the form of short messages, and is often carried out with multiple people at the same time [8]. This kind of communication is optimal for a chatbot with an understanding of natural

languages. Because of this and the immense amount of data that is available, enterprises have developed bots for their own services. The bots mentioned in section 2, that are currently the latest in the modern era, Google Assistant, Siri, and Amazon Alexa, are constantly getting more sophisticated as their knowledge base grows larger every day. These companies have released frameworks and services making it possible for the average consumer to develop new bots for products e.g Messenger or Skype, which has resulted in thousands of new bots. Some being basic and some advanced with their own set of goals [9].

2.5. Prerequisites

The wiki is composed of multiple articles. Figure 2.2 shows a template of the structure of an article on the wiki. Each article has one top-level heading, which is also the title of the article. Furthermore, each document has zero or more subheadings, each heading containing one or more paragraphs of body text. Further nesting of headings does not occur, i.e. there are no sub-subheadings, etc.

The wiki is a website hosted on Google Sites, and requires the user to be logged in to their company Google account. In this project no access to any backend database was provided. Any required data instead has to be gathered by extracting it from the wiki website.

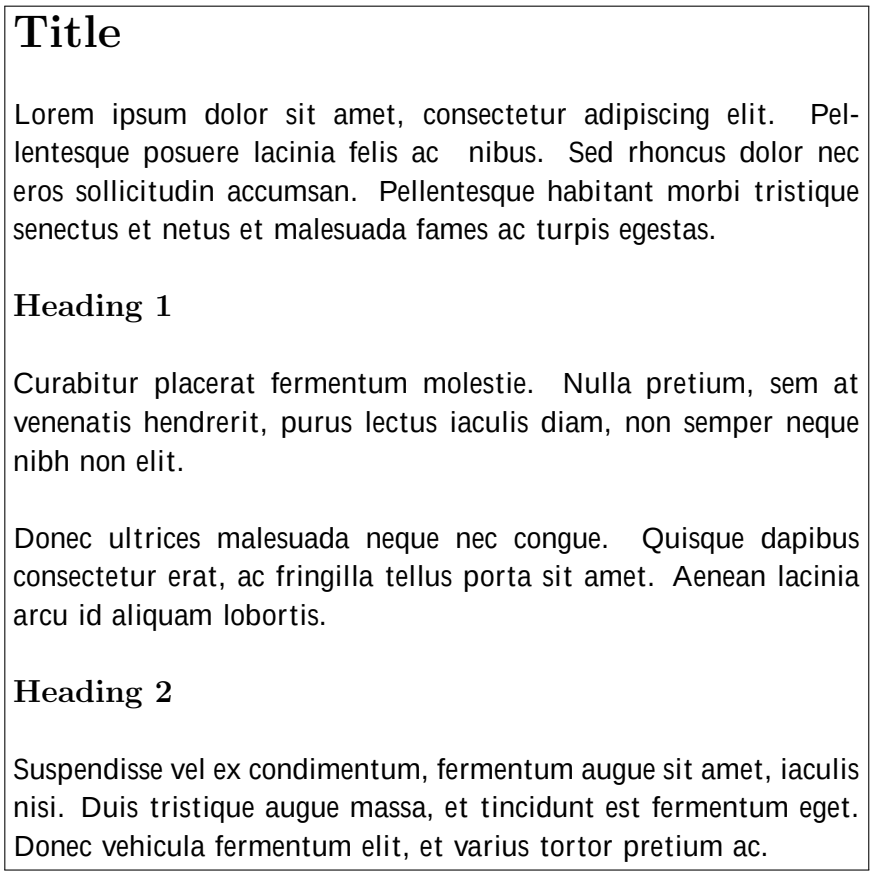

Figure 2.2.: Sample article layout

3. Theory

3.1. Natural language processing

The purpose of making computers understand human language is to make it available for human-machine communication, and to gather information about languages.

To get the desired information, the language used in the text has to be identified. One type of language is formal languages [10]. Programming languages e.g. Java or C are two examples of these, each with their own set of rules. In Java, strings are defined by `String text = "abc";`, while in C it is done by `char text[] = "abc";`. These sets of rules are fixed, and cannot be changed. To process these kinds of languages, one needs to know these exact rules.

Natural languages, such as Swedish or English are significantly more difficult to define with a set of rules, as a sentence may be grammatically correct, but seem incorrect. In their book *Artificial Intelligence: a modern approch* the authors give an illustrative example of this problem. The sentence \Not to be invited is sad" is grammatically correct, and sounds correct to the general public. However, the sentence \To be not invited is sad" is likewise correct, but does not sound correct to the general public. There are also problems with context in natural languages. For example \He moved" can be interpreted to mean that a person moved to a different address, or that the person just physically moved about.

Because of these problems in the processing of natural languages, probabilities are used instead of a fixed set of rules.

3.2. N-Grams

A commonly used method to analyse a language is N-Grams [11; 12]. The method is used to determine the probability of a word occurring in a text after a given sentence. The model consists of either unigram, bigram or trigram. While a unigram checks the probability without considering the preceding words, bigrams and trigrams consider one and two preceding words, respectively.

