VU UNIVERSITY AMSTERDAM

MASTER THESIS

Designing a Dutch financial chatbot

Applying natural language processing and machine learning techniques to retrieval-based question answering

---

Author:
Suzanne WETSTEIN

Supervisors:
Tobias KUHN
Ronald BUITENHEK

A thesis submitted in fulfillment of the requirements
for the degree of Master of Science in Business Analytics

in the

Web and Media Group
Department of Computer Science

July 14, 2017

VU University Amsterdam

Abstract

Faculty of Sciences
Department of Computer Science

Master of Science in Business Analytics

Designing a Dutch financial chatbot

by Suzanne WETSTEIN

Experts have estimated that eighty to ninety percent of data in any organization is unstructured, and this amount is growing significantly. Therefore, techniques that can deal with unstructured data like text need to be developed. In this thesis we have researched techniques that can deal with text by developing a chatbot. A chatbot is a conversational agent that can interact with humans turn by turn using natural language. The main objective of this graduation project was to build a chatbot to answer highly complex questions, which often require even more complex answers, in a well-defined domain. The focus was on only answering questions to which high-quality answers can be given. Three chatbot implementations based on different underlying models have been built and compared. The first model was a vector space model consisting of a self-constructed Natural Language Processing (NLP) pipeline, a term weighting scheme and a similarity measure based on the cosine between two question vectors. The second was a machine learning model based on many features, like sequence matching and TF-IDF. The third model was a combination of the first two. The best performing chatbot was based on a combination of the vector space model and the machine learning model and answers 23% of user questions (the rest is sent through to human experts), and 74% of these questions are answered correctly.

We have shown that with a combination of NLP techniques, machine learning techniques and a well-thought-out implementation we can solve the problem of building a chatbot to answer highly complex questions at a sufficient quality level. Future work in the area of chatbots is advised to focus on further improving the Dutch NLP pipeline and researching generative-based chatbots.

Contents

Abstract **i**

1 Introduction **1**
- 1.1 Chatbots . . . 1
- 1.2 Kandoor . . . 2
- 1.3 Research goals . . . 2
- 1.4 Report structure . . . 3

2 Literature study **4**
- 2.1 Chatbot types & functionalities . . . 4
- 2.2 Chatbot literature . . . 8
- 2.3 Natural Language Processing . . . 23
- 2.4 Vector space model . . . 28
- 2.5 Machine learning models . . . 28

3 Methodology **35**
- 3.1 Vector space model . . . 35
- 3.2 Machine learning model . . . 36
- 3.3 Chatbot implementation . . . 37

4 Vector space model **39**
- 4.1 Data . . . 39
- 4.2 Natural Language Processing . . . 50
- 4.3 Term weighting . . . 54
- 4.4 Measuring similarity . . . 56
- 4.5 Results . . . 58

5 Machine learning model **60**
- 5.1 Data . . . 60
- 5.2 Natural Language Processing . . . 64
- 5.3 Feature engineering . . . 64
- 5.4 Modelling . . . 70
- 5.5 Results . . . 77

6 Chatbot implementation **82**
- 6.1 Chat flows . . . 82
- 6.2 Confidence levels . . . 83
- 6.3 Dynamic learning . . . 88
- 6.4 Results . . . 88
- 6.5 Model comparisons . . . 96

7 Conclusion & Discussion **98**

A NLP: word lists **101**
A.1 Stop-word lists 101
A.2 Compound word list 103
A.3 Synonym list 103

Bibliography **106**

List of Figures

2.1 A diagram of chatbot types. Based on a figure from [16]. 5
2.2 A timeline with the chatbots described in this section. 8
2.3 An example of a chat with ELIZA, the first chatbot. ELIZA is based on a simple pattern matching algorithm. 10
2.4 This is a representation of the simulated world of SHRDLU, accompanied by a conversation in which the blocks are moved [82]. 11
2.5 An example of chatting with ALICE, a complex pattern matching chatbot. 12
2.6 The general framework behind question-answering systems [84]. . . . 13
2.7 A timeline with the QA systems described in this section. 13
2.8 An overview of the FRACT architecture [38]. 18
2.9 An overview of the DeepQA architecture [23]. 18
2.10 Two examples of e-mail suggestions given by Google Smart Reply [18]. 20
2.11 Example of classification with the kNN algorithm [59]. 29
2.12 An example of finding the maximum margin hyperplane in SVM [39]. 31
2.13 An example of non-linear SVM: using a kernel to linearly divide the data in a higher dimensional space [55]. 31
2.14 The structure of an artificial neural network with one hidden layer [22]. 32
2.15 Diagram of a sequence-to-sequence neural network as used in Google Smart Reply [18]. 33
2.16 Architecture of a dual encoder LSTM network [45]. 34

3.1 A diagram of the chatbot framework. 35
3.2 A diagram of the vector space model. 36
3.3 A diagram of the machine learning model. 37
3.4 A diagram of the implementation of the chatbot. C_l stands for the confidence level of the match between the incoming question and the best matching question in the dataset. If the confidence level is high a question is immediately answered by the chatbot. A lower confidence level can lead to a return question, or the question being sent through to an expert. 38

4.1 A bar chart displaying how many question-answer pairs contain a certain word or hyper-link. 40
4.2 Charts displaying the distribution of the number of words in the questions and answers. 45
4.3 Charts displaying the distribution of the sentences of words in the questions and answers. 45
4.4 Heat maps displaying the relation between the number of words and sentences in question and answer. 46
4.5 A bar chart displaying the frequency of the 25 most used words in the questions and answers. 47

4.6 Histograms displaying the count and Jaccard distance of the common unigrams and bigrams in the question-answer combinations. 49
4.7 A schematic overview of the NLP pipeline. 52
4.8 The chatbot screen with an answer to a newly typed question. 57
4.9 The chatbot screen showing the question that is matched to a newly typed question. 58
4.10 The evaluation framework. 59

5.1 Charts displaying the distribution of the common unigrams Jaccard distance for matching and non-matching questions. 63
5.2 Charts displaying the distribution of the TF-IDF weighted common unigrams Jaccard distance for matching and non-matching questions. . 63
5.3 A visualization of the word2vec model. Only words used at least 400 times in the dataset are displayed. 69
5.4 A bar chart displaying the feature importances in the best performing XGBoost model. 73
5.5 A bar chart displaying the feature importances in the best performing Random Forest model. 74
5.6 The results of all machine learning models on different evaluation metrics. 80
5.7 The average and standard deviation of critical parameters 81

6.1 A schematic overview of the chatbot flow. 82
6.2 Charts displaying the distribution of the cosine similarity for matching and non-matching questions. 84
6.3 A chart displaying the cumulative distribution of the cosine similarity for matching and non-matching questions. 84
6.4 Charts displaying the distribution of the XGBoost score for matching and non-matching questions. 85
6.5 A chart displaying the cumulative distribution of the XGBoost score for matching and non-matching questions. 86
6.6 Charts displaying the distribution of the cosine similarity for matching and non-matching questions in the XGBoost model. 87
6.7 A chart displaying the cumulative distribution of the cosine similarity for matching and non-matching questions in the XGBoost model. . . . 88
6.8 The chatbot screen with a good answer to a newly typed question. . . 89
6.10 The chatbot screen after correctly answering a question in which a return question was answered with "Yes". 90
6.9 The chatbot asking a return question. 90
6.11 The chatbot screen with a question that cannot be answered by the chatbot and is sent through to an expert. 91
6.12 An overview of the results of the vector space model chatbot implementation. 92
6.13 The confusion matrices for the implementation of the vector space model. 92
6.14 An overview of the results of the vector space model chatbot implementation. 93
6.15 The confusion matrix for the implementation of the machine learning model. 93

6.16 An overview of the results of the combination model chatbot implementation at a confidence level of 0.4 with a return question at a cosine similarity of 0.25. 94
6.17 An overview of the results of the combination model chatbot implementation at a confidence level of 0.4 without return question. 94
6.18 An overview of the results of the combination model chatbot implementation at a confidence level of 0.3 without return question. 94
6.19 An overview of the results of the combination model chatbot implementation at a confidence level of 0.25 without return question. 95
6.20 The confusion matrices for the implementation of the vector space model. 95
6.21 The results of the chatbot implementations of all models. 96

List of Tables

2.1 A schema depicting the improvements of and techniques used in the chatbots described in this section. 9
2.2 A schema depicting the improvements of and techniques used in the QA systems described in this section. 14

4.1 Ten randomly sampled questions with answers from the Kandoor dataset. 43
4.2 Inverse document frequencies for the questions in the dataset. 55
4.3 The document-term matrix with term frequencies. 56
4.4 The document-term matrix with TF-IDF scores. 56
4.5 Results of the vector space model per financial category. 59

5.1 The framework of the Q2Q dataset. 60
5.2 The framework of the Q2Q dataset including the questions prepared with NLP techniques. 64
5.3 A table depicting a word and the words most often used in the same context as this word. In this case the five words which are used in the most similar context are shown. 69

A.1 Table of synonyms 105

List of Abbreviations

AI	**A**rtificial **I**ntelligence
FNN	**F**eed-forward **N**eural **N**etwork
IR	**I**nformation **R**etrieval
JSON	**J**ava**S**cript **O**bject **N**otation
kNN	**k** **N**earest **N**eighbours
LAT	**L**exical **A**nswer **T**ype
LSA	**L**atent **S**emantic **A**nalysis
LSTM	**L**ong **S**hort-**T**erm **M**emory
MMI	**M**aximum **M**utual **I**nformation
NER	**N**amed **E**ntity **R**ecognition
NLP	**N**atural **L**anguage **P**rocessing
PA	**P**redictive **A**nnotation
PoS	**P**art **o**f **S**peech
Q2Q	**Q**uestion-to-**Q**uestion
RNN	**R**ecurrent **N**eural **N**etwork
SLM	**S**tatistical **L**anguage **M**odelling
SMT	**S**tatistical **M**achine **T**ranslation
SVM	**S**upport **V**ector **M**achine
TF-IDF	**T**erm **F**requency-**I**nverse **D**ocument **F**requency
UI	**U**ser **I**nterface

Chapter 1

Introduction

In the last few years consulting companies have more and more used machine learning to support their clients in making decisions. Especially in the area of Marketing and Internet of Things models are needed that can deal with a lot of data, unstructured data and text. Experts have estimated that eighty to ninety percent of data in any organization is unstructured, and this amount is growing significantly [29]. Therefore, techniques that can deal with unstructured data like text need to be developed and improved.

Machine learning as a new branch in data science can be further explored. A chatbot, or question-answering system, is an application in this field. Question-answering systems have been around since the 1970s but most systems can only give short and factual answers. There are systems based on shallow NLP and Information Retrieval (IR) techniques and systems based on Statistical Machine Translation (SMT). The first type is limited to giving answers from an underlying dataset and has very little understanding of the connection between words in a sentence. The second type focuses on generating a new word or sentence based on previous words or sentences. This technique is very promising but only recently, with the use of neural networks, these systems have started to give sensible answers.

In this study, we build a chatbot to answer very complex questions, which often require even more complex answers, in a well-defined domain (the financial domain). Here we focus on only answering questions to which high-quality answers can be given. Other questions are directed to human agents. Such a system can be very useful for companies as they often get complex questions and want to give high-quality answers to their clients, while also minimizing the cost of hiring human agents. A chatbot could for example help run their helpdesk, react to social media messages for them or help choose from a series of products. In the process of creating a chatbot this study focuses on techniques in NLP and machine learning.

1.1 Chatbots

A chatbot is a conversational agent that can interact with humans turn by turn using natural language. 'Chatbot' used to be a term just used for chatterbots; natural language dialogue systems designed just for fun with shallow knowledge about a lot of topics. Today however, 'chatbot' is a term used for any chatting robot, automation or chat channel. Chatbots are being developed in many domains and for different purposes. Chatbots give psychological assistance, are used in customer service systems and just for fun. Although chatbots are and have been researched thoroughly, the current systems are nowhere near seeming human or giving mostly correct responses. This is greatly due to the difficulty of interpreting natural language by computers. Designing a computer system which can understand and generate human language is a branch of Artificial Intelligence (AI), more specifically NLP.

Chatbots are immensely popular at the moment. There are numerous online chatbot magazines like Chatbots Journal[1] and communities like Chatbot News[2]. This popularity is brought on by big companies like Facebook, Microsoft, Google and IBM making breakthroughs in the AI world. Microsofts' Xiaoice, a Chinese virtual friend, has been available for more than two years and has about forty million users. The average user interacts with Xiaoice sixty times per month. Facebook is creating an online assistant named M and already has bots for small things, like booking a haircut or sending flowers, on its Messenger app. IBM created Watson, a question answering computer system, which proved its power by winning the popular quiz show Jeopardy! Finally, Google is also working on a chatbot-based app to answer user questions [6]. Another thing making chatbots popular is the ease with which anyone can create one themselves, without any programming skills needed. There are many websites and services that can help you create your own bot. Popular ones are API.ai, Chatfuel, Kit Bot and Facebooks' wit.ai.

1.2 Kandoor

Kandoor is a digital knowledge platform which helps to link financial questions to financial information. The company is a start-up from APG, a big Dutch pension provider. Kandoor offers free financial knowledge to everyone (in Dutch). Financial questions asked by users are classified into financial categories by using machine learning techniques. The questions are then answered by experts in the respective categories.

Currently, Kandoor has received and answered around 3800 financial questions, and for this process a lot of expert time is needed. Therefore, Kandoor is interested in a chatbot that can answer (some of) the questions for them. As a dot on the horizon, they would like to have a chatbot that can answer any question, but for now they would like to create an achievable chat system and do research in what chatbots can mean for them in the future. With an achievable system we mean that if a similar question has already been answered by an expert the bot should answer the new question with the same answer. If a question has never been asked before, it should not be answered but sent through to an expert with the right knowledge. In answering the questions it is very important that the bot gives correct answers, therefore in case of doubt the question should be sent through to an expert.

1.3 Research goals

The main objective of this study is to build a chatbot to answer very complex questions, which often require even more complex answers, in a well-defined domain (the financial domain). We focus on only answering questions to which high-quality answers can be given. This chatbot should use NLP and machine learning techniques and retrieve the answers from a database of questions-answer pairs.

The project consists of four components:

- Text cleaning
- Engineering features from the cleaned text

---

[1] https://chatbotsjournal.com/

[2] http://news.chatbotsmagazine.com/login

- Matching an incoming question to questions in the dataset based on the features
- Evaluating the chatbot answers

In the first part of the project we focus on researching techniques used for each of these four components in existing chatbot systems. Then methods are selected that fit with complex questions and high-quality answers, and with these methods the chatbot is built.

Success criteria for this study are a working chatbot system, use of NLP techniques, use of multiple machine learning techniques and a comparison of these techniques. Furthermore, the results of the chatbot system are to be evaluated in the end.

1.4 Report structure

This thesis is structured as follows.

Chapter 2 is an extensive literature study into chatbots. First, we describe the various types of chatbots and whether these types are appropriate for complex questions. Also discussed here are the technological possibilities and limits of implementing certain chatbot functionalities. Second, we give an overview of important chatbots in history and dive into the techniques behind existing question-answering systems relevant to our problem. In the last sections we explain NLP techniques, the vector space model and machine learning models that can be applied in chatbots.

Chapter 3 gives an overview of the different chatbot systems created. It describes step-by-step how the systems are built and which components are used in which chatbot version.

Chapter 4 presents the vector space model we created. First the dataset is explored and its most notable properties are described. Furthermore, in this chapter NLP techniques are applied to the dataset and their quality for the Dutch language is described. The term weighting and measuring similarity sections describe the techniques with which the input question is matched to a question in the underlying dataset to retrieve an answer. Finally, the results of this model are evaluated.

Chapter 5 describes the chatbot model based on machine learning techniques. A new dataset is created for this model and the process of creation and its properties are described. Furthermore, we describe the engineering of features and the various machine learning techniques used for modelling. Finally, the results of the different machine learning models are presented and compared.

Chapter 6 describes the implementation of the models in a chatbot application. In this chapter we determine the chat flows and set the confidence level for which a question is answered. Finally, we present the results of the implemented models and compare them.

Chapter 7 concludes the work and gives an overview and discussion of the findings. The limitations of the work are presented and future improvements are suggested.

Chapter 2

Literature study

As we consider the literature study, we note that there is a wide variety of chatbots and chatbot techniques. In section 2.1 we describe the various chatbot types and their functionalities. In section 2.2, we review historically ground-breaking as well as more recent chatbots. The NLP techniques and the vector space and machine learning models used in these chatbots are explained in more detail in section 2.3, 2.4 and 2.5, respectively.

2.1 Chatbot types & functionalities

In this section we describe the different chatbot types and functionalities. We discuss which types and functionalities are relevant for domains with highly complex questions and a need for reliable answers. Furthermore we discuss which chat functionalities are technologically achievable at this point in time.

2.1.1 Chatbot types

Chatbots exist in a variety of types. The most commonly used categorization can be seen in figure 2.1. In this figure, chatbots are organized into categories based on the type of domain they operate in and based on the way they give answers. Open domain chatbots can answer questions about a great variety of topics, whereas closed domain chatbots only function inside a specific domain in which they have knowledge. Of course there is not a hard split between open domain and closed domain. A chatbot used to react on Twitter statuses would be seen as open domain and a chatbot used to order pizza would be seen as closed domain, but a chatbot to answer all types of financial questions would be somewhere in between (although more closed domain than open domain).

Chatbots can be either retrieval-based or generative-based. Retrieval-based means that answers are retrieved from an underlying dataset. The retrieval process can be simple using rule-based sentence matching, or advanced using an ensemble of machine learning techniques. These systems will always return an existing answer from a set of pre-defined responses. Generative-based chatbots do not give pre-defined responses; these systems generate the response themselves.

In figure 2.1, we can see that there are roughly four types of chatbots:

1. **Open domain with retrieval-based responses**
   Retrieval-based responses are responses from a fixed set, in an open domain this fixed set should be 'any possible question anyone can think of'. Of course, this cannot be done and therefore this type of chatbot is impossible to create.

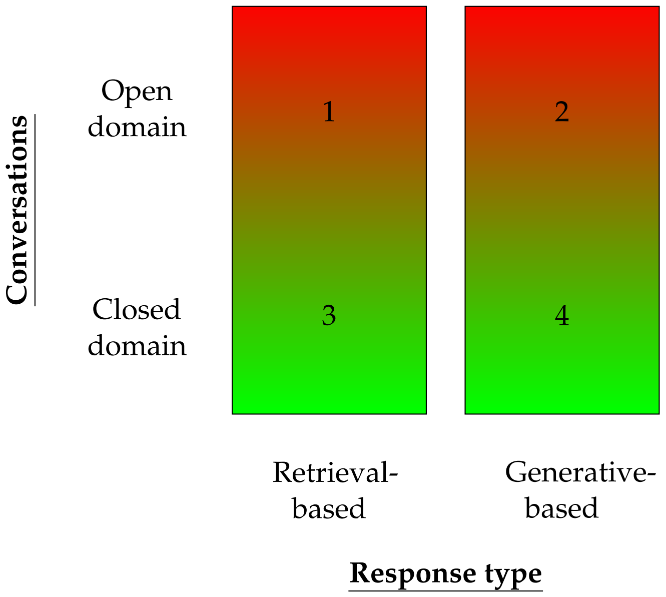


FIGURE 2.1: A diagram of chatbot types. Based on a figure from [16].

2. **Open domain with generative-based responses**
   Here again we want to be able to ask any possible question and get an appropriate response. Solving this problem generative-based is called Artificial General Intelligence (AGI) [26]. This means that the chatbot is a smart machine that can successfully perform the same intellectual tasks humans can. Although this area is being researched intensively, we are not quite there yet.

3. **Closed domain with retrieval-based responses**
   In this area the chatbot is trained on a dataset with text on a certain domain. Questions inside the domain will be answered with one of the responses in the training dataset. The chatbot does not have an answer to questions asked outside this domain, and to unforeseen questions inside the domain. The way most companies deal with this problem is that when there is a question the chatbot cannot answer the question is passed to a human.

4. **Closed domain with generative-based responses**
   Here the chatbot should uses smart machine technology to generate the answer to a question. Generated responses make the chatbot able to handle questions present in the underlying dataset but also new questions. This type of chatbot can also deal with interactions that are longer than one question and one answer and they tend to give more human-like answers and can have their own personality. Downsides here are that generative responses increase the complexity of the problem by a lot, generated answers are usually full of grammatical errors and these chatbots require huge amounts of training data.

As open domain retrieval-based systems are impossible and open domain generative-based systems have not even been successfully built by Google or Microsoft yet, we stick to closed domain chatbots more specifically the financial domain. Both retrieval-based and generative-based closed domain systems are already applied in practice today, though more research to improve the accuracy of responses and to expand the chatbots' domain is needed. We focus on giving reliable and grammatically correct answers, which is why the closed domain retrieval-based chatbot is most appropriate here. In the future, when generative-based techniques have improved and a larger dataset of domain questions and answers is available it might be possible to create a generative-based chatbot for the same case.

Related chatbot terms

In the AI world a lot of different terms are being used for seemingly the same systems. A chatterbot is an online friend with whom you can make light conversation. In this case, the chatbot is just a simple pattern matching system designed to seem as human-like as possible and win competitions like the Loebner Prize.[1] Today, the term 'chatbot' is used for any bot, automation or chat channel, but there are also other terms that are used for the same systems. An Automated Online Assistant (AOA) is a system that assists a user by answering questions and performing some light tasks. For example, an AOA can tell a user what a certain T-shirt costs in its store or it can perform the task of ordering the T-shirt. Chatterbots usually use simple pattern matching over the input to generate their output, but AOA's use more advanced techniques like NLP with word and sentence segmentation, Named Entity Recognition (NER), syntactic and semantic analysis and categorization. These more advanced techniques can also be found in question-answering (QA) systems, but these are usually simpler than AOA's because they are only designed to answer questions and not perform tasks. Also in QA systems interaction is modelled as turn by turn asking a question and giving an answer and not as a chat as a whole.[21] Overall, we think that AOA's, question-answering systems and chatterbots are all types of chatbots. The chatbots designed in this study are most alike to question-answering systems.

2.1.2 Chat functionalities

Chatbots can have many functionalities, and some functionalities are very important for our application. We very strongly need the bot to give a correct answer, we need the bot to collect extra information from the user and we would like a bot that can chat, instead of giving an answer to one question. In this section we discuss the techniques needed to realise these chatbot functionalities and their technological limits.

Methods to enhance the probability of giving a correct answer

For most companies giving a correct answer to their clients is very important. There are different methods to give more certainty that the answer of the chatbot is correct:

- Many machine learning algorithms can give a percentage of certainty with which they think the output for a certain question is correct. We can program the chatbot to only give an answer if this certainty level is high. In case the certainty level is too low we can sent the question through to an expert with the proposed chatbot answer already filled in. If the expert agrees with this answer all he has to do is click sent, if he does not agree he can remove or change it.

- The chatbot matches a new incoming question to an question in the dataset and if these are similar the chatbot gives back the answer an expert gave to

---

[1] The Loebner Prize is an annual AI competition in which chatterbots compete to be judged as the most human-like

the matched dataset question. To be more sure of giving the right answer the chatbot can first ask the user: "Is your question similar to [dataset question]?" If the user answers yes than the chatbot can give the correct answer, if not the question can be sent through to an expert. Technologically this is not more difficult to implement than a chatbot without this system.

- In the question-answer dataset there are a lot of different questions and answers, but some of them will actually be the same question. If we could create a set of standard questions and answers and replace all the similar questions by these it would be easier to classify a new incoming question on the dataset and this would create a bigger chance of giving a correct answer. Technologically, if there are many similar questions in the dataset it is difficult to match a new question to one of these questions, if the answers to the questions are also similar this matching should not be necessary.

Answering methods

There exist different AI systems for solving user problems. Information Retrieval (IR) is one way of answering a question. IR is a query driven approach for accessing information, here the systems does not return an answer but a list of documents related to the user query. The user is responsible for navigating through these documents and finding the right information. An example of IR is Google Search. Question-answering is an answer driven approach for accessing information. The user asks a question in natural language and the system returns a list of short answers.

In our case it is necessary to give one specific answer to a question. It is okay if this answer contains a link to a website with more information, but we should not return a list of documents for the user to search through. Technologically, since we want to return the best answer from our dataset, in the background we still built a ranked list of answers to this question, but we would program the chatbot to only display the highest ranked answer.

Another feature would be to let the bot chat with the user and not just answer only its first question. When programming a chat we need to keep context (previous questions) in memory and not see the new question as a stand-alone query. Technologically, we can give the words in the previous questions a smaller weight than the words in the current question but still compose a word vector also taking into account the previous questions. Now we can match this word vector to the questions in our database.

Returning a question

For answering domain-specific complex questions sometimes extra information is needed from the user. For example, if a client asks in which year he can get a pension, we need to know in what year this person was born. It would be a good feature if the chatbot could recognize this and return a question asking the user about his date of birth.

Technologically, returning a relevant question is quite difficult. This is because the chatbot needs to understand what kind of situation the client is in, what kind of information is needed to be able to answer that question, and which information the client is and is not providing. To make the implementation a bit easier it would be an option to work with a template, this would work like: if a question is matched to this specific frequently asked question than check if there is a birth year mentioned in the question, if not ask for a birth year. Of course, this is a non-flexible rule-based solution which only works in this specific case, but at the moment the technology to solve this in another way has not been invented yet.

2.2 Chatbot literature

We first give an overview of the most ground-breaking chatbots in history. Then we go deeper into the techniques behind question-answering systems. Here we focus on the technologies relevant to the chatbot we create. We describe the inner workings of question-answering systems based on databases with text documents (instead of structured knowledge bases). Relevant to this project are retrieval-based chatbots in the domain of question-answering systems. When generative-based chatbots further improve these systems might become relevant. Therefore in the third part of this section, we give an overview of future chatbot possibilities in the form of generative-based systems. In the last part of this section, we review chatbot evaluation methods.

2.2.1 Chatbot history

In this section we describe the most ground-breaking chatbots in history. A time line with the chatbots described is shown in figure 2.2.

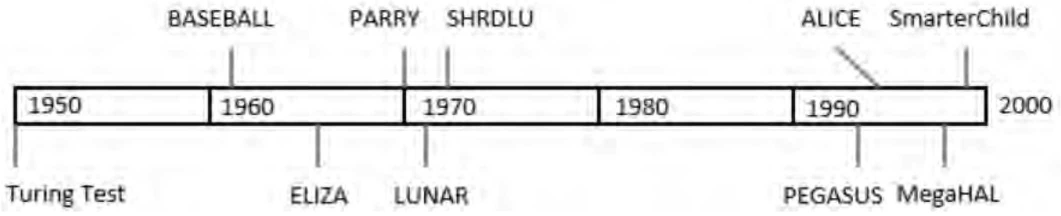


FIGURE 2.2: A timeline with the chatbots described in this section.

The techniques used in these chatbots and the main improvement of this chatbot compared to its predecessors can be found in table 2.1.

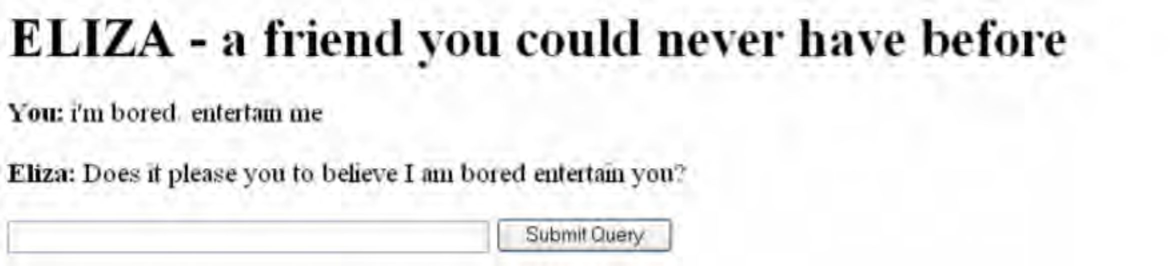
In the 1950s and ’60s computer scientists Alan Turing and Joseph Weizenbaum experimented with the concept of computers communicating like humans do. Alan Turing developed the **Turing test** [75], a test of a computer’s ability to portray intelligent human-like behaviour. This test involves two humans and a computer. A human enters questions into a computer and will receive answers from either the other human or the computer, the challenge here is for the question-enterer to correctly identify whether he is talking to a human or a computer. At the time this game was invented no computer came close to passing this test and, although we are getting closer, at this time still no computer has passed this test. Most computers that come close do so by using tricks like giving responses that match with any input question and not by actually understanding the user’s question or statement.