Bigrams are de ned as $P(w_n|w_{n-1})$ where w is any arbitrary word in the text that is analysed. For instance, given the sentence \Cats like food", the bigram probability that the following word is \with" would be

$$P(with|food)$$

An estimate for the probability can be calculated using the formula

$$P(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)}{\sum_w C(w_{n-1}w)}$$

This counts the number of times w and w_{n-1} occur and divides it with the number of times w_{n-1} occurs. This is based on the maximum likelihood estimation method.

3.3. Term Frequency–Inverse Document Frequency

Full-text search engines which are suitable for this project were brie y presented in section 2.3. They make use of the theory *Term Frequency{Inverse Document Frequency* or TF-IDF to determine the importance of documents depending on a given query. A query consists of words (terms), and each term will be weighted depending on its occurrence in each document, hence being called term frequency. A problem arises when a query consists of common terms e.g. \the" or \is". This may result in documents having a high weight because of non-important terms. To solve this inverse document frequency complements the term frequency. It does this by measuring the importance of the term by setting a weight that decreases the importance of terms that appear frequently in a document [13].

Term frequency, in its essence, is a score based on weight of the terms from the query w over each word in a document w_d.

$$\text{tf}(w) = \frac{w}{w_d} \tag{3.1}$$

Inverse document frequency as described earlier solves the problem with every term being weighted equally in a given query. Having N documents and term w_i occurs in n_i of the documents, then the score for the inverse document frequency of the term is given by

$$\text{idf}(w) = \log \frac{N}{n_i}$$

thus resulting in rare terms giving a high idf score.

TF-IDF is the combination of these, and the score is given by

$$\text{tfidf}(w) = \text{tf}(w) \quad \text{idf}(w)$$

The full-text search engine that is used in this project is built on the Apache Lucene library which uses TF-IDF, but also complements it with algorithms to be more e cient and accurate. Lucene uses the boolean model to simply search all documents that contain any of the terms in the query, and then gives those document value of 1. Then the TF-IDF only iterates the documents with a value of 1. To give an even more accurate score, the Apache Lucene library also uses *eld length normalization*, which reduces the gap between documents of di erent sizes. The reason this is used is because a document that has few words and a document that has many words will result using the equation 3.1 in the document with fewer words getting a higher score. The *eld length normalization* is given by

$$kdk_F = p\frac{1}{N_w}$$

Were N_w is the amount of words in the document [14].

4. Method

The complete chatbot is composed of multiple interacting parts, as illustrated in gure 4.1. When a user asks the bot a question (1), Dialog ow interprets the input. If Dialog ow may respond directly to the user, for example asking the user to try again if the input could not be understood. If required, Dialog ow can also make a webhook request for information from the data source (2). This request is handled by a program on the backend server, and passed on to Elasticsearch (3). Elasticsearch responds with a search result (4), which is sent back to Dialog ow by the request handler (5). Finally, Dialog ow uses the data to construct a response, and sends it to the user (6). The system also contains a scraper, which collects data from the data source, converts it to an appropriate format, and indexes it in Elasticsearch.

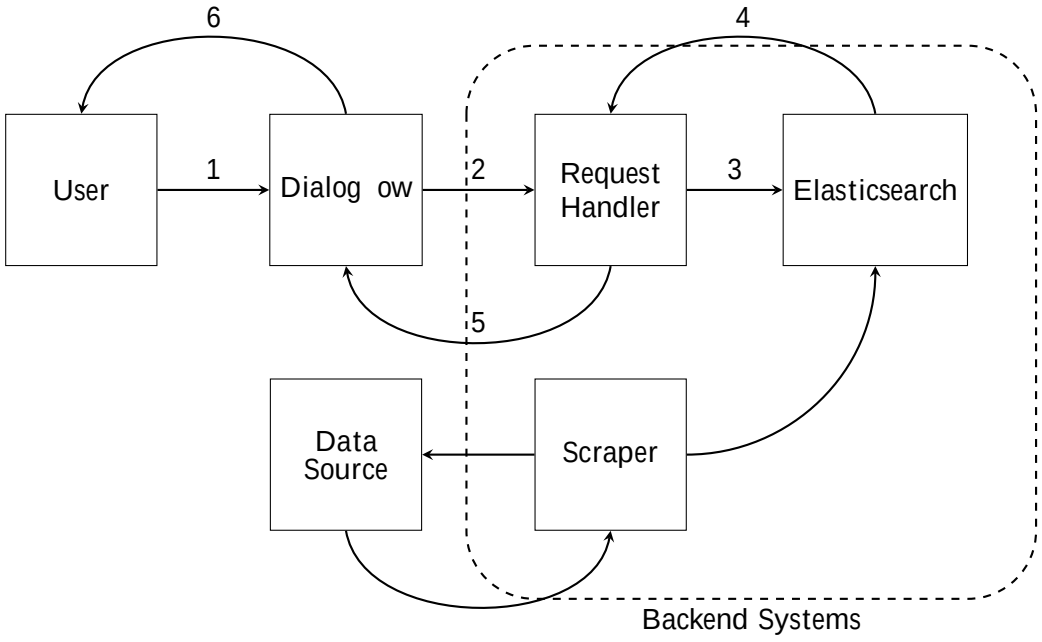

Figure 4.1.: The di erent parts of the chatbot

4.1. Specification

The speci cation of this project is to develop a fully functional chatbot that will be able to give a correct answer at least 50% of the time. The chatbot has to communicate using natural language and should respond to questions in a way that resembles a dialogue more than a typical answer from a search engine. The chatbot has to retrieve the keywords from an input given as a sentence.

4.2. Analysis

After a chatbot was developed that ful lled the speci cation a set of tests was performed. The rst test was to ask suitable questions, e.g \What is the wi password?", and then determine if the answer was correct. This result was used to determine the probability of a correct answer. The same tests were performed on earlier versions of the chatbot and the results were compared to determine how much more accurate the nal version was compared to the previous ones.

The nal version was also released for Jayway employees to evaluate. After each answer from the chatbot, the employees were asked by the chatbot if the answer was good, and their answers were logged.

4.3. Scraper

A Python script was created to scrape data from the wiki, using Selenium WebDriver[1] to control an instance of the Chromium[2] browser, and Beautiful Soup[3] to extract the desired elements of the HTML les. This approach was chosen due to the wiki being hosted on Google Sites, and thus requiring a logged in Google account to view. Selenium was chosen in favour of more lightweight solutions such as Scrapy[4], as Google requires browsers to support JavaScript to log in. Furthermore, Google disallows completely automating the login process, which would further complicate using a simpler scraper.

The scraper starts by visiting the main page of the wiki, causing a Google login screen to be shown. It inputs an email address automatically, but then

[1] Selenium WebDriver https://docs.seleniumhq.org/projects/webdriver/
[2] Chromium - The Chromium Projects https://www.chromium.org/Home
[3] Beautiful Soup: We called him Tortoise because he taught us. https://www.crummy.com/software/BeautifulSoup/
[4] Scrapy | A Fast and Powerful Scraping and Web Crawling Framework https://scrapy.org/

pauses and waits for a user to input the password before continuing. This is required to prevent Google from interpreting the login as automatic, and denying access. After the main page of the wiki is loaded it extracts all the links from the navigation pane and stores them in a list. The links are then each visited, and body text and header elements are extracted. The text to be indexed is then extracted from these elements.

Using Selenium has some drawbacks. It is relatively slow, as all websites are rendered graphically, all scripts are executed, and all images are loaded. Sites also cannot be loaded in parallel. Scraping took between one and two seconds per article, which was not a problem in this case, as the wiki only had 102 articles. The scraping process could, however, take considerable time on larger sites with thousands of articles. Another drawback is that the scraping process could not be fully automated due to the need for a user to log in manually each time. This makes it impossible to automatically scrape the website to keep the database up to date. A user would be required to regularly start the scraping process, and log in with their credentials to keep the database up to date.

After each article is processed, the scraper uploads it to Elasticsearch using the o cial Elasticsearch client library for Python[5].

4.4. Elasticsearch

In order to be able to e ciently nd relevant documents a search engine has to be used. Currently there are two widely used search engines, Elasticsearch [6] and Apache Solr [7]. Both of these search engines are open source and both are suitable for the project. Apache Solr and Elasticsearch are both based on the information retrieval library Apache Lucene. In the article *Comparing Apache Solr and Elasticsearch search servers* [15] the authors compare di erent aspects of these search engines. The main di erence is simplicity of developing. Solr is mainly used in enterprise solutions and requires extensive con guration to develop software. In comparison, Elasticsearch has a simple base with a modular design, requiring little con guration to develop a nished product. Because of its modular design there is still a possibility to make Elasticsearch as advanced as Solr. Because of this Elasticsearch will be used to build the search engine.

To retrieve the desired information Elasticsearch's built in search APIs will be used. These APIs make use of the TF-IDF as described in the section 3.3

[5] elasticsearch-py | Elastic https://www.elastic.co/guide/en/elasticsearch/client/python-api/current/index.html

[6] Elasticsearch https://www.elastic.co

[7] Apache Software Foundation Solr https://lucene.apache.org/solr

to give a good match for full text search. If multiple matches are found the match with highest score will be passed to the NLP service.

Elasticsearch also has a feature called analysers. Analysers transform text indexed in Elasticsearch to improve search results. The built-in analyser *english* was used for all text fields. It removes punctuation and so-called stop words, which are common words that do not add any information, such as \the", \a" and \is". It also stems words, meaning that it reduces words to their root form. For example, it turns \running" into \run". Furthermore, it turns all letters into lower case.