Joseph Weizenbaum created **ELIZA** [80] the first chatterbot. ELIZA was not intended to be a useful or intelligent chatbot, but as a demonstration of how superficial human-computer interaction tends to be. The bot was created in 1966 and the program used only 200 lines of code to imitate the language of a psychotherapist. Although ELIZA was not built to be human-like, and its users knew they were interacting with a chatterbot, still, due to its emotional responses, people would grow attached to ELIZA. The algorithm behind ELIZA was simple and based on keyword matching. The users’ input is analysed and keywords are detected. If a keyword

Chatbot	**Improvement**	**Main technique**	**NLP techniques**
BASEBALL	First QA system	Knowledge database and pattern matching	None
ELIZA	First chatbot system	Pattern matching	Keyword matching
PARRY	Added tricks to seem more human	Pattern-matching	Keyword matching
LUNAR	A manually written knowledge base	Based on knowledge database and pattern-matching	None
SHRDLU	Large input flexibility	Pattern matching	Parsing and keyword matching
PEGASUS	Speech recognition	Speech recognition and pattern matching	Keyword matching
ALICE	More complex pattern matching using XML	Pattern matching	Keyword matching
MegaHAL	First chatbot using machine learning	Machine learning: Markov models	Tokenization, stopword removal and keyword matching
SmarterChild	First chatbot distributed over messaging platforms	Pattern matching	Keyword matching

TABLE 2.1: A schema depicting the improvements of and techniques used in the chatbots described in this section.

is found, the input sentence is mapped according to a rule associated with the keyword, if no keyword is found one of a set of fixed phrases is returned, or under certain conditions an earlier remark is retrieved. The set of fixed phrases to answer when no keyword is found exists of comments such as "Can you elaborate on that?" and "Very interesting, please go on." An example when a keyword is found and simple pattern matching is applied would go as follows: User:"I like X", ELIZA:"That's great! I like X as well!", where X can be any word. An example of a chat with ELIZA can be found in figure 2.3.



ELIZA - a friend you could never have before

You: i'm bored entertain me

Eliza: Does it please you to believe I am bored entertain you?

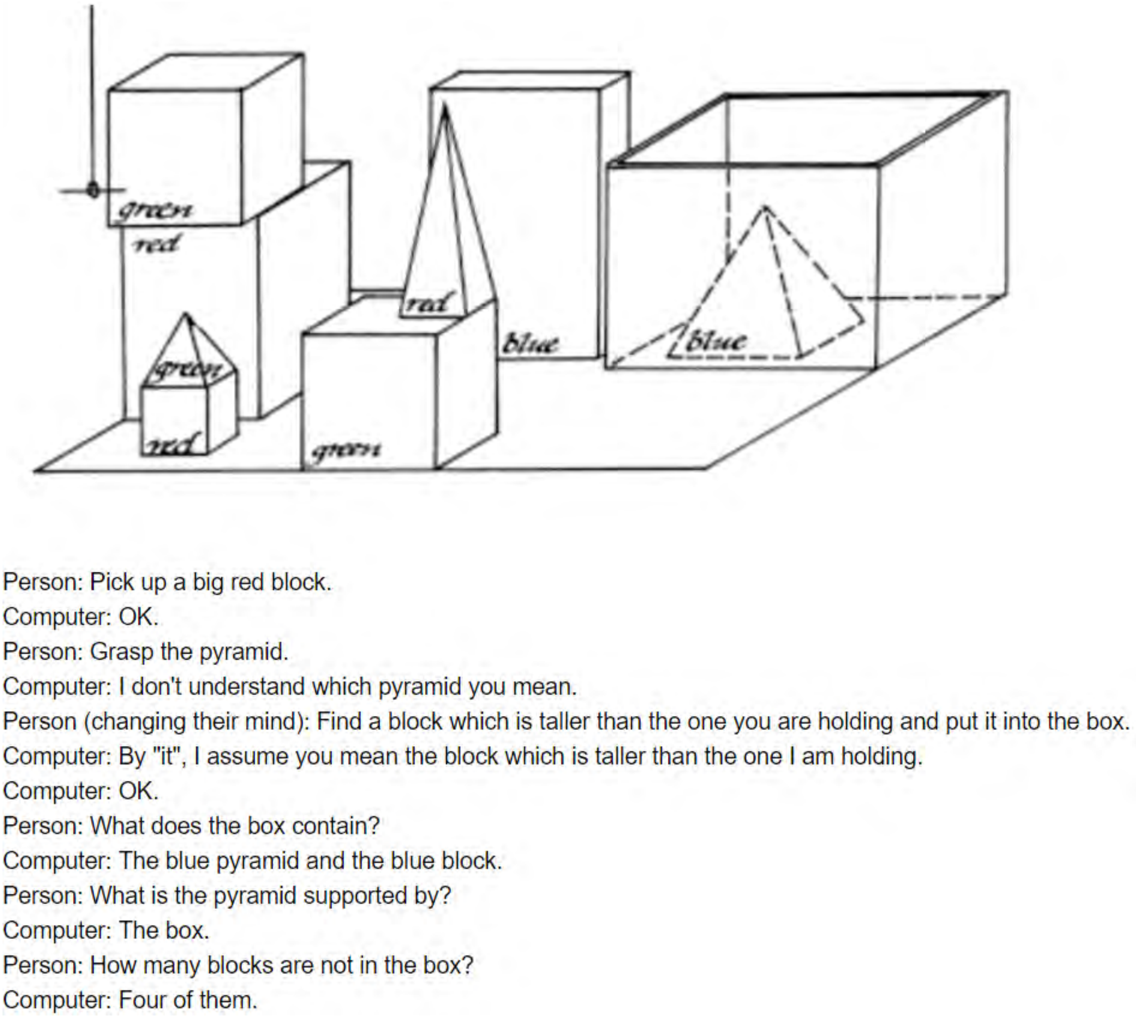
FIGURE 2.3: An example of a chat with ELIZA, the first chatbot. ELIZA is based on a simple pattern matching algorithm.

After ELIZA the next major chatbot was **PARRY** [17], implemented by Colby in 1970. PARRY, in contrast to ELIZA, was modelled as a paranoid patient as he spoke to his therapist. He was initially believed to have passed the Turing test. In the test a group of psychiatrists had conversations with real patients and with PARRY, and another group of psychiatrists reviewed the transcripts of these conversations and judged whether the conversation was with PARRY or a human. The psychiatrists only guessed correctly for 48 percent of the conversations which is consistent with random guessing. Later, this result was subtracted from a passing of the Turing test because in longer one-on-one conversations PARRY could not hold up. PARRY relied on pattern matching techniques and a few tricks, like telling short stories, for if no answer was found. This rule-based system searched for a condition in the input phrase and then executed the corresponding action.

Early question-answering systems were **BASEBALL** [28] and **LUNAR** [85]. BASEBALL was a system that could answer questions about the US baseball league and LUNAR could answer questions about the geology of moon rocks and soil gathered by the Apollo 11 missions. In 1971 LUNAR was demonstrated at a science convention where untrained people asked the system domain-focused questions and it managed to answer 90 percent of the questions. Both systems used techniques similar to the ELIZA chatterbot and were based on a manually written knowledge database.

SHRDLU [83] was a question-answering machine implemented by Terry Winograd in 1972. This program showed a lot of flexibility in the input and common sense compared to systems like ELIZA, PARRY and the early question-answering systems. SHRDLU operates in a simulated world where it can move different coloured and shaped blocks. A representation of this block world and an accompanying conversation can be found in figure 2.4.

In the conversation in figure 2.4 SHRDLU seems to give some quite intelligent responses. It looks like SHRDLU shows a genuine understanding for what the instructor is saying, but all this is based on is a form of symbolic manipulation. SHRDLU uses a left-to-right, top-down parser which analyses the pattern of the instruction, identifies the structure, and also recognizes the sentences' features and grammar. To make SHRDLU understand what to do, the instruction is represented in terms of the objects, their color and shape, and the relations between them.

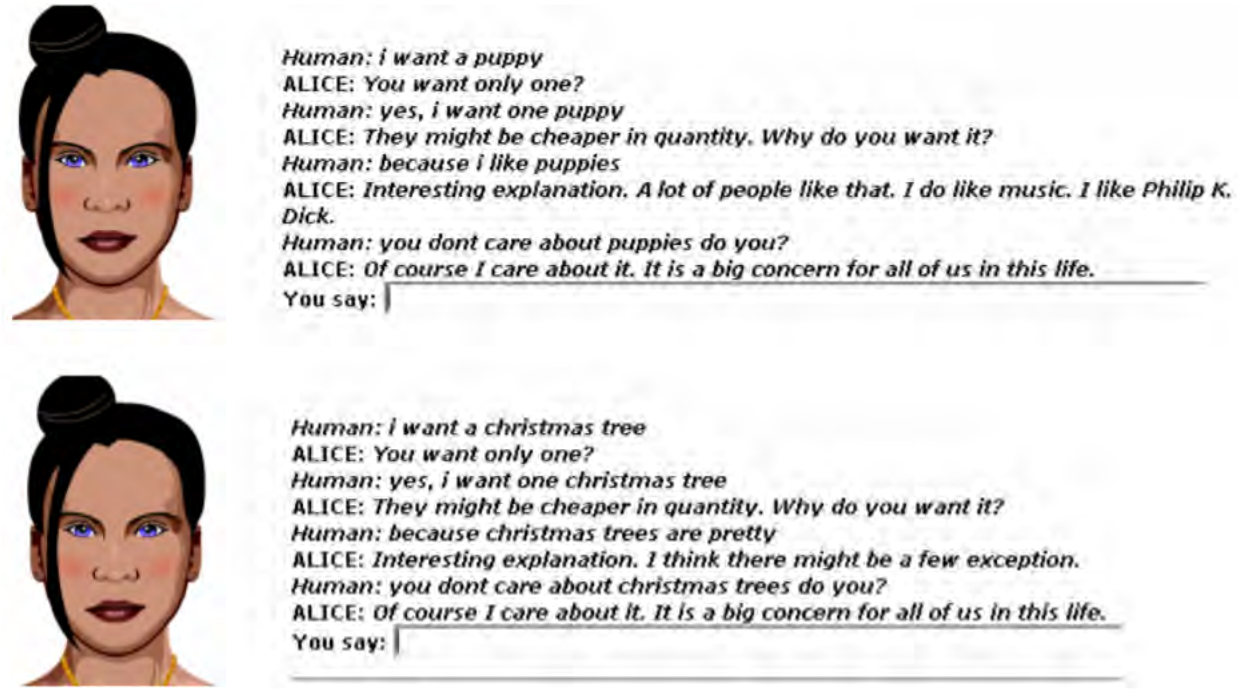


Person: Pick up a big red block.
Computer: OK.
Person: Grasp the pyramid.
Computer: I don't understand which pyramid you mean.
Person (changing their mind): Find a block which is taller than the one you are holding and put it into the box.
Computer: By "it", I assume you mean the block which is taller than the one I am holding.
Computer: OK.
Person: What does the box contain?
Computer: The blue pyramid and the blue block.
Person: What is the pyramid supported by?
Computer: The box.
Person: How many blocks are not in the box?
Computer: Four of them.

FIGURE 2.4: This is a representation of the simulated world of SHRDLU, accompanied by a conversation in which the blocks are moved [82].

One of the first AOA's was **PEGASUS** [89] implemented in 1994, on which users could search and book flights. PEGASUS was speech-based and worked for booking flights with American Airlines. The system was tested by letting ten users book a return flight of these ten seven were successful. It did take these seven users nearly twenty-five queries and an average of thirteen minutes to book their round-trips.

The clear inadequacies that chatterbots like ELIZA and PARRY posses are predictable and redundant responses and a lack of personality. Also, these chatterbots usually do not memorize previous responses which can lead to having the same conversation over and over. In 1995 chatterbot **ALICE** [78] (Artificial Linguistic Internet Computer Entity) was implemented, but even its complex pattern matching structure provided very limited types of responses, often leading to uninteresting conversations. An example of ALICE's limited response vocabulary can be found in figure 2.5. ALICE was based on AIML (Artificial Intelligent Mark-up Language), a version of XML which makes it very easy to write your own chatbot without any programming knowledge. The aim for this project is that, if a lot of people create a



Human: *i want a puppy*
ALICE: *You want only one?*
Human: *yes, i want one puppy*
ALICE: *They might be cheaper in quantity. Why do you want it?*
Human: *because i like puppies*
ALICE: *Interesting explanation. A lot of people like that. I do like music. I like Philip K. Dick.*
Human: *you dont care about puppies do you?*
ALICE: *Of course I care about it. It is a big concern for all of us in this life.*
You say:

Human: *i want a christmas tree*
ALICE: *You want only one?*
Human: *yes, i want one christmas tree*
ALICE: *They might be cheaper in quantity. Why do you want it?*
Human: *because christmas trees are pretty*
ALICE: *Interesting explanation. I think there might be a few exception.*
Human: *you dont care about christmas trees do you?*
ALICE: *Of course I care about it. It is a big concern for all of us in this life.*
You say:

FIGURE 2.5: An example of chatting with ALICE, a complex pattern matching chatbot.

chatbot with each their own knowledge base, ALICE could use all these knowledge bases together and become some sort of all knowing bot.

As is clear, pattern matching techniques did not seem to lead to breakthroughs in AI. A new direction of research was entered with the introduction of **MegaHAL** [34] in 1998. MegaHAL was the first chatbot based on machine learning techniques. It has a language model that consists of two Markov models. The first model predicts a word that will follow a given string and the second model predicts a word that precedes a given string. A user's input is tokenized and stop-words (frequently used words with little meaning) are removed leaving only keywords. Each of these keywords is used as a starting point for generating a response with the forward and the backward sentence creating model. This leads to the creation of a lot of candidate replies, which are ranked based on the amount of information they contain. The reply with the highest information content is returned to the user. The definition of 'highest information content' can be filled in in different ways. In MegaHAL it was implemented in a way that would rank the most surprising reply first to create an original conversation.

Around the end of the 90s bots were also developing in other ways than their back-end techniques. **SmarterChild** [58] was the first chatbot distributed over instant messaging and SMS networks. The bot started as a text-based adventure game, but grew into providing features like instant news, weather and stock price access and hosting tools like calculators and personal assistants. SmarterChild can be seen as a precursor to Apple's Siri.

2.2.2 Question-answering systems

Question-answering systems provide responses to natural language questions by extracting the answers from sources (like text documents, knowledge bases or web pages) or by giving explanations if no answer was found. The goal in QA systems is to find a specific answer matching the question [43] and not to find a list of web pages or text documents containing the answer (like in search engines). Question-answering systems come in different types and can be classified in many different ways. They can be grouped [54] based on question type, response type, analysis technique, structure of the information source, question domain and evaluation method as can be seen in figure 2.6. In this study we look at closed-domain QA systems with a variety of question and response types, a variety of analysis techniques and semi-structured data (FAQ files).

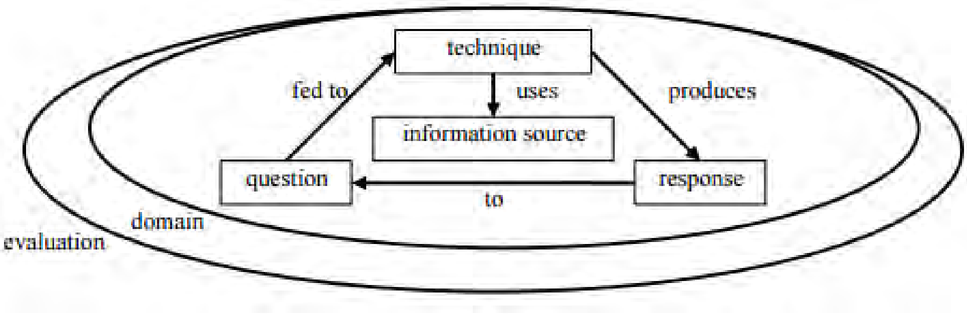


FIGURE 2.6: The general framework behind question-answering systems [84].

The earliest QA systems like LUNAR and BASEBALL differed from SHRDLU because of SHRDLU's ability to have a conversation. Modern QA systems more and more tend towards conversational interfaces but not by actual understanding of the questions. Due to the difficulties in natural language understanding in the end of the 1970s, QA systems were built using an information retrieval approach based on keyword matching. This process doesn't rely on deep natural language processing, as it mostly looks at keyword frequencies. A statistical method is used to process the keywords in a question and rank them by relevance. This ranking is than used to search through documents, of which the one with the best matching keywords is said to give the best answer. Later, many studies have added different natural language processing techniques to this framework and found that this improved the answers given by the QA system [13]. Therefore most QA systems nowadays are based on a combination of shallow NLP and IR techniques. Research in the field of question-answering systems has been mostly driven by the Text Retrieval Conference (TREC) QA track [77].

In this section we give an overview of important and renewing QA systems. A timeline with the QA systems described is shown in figure 2.7.

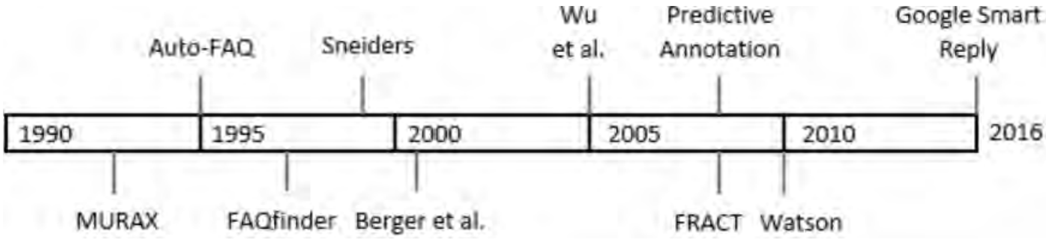


FIGURE 2.7: A timeline with the QA systems described in this section.

The techniques used in these QA systems and the main improvement of this system compared to its predecessors can be found in table 2.2.

QA system	Improvement	Main technique	NLP techniques
MURAX	First QA system based on NLP and IR	IR: lexico-syntactic pattern matching	Part-of-Speech tagging
Auto-FAQ	Maintains its own FAQ set	Pattern matching	Shallow NLP techniques
FAQfinder	First to use semantic knowledge using WORDNET	A scoring system based on NLP techniques and kNN	Tokenization, stemming, stop-word removal, TF-IDF, WORDNET and question-type classification
Sneiders	Prioritized keyword matching	Pattern-matching	Prioritized keyword matching
Berger et al.	New statistical approach to answer finding	Statistical translation models and latent variable models	Adaptive TF-IDF and automatic query expansion
Wu et al.	QA in a specialized domain	Probabilistic mixture model and k-means clustering	LSA
FRACT	Solves lexical disagreement problems	FAQ Retrieval and Clustering Technique	LSA
Predictive Annotation	New QA technique	Annotation	NER and bag-of-words
Watson	Open-domain QA based on enormous knowledge base	Machine learning and information retrieval	Parsing, question-type classification, NER, semantic labelling, lAT, relation detection and more
Google Smart Reply	Retrieval-based after generating responses	Neural networks	Word embedding

TABLE 2.2: A schema depicting the improvements of and techniques used in the QA systems described in this section.

MURAX (1993) [40] was one of the first QA systems based on NLP and IR techniques. MURAX can answer general-knowledge questions as it uses an encyclopedia as knowledge base. A restriction of the system is that it can only answer closed-class questions, meaning that it can only answer questions of which the answer is easily expressible in a noun phrase, eg. Trivial Pursuit questions. The NLP technique used in this system is part-of-speech tagging, of which an explanation can be found in section 2.3. Tagged words in the input question and the encyclopedia are matched to find document matches. Answers are then extracted from these documents by finding all noun phrases (as tagged by the Part-of-Speech tagger) in the document. Finally, lexico-syntactic pattern matching is used to check if the question type matches the answer type, eg. the question "Who is the President of the USA?" needs an answer type "name of person".

Auto-FAQ (1995) [81] is a QA system based on a set of news FAQs. This system maintains it own FAQ set by publishing to a list if a gap in knowledge is encountered. This list can be searched and the knowledge gaps can be filled by volunteers with expertise about this gap. In Auto-FAQ the input question is matched to the FAQ based on keyword comparison aided by shallow language processing techniques. Unfortunately, the NLP and IR techniques used are not published in the paper.

FAQfinder (1997) [12] is a QA system based on information retrieval that does not just use statistical metrics of similarity between question and response, but also uses a semantic knowledge base. This semantic knowledge base, or wordnet, improves FAQfinder's question-answer matching ability. The system's knowledge base consists of files with frequently asked questions and their answers. FAQfinder works in two stages. First the search for an answer is narrowed to a small set of QA pairs out of the whole database. Next, each retrieved QA pair is compared with the input question to find the best matching answer. In the first step an already existing technology called SMART [11] is used. This system stems and removes stopwords from the input question and then forms a term vector. This vector is matched against similar vectors created for the underlying knowledge base. The QA pairs ranked highest in this procedure are further processed in the second step. In the second step, the retrieved questions are matched against the input question and scores are given for their similarity. Three types of scores are used:

1. A statistical similarity score t
   Here the term vectors of the input question and a QA pair are given a score based on TF-IDF, an NLP technique explained in section 2.3. The term vectors are later compared using the cosine of the angle between the vector of the input question and the vector of the QA pair. This score takes into account the frequency of terms occurring in the input question and the knowledge base and therefore judges the overall similarity between input question and the QA pairs, but it does not look at an interpretation of the input question in any way.

2. A semantic similarity score s
   This score is designed to look at a high-level interpretation of the input question. In natural language the same question can be expressed in many different ways. For example, a question with the word "spouse" and a question with the word "husband" might have the same meaning, but this is not included in the statistical similarity score. To include this score the WORDNET [53] database is used in combination with a marker-passing algorithm [62]. WORDNET is a semantic network of English words, which provides relations between words and synonym sets.

3. A coverage score c
   This score makes sure that important concepts in the input question are also present in the question it is matched to. The coverage score is the percent of keywords in the input question that are also present in the question from the database. Here semantics are also used as synonyms of the keywords are also taken into account.

To calculate the overall similarity between the input question and the QA pairs a weighted average of the three scores is used

$$\text{Total score} = \frac{tT + sS + cC}{T + S + C}, \tag{2.1}$$

where *T*, *S* and *C* are constants with which one can adjust the importance of each of the scores for the overall outcome.

FAQfinder was improved a few years later by adding a score based on question type [48]. Input and database questions were classified using a distance-weighted k-nearest neighbours (KNN) algorithm. Twelve different question types were used, like definition, location, reason, manner and yes-no question. The final score on this part was calculated using a similarity matrix defined for the question types.

Other QA systems also successfully used question type classification to extract appropriate answers [31][33]. Question type classification was further researched by Tomuro [74]. He found that using only lexical features, like using keywords, worked best in question classification. Adding semantic features, which give an understanding of the question, caused more harm than good.

Sneiders' QA system (1999) [70] follows the approach of Auto-FAQ in maintaining its own FAQ set. New in this system is the use of prioritized keyword matching, a technique developed by Sneiders himself. The main idea in prioritized keyword matching is that there are three types of words in a sentence:

1. Required keywords
   The words that are essential to understand the meaning of the sentence.

2. Optional keywords
   Words that help in communicating the meaning of the sentence, but are not strictly necessary. They could be omitted without the sentence losing its meaning.

3. Irrelevant words
   These are words that are very common in normal language or in the specific domain of questions. For example, words like "the", "is" or "can".

The words in an input question are organized into these categories and matched to FAQ in the database containing the same required and optional keywords. The words in the FAQs in the database can also have the label 'forbidden keywords'. Forbidden keywords are use to distinguish between sentences with a different meaning but having the same required keywords, eg. "Why do we pay tax?" and "How do we pay tax?". If a question from the database has a forbidden keyword, then it will not be matched with an input question containing this word.

Berger et al. (2000) [8] used a statistical approach to finding answers in a large collection of answered questions. They use techniques inspired by machine learning to 'learn' their system to locate answers and find that this works better than using the standard TF-IDF algorithm for answer-finding. The techniques used are adaptive TF-IDF, automatic query expansion, statistical translation models and latent variable models. The first two techniques are explained in section 2.3. Unfortunately the paper does not state which statistical translation and latent variable models were used. All of these techniques are found to enhance the performance of a question-answering system based solely on TF-IDF.

Wu et al. (2005) [86] researched which independent techniques are useful for question-answering in the medical domain. Medicine, like the financial domain, is a specialized domain but spans quite a broad array of questions. Also, medical and financial questions both use a lot of domain jargon. The paper presents a probabilistic mixture model. The mixture model is used to represent subgroups in the QA database without requiring an identification of which questions are in which subgroup. To be able to give a very precise answer to a new question the answers in the database were divided into multiple sections. These sections are then clustered with Latent Semantic Analysis (LSA) and K-means clustering. Both of these techniques are explained in section 2.5. Under the probabilistic mixture model, retrieving the right QA pair can be seen as a maximum likelihood estimation problem. To find the optimal weights in the mixture model an expectation-maximization model is used. The model used in this paper outperforms the techniques used in FAQfinder for the medical domain.

FRACT (2008) [38] stands for FAQ Retrieval and Clustering Technique. This technique was developed to solve lexical-disagreement problems that can occur in keyword search. For example, the questions "When will I receive my pension?" and "At what time can I get retirement money?" have a similar meaning but no overlap in keywords, thus keyword based QA systems will not return the same answer for these questions. FRACT uses LSA techniques which have been shown to be able to bridge lexical gaps between words by mapping them into latent semantic space [20]. An overview of the architecture used in FRACT can be found in figure 2.8. FRACT collects users' question logs and clusters these logs into predefined categories to use them as knowledge sources. When an answer needs to be retrieved words associated with the clusters help in ranking the answers and overcoming the lexical-disagreement between input question and QA pair.

As can be seen in figure 2.8, FRACT consists of two parts: the query-log clustering system and the cluster-based retrieval system. The query-log clustering system consists of FAQs and question logs in latent semantic space. LSA is used to calculate the distance between the FAQs and question logs. With these distances the question logs are grouped around the FAQs. The answer retrieval system uses the query-log clusters to smooth vector representations of input questions and FAQs, to rank the smoothed input questions and FAQs, and to return a ranked list of relevant FAQs. FRACT outperformed traditional IR systems in experiments, but unfortunately it is difficult to implement FRACT in most systems, as these systems do not use question logs.

Prager et al. (2008) [61] from IBM invented a new question-answering technique called Predictive Annotation (PA). This technique scans text for potential answers to questions, annotates these potential answers and indexes them. During question analysis question words are replaced with QA-Tokens, essentially named entity identifiers. Also during question analysis, a simple scoring system is used to do bag-of-word matching of short paragraphs. PA is very effective at answering factual questions, but fails when any semantic understanding of the question is needed.

Watson (2010) [23] is a question-answering supercomputer famous for beating the best players on the quiz show Jeopardy!. Competing on Jeopardy! was actually just a means to an end for IBM, the real goal was to develop a good solution to the open-domain QA problem using techniques like natural language processing, information retrieval and machine learning. DeepQA is the core technology underlying Watson and this technology can be used in many real-life situations. IBM is now focusing on applying it in the medical and healthcare domain, using DeepQA to aid in differential diagnoses [3]. In figure 2.9 an overview of the inner workings of the DeepQA technology is given. For Watson an enormous knowledge base containing publicly available data from Wikipedia, encyclopaedias, the Bible and many other sources was created before the show. When a question enters the DeepQA pipeline it is first used as a search query (like in Google Search) to search the underlying database. Only the best results are kept. Then, the question and these search results combined are used to retrieve supporting evidence for each search result from the database. Each search result is now an answer hypothesis and is evaluated (in many

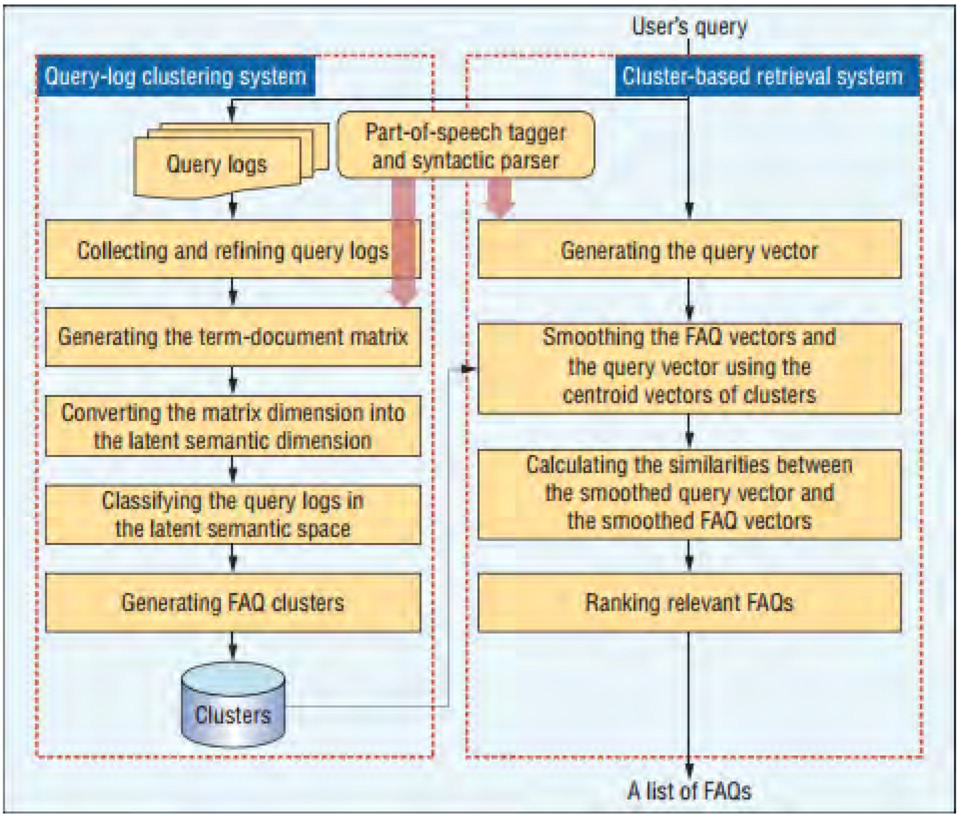

User's query
Query-log clustering system
Cluster-based retrieval system
Query logs
Part-of-speech tagger and syntactic parser
Collecting and refining query logs
Generating the term-document matrix
Converting the matrix dimension into the latent semantic dimension
Classifying the query logs in the latent semantic space
Generating FAQ clusters
Clusters
Generating the query vector
Smoothing the FAQ vectors and the query vector using the centroid vectors of clusters
Calculating the similarities between the smoothed query vector and the smoothed FAQ vectors
Ranking relevant FAQs
A list of FAQs

FIGURE 2.8: An overview of the FRACT architecture [38].

dimensions) on the evidence found in the database. The answers are ranked using multiple machine learning algorithms and the highest ranked answer is outputted if it reaches the minimum confidence level.