4.5. Dialogflow

To make the system communicate using natural language the service Dialogflow was used. It was used in the analysis of the input and in the creation of the response. The inputs are in the form of one or multiple sentences, and Dialogflow retrieves the keywords of the input using NLP. There was no information on what methods Dialogflow uses to retrieve this information. If the input can be interpreted as a question a query will be sent to Elasticsearch based on the keyword via the request handler. Elasticsearch will then send the part in the database were the question was answered, and Dialogflow will then present the answer.

To solve the problem with NLP Dialogfow has different kind of mechanisms. The ones that were used were *entities*, *intents* and *contexts*. Entities were used to retrieve desired information from natural language. An entity was created, which consisted of subjects to be retrieved from the input. Dialogflow tries to identify these entities from a input, if an entitiy is found an intent is triggered. An intent is Dialogflow's way of mapping a user's input to a predetermined step in the conversation. It works by giving Dialogflow a number of examples of what a user may write. Dialogflow then uses machine learning to also match inputs that are similar to the example inputs. Responses to these intents can be created by defining words or sentences that will be presented as a response to a specific intent, Dialogflow then combines these sentences to create a larger set of responses. This method of building responses can be used in combination with entities. An example might be if the input \I really like living in Halmstad" is given and Dialogflow is configured to extract the city the user lives in as an entity. The response could then be \I really like Halmstad as well", where \Halmstad" is inserted by Dialogflow. Contexts are a way for Dialogflow to carry entities from one dialogue intent to another, and to maintain a state in the conversation.

A big reason for choosing Dialogflow instead of another service was that Jayway highly recommended it. There was also an article, *A Comparison and Critique*

of Natural Language Understanding Tools [4], in which the authors compare some of the most used services. The authors present an evaluation on the usability which refers to ease of use for developers when it comes to working with the service. Dialog ow and the service *Watson Conversation* are the only services that get the highest result in usability. Because of the time frame for the project a decision was made that a service that makes it possible to start development as soon as possible would be the optimal route.

4.6. Request Handler

A program was built to receive webhook requests from Dialog ow, and return search results from Elasticsearch, thereby connecting the two components. This request handler was written in Javascript, and utilises the Node.js[8] runtime. When Dialog ow needs to search for an answer, it sends a HTTP POST request containing the user's query and other data about the conversation to the backend server. This request is then received by the request handler, which uses the o cial Dialog ow ful lment library[9] to handle parsing of the conversation data. The user's query is extracted, and a search request is sent to Elasticsearch using the o cial Elasticsearch client library[10]. The search result with the highest score is then sent back to Dialog ow, also using the ful lment library. If there are no search results, a message asking the user to rephrase their question is sent instead. The web application framework Express[11] was used to handle the HTTP requests, as the ful lment library only handles the processing of data in the requests.

Whenever a request is received by the request handler, the user's question, the topic of the conversation, and the answer from Elasticsearch are logged, along with a unique user ID. Users are also asked if the bot's answers are good, and the users' responses to this question are also logged by the request handler.

4.7. Security

An important security consideration when building a server is preventing access by unauthorised users. In order to ensure that the request handler can only be called by Dialog ow, and that the Elasticsearch database can not be

[8] Node.js https://nodejs.org/en/

[9] Dialog ow https://dialogflow.com/docs/reference/fulfillment-library/webhook-client

[10] elasticsearch.js [16.x] { Elastic https://www.elastic.co/guide/en/elasticsearch/client/javascript-api/16.x/index.html

[11] Express - Node.js web application framework https://expressjs.com/

accessed by outside users, a reverse proxy with authentication was used. Elasticsearch and the request handler were con gured to only accept connections coming from the same host, so that they are not directly accessible from the outside. The only service on the server that is accessible from the outside is the web server Nginx[12], con gured to act as a reverse proxy. To access these services, connections have to pass through Nginx, which uses Basic HTTP authentication [16] to reject unauthorised requests. Basic HTTP authentication requires the HTTP client to send a user ID and password to the server with each request. The server then either ful ls or rejects the request depending on if the sent credentials are correct. If the client successfully authenticates, Nginx passes the request along to either Elasticsearch or the request handler, depending on the URL speci ed in the request.

A further security concern is con dentiality of the transmitted information. Information from the wiki being transmitted over the internet without any encryption could result in sensitive data being intercepted by malicious actors. This problem was also solved with the Nginx reverse proxy, by con guring it to only allow incoming connections that use HTTPS. This is also critical for the authentication, as the user credentials are not encrypted when using plain HTTP.

4.8. Programming languages

Python was chosen as the programming language for the scraper due to wide availability of Python libraries, and due to the ease of writing simple programs in Python. A potential drawback of Python is that being a interpreted language, it is relatively slow, but since the scraper will only be ran periodically this should not be an issue.

Node.js was chosen for the request handler because Dialog ow's o cial ful lment library greatly simpli ed building the handler, and only was available for Node.js.

4.9. Platform

A virtual private server running the Ubuntu Linux distribution was chosen for the backend server mainly due to wide availability of packages. This server is hosted on Google Cloud Platform and is equipped with one CPU core and 4 gigabytes of RAM. Software used in the project is listed in table 4.9.

[12] NGINX *j* High Performance Load Balancer, Web Server & Reverse Proxy `https://www.nginx.com/`

Software	Version
Ubuntu	19.04.2 LTS
Elasticsearch	6.7.2
Python	3.7.2
Beatiful Soup	4.7.1
Selenium	3.141.0
elasticsearch-py	7.0.0
Chromium	72.0.3626.119
Node.js	8.10.0
Express	4.16.4
elasticsearch.js	15.74.1
Dialog ow Ful llment Library	0.6.1
body-parser	1.18.3

Table 4.1.: Software used

5. Results

In total, three versions of the chatbot were developed, each version being a further development on the previous. In this chapter, how the di erent versions work, and a statistical comparison of the quality of responses from the di erent versions will be presented.

5.1. The chatbot

This section presents each version of the chatbot. A owchart illustrating the process that each version uses from input to a response, as well as a description of the process.

5.1.1. Version A

Version A was the rst functional chatbot. Figure 5.1 shows a owchart describing the process that it uses in a conversation. It starts by greeting the user with a welcome message. Afterwards, any input from the user is interpreted as a question, and the whole input string is sent as a query to Elasticsearch using the request handler as explained in section 4.6. Elasticsearch then searches for a matching document as described in section 3.3. When matching documents have been found, one or more sentences are extracted from the document with the highest score using Elasticsearch's highlighting feature. Highlighting nds a snippet of text in the document which contains words in the query, thereby attempting to nd an answer to the user's question. The result from the highlighting is then sent to Dialog ow by the request handler. Dialog ow then presents this result to the user. If Elasticsearch did not manage to nd a match, a response asking the user to rephrase their question is presented, e.g. \I didn't get that. Can you say it again?".

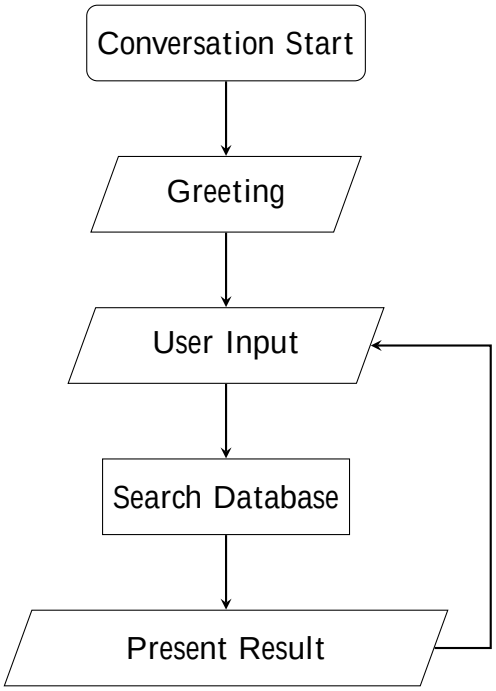

Figure 5.1.: Flowchart for version A

5.1.2. Version B

Version B was the second version of the chatbot. The di erence from version A is that this version of the chatbot needs two inputs. As seen in gure 5.2, the chatbot rst asks the user what topic they want to ask questions about. Each document has a corresponding entry with a title (topic), for example \Vacation" or \WiFi", as explained in section 2.5. To identify a topic from the input, entities were used. In this case an entity called *subject* was created, this subject had every possible topic and suitable synonyms for the topic. A example of this is the topic \Wiki Admin" which has a synonym of \Administration". When a topic is found, Dialog ow matches an intent that asks the user what information they would like know about the topic. After a topic is found, any inputs given to Dialog ow that do not trigger any other intent are sent to Elasticsearch using the same document search process as in Version A. However, the full text search algorithm only searches in the document with the corresponding title. If a user would like to ask a question about another topic, an intent was made that matches inputs similar to \I have another question", which then allows the user to input a di erent topic.