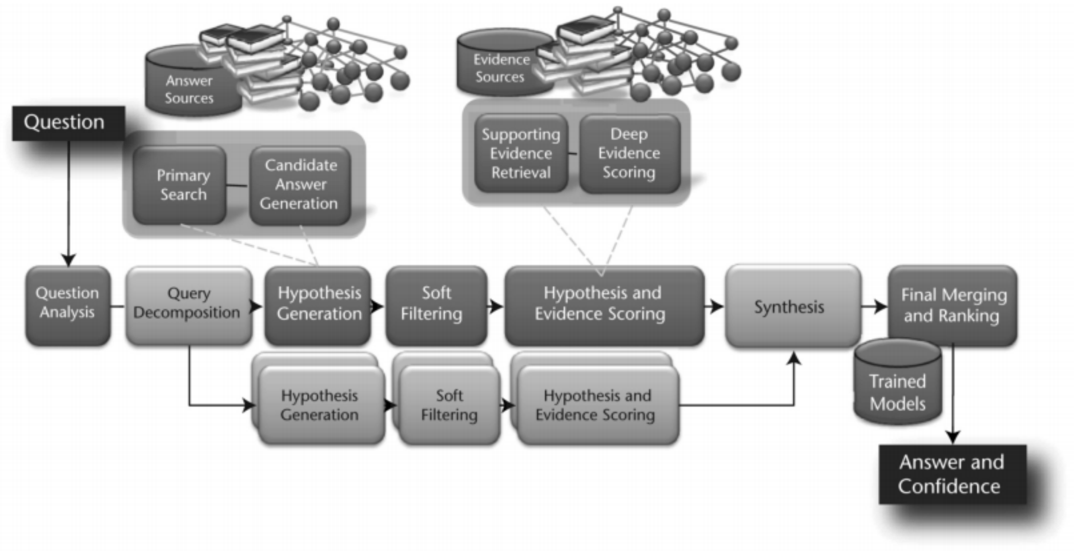


FIGURE 2.9: An overview of the DeepQA architecture [23].

We will now go deeper into the methods underlying the steps of the DeepQA technology.

1. Question analysis
   The first step in figure 2.9 is question analysis. Here all components of the question are analysed to come to an understanding of what is actually asked. This is done by using many different technologies like shallow and deep parses, question classification, named entities, semantic labels, LAT and relation detection and many others. These techniques are explained in detail in section 2.3.

2. Query decomposition
   Query decomposition is needed for questions that include sub questions or are made up out of multiple clues. To determine if and how a question should be decomposed DeepQA uses parsing and statistical classification techniques. If a question is decomposed the fragments run through the whole DeepQA system and hypotheses to answer the question fragments are found. In the synthesis step, later in the process, the answer hypotheses to the fragments are combined to form answer hypotheses for the whole question.

3. Hypothesis generation
   After the question analysis and decomposition, the system's data is searched and answer-sized pieces are extracted from these search results to form candidate answers in the hypothesis generation step. In the primary search the database is searched using multiple search techniques like using text search engines, document and passage search and searching based on named entities in the question. Then, in the candidate answer generation step hypotheses are subtracted from the search results.

4. Soft filtering
   Soft filtering is used to make the large set of answer hypotheses smaller without using intensive time-spending techniques.

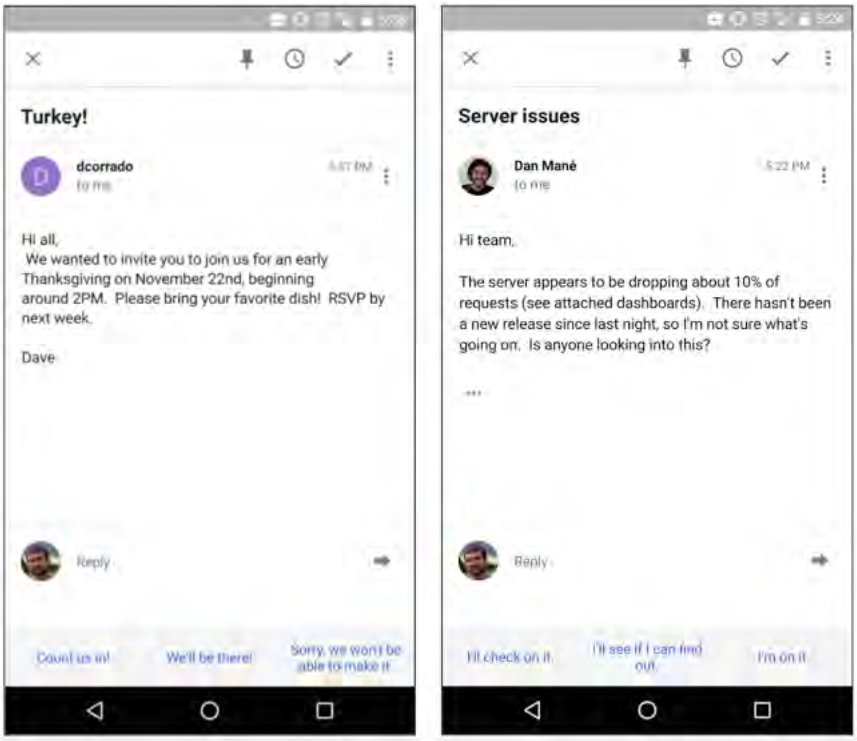
5. Hypothesis and evidence scoring
   Hypotheses that pass the soft filtering step are now subjected to a harder evaluation in which evidence for each hypothesis is searched in the database. In the evidence retrieval step many techniques are used. One of the most important techniques is passage search; the query derived from the question and the hypothesis are combined in a search, retrieving the hypothesis in the context of the question as evidence. In the deep evidence scoring step a certainty level for how well the found evidence supports the answer hypothesis is created. Many different score systems are used in this process which look at measures like the hypothesis' correlation with the terms in the question, the degree of the match between the question and the found property and hypothesis in the evidence (e.g. IDF-weighted terms in common or the length of longest similar character sequence as explained in section 2.3) and source reliability.

6. Synthesis
   In this step the answer hypotheses of questions that were decomposed in the query decomposition step are combined to form an answer hypothesis to the whole question.

7. Final merging and ranking
   Here all hypotheses and their scores on many dimensions are evaluated to

find the best hypothesis and its confidence. In the set of hypotheses there can be many hypotheses which may be written differently but are actually the same. This is very hard to process for ranking algorithms, therefore in DeepQA equivalent hypotheses are merged using an ensemble of normalization, matching and co-reference resolution algorithms. Finally, the hypotheses are ranked and their confidence is calculated using a machine learning approach. The machine learning models are trained by running the entire DeepQA program on existing question-answer sets. For the development of Watson the DeepQA team experimented with many different machine learning techniques but found that regularized logistic regression consistently gave the best results [27]. Logistic regression is explained in section 2.5.

Google Smart Reply (2016) [37] is a feature in Google's Inbox app that suggests short replies to incoming e-mails. Examples of the suggestions of Smart Reply can be found in figure 2.10. The idea is to make answering e-mails on smart phones easier and faster. The system is built up out of two Recurrent Neural Networks (RNNs) also called sequence-to-sequence learning which is explained in section 2.5. The first RNN is for encoding an incoming e-mail, and the second to generate the responses. The first network processes the text in the e-mail word-for-word to create a thought vector. This vector is used by the second RNN to generate a reply word-for-word. The e-mails in figure 2.10 are quite short, but in reality people also receive a lot of longer e-mails which are harder to process. To also be able to process longer e-mails Google has used a Long-Term-Short-Term-Memory (LSTM) network, a specific type of RNN. An LSTM network can grasp the context of the e-mail without getting lost by off-topic sentences.



Turkey!

Hi all,
We wanted to invite you to join us for an early Thanksgiving on November 22nd, beginning around 2PM. Please bring your favorite dish! RSVP by next week.

Dave

Server issues

Hi team,

The server appears to be dropping about 10% of requests (see attached dashboards). There hasn't been a new release since last night, so I'm not sure what's going on. Is anyone looking into this?

...

FIGURE 2.10: Two examples of e-mail suggestions given by Google Smart Reply [18].

Smart Reply gives a user three reply options and to maximize the likelihood of one of these replies to be chosen it is best if the replies are different in meaning. If someone asks for a meeting next week getting the three options: "Sure, let's meet!","Yes, I can." and "See you next week!" is not much of a choice. A diversity like "Sure, let's meet.","When do you want to meet next week?" and "No, I'm sorry next week I'm too busy." is much more practical. To achieve this diversity of options Google researched the semantic similarity of the generated options. This means the similarity in the underlying meaning of the replies, not just the similarity in the words the reply is made from. Another challenge Google encountered was an early version of the system replying with "I love you" to random messages. This was not a mistake in the model, as responses like "Thanks" and "I love you" are very common and turned out to be a big part of the training set. This problem was solved by normalizing the likeliness of a reply by a measure of the replies' prior probability. This normalization made sure that the predicted replies are not just very likely to be responses, but are also highly relevant to the incoming message [18].

A challenge Google had, and we will definitely also have, is that they have to ensure that the response quality is high. Google solved this by only selecting replies from a reply space generated offline by a semi-supervised graph learning approach. To generate this response space they first converted answers that have more than one possible representation into a standard form. Like converting "Thank you!", "Thanks!" and "Thank you very much" all to "Thank you." In the next step they performed semantic intent clustering on all response messages, meaning they formed clusters of responses with different meanings like a thanking-cluster, a funny-cluster and a sorry-cluster. This was done by using scalable graph algorithms in a semi-supervised setting. For each cluster some examples replies were seen as seeds and a base graph with frequent replies was constructed. In this graph replies with same features (like keywords or n-grams) were connected by lines. Finally, all replies are given a semantic intent label by an algorithm that was trained on some manually added labels. To ensure the quality of the labelling the semantic labels of a part of the response dataset were checked manually.

2.2.3 Generative-based chatbots

Generative-based chatbots generate their answer word-by-word or sentence-by- sentence. These systems are in the field of Statistical Language Modelling (SLM). In SLM the probability distribution of lexical entities, like words or sentences, is estimated. One task in this field is assigning a probability of a certain word appearing next, based on its relation with the previous words in the sentence. One of the earliest techniques in this field was using N-gram models (a technique explained in section 2.3) to model the probability of a certain word following already given words [15][10][64]. Later, more advanced solutions based on decision trees [60] and maximum entropy models [65][79] were used. These machine learning models could be trained on various features (like Part-of-Speech (PoS) tags [32] or other semantic knowledge), instead of just keywords.

Recent research in SLM is focused on the use of artificial neural networks (a technique explained in section 2.5). This started with the work of Bengio et al. [7], who recognized and proposed to fight the curse of dimensionality. With this curse, they meant that language is so broad that word sequences on which the model is applied will likely be different from the word sequences used in training. Their solution was to learn a distributed representation for words in which each training sequence learns the model about many semantically neighbouring sequences. Word embeddings are used as input for the neural network. With a feed-forward neural network

(FNN) this model simultaneously learns a distributed representation for each word, and the probability function for word sequences. Thus this model learns itself which features are appropriate, whereas using decision trees or other models require features to be manually selected and created before training, a big advantage of using neural networks. The FNN approach showed a significant improvement over state-of-the-art N-gram models in 2003.

Neural networks work better than N-grams for sequences never encountered in the training data, because they can project the vocabulary into hidden layers, creating clusters of semantically similar words. Vector representations of words or sentences can hold important linguistic information. For example, Mikolov et al. [50] show that vector representations can be compositional, meaning that given the vector *Queen*, if you subtract the vector *Woman* and add the vector *Man*, you can get a vector very similar to the one for *King*.

FNNs as used in the study by Bengio et al. [7] have a big drawback in modelling language. These networks can only take a fixed number of previous words into account when predicting the next word in the sequence. This problem can be solved by using a recurrent neural network (RNN), a network that can handle big contexts and is explained in section 2.5.

A widely used RNN framework called sequence-to-sequence learning (explained in section 2.5) is first proposed by Sutskever et al. [73]. This network is also applied in [69][68][71], in Google Smart Reply [37] and in *A neural conversational model* [76], where the technique is used to model technical support chats.

RNNs can learn vector representations for words and remember a huge context, but they also have some disadvantages to which different solutions have been researched. One disadvantage of RNNs is that models tends to generate meaningless and trivial sentences like "I don't know" and "Okay" as found in [71], [37] and [68]. Li et al. [42] successfully implemented Maximum Mutual Information (MMI) as the objective function of neural networks to overcome these trivial sentences. Another limitation of RNNs is that the model will start computing very large vocabularies when trained on full conversations. A solution to this is to create a vocabulary with a maximum amount of words and replace other words with 'unknown'. This unknown word can later be translated by using a separate dictionary as in [47][35]. RNNs are also limited in their understanding of copying, knowing in which cases words from the input sequence need to be copied to the output sequence. To overcome this limitation CopyNet [30] has a mechanism that can choose sub-sequences from the input sequence and place these a the right places in the output. Other proposed modifications to RNNs are adding a third network to model the intention of words [88] and adding contextual features, like topics, to the model [25].

Generative-based models require a very large training set to give a coherent outcome, due to the immense amount of possible words and sentences in human language. Often when training models for a specific domain not enough training data in this domain is available, and therefore other data is also used. Such a domain mismatch can be solved by first training the neural network on all training data, and then running a few iterations on only the domain-specific training data as shown by Luong and Manning [46].

2.2.4 Chatbot evaluation methodologies

Whereas automatically evaluating models with one correct outcome is easy, evaluating models which can give numerous different but all correct outcomes is very difficult. Evaluation of the response of a chatbot can in practice be applied in two ways: supervised and unsupervised. First, supervised evaluation of the chatbots' answers can be done manually by human experts giving scores to each answer. It is also possible to ask users of the chatbot system to give a user satisfaction score at the end of their conversation. Both these methods of supervised evaluation are user intensive.

Unsupervised chatbot evaluation metrics can automatically evaluate the quality of the chatbots' responses, but the quality of these metrics is often poor. Popular metrics for evaluating dialogue systems are BLUE [57] and METEOR [5], and adaptations of these techniques. An assumption of these metrics is that right answers have a big word overlap with the assumed correct answer to a question. In chatbot systems this is a very strong, possibly too strong, assumption. An unsupervised evaluation metric based on another technique than statistical word-overlap is Word2Vec [51] in which word embedding is used.

In 2016 Liu et al. [44] researched the correlation between scores given to chatbot replies by these unsupervised evaluation metrics and by humans. They found that all unsupervised metrics show weak or no correlation with human scoring. On a Twitter dataset, a small positive correlation between the unsupervised metrics and human judgement was found, but on the big and technical Ubuntu Dialogue Corpus no correlation could be found at all. This shows that unsupervised metrics are still far away from replacing human judgement in evaluating dialogue systems.

2.3 Natural Language Processing

Natural Language Processing (NLP) is a set of techniques that enables computers to analyse and have an (shallow) understanding of human language. Unlike common word processors, NLP takes the hierarchical structure of a language into account (like letters forming words and words forming a sentence). NLP is seen as a hard problem in computer science, because of the ambiguity found in languages. When a computer needs to understand a language it needs to not only know the meaning of the words, but also the concepts and how these concepts are linked together to create the meaning of a text.

We divided the process of understanding natural language into three stages: morphological analysis, syntactic analysis and semantic analysis. These three stages and the corresponding NLP techniques are discussed here.

2.3.1 Morphological analysis

Morphological analysis focuses on studying the elements within words. In this first stage of NLP, individual words are extracted from text, ignoring non-word tokens like punctuation. The words are then analysed down to their components. Important terms in morphological analysis are:

- **Tokenization**
  Tokenization is the process in which a text is broken down into symbols, words, phrases, or other text elements called tokens. The list of processed tokens becomes input to further processing steps. These further steps need a tokenizer to remove or transform text components like punctuation, abbreviations and white space. Tokenization is mainly used to identify meaningful keywords in a text.

- **Stop-word removal**
  Stop-words are the most common words in a language, like 'a', 'the' and 'are'.

These words occur very frequently and cannot be used to separate the topic of one text from another. If these words are not useful in the classification of certain texts (this can be different per domain), they can be removed. Because of the inconsistency of words being meaningful or meaningless in certain domains, it is difficult to create standard stop-word lists. The process of stop-word removal is needed because it leads to a smaller data size and it improves the performance of text classification.

- **Stemming**
  Stemming is the process of reducing conjugated words back to their original word stem. For example, the conjugated words "walking", "walk" and "walked" can all be stemmed to the word stem "walk". This technique is used to make matching of text documents with the same content easier. There are four common types of stemming algorithms [4]:

  1. Table look-up stemmers
     A table look-up stemmer is based on a table that stores as many conjugated words and their stems as possible. A query is used to look-up a conjugated word in this table. Look-ups are very fast, but disadvantages of this method are the storage capacity needed and the lack of complete conjugated word tables in most languages and domains.
  2. Successor variety stemmers
     Successor variety stemmers segment words into their stems by looking at successor letter variety counts. For example, if we have the word-set: "walk","world","warm" and "like" and we look at the successor letter variety counts of the word "warm" we start with the first letter w, in the word-set this letter is followed by an a and an o, thus the successor letter variety count is two. If we then look at the variety count of "wa", we see that this combination is followed by an l and an r, thus this successor letter variety count is also two. The combination "war" is only followed by an m, and thus from now on the successor letter variety count is only one. This way we can create successor letter variety counts for every word in the text. These counts are then used to segment the words into the correct stems.
  3. N-gram stemmers
     N-gram stemmers divide words into parts with N consecutive letters. These parts are then compared between the words in a document. For example, the words "walking" and "walker" can be compared by creating bi-grams like: walking to wa al lk ki in ng and walker to wa al lk ke er. The word walking has six unique bi-grams and the word walker has five unique bi-grams. The words share three bi-grams. Now a similarity measure is used to compare these two words, called the dice coefficient S.

     $$S = \frac{2C}{A+B} \tag{2.2}$$

     Here A and B are the number of unique bi-grams in the first or second word, respectively. C is the number of common bi-grams. In our example, the dice coefficient is $\frac{6}{11}$. A matrix of all dice coefficients in a text is called a similarity matrix. The chosen cut-off for the dice coefficient determines if words are stemmed to the same base.

4. Affix removal stemmers
   Affix removal stemmers focus on removing pairs of letters from the end of words. For example, the stemmer could always remove "ing" from the end of a word, so "walking" becomes "walk", but also "aging" becomes "ag". Clearly, this is not a very precise method.

All types of stemming algorithms can make mistakes, definitely in hard to stem languages. These errors can be divided into two types: over-stemming and under-stemming. Over-stemming means that two words that stem from different roots are stemmed to the same word base. Under-stemming means that two words that stem from the same root are not stemmed to the same word base.

- **Lemmatization**
  Lemmatization is also a process of reducing conjugated words back to their original word stem. However, where stemming often crudely chops off the end of a word to get to the word stem, lemmatization does this more properly using a vocabulary and aiming to transform the word to its base or dictionary form, called the lemma. For example, if a stemmer encounters the word 'wound', the past form of 'to wind' it will probably leave this word as is, where lemmatization will either transform this word to 'wind', its base form, or, if it is a noun, it will also leave the word as 'wound' [49].

- **Automatic query expansion**
  Query expansion is the process of reformulating a query to encourage matching between texts. This involves evaluating the input query and expanding this query to match additional texts. Some techniques used to expand queries are: using synonyms of words, stemming all words and fixing spelling errors [87].

- **Part-of-Speech (PoS) tagging**
  PoS tagging is annotating each word in a sentence with its part-of-speech. For example, a word can be tagged as a noun, verb, pronoun or article. Assigning such a part-of-speech to words in a sentence can be done in two ways: with rule-based approaches [9] and with statistical approaches. Statistical approaches often make use of Markov models, which capture the linguistic and contextual information. Parameters of this model can then be estimated from (un)tagged text [19]. The output of PoS tagging is very useful in sentence parsing and for use in word sense disambiguation in NLP stages two and three, respectively.

2.3.2 Syntactic analysis

Syntactic analyses focuses on studying the elements within sentences. In this second stage of NLP, sequences of words are transformed into structures that show how the words in the sequence are related to each other. Important terms in syntactic analysis are:

- **Parsing**
  Sentence parsing is the process of converting a sequence of words into the structure of a formal grammar. In computational linguistics parsing usually outputs a parse tree showing the syntactic relation between the words in the

input sentence. Shallow parsing consists of building only partial trees for one sentence, whereas deep parsing builds complete trees [1].

- **Bag-of-words**
  A bag-of-words model is a very simple representation of a text. Here a text is portrayed as the set of its words, without any relations between words and also disregarding grammar. Bag-of-words only represents the words in the text and their multiplicity. This model is often used for document classification.

- **N-grams**
  Whereas bag-of-words is an orderless representation of a text, N-gram models are used to store this spatial information. N-grams were briefly mentioned in section 2.3.1 where this technique was used for stemming. Using N-gram models the probability of a certain word following a sequence of words is estimated. The N in N-gram stands for the amount of words considered. For example, a bi-gram (N=2) transforms the sentence "When do I get my pension?" into "When do","do I","I get","get my" and "my pension". The estimation of the probability of a word following another word, results from the frequency of word co-occurrences. This means that the probability that a given word appears next is equal to the number of times that this word follows the studied sequence in the training data divided by the number of times that the studied sequence is present in the training data (with and without the new word following). In practice, not every word combination can be present in the training data. This problem can be solved by smoothing the probability distribution for a word appearing next [14].

- **Term Frequency-Inverse Document Frequency (TF-IDF)**
  TF-IDF is a statistic that reflects how important a certain word is to a text in a collection of texts. This statistic was defined by Salton and McGill [67] in 1983, and is now one of the most used term-weighting metrics. The TF-IDF statistic increases when a word appears more often in a text, but this is offset by the amount of times the word appears in all texts. If t is the term frequency in the input, d is the number of text documents the term appears in and D is the total number of text documents, then

  $$\text{TF-IDF} = t \log \frac{D}{d}. \tag{2.3}$$

  With this definition, a term that appears in every document and is thus of little value for separating the documents, will have an almost zero IDF ($\log \frac{D}{d}$) value, leading to a small TF-IDF value. However, a term that only appears in one document will have a very high IDF and thus also a higher TF-IDF.

  Berger et al. [8] used an adapted version of TF-IDF in their question-answering system. This adaptive TF-IDF algorithm had adjusted IDF-weights of each word, so it could be used to maximize the probability of retrieving a correct answer for every question. The IDF-weights were adjusted by applying gradient descent to bring questions and answers closer together.

2.3.3 Semantic analysis

Semantic analyses focuses on assigning meaning to words, sentences and texts. In this third stage of NLP, structures are created to represent the meaning of words and combinations of words. A lot of research is going on in this stage as there are still no optimal solutions for automatically deriving meaning from text. Important terms in semantic analysis are:

- **Named Entity Recognition (NER)**
  NER maps words in the text to proper names, like people or places, and recognizes the type of each proper name, like a person, organization or location. NER is still being researched a lot, as better approaches are needed. The existing approaches are list look-up, triggering and pattern matching. In list look-up the system only recognizes the entities stored in a list . This system is fast and easy to adapt to another text, but there are high costs for collecting and maintaining the entities and it gives problems handling name variants. The triggering approach uses structures that are often embedded in names. For example, "Mount Everest" can be recognized through the rule "Mount + Capitalized Word". Lastly, in the pattern matching approach, the whole pattern is manually constructed. For example, the pattern "<Name> stays in <Location> on <Date>", will find the named entities in the sentence "Maria stays in New York on May 3" [2].

- **Word sense disambiguation**
  Word sense disambiguation is used to determine which meaning of a word is used in a text. A lot of words have more than one meaning, and differentiating between them is difficult. Methods used in word sense disambiguation include empirical methods, knowledge-based methods and AI methods [72].

- **Latent Semantic Analysis (LSA)**
  LSA is a technique in NLP that assumes that words that are alike in meaning will occur in similar texts. In LSA relationships between documents sets and the terms they contain are analyzed by producing a concept set related to the terms and documents. A word-document matrix is constructed which contains word counts, here the unique words are in the rows and the text documents in the columns. Singular value decomposition is used to reduce the amount of unique words, while maintaining the similarities between the text documents. The words that remain are compared by taking the cosine of the angle between two word count vectors. A cosine value close to one means that words are very similar, while a cosine value close to zero means that words are very different [41].

- **Semantic (role) labelling**
  Semantic labelling detects the semantic roles of words in a text. For example, in the sentence "Maria stays in New York", semantic labelling would recognize "Maria" as the one who stays, the verb "to stay in" and "New York" as the place to stay in. Being able to recognize these relationships makes it easier to understand the meaning of the sentence [56].

- **Lexical Answer Type (LAT)**
  LATs are the terms in a question that indicate what kind of entity is begin asked for [23]. For example, in the question "Who is the president of the US?", the LAT is "Who" which indicates that the answer will be the name of a person.

- **Relation detection**
  Relation detection is used to find semantic and syntactic relationships between entities in a sentence [23]. For example, in the question "Who is the president of the US?" the relation is(president, ?x, the US) can be found.

2.4 Vector space model

The vector space model is a mathematical model used to represent text documents as vectors. Using this model vectors of text documents can be compared and thereby we can calculate the similarity of two documents. This model was first used in the SMART Information Retrieval System [66].

The vector space model leads to a document-term matrix by assigning values to the terms that appear in the text documents that are to be compared. There are several ways to do this term weighting, but the most popular one is TF-IDF weighting as explained in the previous section. The dimensionality of a vector is the number of words in the vocabulary (all words used in the text documents to be compared).

To compare text documents we can use vector operations. These operations are also called similarity measures. The similarity measures can lead to a ranking based on relevance when comparing a text document to a set of text documents. The most popular similarity measure is cosine similarity. This measure calculates the cosine between the vectors of two text documents using the following formula:

$$\cos\theta = \frac{d_1 \cdot d_2}{||d_1|| \cdot ||d_2||}. \tag{2.4}$$

Where $d_1 \cdot d_2$ is the intersection of the two document vectors and $||d_1|| \cdot ||d_2||$ is used to normalize this score by the length of the document vectors. This normalization is important as it ensures that the chance of matching is not higher for longer documents (containing more words). A higher cosine value means a higher similarity between two documents.

A limitation of the vector space model is that words in one document must exactly match words in another document. This limitation can be overcome by preprocessing all documents with NLP techniques and by using lists of synonyms.

2.5 Machine learning models

Machine learning is a field in computer science in which computers get the ability to learn without rules for learning being explicitly programmed. Machine learning algorithms make data driven predictions through a model of sample inputs and thereby overcome the need of following strict programming instructions.

Machine learning methods can be divided into categories in multiple ways. One way is by learning method, here a difference is made between supervised and unsupervised learning. In supervised learning the computer is presented with a training dataset with predictive variables and the output obtained by using these variables. The goal here is to learn a rule or pattern that matches the values of the predictive variables to the output. In unsupervised learning there is no output variable or label. There is a dataset with the values for a set of variables for a number of observations, leaving it to the algorithm to find structure in this data. Unsupervised learning can be used to find hidden patterns in data or to engineer a good set of features.

Another way to divide machine learning methods into categories is by looking at the type of output they produce. Classification methods give answers in two or more classes. This method is usually used as a supervised learning method that labels to which class a new instance belongs. Regression methods also solve supervised problems but give continuous outputs instead of categorical ones. Clustering methods divide a set of data into different groups without knowing anything about the groups beforehand, thus this an unsupervised learning method.

In this section we explain the machine learning techniques used for the chatbots in section 2.2 and new upcoming techniques suitable for question matching. There exist many more machine learning techniques, but we focus on the ones that can be used in question-answering systems.