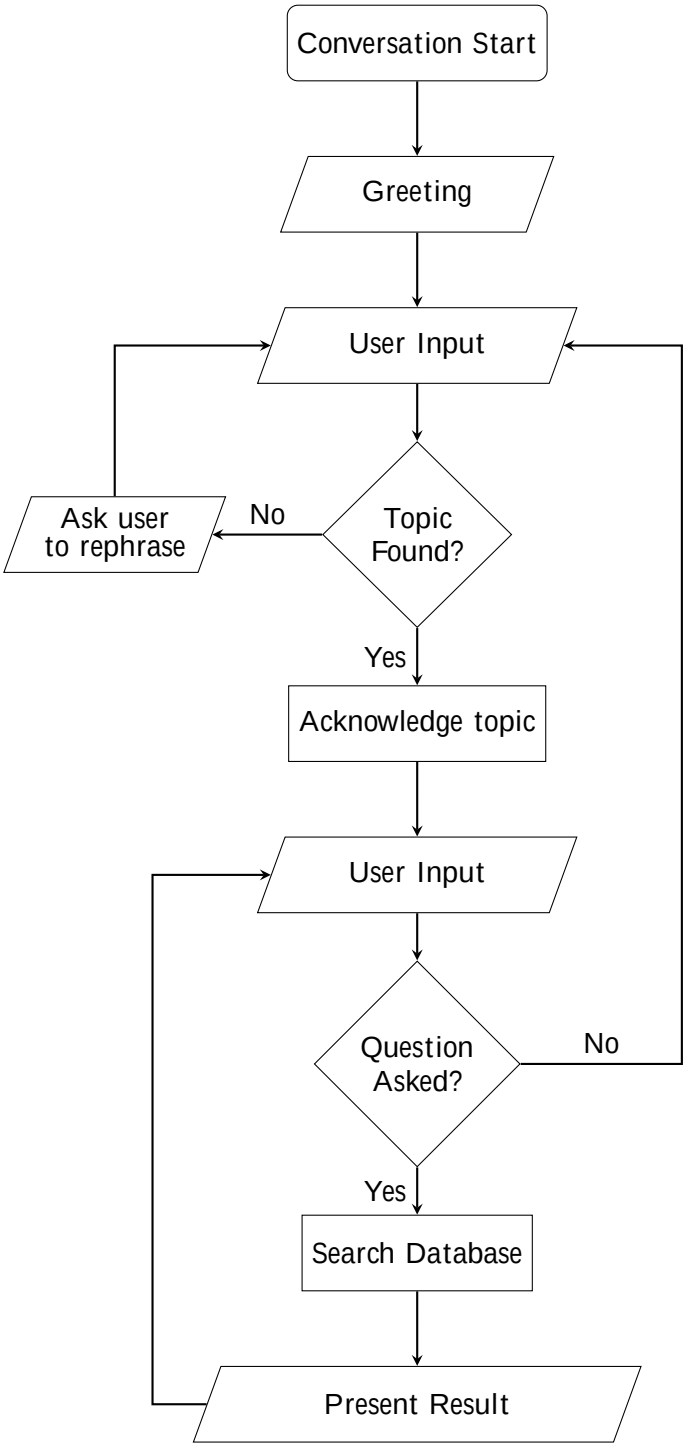

Figure 5.2.: Flowchart for version B and C

5.1.3. Version C

Version C is the final version which was developed. This version used the same process as in Version B but a different search method. Instead of using the highlighter, the document was searched for paragraphs, presenting the paragraph with the highest score. The documents in the database were also structured in a different way. Instead of using the whole document, version C divides the each documents into paragraphs, and pairs each paragraph with the heading preceding it. When searching the document for a match, a weight multiplier was set on each heading, meaning if a heading got a match from the full text search its score would be higher than if a term was found in the paragraph text. The reasoning for doing this is that the headings were often in the form of keywords, that often match the user's question. The reason for using the paragraphs instead of the highlighter as in version A was to reduce the risk of presenting text with insufficient context.

5.2. Analysis

5.2.1. Jayway Feedback

When the chatbot was finished, it was released to the employees at Jayway's Halmstad office. The employees got to try the chatbot and were able to leave feedback on the user experience. They were also asked if the chatbot's answer was correct every time they asked a question.

	Total	Good questions	Unintended questions	User error
Correct	8	8	0	0
Wrong	8	4	1	3
Result	16	12	1	3

Table 5.1.: Jayway feedback

Table 5.2.1 shows the result of the feedback from the employees. In total there were $N = 16$ data points, and 50% of the answers were deemed correct. However, as seen in the table, there are columns with *Unintended questions* and *User error*. The questions that are unintended are questions that the chatbot can not answer. In this case the user tried to small talk with the bot, not presenting a question about the wiki. This kind of small talk is not implemented in to the chatbot and will therefore not have a corresponding answer. *User error* were problems with users asking questions about a different topic than the one they inputted earlier in the conversation. As Elasticsearch then searches the wrong document, this makes it impossible for the chatbot to

retrieve an answer to the question. After filtering out the unintended questions and user error the chatbot answered correctly 66% of the time.

5.2.2. Testing

Version	Correct	Incorrect	Percentage correct	Standard deviation
C	18	7	72%	0.45
B	9	16	36%	0.48
A	6	19	24%	0.43

Table 5.2.: Test result

A test based on a sample of $N = 25$ questions that have an answer on the wiki was also conducted. These questions are listed in appendix A, and the result of this test is presented in table 5.2.2. The result clearly shows that version C achieves the best *percentage correct* compared to versions A and B.

To determine if the different iterations of the chatbot improved upon the previous versions, a two-sample hypothesis test with a significance level of 95% was conducted, testing if version C has twice the accuracy of version A.

$$\begin{cases} H_0 : 2\ _A = \ _C \\ H_1 : 2\ _A < \ _C \end{cases}$$

Assuming the accuracy is normally distributed, the test statistic is given by versions

$$u = \frac{2\ P_A \quad P_C}{\sqrt{p(1 \quad p)\ \frac{1}{n_A} + \frac{1}{n_C}}} \quad 2\ N(0; 1)$$

where p is calculated using

$$p = \frac{2\ \ fx_{Ai} = 1g + fx_{Ci} = 1g}{n_A + n_C}$$

This gives $u = \ 1{:}73205$ and p-value $= 1 \ \ 0{:}9582 = 0{:}0418$, which results in the null hypothesis can be rejected with a significance level of 95%.