2.5.1 K-means clustering

K-means clustering is used to divide n observations into k groups. The idea is that each observation belongs to the group with the nearest mean. The k-means algorithm works in five steps:

1. Choose k: the number of clusters
2. Initialize the centroids (the centers of the clusters)
3. Assign data points to the cluster with the nearest centroid
4. Update the position of the centroids based on the new clusters
5. Re-assign the data points to the cluster with the nearest centroid

Step 4 and 5 are repeated until the centroids stop moving and the data points stop moving from one cluster to another.

Initializing the centroids can be done by random choice, but it is better to use the "farthest" heuristic. Here the first centroid is chosen randomly, but the second centroid is the data point farthest away from the first centroid. The following centroids are also chosen to be as far away as possible from all preceding centroids [24].

2.5.2 k-Nearest Neighbours (kNN)

The kNN algorithm is a supervised learning method that can be used for classification and regression. In classification the k closest data points to a new point are used to determine to which class this new point belongs. A majority vote is used to classify the new point, meaning that it is assigned to the class that most of its k neighbours belong to. An example of classification with kNN can be found in figure 2.11. Here the idea is to classify the red star into class 1 (green) or class 2 (blue). If we look at the three (k=3) nearest neighbours of the red star, it would be classified into class 2 (blue), but if we look at the five (k=5) nearest neighbours the star would be assigned to class 1 (green). This shows how sensitive kNN is to the local structure of the data.

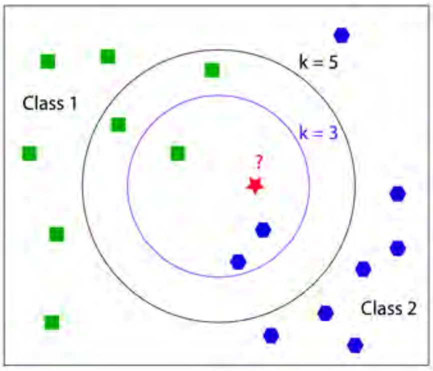


FIGURE 2.11: Example of classification with the kNN algorithm [59].

In regression with kNN the input again consists of the k closest data points to the new point, but the output is now a property value. This property value is the mean of the values of the k nearest neighbours of the new point.

For both classification and regression with kNN, the algorithm can be improved by taking the distance of the neighbours to the new data point into account. We can assign a higher weight to nearer neighbours and a lower weight to more distant ones. Often the approach of giving each neighbour a weight of $\frac{1}{d}$ is used, where d is the distance between the new point and the neighbour [24].

2.5.3 Random Forest

Random Forest is a machine learning model that is very versatile and can perform regression and classification tasks. It is an ensemble learning method, as it combines a group of weak models to form one bigger more powerful model. The models that are combined are decision trees, tree based structures that split the data on a certain condition at every node.

Using Random Forest we grow multiple decision trees, with each there own features and conditions on the nodes. The nodes in the trees are different because when selecting which feature is best to split on, only a randomly selected part of all features are given as options. To appoint a new incoming question pair to a class, each tree gives a classification (a vote) and the Random Forest model chooses the class with the highest number of votes over all trees.

2.5.4 XGBoost

XGBoost (eXtreme Gradient Boosting) is an upcoming gradient boosting algorithm, which is popular because it is ten times faster than current gradient boosting models.

Boosting uses a combination of different weak rules and combines them to a more powerful model by voting (like in Random Forest), by averaging or by weighted voting (one weak rule might be more important than others). An example of a weak rule would be "if two questions have more than 5 words the same, it is the same question". The weak rules are found by applying basic machine learning algorithms to the dataset.

XGBoost is special because of its speed, which is due to the fact that it implements parallel computing. However it also has other benefits over other boosting algorithms like the fact that it works well against overfitting by implementing regularization and that it has a very high flexibility as users can set their own optimization objective and evaluation criterium.

2.5.5 Logistic regression

Logistic regression is a regression model with a categorical output. Here supervised learning is used for classification purposes. Usually, there are two classes in logistic regression, logistic regressions with more classes are analysed with multinomial or ordinal logistic regression.

The goal of logistic regression is to find the best description of the relationship between the independent variables and the response variable. This is done using a logit transformation of the probability of the output being positive:

$$\text{logit}(p) = \ln(\frac{p}{1-p}) = b_0 + b_1X_1 + b_2X_2 + ... + b_kX_k \tag{2.5}$$

Here p is the probability of the response variable being positive, b are the parameters of the model and X are the independent variables [24].

2.5.6 Support Vector Machines (SVM)

An SVM is a binary classifier, based on independent variables it appoints observations to one out of two classes. SVM is a supervised learning method with classification as output. When an SVM is trained a linear boundary is searched which divides the two classes as good as possible. This boundary is called the maximum margin hyperplane. This hyperplane is found by maximizing the margin, i.e. the perpendicular distance between the hyperplane and the closest observation of each class. In figure 2.12 the margin of the bold black hyperplane is shown with the smaller black lines. The bold black line is the maximum margin hyperplane, because it creates a larger margin than the orange or green hyperplanes. When a new observation is added to the dataset, we can easily classify it based on which side of the maximum margin hyperplane it is located on.

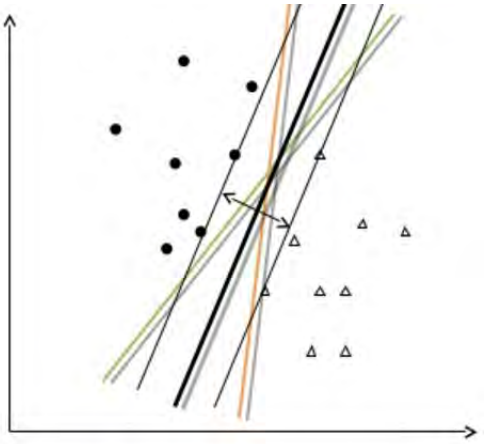


FIGURE 2.12: An example of finding the maximum margin hyperplane in SVM [39].

In some cases it seems hard to divide the data into two classes with a linear hyperplane, as shown in the left figure of figure 2.13. In SVM this problem can still be solved linearly, but in a higher dimension. This higher dimension is calculated using a kernel function, i.e. a mathematical transformation of our data. When we translate the linear boundary back to the input space it becomes non-linear [24].

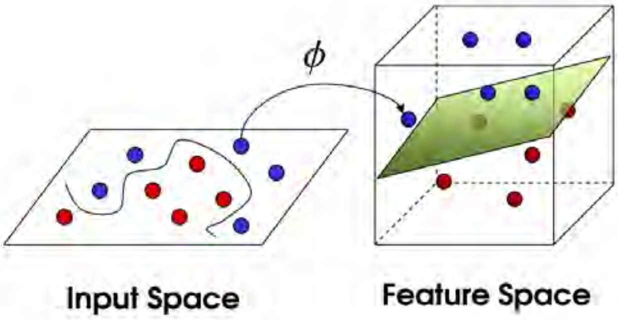


FIGURE 2.13: An example of non-linear SVM: using a kernel to linearly divide the data in a higher dimensional space [55].

2.5.7 Neural networks

Artificial neural networks are based on the structure of human brains. Human brains can solve complicated problems fast because they are composed of billions of cells, called neurons, which are all working together. In figure 2.14 an artificial neural network is depicted. As in the human brain, this network consists of neurons where the information is processed. Signals are passed between neurons over weighted connection links. Each individual neuron applies a certain activation function over its input to determine its output. An artificial neural network can consist of many more layers than in the figure, and all layers between the input and output layer are called hidden layers. The input layer has as many neurons as there are independent variables, and the output layer has as many neurons as their are dependent variables. The amount of hidden layers, and the amount of neurons in the hidden layers depends on the type and amount of data.

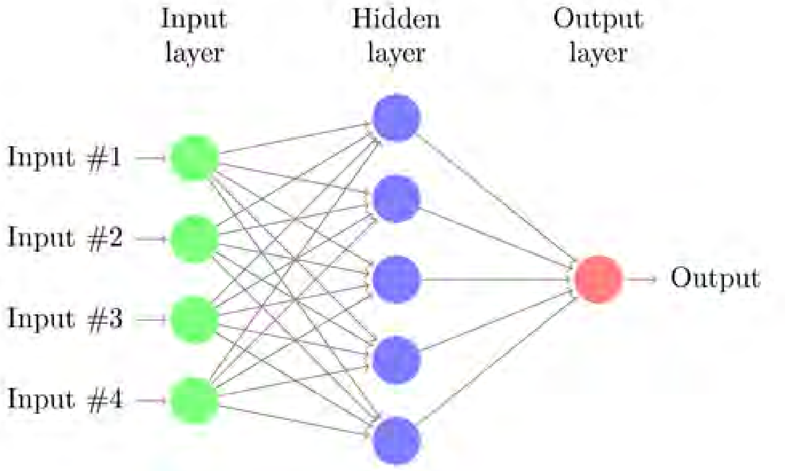


FIGURE 2.14: The structure of an artificial neural network with one hidden layer [22].

When a neural network is trained, the weights on the connection links between the neurons are modified, as to reach the optimal model for the training dataset. Of course, for learning to work there needs to be an element of feedback, the network needs to know what it is doing right or wrong. With this information it can modify the weights on the connection links, and see if the result is better. The larger the difference between the model outcome and the actual outcome, the more radically the connection weights will be altered. In neural networks this feedback process is called backpropagation, as it works back from the output neurons through the hidden neurons to the input neurons. Once the network is done training, it can be presented with new inputs to generate responses.

There are many types of artifical neural networks. The network in figure 2.14 is a feed-forward neural network (FNN) as the information propagates only in one direction. In chatbot systems recurrent neural networks (RNNs) are used because these networks can learn vector representations from words and can remember a huge context compared to other neural networks. RNNs, in contrast to FNNs, have bi-directional data propagation. Data is propagated from input to output, but data from later processing stages is also propagated to earlier stages [24].

When a RNN is trained on a large dataset it is usually not possible to optimize the weights of the connection links in a way that the model output is exactly the same as the actual output for each observation. The sum of the difference between these outputs for all observations is the total error of the model. To minimize this error, gradient descent can be used to change the weights on the connection links in proportion to its derivative. A lot of different RNN architectures were developed to minimize the total error, and the one used for chatbot systems is Long Short-Term Memory (LSTM).

LSTM networks are good at remembering values, in the chatbot case context, for long durations of time. These networks contain gated cells outside the normal flow of a RNN. These gated cells are used to store, write and read information from. The cell itself learns what to store, write and erase and when to allow reads, via gates that can open and close. The gates are implemented with element-wise multiplication by sigmoids, giving analog output in the range of zero to one. The gates work based on signals they receive, and block or pass information based on the strength and sign of the signal, which is processed with the gates own set of weights. These weights, like the weights on the connection links of the RNN, are adjusted during training. Thus the gated cells learn when to import, leave or delete data. In figure 2.15 an example of the structure of a gated cell is shown at the word "free".

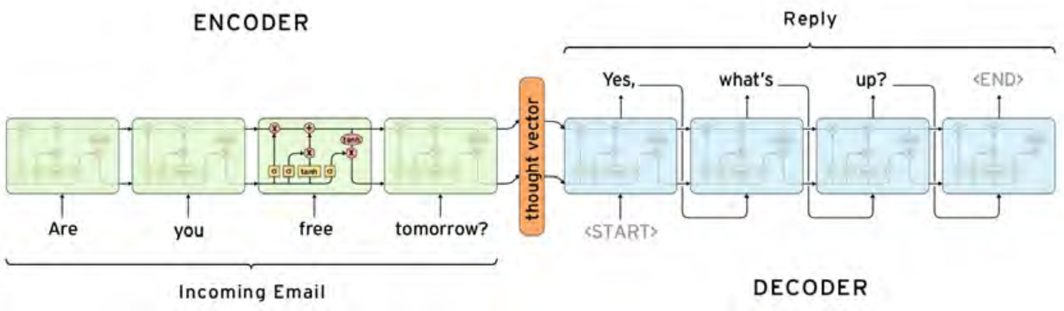


FIGURE 2.15: Diagram of a sequence-to-sequence neural network as used in Google Smart Reply [18].

The neural network shown in figure 2.15 is an example of an end-to-end memory network. These networks allow the model to read the input multiple times before creating an output. This way the memory content can be updated at each step. The specific end-to-end network in the figure is called a sequence-to-sequence network. This network is constructed of two LSTM RNNs, as used in Google Smart Reply. The first RNN encodes the data and the second RNN generates the output. Each green or blue box in figure 2.15 represents a cell of the LSTM RNN. The encoder and decoder networks can share weights, or, more commonly, use different parameters [18].

Another popular type of RNN in NLP is the dual encoder LSTM network, which is shown to give good performance on the Ubuntu dialogue corpus [36]. The architecture of a dual encoder LSTM is shown in figure 2.16. To use this network both question and answer are split by words, and each word is embedded in a vector. These vectors are fed into the same RNN word-by-word. In the figure c represents the question words and r the response words. The RNN now generates a vector that represents the meaning of the question-answer pair. We can multiply c with a matrix M to generate a response r'. This matrix M will be learned during the training of the network. Finally, we can measure the similarity of the actual answer r and the predicted answer r' by taking the dot product between their respective word vectors.

A sigmoid function can be used to convert the dot product outcome to a similarity score [45].

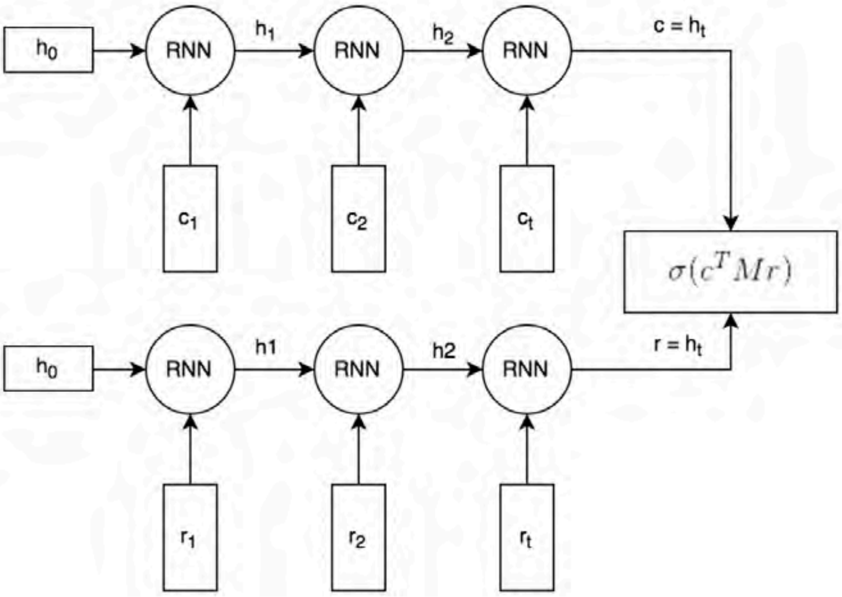


FIGURE 2.16: Architecture of a dual encoder LSTM network [45].

Chapter 3

Methodology

In this chapter we give an overview of the models used to create a chatbot and the chatbot implementation. Figure 3.1 shows a simple diagram of a chatbot. A user can sent a message to the chatbot to which the chatbot formulates a reply using the question-answer dataset. The user and chatbot can sent multiple messages and replies, creating a chat conversation. We have modelled the chatbot using a vector space model, as explained in section 3.1, and using machine learning models, as explained in section 3.2. For both models we have created a chatbot implementation, which is explained in section 3.3. The chatbot implementation section describes the physical manifestation of the end product and its features.

In this thesis financial question-answer and question-to-question datasets are used as the basis for the chatbot, but the same models can also be applied to other domains. The chatbot is very general as it can be applied to any domain for which question-answer and/or question-to-question datasets exist.

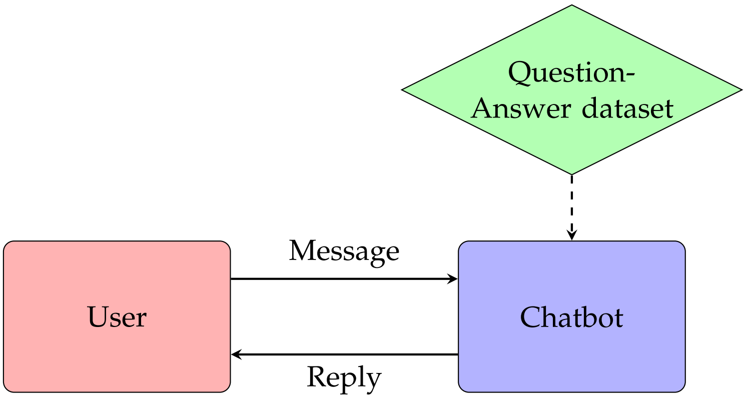


FIGURE 3.1: A diagram of the chatbot framework.

3.1 Vector space model

In figure 3.2 the same diagram as for the simple chatbot model is used to show the inner workings of the vector space chatbot model. The vector space model is a mathematical model used to represent text documents (in this case questions) and is often applied to relevancy rankings. Our vector space model consists of four steps.

The first step in this model is data preprocessing, in which the question goes through an NLP pipeline.

After data preprocessing, the terms in the questions are weighted using term frequency-inverse document frequency (TF-IDF). In this process we create term vectors for the incoming question and the questions in the dataset.

In step 3, the similarity between the question vectors is calculated using cosine similarity.

Finally, the dataset question with the highest similarity to the incoming question will be ranked highest, and the answer to this dataset question will be returned as an answer for the incoming question. We describe the details of the vector space model in chapter 4.

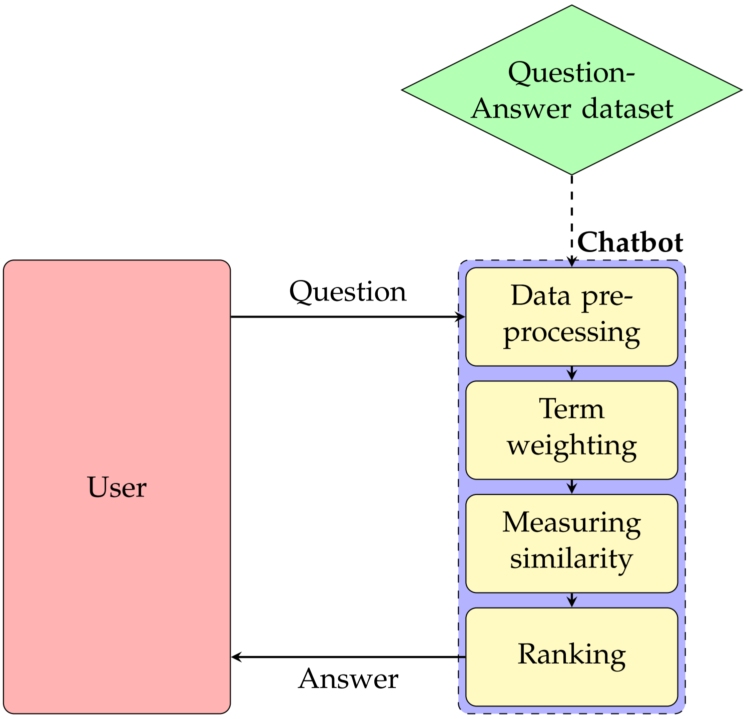


FIGURE 3.2: A diagram of the vector space model.

3.2 Machine learning model

Figure 3.3 gives an overview of the inner workings of the machine learning chatbot model. The first step in this model is data preprocessing, which is done with the same NLP pipeline as in the vector space model.

In the second step, features are created from the preprocessed data. These features include question length, TF-IDF vector similarity, TF-IDF vector similarity using synonyms, TF-IDF score similarity for the most used words, and more.

In the machine learning step a machine learning model trained on the question-to-question dataset is used to score how well two questions match. In this step a confidence level is calculated for the match between the incoming question and each question in the dataset. The machine learning models used are XGBoost, neural networks, random forest and SVM.

In the ranking step, we retrieve the dataset question with the highest confidence level for matching. The answer to this dataset question is returned to the user. We describe the details of the machine learning model in chapter 5.

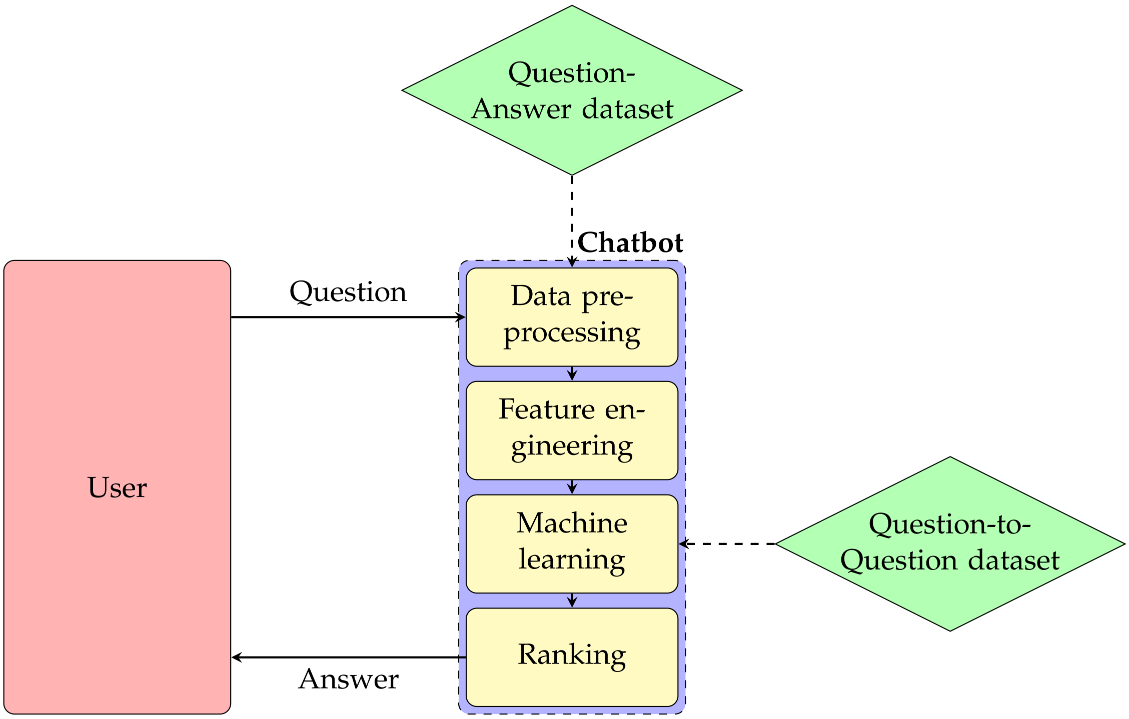


FIGURE 3.3: A diagram of the machine learning model.

3.3 Chatbot implementation

The implementation of the vector based model and the machine learning model in a chatbot application is shown in figure 3.4. Both models output the best matching question from the dataset and the cosine similarity (vector space model) or confidence level (machine learning model) with which this question matches the user question. If the confidence level for this question is high enough (bigger than x in the figure) an answer can be given. This answer is the answer given to the matched dataset question.

If the confidence level is not high enough different terms will be added to the question. By running the newly formed question through the chatbot system again and again (with different terms), we can see if adding a certain term to the question increases the confidence level of matching enough to reach the threshold (x). If the threshold is reached, a return yes/no question is asked to the user. For example, if adding the term "AOW" to the question makes it reach the threshold, the return question is "Does your question have to do with AOW?" If the answer to this question is yes, then the answer to the dataset question that reached the confidence level will be given. If the answer is no, then the question will be sent through to an expert. Also, if adding words to the question does not lead to reaching the confidence level than the question will be sent through to an expert. An expert, in this case, is a human trained in the financial domain.

We describe more details about the chatbot implementation and the results of the chatbots in section 6.

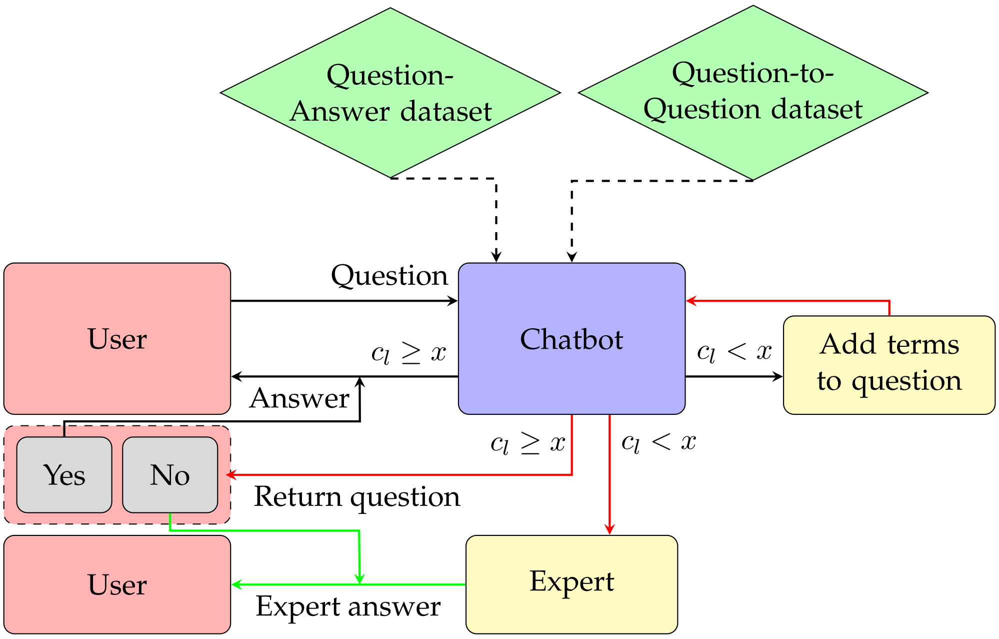


FIGURE 3.4: A diagram of the implementation of the chatbot. C_l stands for the confidence level of the match between the incoming question and the best matching question in the dataset. If the confidence level is high a question is immediately answered by the chatbot. A lower confidence level can lead to a return question, or the question being sent through to an expert.

Chapter 4

Vector space model

In this chapter we create a vector space model to answer questions. The basis of the vector space model is described in section 2.4. Vector space models are most often used for document retrieval. In this chapter we do not use the model for retrieving documents, but for retrieving an answer to a question. Answers are retrieved from a question-answer dataset, which is described, cleaned and explored in section 4.1. Natural Language Processing techniques that are applied to this dataset and incoming questions are described in section 4.2 and we describe the creation of the document-term matix using TF-IDF in section 4.3. The cosine similarity measured used to compare questions and select an answer, and the final ranking is described in section 4.4. Finally, the results of the vector space model on the dataset are described in section 4.5.

4.1 Data

The data used in this study comes from Kandoor, an online platform on which Dutch people can ask financial questions to experts. The framework in which the questions are asked is a chat setting. Therefore the conversation between user and expert is not limited to one question and one answer, but can also contain an entire chat session.

The data is stored in three JSON[1] files. The first one contains the first question a user asked, the second one contains all messages in the rest of the chat session and the third one contains information about who typed which message. The data contains 3936 first questions in total.

4.1.1 Data cleaning

A first glance at the data reveals that there are many unanswered questions. This is not because the experts did not answer (they almost always did), but because the experts asked a return question to the user to be able to answer their question and this return question was never answered. Since our model returns answers from the dataset as an answer to a new question, we need the set of answers to be as correct as possible. To achieve this we clean the dataset by removing question-answer pairs that do not meet our criteria.

In the chat sessions it is difficult to distinguish which chat message is the real question and which message is the real answer (instead of the expert just greeting the user). To make sure our final dataset consists only of financial question-answer pairs (and not chat sessions), we select all sessions in which the first reply was given by an expert and the second reply was either non-existent or given by a user. From these

---

[1] JavaScript Object Notation

sessions we extract only the question and the first reply (the answer). After this procedure we are left with 3592 question-answer pairs. These pairs are further manually cleaned to remove test questions, double questions which got an answer like 'This is a double question. Please, check my previous response.', vaguely posed questions, questions which do not include enough background information and other non-representative questions. Finally, we are left with 3153 question-answer pairs.

4.1.2 Data exploration

To provide an insight into the dataset, consider table 4.1 for a random sample of ten question-answer pairs from the cleaned dataset. Some observations we make from this table are:

- There is a large variation in question and answer length
- There is a large number of spelling mistakes in the questions
- Many answers contain hyper-links
- Answers are often formulated for very specific cases
- Answers often contain a salutation with the user's name, and a greeting with the expert's name

In the process of cleaning the data we found that there were some questions and answers that appeared more frequently than others. A lot of questions were about pensions and AOW (a basic state pension in the Netherlands). Many questions were also about insurance, unemployment benefits and an AOW gap for which a bridging benefit exists. In the answers we saw that many users were sent through to a website where they could find their pension information and another website where they could find their AOW age. We have counted how many times these questions and answers were given and the results can be found in figure 4.1. As this figure shows, more than half of the question-answer pairs contain the word 'pension' and more than 30% of the question-answer pairs contain a hyper-link.

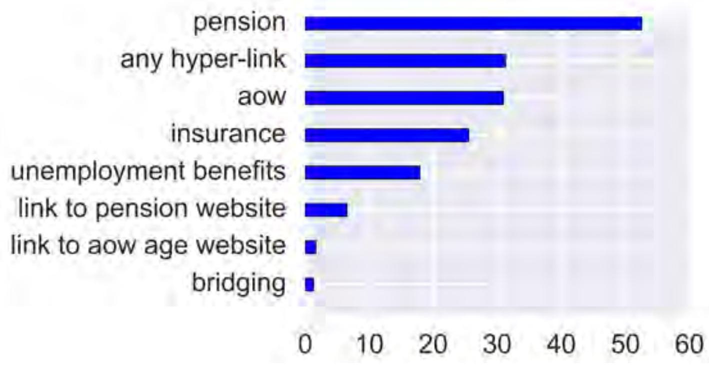


FIGURE 4.1: A bar chart displaying how many question-answer pairs contain a certain word or hyper-link.

Question	**Answer**
kan ik met deeltijd pensioen . 3dagen werken 2 dagen pensioen. hoe wordt er dan gerekend	Beste Jan, Als u met u werkgever kunt afstemmen dat u 3 dagen gaat werken en u voldoende pensioen heeft opgebouwd, kunt u een deeltijd pensioen aanvragen bij uw pensioeninstelling. Het rekenen gaat ongeveer als volgt: U werkt 3 dagen in de week: dit is meestal een 60% (3/5) parttime. U bouwt verder pensioen op over uw parttime loon. U neemt twee dagen pensioen in de week. Dat is dan ca. 40% van uw voltijdspensioen. U kunt het beste een pro forma berekening aan uw pensioeninstelling vragen. Dan weet u waar u aan toe bent. Met vriendelijke groeten, Jurgen Kandoor.nl is een platform dat mensen met een financiële vraag verbindt aan mensen die meer weten. Deze mensen zijn een soort gidsen en helpen anderen in hun vrije tijd met financiële kwesties. Ben je goed geholpen? Sluit de vraag dan af! Bedankt!
Mijn moeder van 94 ontvangt van het begin al aow, sinds een paar maanden ontvangt ze geen aow meer, er is niks veranderd in haar situatie, kan dat?	Geachte Frans, Ik zou geen redenen weten zo direct. Hebt u ingelogd bij het SVB om te zien wat er gaande is. Kunt daar de status zien. U kunt inloggen met het digid `http://www.svb.nl/int/nl/`. Mocht u daar rare dingen tegen komen kunt u direct mailen of bellen. vr.gr. Andre
Ben ik als ik voor onderzoek naar het Albert Sweitzer ziekenhuis in Dordrecht moet, ook verzekerd? Mevr. A Verstraten-Mies	Beste mevrouw Verstraten, Op de website van het Albert Schweitzerziekenhuis kunt u vinden met welke verzekeraars allemaal een contract is afgesloten. U vindt dit als u dit internetadres in uw adresbalkje op internet plakt: `https://www.asz.nl/patienten/Zorgkosten/Zorgverzekering/gecontracteerde-zorgverzekeraars-2016/` Als uw verzekeraar er niet bijstaat adviseer ik u even contact op te nemen met de verzekeraar. Zij moeten u in dat geval vertellen: - onder welke voorwaarden u toch naar het ASziekenhuis kunt -welk ander ziekenhuis bij u in de buurt bezocht kan worden. Veel succes. Ellen den Boer
Mijn vraag ,worden pesioenen ergens centraal geregistreerd.	Geachte mevrouw of heer, Veel pensioenen worden in het pensioenregister geregistreerd (niet allemaal). U kunt dat raadplegen op de volgende website. `https://www.mijnpensioenoverzicht.nl/pensioenregister/` Met vriendelijke groet, Ernst de Reus

Mag een reisverzekering (abnamro) bij twijfel over een aankoop een machtiging toesturen voor de site waar je je aankoop hebt gedaan. Terwijl dat de facturen al zijn gezien en meegenomen door een expert die thuis is geweest van de verzekering. Hoe zit het met privacy wetgeving en waar ligt de grens. Tot hoe ver moet ik meewerken?	Beste heer/mevrouw, De reisverzekering (ABNAMRO) vraagt niet zomaar gegevens van u op. Dat zou in het kader van de privacywet niet moeten mogen. Maar daarom sturen ze u eerst een machtiging om toestemming te krijgen om gegevens op te vragen. Het kan zijn dat zij dit nodig achten voordat zij tot vergoeding van een gedane aankoop overgaan. Verzekeringsbedrijven willen door fraude die helaas vaak voorkomt niet teveel risico's nemen. Het is in die zin dus wel begrijpelijk dat ze onderzoek willen doen. Maar u bepaalt of u de machtiging wilt geven of niet. U kunt misschien wel bij ABNAMRO navragen voor welke gegevens de machtiging geldt. En, voor zover ze dit niet gedaan hebben, of ze op de machtiging willen aangeven welke info ze precies gaan opvragen zodat u alleen daarvoor tekent. mvg Daniëlle
Hoeveel belasting is er ingehouden van het vakantiegeld van een alleenstaande AOWer?	Hoi Dit is o.a. afhankelijk van: "bent u altijd verzekerd geweest" en "is er loonheffingskorting" toegepast. Uw betaalspecificatie zou daar opheldering over moeten geven. Tip: op de site `https://www.rekenkeizer.nl/pensioen-aow-leeftijd/aow-vakantietoeslag-vakantiebijslag-bruto-netto-berekenen` kunt u eea precies narekenen! Succes! Patrick
Ik kreeg net een mail van Salarisadministrateur. Hierin staat: Voorheen betaalde het pensioenfonds meer premie dan de werknemer over deze opbouw. Echter, per 1 januari 2016, heeft het pensioenfonds dit aangepast naar 50% voor beide partijen. Dus 50% voor de werknemer en 50% voor het pensioenfonds. Voor jou betekent dit dat wij vanaf januari t/m juni een correctie moeten doen. Het gecorrigeerde bedrag hebben wij verdeeld over de rest van het jaar of tot eind van de ouderschapsverlof periode indien deze voor eind van het jaar eindigt. Vanaf juli is het bedrag dat wij inhouden voor het pensioen dus hoger ivm de verhoging en de correctie. Het verschil voor de medewerkers ligt tussen de +-€5 en €28. Ik heb vanaf 01-01-2016 ouderschapverlof. Vraag mag dit zomaar?	Beste Quinten, Pensioen is een arbeidsvoorwaarde. Als sociale partners afspreken (werkgever of werkgeversorganisatie en vakbonden) om de premieverdeling te wijzigingen dan kan dat. In die zin is pensioen vergelijkbaar met andere arbeidsvoorwaarden, waar soms ook verbeteringen en verslechteringen worden afgesproken. Het zou slechts anders zijn als u niet onder een CAO valt, maar een individuele afspraak heeft gemaakt met uw werkgever over uw arbeidsvoorwaarden. In dat geval zou het alleen in onderling overleg kunnen. Met vriendelijke groet, Ernst de Reus

Ik ben 73 mijn vrouw is 60 ik krijg nu een extra AOW bijdrage voor mijn vrouw wij overwegen om in Suriname te gaan wonen. Kunt u mijn zeggen wat het gevolg is voor deze extra bijdrage en de aow opbouw van mijn vrouw. Als mijn vrouw in nederland blijft ingeschreven en 3 mnd in suriname verblijft en dan weer 3 mnd in Nederland en zo verder wat is dan het gevolg	Beste heer Vernooij, U kunt op de website van de sociale verzekeringsbank informatie vinden over het ontvangen van toeslag op uw AOW ivm. een jonger partner. Zover ik kan zien, kunt u dit behouden als u naar Suriname verhuist. Zie: `http://www.svb.nl/int/nl/aow/wonen_buiten_nederland/beu/index.jsp` Met vriendelijke groeten, Jurgen Kandoor.nl is een platform dat mensen met een financiële vraag verbindt aan mensen die meer weten. Deze mensen zijn een soort gidsen en helpen anderen in hun vrije tijd met financiële kwesties. Ben je goed geholpen? Sluit de vraag dan af! Bedankt!
Is extra aflossen van mijn hypotheek een goed idee op de lange termijn?	Je kunt 30 jaar hypotheekrente aflossen, dus uiteindelijk stijgen je kasten bij niet aflossen, om voor uw persoonlijke situatie af te wegen of dit interssant is kan ik niet direct beoordelen. Ik raad je aan om bijv de calculator op fiscalert.nl te gebruiken
Tussenpersoon van verzekering geeft per mail door dat de schade uitbetaald wordt ik regel een bedrijf dat de hele schade gaat maken , het bedrijf besteld alles en begint met de reparatie , op dat moment krijg ik een mail (3 weken na het eerste mailtje) dat de schade niet wordt vergoed ivm te laat betalen van een premie ?	Beste heer Vink, Als u de premie voor je autoverzekering niet of (te) laat hebt betaald zult u in eerste instantie incassoherinneringen ontvangen hebben. Als u daarop niet gereageerd heeft. Dan zal uw verzekering opgeschort worden. Dat betekent dat uw verzekering als het ware in de pauzestand wordt gezet. Als u dan gaat auto rijden, bent u dus niet verzekerd! De gevolmachtigde agent maakt dan een een extra polisblad waarin staat dat de verzekering per de premievervaldatum is opgeschort. Uw kenteken wordt dan ook afgemeld bij de RDW. Zij melden in hun systeem (het kentekenregister) dat de verzekering voor jouw auto is beëindigd. Echter ook nadat een verzekering is beëindigd of opgeschort wegens wanbetaling blijft een verzekeraar aanspreekbaar voor eventuele schade die met het onverzekerde voertuig wordt toegebracht. De verzekeraar moet zich namelijk houden aan wettelijke bepalingen. Zo moet de verzekeraar het kenteken afmelden bij de RDW, maar dat moet gebeuren binnen 30 dagen nadat de premie betaald had moeten worden. Is uw verzekering opgeschort? Dan blijft de verzekeraar wel tot 16 dagen daarna nog aanspreekbaar voor de schade. Zo worden eventuele slachtoffers zoveel mogelijk beschermd. Let op: verzekeraars kunnen de schade-uitkering die ze in deze periode moeten doen aan een onschuldige tegenpartij, wel verhalen op de onverzekerde persoon. In uw situatie heeft u schade. De verzekering hoeft de schade cf bovenstaande dan niet te betalen. Geldt er een bijzondere reden dat u de premie niet betaald heeft of heeft u geen herinnering gehad, dan zou ik in ieder geval bezwaar maken tegen het besluit van uw verzekeraar. Met vriendelijke groet, Daniëlle Blezer

TABLE 4.1: Ten randomly sampled questions with answers from the Kandoor dataset.

Word counts and sentences counts

To analyse the text in the questions and answers we first gathered statistics and made visualizations of the number of words and sentences they contain. In figure 4.2 histograms and box plots of the number of words in the questions and answers can be found. Figure 4.3 show the same figures for the number of sentences.

In figure 4.2 we see that questions containing 0 to 30 words already span more than 50% of the dataset. This histogram is fat-tailed as it shows a large skewness to the right. In this histogram we cut off the number of words at 300, while the actual word maximum was 411. In this procedure we cut off 0.13% of the question data. The questions have an average of 36 words and a median of 27 words. Considering the histogram we expected the average to be higher than the median, as questions exist with up to 411 words.

Comparing the number of words in the questions and answers, we see that the answers are on average more than twice as long as the questions. This can have multiple causes, for example because financial answers are complex and the explanation of financial measures or procedures can take up many words. The histogram for the answers shows that the number of words most frequently used in answers is between 30 and 70 words. The answers with 30 to 70 words also make up more than 50% of the dataset. The median is a bit higher with 71 words in an answer. The cut off of the histogram at 300 words removes 2.25% of the answers from the histogram.

The box plot in figure 4.2 shows that questions have a statistically significant lower number of words than answers. This can be seen by the interquartile ranges of the box plots not overlapping each other.

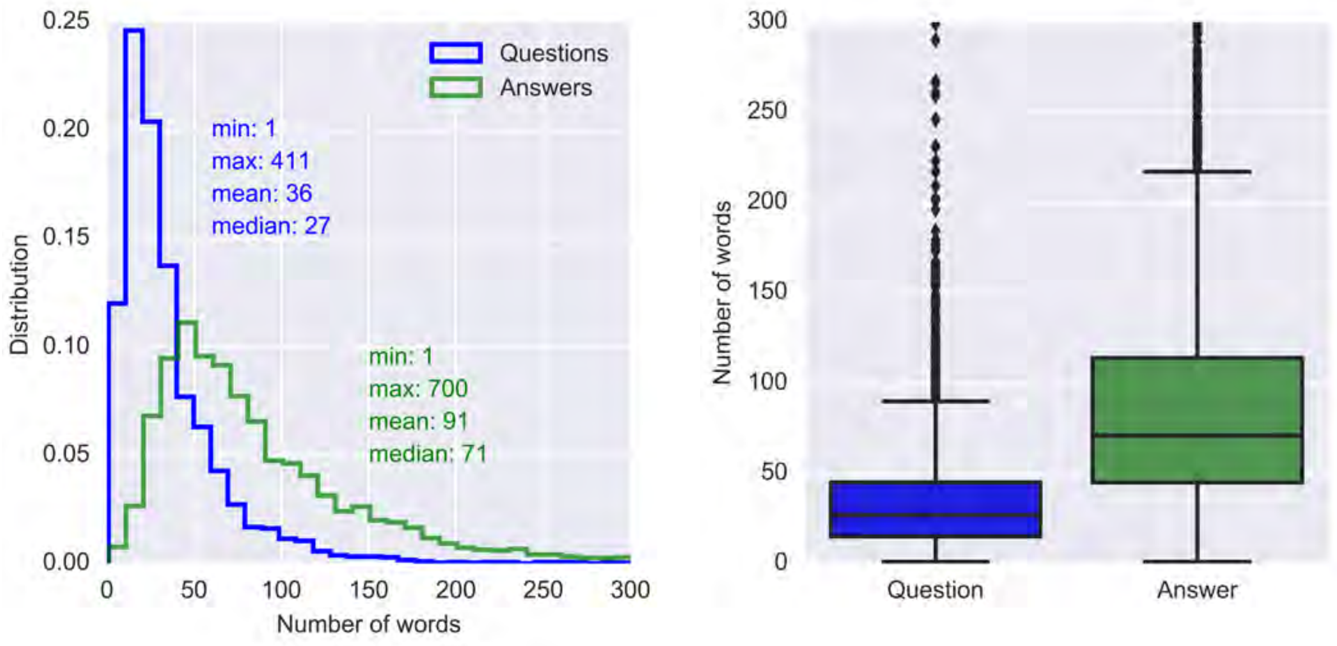
Figure 4.3 shows that about 40% of the questions consist of one sentence and that most answers consist of 3 to 6 sentences. The histograms are fat-tailed and show a large skewness to the right as the mean is higher than the median. The median number of sentences in questions and answers is 2 sentences per question and 6 sentences per answer, respectively.

The box plot in figure 4.3 shows that questions have a statistically significant lower number of sentences than answers. This can be seen by the interquartile ranges of the box plots not overlapping each other. This was also expected as the questions also have a significantly lower number of words.

After these first word and sentences counts, we became interested to see whether there was a relation between question and answer length. To explore this relation we created the heat maps in figure 4.4. From these heat maps we can see that there is no relation between question length and answer length. We expected differently, as longer questions can describe a more specific situation to which a very specific and short answer, can be given.

To research the keywords and topics of the questions and answers more closely we made an ordered bar chart with the 25 most used words in the questions and answers. To generate these bar charts we first made all letters in the questions and answers lowercase and then removed stop-words. The removed stop-words contained a standard list of Dutch stop-words and we added some frequently occurring words which had little meaning in our dataset to this list. These were salutations, frequently mentioned names of experts, and Dutch stop-words that were not in the standard list.

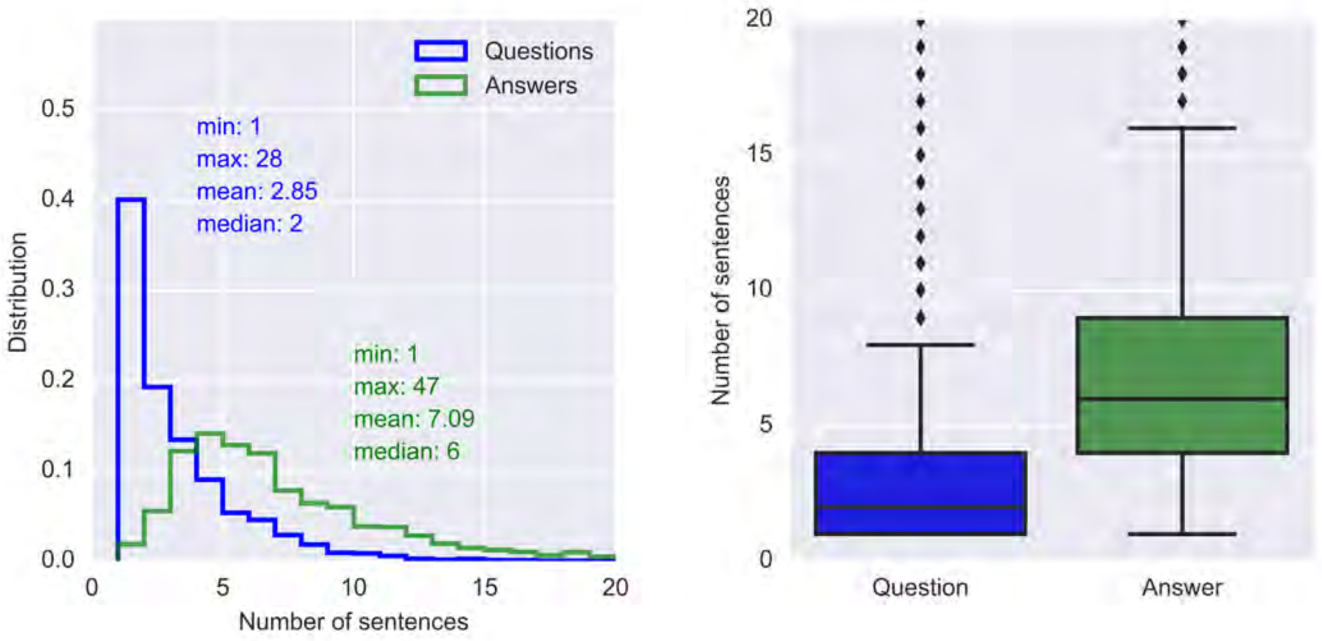
In figure 4.5 we can see that the most used words in the questions are ′pensioen′, ′jaar′, ′aow′ and ′uitkering′. The most used words in the answers are ′pensioen′,′aow′, ′pensioenfonds′ and ′jaar′. In the questions ′pensioen′ makes up 3.1% of all words in all questions and in the answers it makes up 2.3%.



(A) Histogram

(B) Box plot

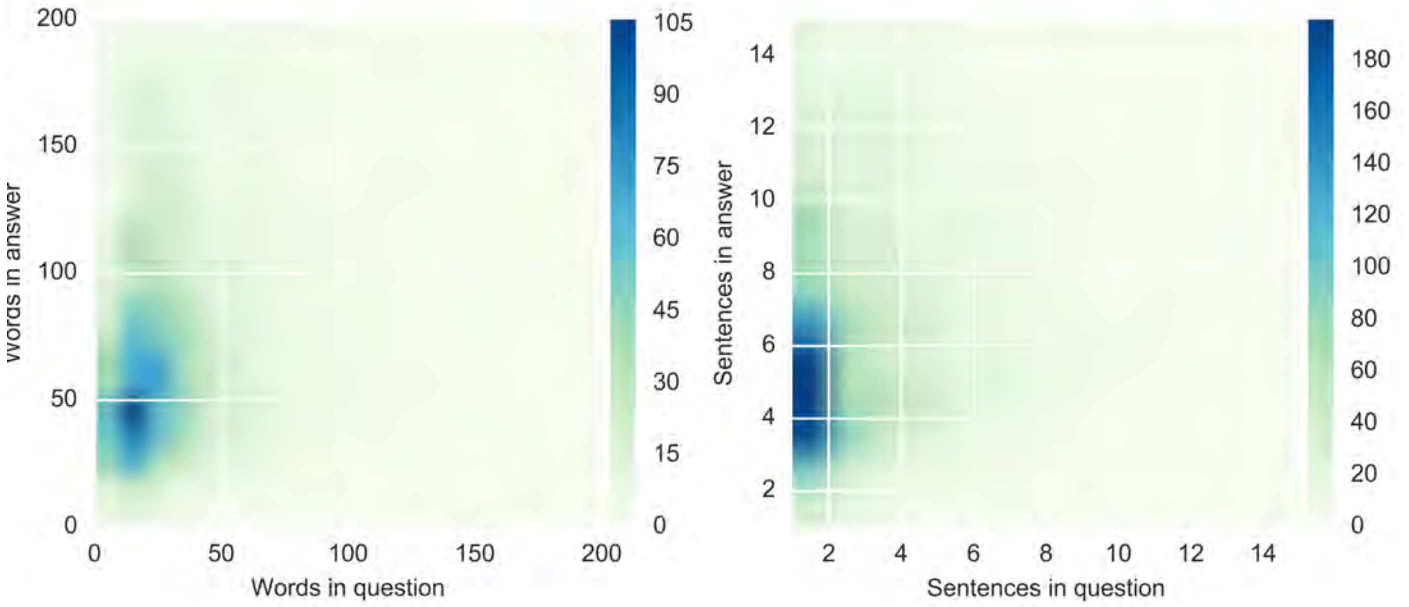
FIGURE 4.2: Charts displaying the distribution of the number of words in the questions and answers.



(A) Histogram

(B) Box plot

FIGURE 4.3: Charts displaying the distribution of the sentences of words in the questions and answers.



(A) Words

(B) Sentences

FIGURE 4.4: Heat maps displaying the relation between the number of words and sentences in question and answer.

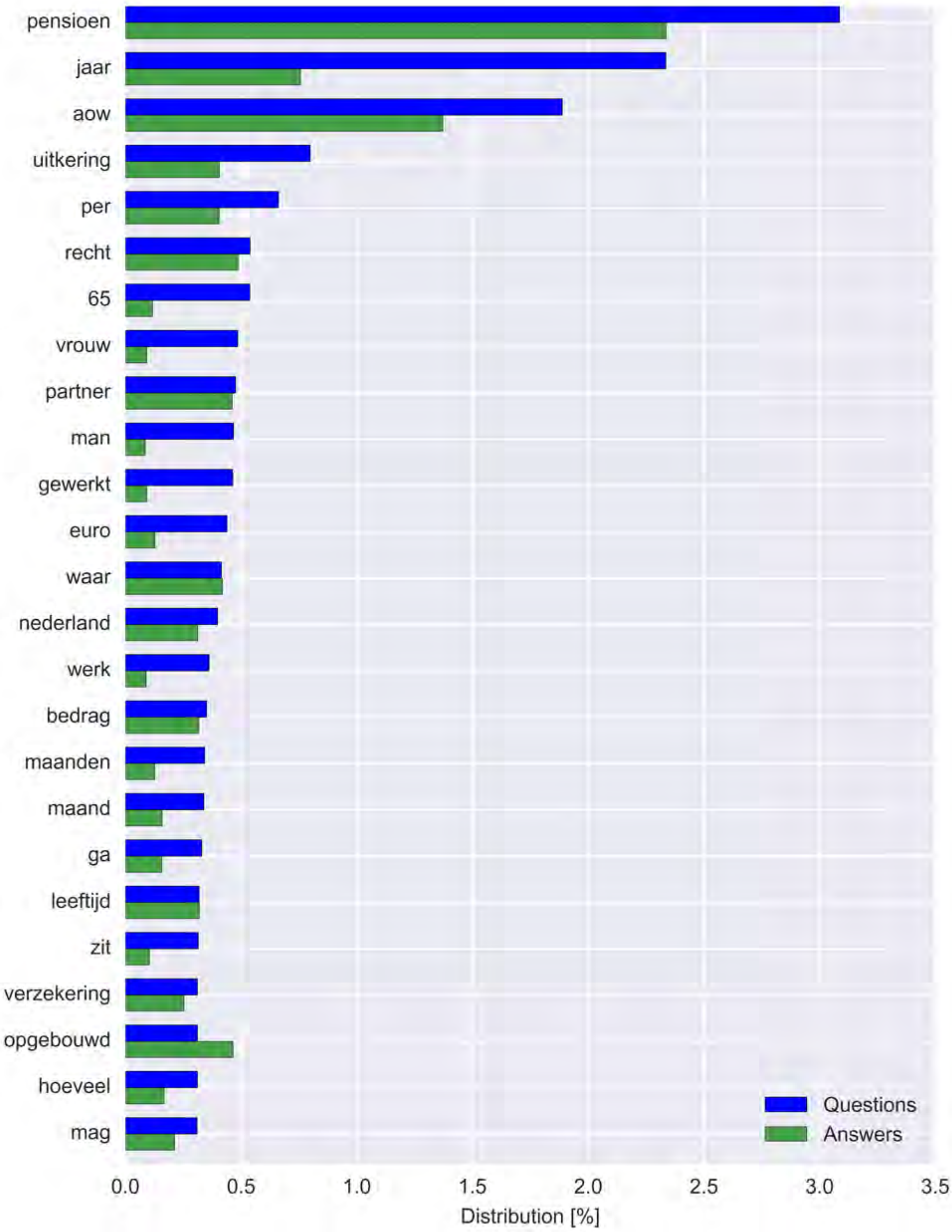


FIGURE 4.5: A bar chart displaying the frequency of the 25 most used words in the questions and answers.

N-grams

In this section we look at the unigrams and bigrams in the questions and answers. When looking at common unigrams in question and answer, we are essentially looking at words (after stop-word removal) that occur in the question and the corresponding answer. The number of common unigrams and bigrams in our dataset can be seen in the histogram in figure 4.6(A). In this figure, we see that question-answer pairs with 0 to 3 words in common make up more than 50% of the dataset. Surprisingly, we also see that about 12% of the question-answer pairs do not contain any of the same words. An example of this is:

Question Ik ben weduwe en ontvang partnerpensioen vervalt die als ik hertrouw

Answer Geachte mevrouw, Dat valt niet zo te zeggen maar hangt af van de pensioenregeling. In sommige regelingen stopt de uitkering op het moment dat de overgebleven partner opnieuw gaat samenwonen of trouwen. U kunt dat het beste navragen bij de pensioenuitvoerder hoe dat voor u geregeld is. Met vriendelijke groet, Ernst de Reus

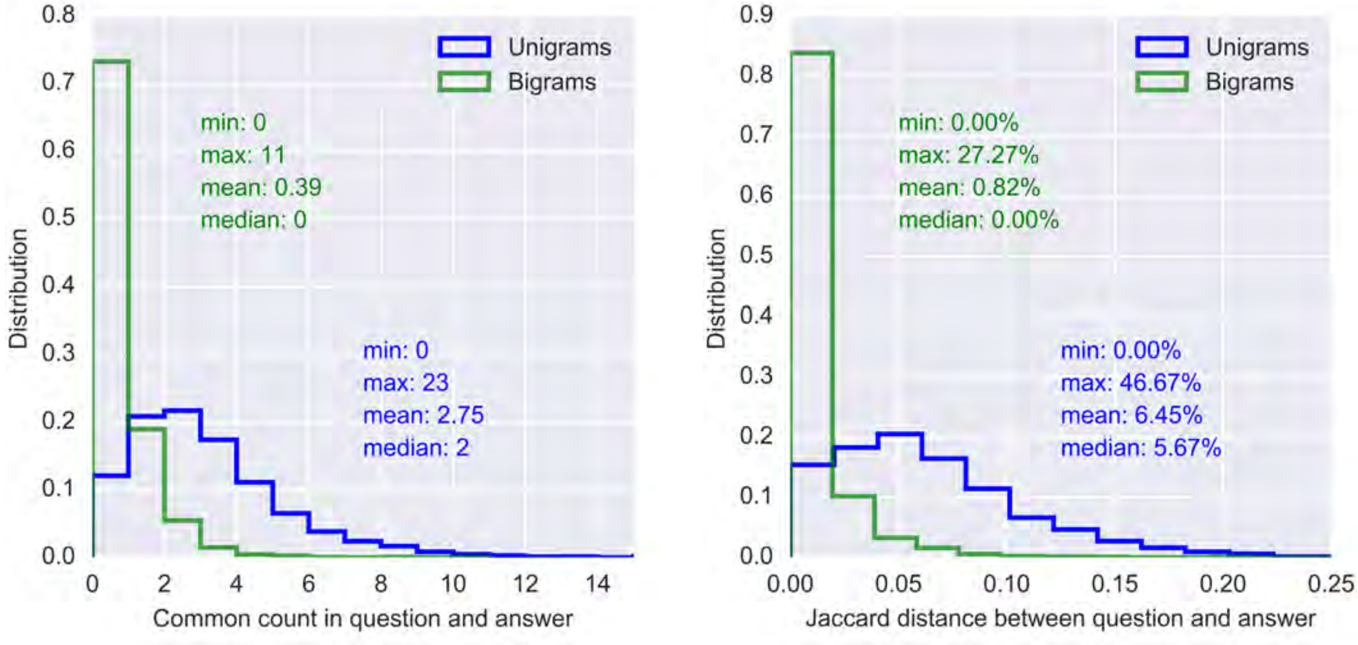
This question-answer pair does not contain words that are exactly the same, but it does contain compound words, like 'partnerpensioen' in the question of which both parts are used in the answer. Also, the question contains the word 'weduwe' where the answer has the synonym 'overgebleven partner'.