6. Discussion

The chatbot that was developed can successfully search and answer questions from the wiki in a natural dialogue, however a more general solution that is easier to adapt for use with di erent data sources would have been more desirable.

6.1. Milestones

As presented in section 1.2 the project had four milestones that had to be accomplished. This section will discuss the results.

6.1.1. Scraper

When building the database using the web scraper each article was successfully retrieved from the wiki without any problems, however, when building the database for version C from the wiki there were problems with the HTML structure of the wiki. It was quite clear that the di erent pages did not have a common structure. Sometimes subheadings were written with heading tags, and other times they were written as bold text in a paragraph tag. This led to problems when dividing each document into paragraphs and pairing the paragraphs with their headings. Another issue with the retrieval of the text was that the web scraper has to be run manually. As explained in section 4.3, Google authentication does not work with an automatic solution. This leads to inconvenience as someone has to manually update the database periodically. The optimal solution would be that it updates every time a change is made to the wiki, but this would require access to the wiki's backend which was not given in this project.

6.1.2. Search engine

Using Elasticsearch as the search engine was a good decision, as it had a lot of good documentation, and a lot of information on how it works. The underlying algorithms and methods that are being used to search a document are also well documented. The search engine was the most di cult part in the project, and

it turned out to be very accurate when the decision was made to not use the built in highlighter but instead break up the documents into paragraphs.

6.1.3. Natural language processing

In the research phase of the project the decision was made to use Dialog ow as the service to handle NLP. Contrary to expectations, a custom NLP solution for handling the input and output using one of the methods presented in chapter 3 was not needed, as Dialog ow handled all of it. When Dialog ow was integrated with the search engine it managed to do this in a way that exceeded expectations. Because of this, the tools available in the service could be used, and no custom NLP solution had to be created. Dialog ow is closed source, and because of this there is little to no information about how it analyses natural language, therefore it was not possible to compare it with other methods. One of the consequences of using entities to represent the di erent articles on the wiki is that synonyms for the names of the articles have to be entered manually. Compared to a more generic method, this results in additional work when adding a new source of data. This means this solution is not optimal for the implementing of other data sources, which was one of the possible extensions listed in section 1.1.

6.1.4. Chatbot

The nal chatbot ended up working very well. It achieved the aim that was set for the project, as it succeeded to retrieve correct answers from the wiki. With the testing described in section 5.2.2 it answered correctly 72% of the time, the test users were satis ed with its answers 66% of the time, which accomplishes the goal of an accuracy of at least 50%, as described in section 4.1. Using a two-sample hypothesis test with a signi cance level of 95%, it was proven that the nal version of the chatbot achieved more than twice the accuracy of version a, with the p-value = 0.0418.

There was no time to integrate it with a voice assistant, and it was not possible to make a generic solution that works with other data sources.

Overall the services that were used were very suitable and produced good results. They were easy to use and get started with, and made it possible to complete the project in the given time frame. However, Dialog ow was lacking in documentation and it was not possible to nd any technical information on its technical workings.

6.2. Societal demands on technical product development

A major bene t of chatbot interfaces is that they are very accessible. Since there is essentially no interface, just human language, anybody who is uent the same language as the chatbot is able to use it. A chatbot that users can speak to using a voice interface is even more accesible, allowing users who cannot use a keyboard to interact with it.

A big consideration when developing any system that handles potentially sensitive company data is preventing unauthorized access to the aforementioned data. This was accomplished by using a reverse proxy with authentication as described in section 4.7.

7. Conclusion

By manual testing and the feedback from Jayway, version C of the chatbot managed to present a correct answer from the wiki with a probability of 72% by the manual testing, and 66% from the Jayway feedback. Using a two-sample hypothesis test with a signi cance level of 95%, it was proven that the nal version of the chatbot achieved more than twice the accuracy of version a, with the p-value = 0.0418. The chatbot communicates using natural language, and can extract keywords from the user's sentences. The chatbot is not suitable as a generic solution and therefore other data sources for information retrieval were not used. The chatbot was not integrated with Google Home because of the time frame of this project.

New knowledge has been gained on how a chatbot works, as it is more basic than one would believe. Further knowledge has been gained on both Javascript and Python, and how di erent services can be combined to form a functional project.

The chatbot could also be extended to somehow give the user suggestions for topics, as the current iteration requires the user to know what information exist on the wiki.

Possible further research could be how chatbots might help the elderly or people with disabilities use the web with a natural dialogue instead of a traditional user interface.

Bibliography

[1] Petter Bae Brandtzaeg and Asbj rn F lstad. Why people use chatbots. In *International Conference on Internet Science*, pages 377{392. Springer, 2017.