Figure 4.6(A) also shows the bigrams occurring in the question and the corresponding answer. An example of a common bigram is when the question contains the bigram 'accrued pension' and the answer too. As we can see in the figure, common bigrams do not occur very often. About 72% of question-answer pairs do not have any common bigrams. This is probably due to the fact that words that are often a bigram in English are compound words in Dutch. For example, in English 'pension fund' can be a bigram, where this in Dutch becomes the compound word 'pensioenfonds'.

A longer question (or answer) has a higher chance of containing unigrams and bigrams from the corresponding answer (question). Therefore displaying a ratio of unigrams and bigrams in comparison to the total number of words gives a better representation. To do this we used the Jaccard index:

$$J(Q, A) = \frac{|Q \cap A|}{|Q \cup A|} \tag{4.1}$$

This index takes the number of common unigrams $|Q \cap A|$ in the question Q and the answer A and divides this by the set of the total number of words $|Q \cup A|$. Figure 4.6(B) shows a histogram of the Jaccard distances for the unigrams and bigrams. Here the Jaccard index takes the number of common bigrams and divides this by the set of the total number of bigrams. This histogram shows that about 70% of the question-answer pairs in the dataset have a Jaccard distance smaller than or equal to 0.06 for unigrams. The Jaccard distance for bigrams is smaller than or equal to 0.02 in 83% of the dataset.



(A) Count

(B) Jaccard distance

FIGURE 4.6: Histograms displaying the count and Jaccard distance of the common unigrams and bigrams in the question-answer combinations.

Conclusion

After this exploration of the dataset, we have found some key points that we need to pay attention to when applying NLP and when modelling:

- Many answers contain salutations with names in them, and sometimes very personal messages, for example if someone passed away. Therefore we have to make sure that answers we return to new questions are directed towards this new user and do not contain messages that are not suited to this user.
- There is a large number of spelling mistakes in the questions
- Compound words and synonyms make it seem like question and answer are not similar
- Question and answer are very dissimilar. Common unigrams do occur often, but common bigrams are very rare.

As question and answer are very dissimilar, we do not focus on matching a new question to an answer from the dataset. We only focus on matching a new question with the questions in the dataset, and if we find a match we return the answer that corresponds to the matched question.

4.2 Natural Language Processing

In this section NLP techniques are applied to the questions and answers to prepare them for modelling.

In the data exploration phase we found that answers often contain personalised salutations.To improve the dataset from which we return answers, we replaced these personalised salutations with general salutations. This process is further specified in section 4.2.1.

The questions go through our NLP pipeline. An overview of the NLP pipeline and a more detailed explanation of the techniques in the NLP pipeline can be found in section 4.2.2.

4.2.1 Salutation replacement in the answers

During data exploration we found that the answers in the dataset contained personal salutations and closings. For example, an answer could start with "Dear Jonathan," and end with "Kind regards, Suzanne". To make sure that all questions are suitable as an answer to any user we changed all salutations into "Dear Sir/Madam," and endings into "... regards, KanBot". In the ending we kept the word in front of 'regards', so for example it could be just "regards" or "kind regards" or "best regards" or another variation. KanBot is the name of the chatbot, which was already thought of by Kandoor. The replacement of the opening salutations was done with the pseudocode in algorithm 1 and the replacement of the greetings was done with algorithm 2. With algorithm 1, we managed to replace 90% of our greetings with the standard "Dear Sir/Madam, " greeting. Most answers that were not affected by the algorithm did not start with a greeting to begin with. With algorithm 2, 80% of answers was altered to end with "KanBot" instead of the name of an expert. The unaffected answers can still contain names of experts, but these are hard to replace with an algorithm without also replacing parts of the last sentence that should not be replaced. An example of this is an answer ending with just the name of an expert without a greeting word like "Groeten". We cannot remove all last words from answers as this could also mean removing the last word of answer that has no greeting or expert name at the end. In that case we would accidentally remove the last word of an answer.

Algorithm 1 Replacement of opening salutations

```
Break up each answer into its sentences.
Collect "greeting words" from the first sentence of each answer. "Greeting words" is a list of words that we composed specifically for this dataset.
If the first sentence of an answer contains either a greeting word or the second word in the answer is a comma, then substitute everything before (and including) the first comma in the answer with "Dear Sir/Madam, ". Else do nothing.
```

Some examples of original sentences are:

- "Dear Jonathan, this is a financial answer. Best regards, Suzanne"
- "Jonathan, this is a financial answer. Regards, Suzanne"
- "Hi Jonathan, this is a financial answer. Good luck with it! Suzanne"

After salutation replacement these sentences would transform into:

- "Dear Sir/Madam, this is a financial answer. Best regards, KanBot"

Algorithm 2 Replacement of closing salutations

```
Break up each answer into its sentences.
Collect "closing words" from the last sentence of each answer. "Closing words" is again a list of words that we composed specifically for this dataset.
If the last sentence of an answer contains a closing word, then substitute everything from (and including) the closing word with "closing word, KanBot". If else, the last sentence only contains one word than replace this word by "KanBot". Else do nothing.
```

- "Dear Sir/Madam, this is a financial answer. Regards, KanBot"
- "Dear Sir/Madam, this is a financial answer. Good luck with it! KanBot"

4.2.2 The NLP pipeline for the questions

During the data exploration phase, we found some key points we needed to take into account in NLP. This resulted in the NLP pipeline for the questions in figure 4.7. The order of applying these NLP techniques is very important. For example, if we would want to extract a word from a compound word before spelling correction, we could miss extracting words in which the compound words are misspelled. The word "pensjoenfonds" will not lead to "pensioenfonds" if compound word extraction is done first and it will lead to "pensioenfonds pensioen" if spelling correction is done first. Of course "pensioenfonds pensioen" can lead to more matches than only "pensioenfonds".

That said, the placement of stop-word removal in this order of techniques is only dependent on spelling correction. Stop-word removal has to take place after spelling correction, as also stop-words can be misspelled. We apply stop-word removal early on in the process to shorten the calculation time of later parts but the actual outcome is not influenced by this placement. Therefore, if later on we decide to expand the list of stop-words to try and achieve better results we can just subtract these extra stop-words from the already prepared dataset.

Tokenization

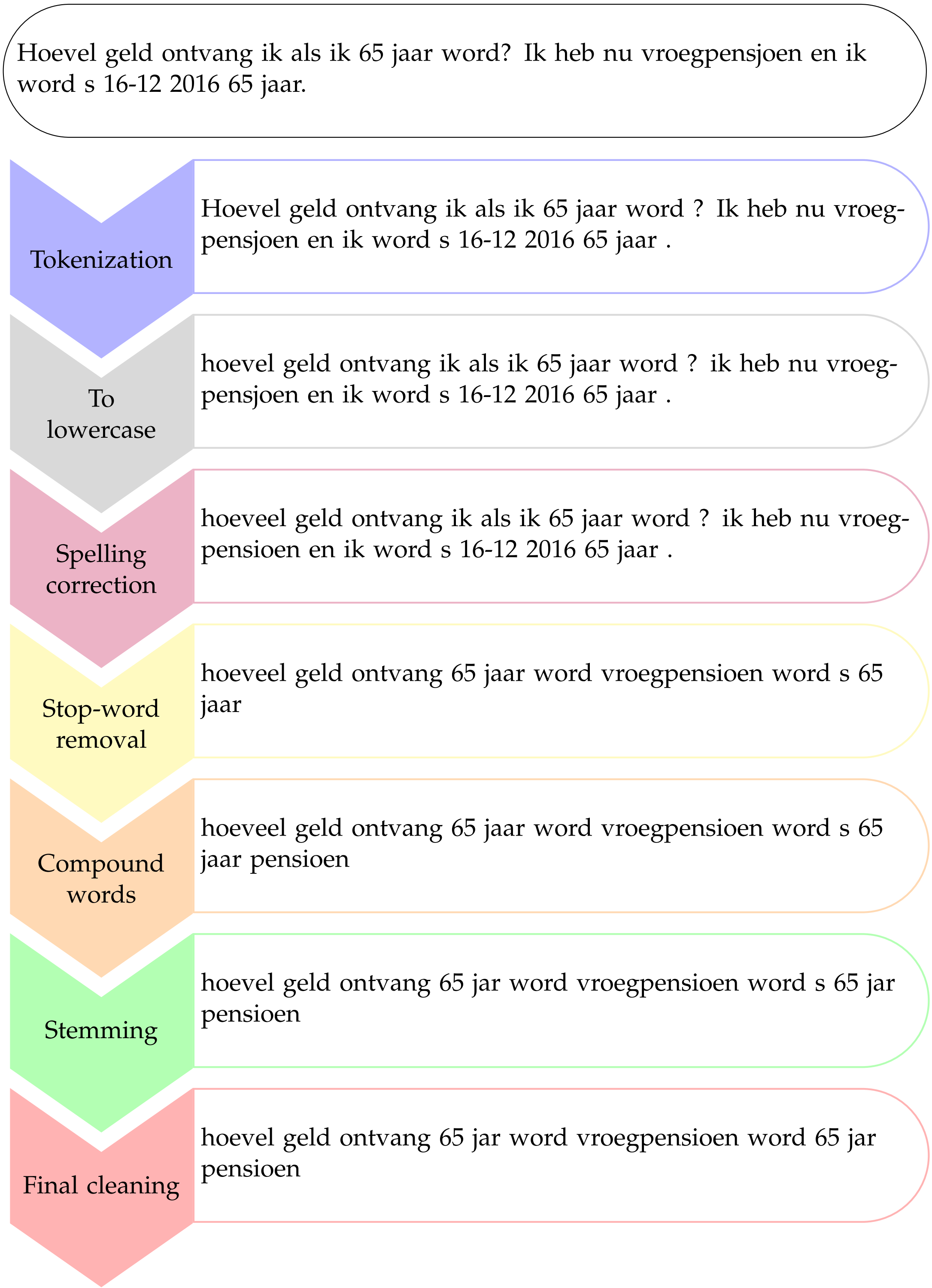
Tokenization, as explained in section 2.3.1, is the process of breaking down a text into words and symbols. We apply this by separating all words and punctuation with white space.

Lower-casing

From this point on, we changed all letters in our dataset to lower-case to make sure that identical words match each other, regardless of the letters begin lower-case or upper-case.

Spelling correction

In the data exploration we found that the questions in our dataset contained many spelling mistakes. These spelling mistakes can influence matching a new question with the questions in the dataset. Therefore, we build a spelling corrector. To do this we started with a list of all the words currently used in our dataset. Now we removed the words from this list that were used 5 times or less (these words are very



Hoevel geld ontvang ik als ik 65 jaar word? Ik heb nu vroegpensjoen en ik word s 16-12 2016 65 jaar.

Hoevel geld ontvang ik als ik 65 jaar word ? Ik heb nu vroegpensjoen en ik word s 16-12 2016 65 jaar .

hoevel geld ontvang ik als ik 65 jaar word ? ik heb nu vroegpensjoen en ik word s 16-12 2016 65 jaar .

hoeveel geld ontvang ik als ik 65 jaar word ? ik heb nu vroegpensioen en ik word s 16-12 2016 65 jaar .

hoeveel geld ontvang 65 jaar word vroegpensioen word s 65 jaar

hoeveel geld ontvang 65 jaar word vroegpensioen word s 65 jaar pensioen

hoevel geld ontvang 65 jar word vroegpensioen word s 65 jar pensioen

hoevel geld ontvang 65 jar word vroegpensioen word 65 jar pensioen

FIGURE 4.7: A schematic overview of the NLP pipeline.

likely to be misspelled). We also removed some extra words that we found in the list for this dataset specifically, that were misspelled even more than 5 times. Now that we have our list of correct words, we ran the words in all of our questions through the spelling corrector in algorithm 3. This algorithm made one or more spelling corrections in 42% of the questions.

Algorithm 3 Spelling correction

```
1: Calculate the probability of this word occurring for this specific dataset. This is done by dividing the frequency with which the word occurs in the dataset and dividing it by the total number of words in the dataset.
2: Find all edits that are only one correction away from the word. One edit can be a deletion of a letter, an insertion of a letter, a swap of two adjacent letters or a replacement of one letter for another.
3: Find all edits that are two corrections away from the word. This means running step 2 again on its own outcome.
4: Restrict the sets of words (one and two edits away from the initial word) to only include words that are in our existing word list.
5: Generate the corrected spelling candidates for the word. These candidates are (in order of replacement): the word itself (if it is in the word list), words in the word list that are one edit away, words in the word list that are two edits away, and finally if none of these exist it will be the word itself.
6: Correct the word. This is done based on the word candidates generated in the previous step. If the word itself is in the word list it will not be altered. If the word is not in the word list the first option is the words that are one edit away and in the word list. If these exist we choose the one with the highest probability of occurring in our dataset, as calculated in step 1. Otherwise, we look at the words that are two edits away and in the word list, again choosing the one with the highest probability of occurring. If all of these options do not generate words, we just give the word that was supplied back as this is then a rare word that is potentially important for the meaning of the question.
```

Stop-word removal

After spelling correction, we used algorithm 4 to remove stop-words and other characters with little meaning from our dataset. The standard Dutch stop-word list and a list of stop-words specific to our dataset can be found in appendix A.1. The standard Dutch stop-word list contains 101 words, and the list with extra words specific to our dataset contains 37 words.

Extracting words from compound words

In the Dutch language many compound words exist, making a match between two similar phrases more difficult. For example, 'pensioenfonds' is a word in Dutch which will not match with 'pensioen' while questions containing these words can be very similar. Therefore we have extracted some words from their compounds and placed this extra word at the end of the sentence. This way, we can now match this sentence with other sentences containing either the compound word (e.g. pensioenfonds) or the extracted word (e.g. pensioen). The list with words to extract was manually constructed and contains 54 words. This list can be found in appendix A.2.

Algorithm 4 Stop-word removal

```
Remove all standard Dutch stop-words using a pre-made Dutch stop-word list.
Remove extra words that are specific to the dataset and contain little meaning in this case. For example, "hello" or "question".
Remove all loose punctuation (the punctuation that is separated from words, like a dot at the end of a sentence).
Remove all words which contain numbers except for "65" and "67", because these have special meanings (pension age) in our dataset.
Remove all hyper-links.
Change all punctuation that is still present (this is only possible inside words), to spaces. For example, "pension-age" is changed to the words "pension" and "age".
```

Some words appear in compound words but not in the list, because these words are not relevant in our dataset. This is for example the word "vroeg" in figure 4.7.

Stemming

A stemming algorithm was applied on the words in the questions to reduce words back to their original stem. This is done to make matching of questions with the same content easier. We used an already existing popular stemming package for the Dutch language, called Snowball[2]. This stemming algorithm does not always stem to the original stem of a word, but it does get words with the same stem to the same form, which also works. For example, in figure 4.7, the word "hoeveel" is stemmed to "hoevel" which is not the original stem, but as "hoeveel" is stemmed to "hoevel" in every question the words will still match.

Final alterations

Finally, upon inspection of the words left in the dataset, we noticed that there were some words left that consisted of only one character. In the Dutch language no word consists of one character therefore we removed all words with only one character.

4.2.3 Situation after NLP

After applying the NLP pipeline we are left with a dataset containing three columns: the original question, the terms left in the question after data preparation, and the corresponding answer to give back if this question is selected (with improved salutations). Our prepared questions still contain a total of 4978 different words. This is still a large set of different words, therefore on further inspection we might remove more words which seem less useful in the matching of the questions. This can be done in steps in creating different chatbot versions to see the impact of these decisions.

4.3 Term weighting

In our vector space model we use the TF-IDF score to create the document-term matrix. The theory behind TF-IDF was already explained in section 2.3.2, but here we will give an intuitive example of how this score works in practice. Consider a

---

[2] `http://snowballstem.org/`

new question and a dataset of three questions to match it to:
Question: Wat is mijn pensioenleeftijd? Geboortedatum 13-07-1952.
Dataset:

- **Question 1:** Wanneer krijg ik mijn pensioen? Mijn leeftijd is 65 jaar.
- **Question 2:** Wat is mijn AOW-leeftijd?
- **Question 3:** Hoeveel studiefinanciering krijg ik? Geboortedatum 17-05-1992.

Of course, these questions all first go through our NLP pipeline and this results in:
Question: pensioenleeftijd geboortedatum leeftijd pensioen
Dataset:

- **Question 1:** wanner pensioen leeftijd 65 jar
- **Question 2:** aow leeftijd
- **Question 3:** hoevel studiefinancier geboortedatum studie

Some terms appear in only one question in the dataset, other terms appear in multiple questions. The total number of questions in the dataset is 3. With this we can calculate the IDF (Inverse Document Frequency) for all terms, by taking the logarithm of the total number of questions divided by the amount of questions in the dataset the term appears in. The IDF can be found in table 4.2.

Term	IDF
65	$log(\frac{3}{1}) \approx 0.48$
aow	$log(\frac{3}{1}) \approx 0.48$
geboortedatum	$log(\frac{3}{1}) \approx 0.48$
hoevel	$log(\frac{3}{1}) \approx 0.48$
jar	$log(\frac{3}{1}) \approx 0.48$
leeftijd	$log(\frac{3}{2}) \approx 0.18$
pensioen	$log(\frac{3}{1}) \approx 0.48$
pensioenleeftijd	$log(\frac{3}{0})$set to 0
studie	$log(\frac{3}{1}) \approx 0.48$
studiefinancier	$log(\frac{3}{1}) \approx 0.48$
wanner	$log(\frac{3}{1}) \approx 0.48$

TABLE 4.2: Inverse document frequencies for the questions in the dataset.

To calculate the TF (Term Frequency), we count how many times a term appears in a question. To normalize this term count we divide the term count by the maximum term count in that question. In this example none of the questions have a term that appears more than once, so no normalization is needed. The term frequencies can be found in table 4.3.

	Question	Question 1	Question 2	Question 3
65	0	1	0	0
aow	0	0	1	0
geboortedatum	1	0	0	1
hoevel	0	0	0	1
jar	0	1	0	0
leeftijd	1	1	1	0
pensioen	1	1	0	0
pensioenleeftijd	1	0	0	0
studie	0	0	0	1
studiefinancier	0	0	0	1
wanner	0	1	0	0

TABLE 4.3: The document-term matrix with term frequencies.

We now have the TF and the IDF scores and can calculate the TF-IDF score by multiplying the TF scores with the IDF values belonging to each term. Doing this we obtain the document-term matrix in table 4.4.

	Question	Question 1	Question 2	Question 3
65	0	0.48	0	0
aow	0	0	0.48	0
geboortedatum	0.48	0	0	0.48
hoevel	0	0	0	0.48
jar	0	0.48	0	0
leeftijd	0.18	0.18	0.18	0
pensioen	0.48	0.48	0	0
pensioenleeftijd	0	0	0	0
studie	0	0	0	0.48
studiefinancier	0	0	0	0.48
wanner	0	0.48	0	0

TABLE 4.4: The document-term matrix with TF-IDF scores.

4.4 Measuring similarity

In the previous section we showed how to calculate a TF-IDF vector for a question given a certain dataset. Many measures exist to compare these question vectors to each other. We use cosine similarity as this is an often used measure and it works well for scores between 0 and 1. Also, cosine similarity corrects for document length (or in our case question length) whereas many other measures do not keep this into account.

We revisit the example from the previous section. To calculate the cosine similarity between the new question and the questions in the dataset we first need to calculate the length of each question vector:

$$\begin{aligned}\text{Length of input question vector} &= \sqrt{0.48^2 + 0.18^2 + 0.48^2} = 0.70\\ \text{Length of question 1 vector} &= 0.98\\ \text{Length of question 2 vector} &= 0.51\\ \text{Length of question 3 vector} &= 0.96\end{aligned}$$

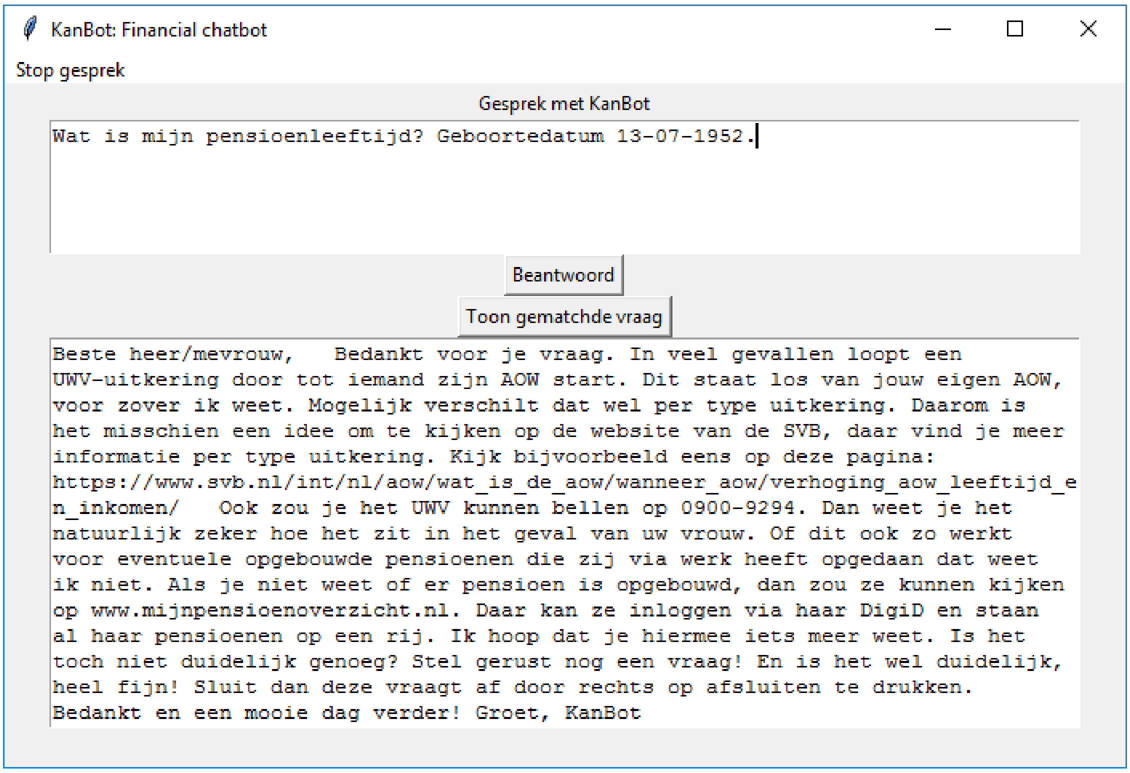
The cosine similarity between two question vectors is defined as taking the dot product of the two question vectors and dividing this by the lengths of the respective vectors:

$$\text{CosSim(Q,Q1)} = \frac{0 \cdot 0.48 + 0 \cdot 0 + 0.48 \cdot 0 + \ldots + 0 \cdot 0 + 0 \cdot 0 + 0 \cdot 0.48}{0.70 \cdot 0.98} = 0.38$$

$$\text{CosSim(Q,Q2)} = 0.09$$

$$\text{CosSim(Q,Q3)} = 0.34$$

The cosine similarity between the input question and question 1 from the dataset is the highest, therefore question 1 is chosen as the best match to the new question. The second best match is question 3 and the third best match is question 2. In the chatbot model, the answer to question 1 would now be returned as an answer to the question. However, we do not have exactly these three questions in our dataset but a set of 3153 other financial questions. The actual answer returned to the question using our vector space model can be seen in the chat screen in figure 4.8.

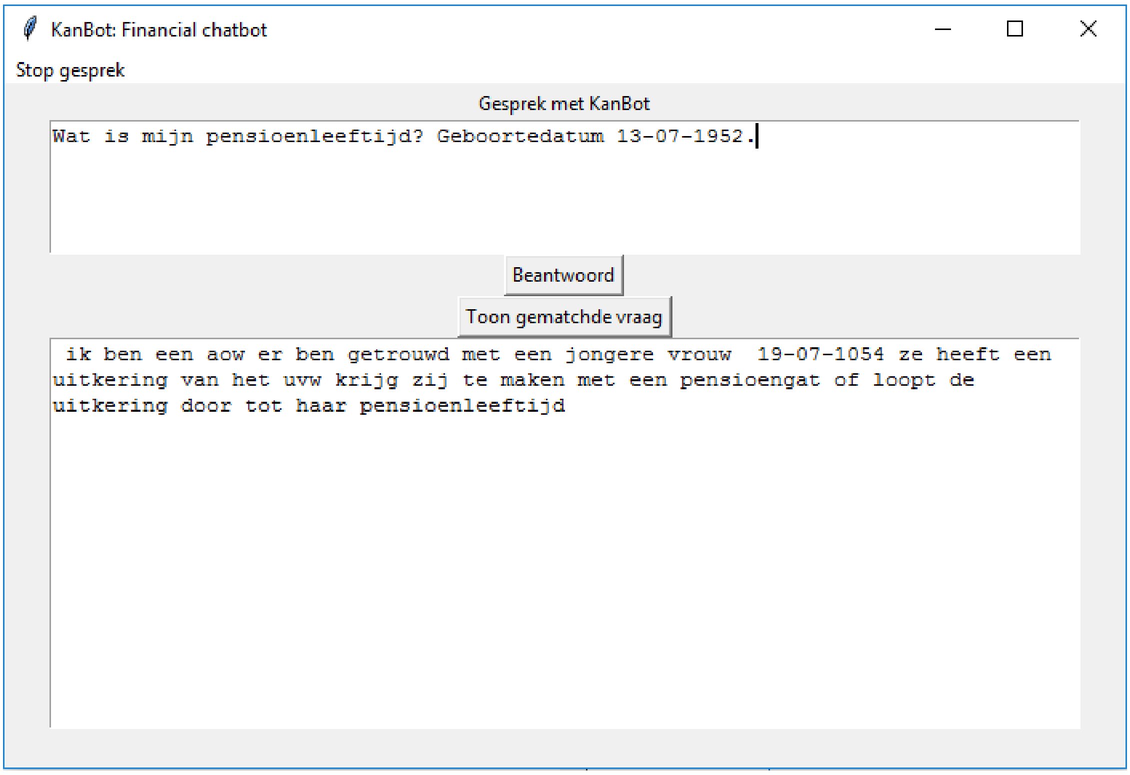


```
Wat is mijn pensioenleeftijd? Geboortedatum 13-07-1952.
```

```
Beste heer/mevrouw,   Bedankt voor je vraag. In veel gevallen loopt een
UWV-uitkering door tot iemand zijn AOW start. Dit staat los van jouw eigen AOW,
voor zover ik weet. Mogelijk verschilt dat wel per type uitkering. Daarom is
het misschien een idee om te kijken op de website van de SVB, daar vind je meer
informatie per type uitkering. Kijk bijvoorbeeld eens op deze pagina:
https://www.svb.nl/int/nl/aow/wat_is_de_aow/wanneer_aow/verhoging_aow_leeftijd_e
n_inkomen/   Ook zou je het UWV kunnen bellen op 0900-9294. Dan weet je het
natuurlijk zeker hoe het zit in het geval van uw vrouw. Of dit ook zo werkt
voor eventuele opgebouwde pensioenen die zij via werk heeft opgedaan dat weet
ik niet. Als je niet weet of er pensioen is opgebouwd, dan zou ze kunnen kijken
op www.mijnpensioenoverzicht.nl. Daar kan ze inloggen via haar DigiD en staan
al haar pensioenen op een rij. Ik hoop dat je hiermee iets meer weet. Is het
toch niet duidelijk genoeg? Stel gerust nog een vraag! En is het wel duidelijk,
heel fijn! Sluit dan deze vraagt af door rechts op afsluiten te drukken.
Bedankt en een mooie dag verder! Groet, KanBot
```

FIGURE 4.8: The chatbot screen with an answer to a newly typed question.

This answer is not a correct answer to the question, and therefore we are interested to see what the question was that our question is matched to. The matched question can be seen in figure 4.9.



```
Wat is mijn pensioenleeftijd? Geboortedatum 13-07-1952.
```

```
 ik ben een aow er ben getrouwd met een jongere vrouw  19-07-1054 ze heeft een
uitkering van het uvw krijg zij te maken met een pensioengat of loopt de
uitkering door tot haar pensioenleeftijd
```

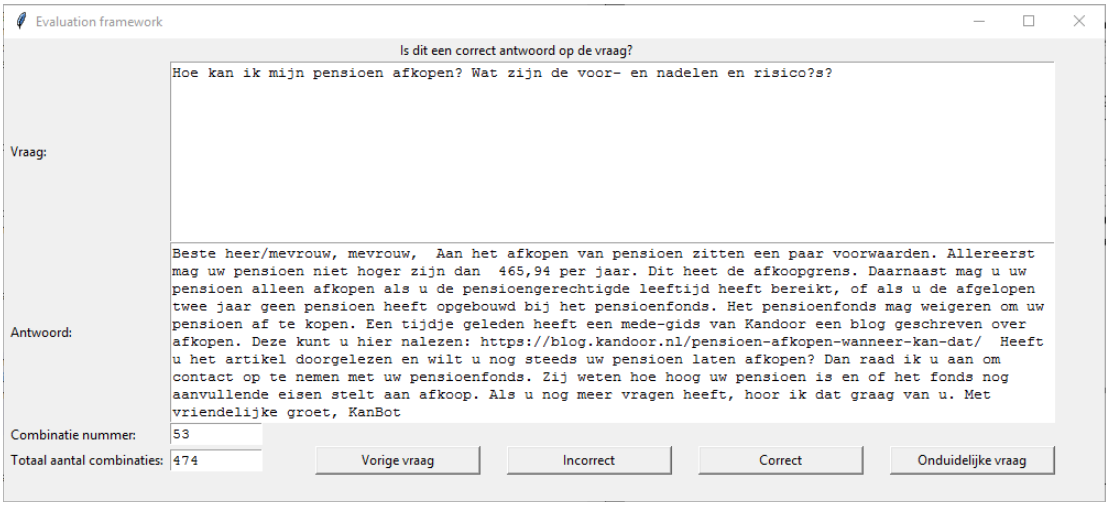
FIGURE 4.9: The chatbot screen showing the question that is matched to a newly typed question.

Comparing the two questions, we do understand that a machine that has no actual understanding of the meaning of the question would find these two questions quite similar. After all, the questions are both about pension and they both contain the words "pensioenleeftijd", "pensioen" en "leeftijd" after NLP, which is 3 of the 4 words in the incoming question after NLP. The difference is that the first question is about "when the pension age is" and the second question is about "pension gap" but the TF-IDF scores could not distinguish this. Machine learning models might be better at making this distinction.

4.5 Results

To apply the vector space model we used our entire question-answer dataset as the training set, and a new set of questions from Kandoor as the test set. We have 3153 questions to match to, and answers to return, and the test set contains 400 new questions. Special about this test set is that the questions are labelled by category and therefore we can see how well the model scores on questions in 14 different financial categories.

Evaluation is done manually in our evaluation framework, which can be seen in figure 4.10. The evaluation framework shows a question from our test set and the answer selected for it. We now have 3 options in evaluating this question-answer pair and one button to go back to the previous question. The 3 options are a correct answer, a wrong answer or the question being unclear. This last option is added because we did not clean the test set before evaluation and therefore it still contains questions that cannot be answered. For example, the question "De 1 weeks Euribor rente is in 1 werkdag fors gestegen. Was deze rente afgelopen vrijdag nog 0,182%, vandaag noteert deze rente 0,242%. Daarmee stijgt deze rente in 1 dag met 33%." which cannot be answered because there is no actual question but a stated fact.



Is dit een correct antwoord op de vraag?

Vraag: Hoe kan ik mijn pensioen afkopen? Wat zijn de voor- en nadelen en risico?s?

Antwoord: Beste heer/mevrouw, mevrouw, Aan het afkopen van pensioen zitten een paar voorwaarden. Allereerst mag uw pensioen niet hoger zijn dan 465,94 per jaar. Dit heet de afkoopgrens. Daarnaast mag u uw pensioen alleen afkopen als u de pensioengerechtigde leeftijd heeft bereikt, of als u de afgelopen twee jaar geen pensioen heeft opgebouwd bij het pensioenfonds. Het pensioenfonds mag weigeren om uw pensioen af te kopen. Een tijdje geleden heeft een mede-gids van Kandoor een blog geschreven over afkopen. Deze kunt u hier nalezen: https://blog.kandoor.nl/pensioen-afkopen-wanneer-kan-dat/ Heeft u het artikel doorgelezen en wilt u nog steeds uw pensioen laten afkopen? Dan raad ik u aan om contact op te nemen met uw pensioenfonds. Zij weten hoe hoog uw pensioen is en of het fonds nog aanvullende eisen stelt aan afkoop. Als u nog meer vragen heeft, hoor ik dat graag van u. Met vriendelijke groet, KanBot

Combinatie nummer: 53

Totaal aantal combinaties: 474

FIGURE 4.10: The evaluation framework.

Out of the 400 questions, 145 questions (36%) were unclear questions and were therefore removed from the results. Out of the remaining 255 questions, 171 were answered incorrectly and 84 (33%) were answered correctly. Thus the overall result of the statistical chatbot model is that the chatbot can answer 33% of questions correctly. The percentage of questions answered correctly in the 14 financial categories can be found in table 4.5. These results should be interpreted with caution, because many financial categories have very few samples which might not be representative for this category. As shown, the category "Buitenland wonen/werken" has the highest percentage of correctly answered questions, but this category only contains 6 questions. More reliable is that the category "Hypotheek" has a quite high score with 52% of questions answered correctly as there are 33 samples in this category. The lowest score is for the category "ZZP & ondernemen" with only 11% of questions answered correctly.

Category	Number of samples	Correctly answered
Pensioen werkgever	76	32%
Pensioen overheid (AOW)	34	38%
Hypotheek	33	52%
Impact werksituatie	24	17 %
Verzekeringen	18	22%
Lenen & schulden	10	40 %
Belasting	10	30%
Pensioen individueel	9	33%
ZZP & ondernemen	9	11%
Beleggen	7	29%
Sparen & vermogen	7	29%
Huisvesting	7	29%
Buitenland wonen/werken	6	67%
Wet & recht	5	20%

TABLE 4.5: Results of the vector space model per financial category.

Chapter 5

Machine learning model

In this chapter we apply multiple machine learning models to classify whether two questions are the same or not. The machine learning models that are compared are XGBoost, Random Forest, Neural Networks, SVM and logistic regression.

The chapter starts with a description of the dataset in section 5.1. Then we apply NLP techniques to our dataset in section 5.2 and create features from the textual data in section 5.3. Furthermore, in section 5.4 we use these features in our machine learning models with which we classify whether two questions match or not. Finally, in section 5.5 we show the results of our machine learning models.

5.1 Data

5.1.1 Data creation

The data used in the machine learning model is created with the data used in the vector space model in chapter 4. This dataset is cleaned and explored in sections 4.1.1 and 4.1.2, respectively. To be able to train a machine learning model, we needed labeled data. Data with which we could learn a computer whether two (financial) questions are the same or not. A question can be asked in many different ways, and we consider two questions to be the 'same' when they ask for the same financial information, regardless of the personal information given. For example, the question "Wanneer krijg ik AOW? Ik word morgen 65." and the question "Op welke leeftijd gaat mijn staatspensioen in? Ik ben geboren op 17-07-1952." are considered the same because both questions ask for the state pension age. To be able to learn a computer whether two questions are the same or not we create a question-to-question (Q2Q) dataset, as shown in table 5.1.

ID Q1	Question 1	ID Q2	Question 2	Match
1	Wanneer krijg ik...	33	Op welke leeftijd...	1
1	Wanneer krijg ik...	2987	Bij welke pensioenfondsen...	0
1	Wanneer krijg ik...	3	Wanneer ga ik...	1
1	Wanneer krijg ik...	435	Wat is mijn...	0
1	Wanneer krijg ik...	875	Is prinsjesdag op...	0
1	Wanneer krijg ik...	1542	Wordt mijn AOW...	0
2	Krijg ik studie...	16	Is studiefinanciering ook...	1
...	...	...	...	...

TABLE 5.1: The framework of the Q2Q dataset.

Ideally the Q2Q dataset is a dataset consisting of all combinations of the questions in the Kandoor dataset. However, this dataset contains 3153 questions (after cleaning) and therefore all combinations would give us $\frac{3153 \cdot 3152}{2} \approx 5$ million combinations of questions. Since we need to manually label whether two questions match (are the same) or not, we created a smaller set of questions to label. In creating this dataset we tried to make sure that the class imbalance (between matching and non-matching questions) was as small as possible. If we randomly picked two questions, then the chance that they match is small. Therefore, to increase the chance of matching we used the vector space model created in chapter 4 to match the questions. To not overly focus on matching questions with similar words we also added some random question combinations to the Q2Q dataset. Finally, the Q2Q dataset was created by taking each question, its 4 closest matches according to the cosine similarity measure in the vector space model, and 2 random questions. This means that each question was in the dataset at least 6 times, and in total there were $6 \cdot 3153 = 18918$ combinations. However, there were also some duplicates in this dataset. For example, if question 1 was matched with question 3 by cosine similarity but also randomly, or if question 1 was matched to question 3 by cosine similarity but question 3 was also matched to question 1 by cosine similarity. These duplicates were removed and then we were left with 16050 question combinations. All these combinations were manually labelled on whether they match or not.

5.1.2 Data exploration

The Q2Q dataset is created from the question-answer dataset already explored in section 4.1.2. Therefore, we need not explore the questions themselves again but concentrate on exploring the data with regard to the matching (or non-matching) of two questions. The first observation after creating the Q2Q dataset, is that only 15% of the question combinations match. This class imbalance needs to be taken into account when modelling, as predicting all question combinations to not match would lead to a model accuracy of 85%. Clearly, accuracy is not a good measure here.

Word counts and sentences counts

We studied the influence of question length on the matching of questions to find out if longer questions match more or less often to other questions than shorter questions. When it is the case that longer questions match more or less often, then question length would be a good feature in our machine learning models.

We studied the relation and found no significant effect of question length on the question matching or not. Therefore we conclude that question length is not a good predictor for whether a question matches or not.

We also researched the most used words in our dataset and their distribution over matching and non-matching questions. Gathering all words in questions that match (do not match), it turns out that 3.2% (2.3%) of these words are the word "pensioen". We find that the top 3 terms: "pensioen", "jaar" and "aow" appear relatively more often in matching questions than in non-matching questions. This can happen because questions involving these terms occur often in our dataset and therefore also match with each other often. From this result we conclude that the appearance of certain most used words in questions could be good predictors for whether questions match or not.

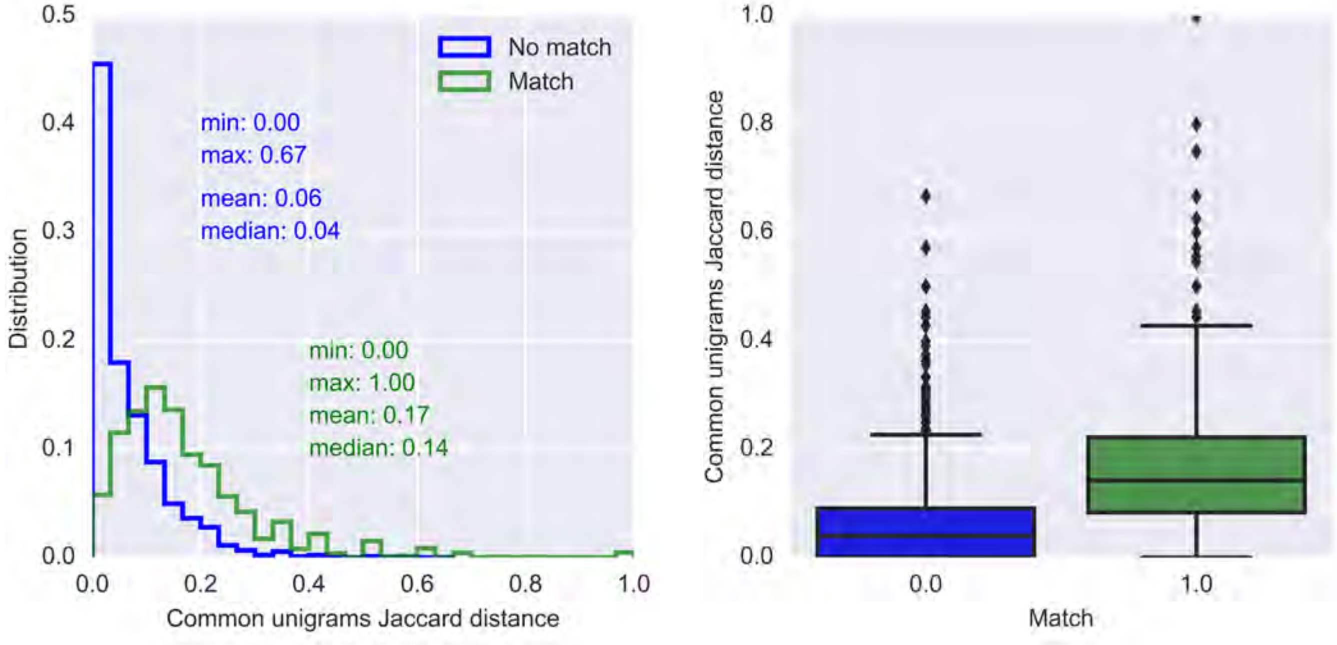
N-grams

In this section we look at N-grams in the matching and non-matching question combinations. When looking at common unigrams in the questions, we are essentially looking at words (after stop-word removal) that occur in both questions. The ratio of common unigrams in the question combinations in our dataset can be seen in the histogram in figure 5.1. In this figure, we see that matching question pairs have a distribution that is less right skewed than non-matching questions. Looking at the box plot in the same figure shows that matching questions are statistically likely to have a higher common unigram ratio. We cannot say this with certainty though as the interquartile ranges of the box plots do overlap a bit.

To see if we can improve the prediction of questions matching or not using common unigrams we applied TF-IDF weighting to the unigrams. This means that we value words that appear in both questions, but do not appear a lot in the dataset, as more important than words that appear a lot in the dataset. Thus we care more about uncommon words than about frequently appearing words. Comparing the histogram in figure 5.2 to the histogram in figure 5.1, we see that it seems like the separation between matching and non-matching questions actually became less clear with the TF-IDF weighting. The box plot in figure 5.2 still shows that it is statistically likely that the common unigrams ratio of matching questions is higher than the common unigrams ratio of non-matching questions. However, also here the separation between matching and non-matching questions seems smaller than without the TF-IDF weighting.

To further investigate the prediction of questions matching or not using unigrams and TF-IDF weighted unigrams we calculated the AUC score for both features. AUC stands for Area Under the Curve and is used in classification analysis to determine which of the used features or models predicts the classes best. When predicting whether questions match in our dataset, AUC is a better measure than accuracy because of the large class imbalance. The AUC is the expectation that a uniformly drawn random positive is ranked above an (also uniformly drawn) random negative. In our dataset this area can show the predictive power of our (TF-IDF weighted) unigram feature on whether questions match. The AUC for our non-weighted unigram feature is 0.81 and the AUC for our TF-IDF weighted unigram feature is 0.76 meaning that, as expected from the figures, the non-weighted unigram feature actually has a better predictive power than the TF-IDF weighted unigram feature. Regardless of this result we do still think that the TF-IDF weighted unigram feature can provide extra information in a prediction model next to the non-weighted unigram feature.

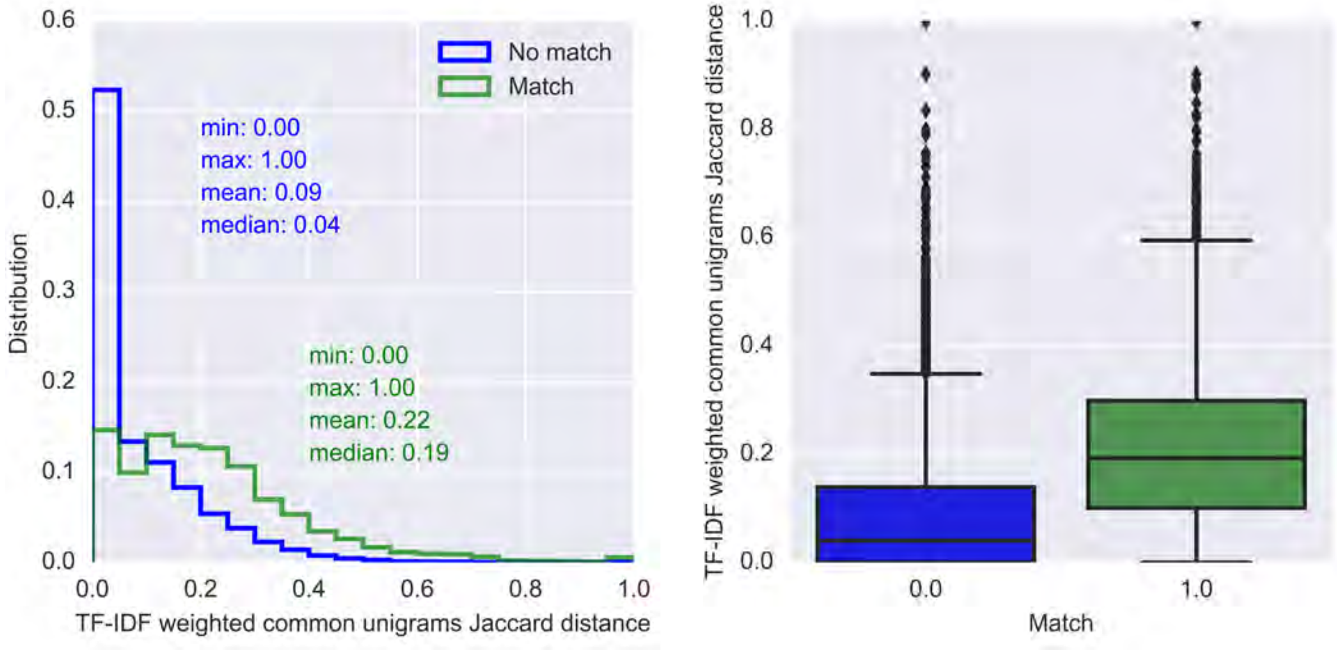
We also studied the common bigrams in our question combinations, but common bigrams occur so little that we could not draw any valid statistical conclusions for this feature. It might be the case that for the Dutch language bigrams are less important predictors of similarity than for, for example, the English language because of compound words. In English "pension fund" would be an often occurring bigram in our dataset, but in Dutch this just becomes one word "pensioenfonds" which is a unigram.



(A) Histogram

(B) Box plot

FIGURE 5.1: Charts displaying the distribution of the common unigrams Jaccard distance for matching and non-matching questions.



(A) Histogram

(B) Box plot

FIGURE 5.2: Charts displaying the distribution of the TF-IDF weighted common unigrams Jaccard distance for matching and non-matching questions.

Conclusion

In the exploration of the Q2Q dataset we have found that the amount of words or sentences in a question combination does not have an influence on the chance of the combination matching. We have also found that certain most used words in the dataset occur relatively more often in matching questions than in non-matching questions. Lastly, we found that common unigrams and TF-IDF weighted common unigrams are likely to be good features for predicting whether questions match or not.

5.2 Natural Language Processing

The data was prepared for modelling by applying the same NLP pipeline used in the vector space model to the questions in our dataset. A description of the NLP pipeline can be found in section 4.2.2. Figure 4.7 shows an example of a prepared question.

After applying the techniques in the NLP pipeline we added one extra NLP step. In this step synonyms of words are added to a question to make sure that two questions about the same topic, but containing different words, match. For example, synonyms of the Dutch word "pensioenoverzicht" are "upo" and "pensioenbrief". If a question contains either the word "upo" or "pensioenbrief", the word "pensioenoverzicht" will be added to this question to make it match to other questions containing one of these three words. We do not remove the original word "upo" in the question, as the meanings of synonyms might differ slightly and two questions containing "upo" might match slightly better than a question containing "upo" and a question containing "pensioenoverzicht". The list with synonyms was manually constructed and contains 62 synonym groups. This list can be found in appendix A.3.

As a result of applying these NLP techniques our Q2Q dataset is now build up as shown in table 5.2.

ID Q1	Question 1	Prep Q1	ID Q2	Question 2	Prep Q2	Match
1	Wanneer...	Wanneer...	33	Op...	leeftijd...	1
1	Wanneer...	Wanneer...	2987	Bij...	pensioenfondsen...	0
1	Wanneer...	Wanneer...	3	Wanneer...	Wanneer...	1
1	Wanneer...	Wanneer...	435	Wat...	pensioenleeftijd...	0
1	Wanneer...	Wanneer...	875	Is...	prinsjesdag...	0
1	Wanneer...	Wanneer...	1542	Wordt...	aow...	0
2	Krijg...	Krijg...	16	Is...	studiefinanciering...	1
...	...	...	...	...	...	...

TABLE 5.2: The framework of the Q2Q dataset including the questions prepared with NLP techniques.

5.3 Feature engineering

In this section, we create the features used in the machine learning models. These features are partly based on features that were successfully used in the cases in our literature study, partly based on our data exploration and partly based on new ideas while constructing the features. In our literature study we found that TF-IDF transformations, document lengths and looking at common and different words are good features. Our data exploration showed that common words (unigrams) are likely to be a good feature, even without TF-IDF weighting and that certain often used words appear more in matching questions than in non-matching questions.

5.3.1 Nouns

Nouns are words that identify a person, place, thing, or idea. For example, in our dataset nouns would be "AOW-leeftijd", "pensioengat" and "studie". Nouns are very important in conveying the meaning of a sentence. Therefore we extracted all nouns from the questions and created a feature that counts how many nouns overlap between the questions.

5.3.2 Lengths

During the exploration of the data we found that the length of a question does not have a significant influence on the probability that it matches another question. Therefore we do not include the length of the original question as a feature. The length of a prepared question however is a measure for how many "important" characters or words there are in a question, and we do include this feature.

We have one feature that describes the length of the first question (after applying the NLP pipeline) in number of characters (including white space) and one feature that describes this length in the number of words. We have the same features for the second question in the pair, giving us four features. Then we also have two difference features which describe the length difference between the two questions in number of characters and number of words, respectively.

5.3.3 Sequence matching

When comparing two questions, not only matching words but also longer sequences are important. For instance, "waar heb ik ergens pensioen opgebouwd?" and "heb ik pensioen opgebouwd?" both contain the exact sequence " pensioen opgebouwd?". The Ratcliff-Obershelp pattern-matching algorithm [63] can be used to calculate a sequence matching ratio between two text documents. This algorithm was introduced in 1983 as a measure to improve educational software and automated answer checking.

The Ratcliff-Obershelp matching ratio returns a value between 0 and 1, where 0 is no match and 1 is a perfect match of the two compared texts. The algorithm is expressed as

$$D_{RO} = \frac{2 \cdot K_m}{|S_1| + |S_2|}, \tag{5.1}$$

where D_{RO} is the matching ratio, K_m is the number of matching characters and S_1 and S_2 are the lengths of the compared texts.

The number of matching characters K_m is found in steps. First, the longest common substring between the two texts is found. This is called the anchor. Then the parts on both sides of the anchor are examined as new strings. This process is repeated until all characters are examined. K_m is the length of the anchor plus the lengths of other matched subsequences.

To clarify the working of the algorithm, consider again the questions "waar heb ik pensioen opgebouwd?" and "heb ik ergens pensioen opgebouwd?". The length of the first question is 31 characters (including white space) and the length of the second sequence is 33 characters. The longest matching substring is " pensioen opgebouwd?", this is the anchor with 20 characters. To the right of the anchor there are no more characters. To the left of the anchor we find the matching substring " heb ik ", with 8 characters. On the right of this substring we find "ergens" in the second question but there is nothing there in the first question. To the left of the substring we find "waar" in the first question, but nothing to match it to in the second question. Now we have examined all characters, and are done. The total number of matching characters K_m is 28 and therefore the matching ratio is $\frac{2\cdot(20+8)}{(31+33)} = 0.875$.

The Ratcliff-Obershelp matching ratio calculated for the question pairs is included in our models as a feature.

5.3.4 Fuzzy sequence matching

In the previous section, we compared two questions using sequence matching. Often questions are similar but contain spelling mistakes, have a different word order or one is a substring of the other. In the Ratcliff-Obershelp algorithm this would lead to a lower matching ratio, but in fuzzy sequence matching these alterations can be overcome.

Our fuzzy sequence matching features make use of the Levenshtein distance to calculate the differences between two questions. The Levenshtein distance is the minimum number of edits (insertion, deletion or substitution of one character) needed to change one question into the other. Seven features are created using the Levenshtein distance:

1. The Q-ratio. The simple ratio of how many characters to edit in the first question to get the second one, divided by how many characters the first question has.
2. The token sort ratio. The questions are broken down into words and then these words are alphabetically ordered. After ordering the Q-ratio is calculated.
3. The token set ratio. The questions are again broken down into words, and then the separate words are split into an intersection (words that are the same in both questions) and a remainder. These sets are used to build a comparison string to which the Q-ratio is applied. This ratio gives good results when sorting does not help because one of the questions has a lot more words. For example, when comparing "pensioen van abp" and "krijg ik pensioen bij abp", when we sort the strings we get "abp pensioen van" and "abp bij ik krijg pensioen" where "abp" and "pensioen" are far apart. In the token set ratio these words are put next to each other to get a better match. The token set ratio gives us the strings "abp pensioen van" and "abp pensioen bij ik krijg", with first the two matching words ("abp" and "pensioen") in alphabetical order and then the rest of the words in alphabetical order.
4. The partial ratio. This feature is the same as the Q-ratio except that it recognizes when the first question is a substring of the second question and then it returns a full match. For example, the words "pensioen" and "pensioenfonds" would get a 100% match with the partial ratio.

5. The partial token sort ratio. This uses the same algorithm as the token sort ratio, but after sorting it uses the partial ratio instead of the Q-ratio.
6. The partial token set ratio. This uses the same algorithm as the token set ratio, but on the sets of tokens it uses the partial ratio instead of the Q-ratio.
7. The W-ratio. This is a weighted ratio of all previous ratio's. The idea is to get the highest score among all calculated ratio's. Some scaling is also used here when one question is much longer than the other.

5.3.5 Word matching

In section 5.1 we explored the data, and found that the amount of common unigrams seem to be a good predictor of whether two questions match. Therefore we have included a feature which computes a ratio of matching words. The ratio is calculated by

$$R = \frac{M_1 + M_2}{|W_1| + |W_2|}, \tag{5.2}$$

where R is the ratio, M_1 and M_2 are the number of words in question 1 (2) that are also in question 2 (1) and $|W_1|$ and $|W_2|$ are the lengths of the questions in the number of words.

Now we consider the questions "over pensioen, waar heb ik pensioen opgebouwd?" and "heb ik ergens pensioen opgebouwd?". Question 1 has 7 words and question 2 has 5 words. The words in question 1 that are also in question 2 are [pensioen, heb, ik, pensioen, opgebouwd], 5 in total. The words in question 2 that are also in question 1 are [heb, ik, pensioen, opgebouwd], 4 in total. Therefore the ratio word match ratio is 0.75.

This word match ratio is calculated for each question pair and included as a feature in our models.

5.3.6 TF-IDF

TF-IDF was already used in the vector space model, and for this model we also create some features with it. We trained the TF-IDF vectorizer on all unique questions and then fed it one question at a time and calculated scores. The TF-IDF transformation of a question is a vector with a TF-IDF score for each unique word and bigram in the question. The sum of this vector, mean of this vector and length of this vector are used as features for question 1 and question 2 separately. From these 6 features we also calculate the difference between the sum, mean and length of the TF-IDF transformations of question 1 and 2, generating three more features.

The sum or mean of a TF-IDF vector in itself does not have a lot of meaning, but more rare words get higher IDF values and therefore a question with many rare words might get a much higher sum or mean. A question with many rare words will not match many questions, and therefore these features might help in predicting this. The length of a TF-IDF vector gives the number of unique words added with the number of unique bigrams in the question.

5.3.7 Similarity scores

In the vector space model, we used the cosine similarity between two questions to predict if they were a match. In the machine learning model cosine similarity is used as a feature. More information about cosine similarity can be found in section 4.4.

Another similarity measure is the Jaccard similarity. We introduced the Jaccard index in section 4.1.2 and the same formula is used here. The Jaccard similarity is the number of words used in both questions divided by the total number of words. This is different from the word match ratio as it does not take into account words that are used twice in one sentence.

5.3.8 Common and different words

In the data exploration section we found that many of the most used words were used more often in matching questions than in non-matching questions. This fact can help us predict whether two questions have a high chance of matching.

To create the common words features we have selected the 200 words that occur in both questions, most often. These are not the most often used words, but words that are most often in common between the two questions. Now for each question pair we find the words in common and, if these words were in the top 200, we fill in their counts in the 200 word features.

The different words features are created similarly, but instead of finding the 200 words that occur in both questions, we use the 200 words that occur most often in only one of the questions (the set difference). Now for each question pair we find the words that are in only one of the questions and fill in their counts in the 200 word features.

Up to this point all features were created using the questions prepared with NLP techniques. However for these common and different words we have also created 100 common and 100 different word features using the original questions. This was done to retrieve some of the information that might have been lost in preprocessing.

5.3.9 Category

The questions in our dataset are all financial questions, and they can be subdivided in 14 financial categories as shown in section 4.5. In a previous project for Kandoor, a classifier was built to appoint one of these 14 financial categories to a new question. We have helped to improve this classifier and applied it on the questions in our dataset. These categories give us a new binary feature, namely whether two questions are predicted to fall into the same financial category or not.