[2] Chatbot - report a bike theft. https://www.helvetia.com/ch/web/en/about-us/about-helvetia/information/chatbot-service.html. Accessed: 2019-4-26.

[3] James F Allen, Donna K Byron, Myroslava Dzikovska, George Ferguson, Lucian Galescu, and Amanda Stent. Toward conversational human-computer interaction. *AI magazine*, 22(4):27{27, 2001.

[4] Massimo Canonico and Luigi De Russis. A comparison and critique of natural language understanding tools. *Cloud Computing*, pages 110{115, 2018.

[5] Lars Holmdahl. Dynamic product development DPD. http://www.larsholmdahl.com/pdf/Dynamic%20Product%20Development%20DPD.pdf, 2014. Accessed: 2019-04-29.

[6] Joseph Weizenbaum. ELIZA | a computer program for the study of natural language communication between man and machine. *Communications of the ACM*, 9(1):36{45, 1966.

[7] Rob High. The era of cognitive systems: An inside look at IBM Watson and how it works. *IBM Corporation, Redbooks*, 2012.

[8] Robert Dale. The return of the chatbots. *Natural Language Engineering*, 22:811{817, 09 2016.

[9] Asbj rn F lstad and Petter Bae Brandtz g. Chatbots and the new world of hci. *interactions*, 24(4):38{42, 2017.

[10] Stuart J Russell and Peter Norvig. *Arti cial intelligence: a modern approach*. Malaysia; Pearson Education Limited, 2016.

[11] Dan Jurafsky and James H. Martin. *Speech and Language Processing*. Prentice Hall, 2018.

[12] Peter F Brown, Peter V Desouza, Robert L Mercer, Vincent J Della Pietra, and Jenifer C Lai. Class-based n-gram models of natural language. *Computational linguistics*, 18(4):467{479, 1992.

[13] Anand Rajaraman and Je rey David Ullman. *Mining of Massive Datasets*. Cambridge University Press, New York, NY, USA, 2011.

[14] Apache lucence - scoring. `https://lucene.apache.org/core/2_9_4/scoring.html`. Accessed: 2019-04-29.

[15] N Luburic and D Ivanovic. Comparing apache solr and elasticsearch search servers. *ICIST*, 2016.

[16] Julian Reschke. The 'basic' http authentication scheme. Technical report, Internet Engineering Task Force, 2015.

A. Questions

#	Topic	Query
1	Vacation	How many vacation days do I get?
2	Malmo	Where is the o ce located?
3	Malmo	Where is the Malmo o ce located?
4	Our Culture	Tell me about Jayway's culture
5	The Box	What is the box
6	Competence Development	competence leads halmstad
7	Meet Wiki	Who is Wiki?
8	KPerson & BoB	What is Kperson
9	Halmstad	what is the code to the door
10	Slack	how do i join the slack channel
11	Health & Wellness	How much money can I spend on friskvardsbidrag
12	Company Holidays	What public holidays are there in Sweden?
13	Leave - Sickness, parental leave and leave of absence	What should I do if I'm sick?
14	About Jayway	when was it founded?
15	Stockholm	Where is the o ce located?
16	WiFi	What is the password in Malmo?
17	Wiki Admin	How do i edit a page?
18	O ce hours & Time reporting	How much am i expected to work
19	US bene ts	How much support do i get for a gym membership
20	Logos & Typography	What font should i use?
21	Company texts	What is the danish text?
22	About Jayway	What is Jayway?
23	Traveling & Accomodation	How do I get reimbursed for travel expenses
24	Mailing lists	What is the mail to group management?
25	Mailing lists	What is the mail address to all in Halmstad?

#	A	B	C
1	Correct	Correct	Correct
2	Incorrect	Correct	Incorrect
3	Incorrect	Correct	Correct
4	Incorrect	Incorrect	Correct
5	Incorrect	Incorrect	Correct
6	Incorrect	Incorrect	Correct
7	Incorrect	Incorrect	Correct
8	Correct	Correct	Incorrect
9	Incorrect	Correct	Correct
10	Incorrect	Incorrect	Incorrect
11	Incorrect	Incorrect	Correct
12	Correct	Correct	Incorrect
13	Incorrect	Incorrect	Correct
14	Correct	Correct	Correct
15	Incorrect	Incorrect	Incorrect
16	Incorrect	Incorrect	Correct
17	Incorrect	Incorrect	Correct
18	Incorrect	Incorrect	Correct
19	Incorrect	Incorrect	Correct
20	Correct	Correct	Incorrect
21	Correct	Correct	Correct
22	Incorrect	Incorrect	Correct
23	Incorrect	Incorrect	Incorrect
24	Incorrect	Incorrect	Correct
25	Incorrect	Incorrect	Correct
Total Correct	6	9	18
Total Incorrect	19	16	7

Falk Höppner, Computer Science and Engineering, 300 credits

Joakim Fredriksson, Computer Science and Engineering, 300 credits

PO Box 823, SE-301 18 Halmstad
Phone: +35 46 16 71 00
E-mail: registrator@hh.se
www.hh.se