5.3.10 Word embedding

Word embedding is mapping words or sentences to vectors containing real numbers. In section 2.2.3, we already explained that these vector representations can hold important linguistic information. A very popular word embedding method is word2vec [52]. Word2vec was created by Google and consists of neural networks which are trained to build and understand the context of words. The input for a word2vec model is a text corpus and the output is a vector space, where each word from the corpus has been assigned a vector. These word vectors are placed in the vector space in such a way that words that occur in similar contexts are placed close to each other in the space.

We trained a word2vec model on the Kandoor dataset to see which words occur in similar contexts and to be able to use this model as a feature in our classifier. In table 5.3, we have shown a few terms and the 5 words most often used in the same context as these terms.

Term	1st most similar	2nd	3rd	4th	5th
Pension	Pension fund	accrued	employer	where	contact
AOW	svb	since	receives	partner	65
65	year	since	age	receives	the Netherlands

TABLE 5.3: A table depicting a word and the words most often used in the same context as this word. In this case the five words which are used in the most similar context are shown.

From table 5.3, we see that the word2vec model works well on our dataset, as we would expect words like ’pension’, ’pension fund’, ’accrued’ and ’employer’ to be used in the same context.

This word2vec model produces a vocabulary in which each word is represented as an 300-dimensional array. To visualize the similarity and dissimilarity between words and their contexts, we compressed these 300-dimensional feature vectors to two-dimensional coordinate pairs using Principal Component Analysis (PCA). This was done with the idea to keep similar words together on the x,y-plane, and maximize the distance between words used in different contexts. The result of this vector compression is shown in figure 5.3. Unfortunately, we cannot draw many conclusions from this figure, as we can see no logical clustering of words in different regions of the plot. This is probably due to only using words which occurred at least 400 times in our dataset (otherwise the plot would be enormous) and due to the information loss from compressing a 300-dimensional vector space to a 2-dimensional vector space.

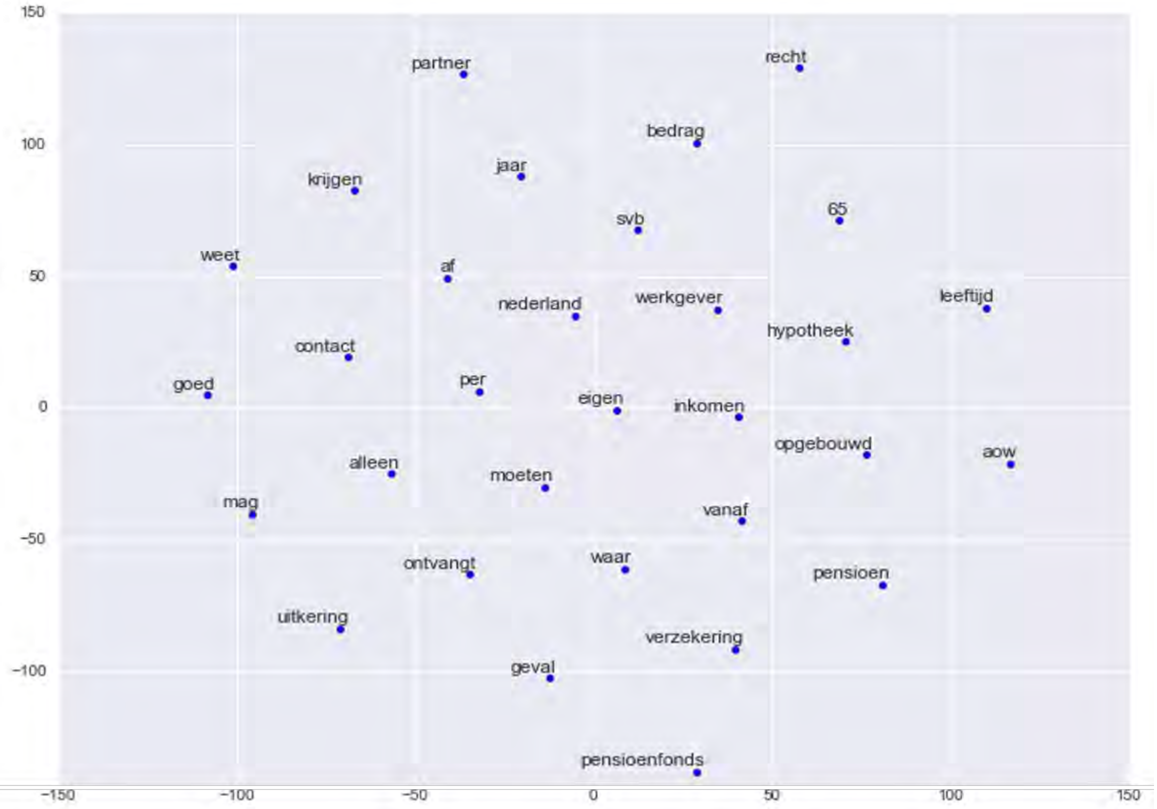


FIGURE 5.3: A visualization of the word2vec model. Only words used at least 400 times in the dataset are displayed.

To create the word2vec features we trained a word2vec model on all unique questions in our dataset with 300 dimensions and used all words that were used at least 40 times. Each word in a question has values for each of these 300 dimensions. To make it more tangible you could image that the word "woman" has semantic features like "femininity", "human" and "powerful".

We created 300 features for both questions (600 in total), by averaging the scores for all words in the question for each dimension. For example, if a question contained both "woman" and "man" it would have a high average score for "human". The difference in each of these dimensions for the two questions might also be of importance and therefore we also generated 300 difference features.

5.4 Modelling

To predict whether questions are the same, we built multiple machine learning models using the features described in the previous section. The machine learning models we use are XGBoost, Random Forest, Support Vector Machines, Neural Networks and logistic regression. XGBoost was selected as it is an upcoming method with great results in applied machine learning challenges like Kaggle[1] competitions. Random Forest, Support Vector Machines and Neural Networks were selected as these are established methods proven to give good results on many machine learning problems. In particular, Random Forest is easy to tune and to interpret, Support Vector Machines work well with a large number of features and Neural Networks can give great results if the dataset turns out to be large enough. Finally, we also apply logistic regression to the dataset as this algorithm gave the best results in Watson [27]. The theory behind all of these machine learning models is explained in section 2.5. In this section we explain how we applied these models on our dataset and which features were important in the models.

5.4.1 Feature selection

Feature selection is the process of selecting which features to use in a model. Feature selection can be applied to simplify the model, to shorten the training time of a model, to avoid the curse of dimensionality and to reduce model variance. In our case especially shortening the training time (a user expects a fast answer from a chatbot) and reducing model variance (reducing overfitting) are important.

We apply two different feature selection methods, namely the chi-squared test and the Random Forest model. We chose these methods as applying them costs little time and the results are easily interpretable. The chi-squared test is a statistical test which determines whether there is a dependency between two variables. For each feature it determines whether it has a significant influence on the outcome variable (match or no match). When using the Random Forest model for feature selection, the best performing number of features is found by eliminating features based on their feature importance (Gini importance) in the model. The Gini importance of a feature is the total decrease in node impurity averaged over all trees when removing this feature.

In applying the chi-squared test to our features we could not test the dependence of the Word2Vec features as they have negative values in them. After chi-squared

---

[1] https://www.kaggle.com/

feature selection we were left with 362 significant features of our original 1528 features. The features with the smallest P-values (most significant features) were the length features and the fuzzy sequence matching features.

When using the Random Forest model for feature selection, the best results were reached when using only features with a minimum Gini importance of 0.01370. This led to the model using only the 14 features shown in figure 5.5.

We are now left with three sets of features: the full set (1528 features), the set selected with the chi-squared test (362 features) and the set selected with Random Forest (14 features). These three feature sets were all tested in every model and results are given for the set with the best performance in the respective model.

5.4.2 Downsampling

As described the Q2Q dataset consists of 16050 question pairs of which about 15% is a match. This means that there is a large class imbalance in our dataset. XGBoost models generally work well with datasets with class imbalances as their evaluation metric can be manually set. However other the standard programming packages for other models like Random Forest generally optimize for accuracy, which will lead to the model predicting almost everything to be negative. To overcome this problem we have downsampled the dataset. This was done by selecting all question-pairs that matched and just as many question-pairs (randomly sampled) that did not match. Now we have a smaller dataset but with an even class balance. For the XGBoost model we also applied the full dataset but the results were better with the downsampled dataset. For all other models we just applied the downsampled dataset.

5.4.3 XGBoost

XGBoost is a boosting algorithm that combines weak rules from simple machine learning models to form one strong model. The theory behind this method is explained in section 2.5.4. While applying XGBoost to a specific dataset many parameters can be tuned.

We now explain the most important parameter settings in our XGBoost model. These parameters were tuned by a grid searching algorithm that finds the optimal combination in a grid of parameter values. For this tuning 5-fold cross validation was used to ensure that the settings are not just optimal for one part of the dataset.

1. **Objective function**
   The objective function defines the loss function to be minimized. We use logistic regression for binary classification as the objective function because we only have 2 classes (matching and non-matching). This objective function returns the probability that two questions match (not in which class the question-pair belongs).

2. **Evaluation metric**
   The evaluation metric is the metric to be used on the validation set. We have used AUC, the Area Under the ROC Curve. Why this measure works well for our dataset is explained in section 5.5.

3. **Learning rate**
   The learning rate (also denoted as eta) makes the model more robust as it shrinks the feature weights for each step. This is done to prevent overfitting.

A higher learning rate leads to a more conservative model. Using our grid optimization algorithm the learning rate was set to 0.05 where the default value is 0.3.

4. **Minimum child weight**
   The minimum child weight is the minimum sum of all instances in a leaf node for which the algorithm will still partition further. If the sum of all instance weights in a leaf node is lower than this value, the building process of the tree will stop there. A higher minimum child weight leads to a more conservative model. Using our grid optimization algorithm the minimum child weight was set to 0.3 where the default value is 1.
5. **Maximum depth**
   The maximum depth specifies the maximum depth of the trees, meaning that it limits the number of nodes in a tree. A higher maximum depth leads to a more complex model and a higher chance of overfitting. Using our grid optimization algorithm the maximum depth was set to 6 which is also the default value.
6. **Subsample**
   The subsample ratio denotes the fraction of observations in the training set to be randomly sampled and used in each tree. A low subsample value makes the algorithm more conservative and prevents overfitting, but a too low ratio might lead to underfitting. Using our grid optimization algorithm the subsample ratio was set to 1 which is also the deafult value.
7. **Column sample by tree**
   Whereas the subsample ratio determines the fraction of observations to be used for each tree, the column sample by tree determines the fraction of features to be randomly sampled and used in each tree. Also here, a low subsample value makes the algorithm more conservative and prevents overfitting, but a too low ratio might lead to underfitting. Using our grid optimization algorithm the column sample by tree was set to 0.3 where the default value is 1. This means that 30% of the features is randomly sampled at each splitting point and out of these features the best one is selected for splitting.
8. **Scaling the positive weights**
   This parameter is used in cases where there is a high class imbalance to help the algorithm converge faster. As a rule of thumb the scale is set to the number of negative cases divided by the number of positive cases, which would be 5.7 for our full dataset. However the downsampled dataset made the performance a lot better and with this dataset the scale was set at the default value of 1 since there is no class imbalance.

In tree based ensemble methods, like XGBoost, each feature is seen as a potential feature to split on. This makes the algorithm quite robust to unimportant features, as these cannot discriminate between classes. A caveat is that the importance given to highly correlated variables might not be a good indicator of their actual performance as a splitting point might have been good for both variables. We expected that feature selection would not improve our XGBoost model, as the algorithm selects the features with the best splitting points. However, the set of features selected with the chi-squared test gave the best results and XGBoost used 340 of the 362 features in this model to reach its optimal AUC score. The top 25 best performing features and there F-scores can be seen in figure 5.4. The most important features are the sums of the TF-IDF vectors of the 2 questions separately, something we did not expect as these sums don't have a clear meaning. Another important feature is the cosine similarity which we did expect as we built an entire model (the Vector Space Model) on this feature.

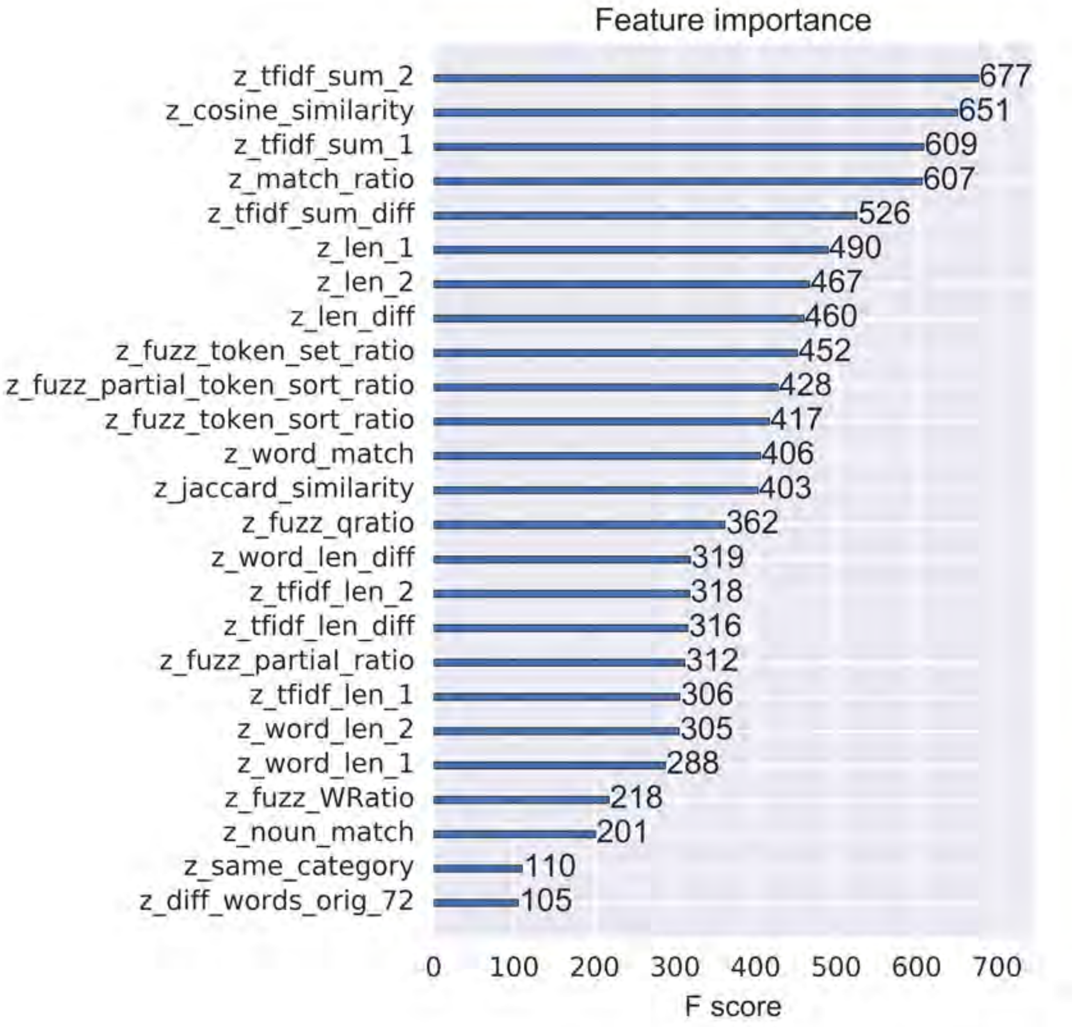


FIGURE 5.4: A bar chart displaying the feature importances in the best performing XGBoost model.

5.4.4 Random Forest

Random Forest combines decision trees to form a forest of trees which each have a vote in deciding which class an object falls in. The theory behind this method is explained in section 2.5.3. While applying Random Forest to a specific dataset many parameters can be tuned.

We now explain the most important parameter settings in our Random Forest model. These parameters were tuned by a grid searching algorithm that finds the optimal combination in a grid of parameter values. For this tuning 5-fold cross validation was used to ensure that the settings are not just optimal for one part of the dataset.

1. **Maximum number of features**
   The maximum number of features Random Forest can try at each split in each

tree. We have created around 1500 features for our dataset and this parameter specifies what randomly sampled fraction of these features can be at each splitting point. A higher maximum number of features can lead to better performing individual trees but also to more similarity of the trees which can reduce the overall performance of the forest as it eliminates some variation in the trees. Using our grid optimization algorithm the maximum number of features was set to the square root of the number of input features which is also the default value.

2. **Number of estimators**
   The number of estimators is the number of trees that a Random Forest grows. When there are more trees in the forest the maximum of the votes is a more robust measure. More trees always give better prediction results, but it makes the computation time a lot longer. We set the number of estimators to 10000 as this was the maximum value that our processor could handle. The default value is 10, and this value is way to small for our dataset.

3. **Minimum sample leaf size**
   The minimum sample leaf size is the minimum number of samples needed in an end node of a tree. When the leaf size is smaller the model has a higher chance of capturing noise in the end leaves. The default value for this parameter is 1, but this will catch a lot of noise. Using our grid optimization algorithm the minimum sample leaf size was set to 30.

Like XGBoost, Random Forest is a tree based ensemble method where each feature is seen as a potential feature to split on. We did perform feature selection while applying our Random Forest model, as the algorithm selects only a fraction (the maximum number of features) at each splitting point. The best performing number of features was found by eliminating features based on their feature importance (Gini importance) in the model. The Gini importance of a feature is the total decrease in node impurity averaged over all trees when removing this feature. The best results were reached when using only features with a minimum Gini importance of 0.01370, and this led to the model using only 14 features. The performance of these 14 features in the model can be seen in figure 5.5.

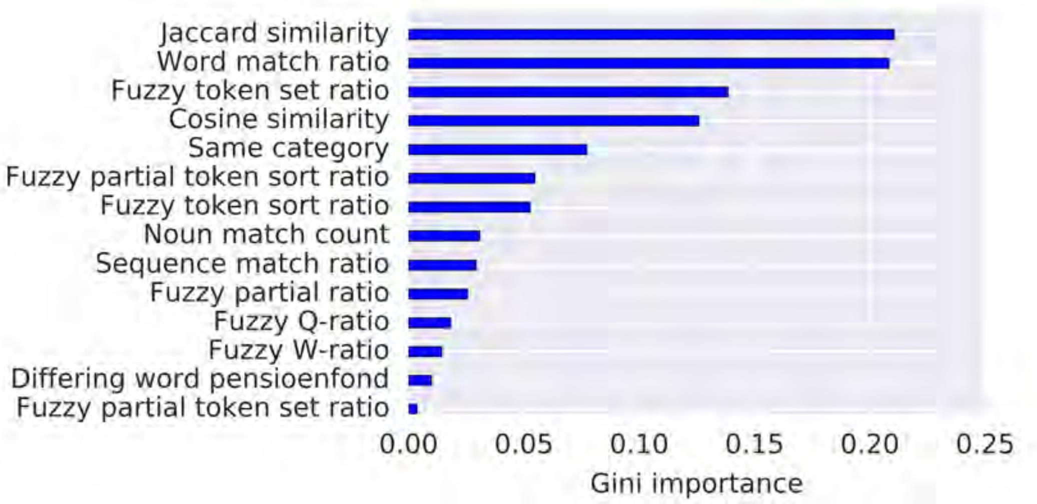


FIGURE 5.5: A bar chart displaying the feature importances in the best performing Random Forest model.

5.4.5 Support Vector Machine

Support Vector Machine (SVM) is a machine learning algorithm that plots each observation as a point in an n-dimensional space, where n is the number of features in the model. The objective of the algorithm is to perform classification by finding a hyper-plane that separates the two classes as good as possible. The hyper-plane does not have to be linear but can take on all kinds of shapes due to transformations of the space using kernels. Further theory behind SVM is explained in section 2.5.6.

We now explain the most important parameter settings in our SVM model. These parameters were tuned by a grid searching algorithm that finds the optimal combination in a grid of parameter values. For this tuning 5-fold cross validation was used to ensure that the settings are not just optimal for one part of the dataset.

1. **Kernel**
   The kernel of an SVM model take the low dimensional input space and converts it to a space in which the classes can be well separated. The kernel transformation options are linear, polynomial, radial basis function and sigmoid. Using the linear kernel is advised when the number of features in the model is large (>1000) as then it is likely that the data is linearly separable in a high dimensional space. Using our grid optimization algorithm we tried all 4 kernel settings and found that the linear kernel was indeed optimal for our dataset.

2. **Gamma**
   Gamma is the kernel coefficient used in the radial, polynomial and sigmoid kernels. A higher value of gamma means that the model will try to fit the training data points more closely. A too high value can lead to overfitting, and a too low value can lead to underfitting. Using our grid optimization algorithm we found that a gamma value of the inverse of the number of features was optimal. This is also the default value

3. **C**
   C is the penalty parameter of the error. The setting of this parameter needs to be a balance between a smooth decision boundary and a correct classification of the training points. Using our grid optimization algorithm we found that a C value of 0.1 was optimal for our dataset. The default value of C is 1.

SVM implements regularisation (ridge regression) to avoid overfitting. Ridge regression is a measure to solve multicollinearity problems between features in a model. Using ridge regression highly correlated features get a small bias to overcome this problem. Therefore if we carefully set the C parameter, the performance of the model should be good without first using feature selection. We found out that this was indeed the case as the best performing model for SVM was the model with the full set of features.

5.4.6 Neural Networks

Neural networks consist of layers with neurons connected by weighted connection links. The weights on these connection links are modified as to reach the optimal model from the training set. Further theory behind Neural Networks is explained in section 2.5.7.

Setting an optimal number of layers and neurons in these layers can be done using a set of rules. The input layer consists of as many neurons as there are features in the model, and the output layer consists of 1 neuron. Then for the large majority of problems and datasets one hidden layer is sufficient. An empirically derived rule-of-thumb says that the optimal number of neurons in the hidden layer is between the number of neurons in the input and output layers.

We now explain the most important parameter settings in our NN model. These parameters were tuned by a grid searching algorithm that finds the optimal combination in a grid of parameter values. For this tuning 5-fold cross validation was used to ensure that the settings are not just optimal for one part of the dataset. We also scaled the features before modelling as this is needed to get meaningful results from a neural network.

1. **Hidden layer size**
   We use one hidden layer as literature shows that this is sufficient for the majority of datasets. According to a rule-of-thumb the number of neurons in this layer optimally lays between the number of features and 1 (the number of neurons in the output layer). We have experimented with layers with all, three quarters, half and one quarter of the number of features as the number of neurons. Using our grid optimization algorithm we found that 1146 neurons (3/4 of the number of features) in the hidden layer is optimal for our dataset.

2. **Solver**
   The solver used for the optimization of the weights on the connection links can be chosen. The options are lbfgs, sgd and adam. Lbfgs is a quasi-Newton method optimizer, sgd uses stochastic gradient descent and adam (default) uses a specific gradient-based optimizer. Using our grid optimization algorithm we tried all three and found that adam is optimal for our dataset.

3. **Alpha**
   Alpha is a regularization parameter, that helps prevent overfitting by constraining the size of the weights on the connection links. A high value of alpha can help in datasets with high variance and a low value of alpha can help in datasets with high bias. Using our grid optimization algorithm we found that an alpha value of 1×10^{-5} is optimal for our dataset. The default value is 1×10^{-4}

Unfortunately, neural network models lack in interpretability. Therefore it is not possible to find out which features had a high contribution in the model without learning an interpretable model around the neural network. This was not further studied in this research. We did find that the full set of features led to better results than the selected feature sets.

5.4.7 Logistic regression

Logistic regression is a regression model with a categorical (in our case binary) output. The theory behind this model is explained in section 2.5.5. While applying logistic regression to a specific dataset many parameters can be tuned.

We now explain the most important parameter settings in our model. These parameters were tuned by a grid searching algorithm that finds the optimal combination in a grid of parameter values. For this tuning 5-fold cross validation was used to ensure that the settings are not just optimal for one part of the dataset.

1. **Penalty**
   The penalty parameter is used to specify what type of penalty is given to a

wrong match. Since most solvers only support I2 penalties we have chosen this penalty.

2. **C**
   As in SVM, C is the inverse of the regularization strength. Thus a higher value specifies a stronger regularization. Using our grid optimization algorithm we found that a C value of 0.1 was optimal for our dataset. The default value is 1.

To get a good prediction using most logistic regression models, feature engineering is crucial. However our logistic regression model already uses regularization. Therefore we expected that the full feature set would return better results than the selected feature sets. This was not the case as the model with the features that were significant in the chi-squared test performed best.

5.5 Results

In this section we will evaluate the best results of the 5 different machine learning models. There are many methods and metrics to evaluate the results of a classification problem. Well-known metrics are the confusion matrix, from which precision, recall, sensitivity, specificity, F-score and accuracy can be derived and AUC score. We know explain the pros and cons of using these metrics for evaluation on our dataset.

The confusion matrix, as shown in figure 5.7a, depicts the performance of an algorithm at a certain cut-off value. We have used our machine learning models for a binary classification: match or no match. However, these models can also give a probability with which they predict that it is a match or not, and this is the value we actually use. The cut-off value is the probability at which we round up to say that the questions match or not. Most generally, a cut-off value of 50% is used, but the choice depends on the business case. In our case a higher cut-off value means that less questions will be considered as pairs, in the larger chatbot scheme this means that we will answer less questions, but also that less questions will be answered wrongly. The optimal cut-off (or confidence level) will be chosen during the implementation of the chatbot in chapter 6. In this section we will use 50% as the cut-off value.

The confusion matrix shows the number of true positives, false positives, false negatives and true negatives. When relating this to our chatbot implementation we can see those terms as follows:

1. **True positives**
   The true positives are question-pairs that match and that are also predicted to match. In a chatbot this means that there is an incoming question to which we have a match and that we find this match and give a correct answer.

2. **False positives**
   The false positives are question-pairs that do not match but are predicted to match. In a chatbot this means that there is an incoming question to which we think we have a match, but is not. This means that we give a wrong answer.

3. **False negatives**
   The false negatives are question-pairs that match but we predict that they do not match. In a chatbot this means that there is an incoming question to which we have a match but we do not find this match. This means that we could have answered the question, but we did not. This is a missed opportunity.

4. **True negatives**
   The true negatives are question-pairs that do not match and that are also predicted to not match. In a chatbot this means that there is an incoming question to which we do not have a match and we also identify it as such. This means that it is a new question to which we do not have an answer yet.

The outcome would be optimal if there are only true positive and negatives and no false positive and negatives. This gives an accuracy of 100%. However, this optimal outcome is not achievable and therefore we have to consider what type of mistakes are worse than others. In our case a false positive means giving a wrong answer, which is something that companies do not want their chatbots to do. However, when minimizing the amount of false positives we also decrease the amount of true positives, which leads to the chatbot not answering most questions which is also costly for the company as experts have to do all the work. Therefore we have to find a balance between the four components in the confusion matrix that works for this specific business case.

Metrics like accuracy, precision, recall (also known as sensitivity), specificity and F-score are used to show the performance of a model. In our case those terms can be interpreted as:

1. **Accuracy**
   The accuracy shows how often the classification outcome is correct. It divides the sum of the correctly answered and correctly not answered questions by all questions asked. This means that it is the percentage of correctly handled questions. A higher percentage of correctly handled questions is of course better, but this measure does not say anything about the incorrectly handled questions. It does not show whether the incorrectly handled questions are mostly wrongly answered questions or missed opportunities.

2. **Precision**
   Precision is the number of correctly answered questions divided by all answered questions. This is an important percentage as wrongly answered questions are bad for the company. We strive for a high precision.

3. **Recall (also sensitivity)**
   Recall is the number of correctly answered questions divided by all questions that we could have been able to answer. Having a low recall means that there are many missed opportunities, i.e. questions that we did have an answer for but we did not find them.

4. **Specificity**
   Specificity is the number of questions that we correctly did not answer divided by all questions that cannot be answered. We strive for this percentage to be high, as it means that we can correctly identify when not to answer a question, and ultimately that we do not give too many wrong answers.

5. **F-score**
   The F-score is the harmonic mean of the precision and recall. It combines the percentage of correct answers and the percentage of missed opportunities.

From these definitions we can say that we find it important to have a high precision. However, when optimizing a model for precision, all questions which are not sure will not be answered. Therefore optimizing for precision makes for giving less answers. A better metric is the F-score, which looks at the percentage of correct answers but also at the percentage of missed opportunities.

A downside of all these metrics derived from the confusion matrix is that we have to choose a cut-off level in the probability of matching before assessing these scores. This way we do not look at the whole performance of the model, but at the performance of the model at a specific confidence level. This confidence level might not turn out to be optimal when implementing these models in a chatbot application.

The Area Under the Curve (AUC) is a metric that does give a score for the whole model at every confidence level. The intended curve under which the area is calculated is the Receiver Operating Characteristic (ROC). One point on this curve specifies the sensitivity of the model in relation to the inverse of the specificity. That means that one point on the curve is the relation between missed opportunities (decreasing further up the y-axis) and the incorrectly answered questions rate. One point on the curve is a representation of the model at a certain cut-off level. The whole curve represents the model at all cut-off levels. Optimizing the Area Under the Curve means optimizing the combination of missing answering opportunities and giving wrong answers.

Since the AUC is a measure for the whole model, at every cut-off level, and it takes the false positive rate into account we have chosen to optimize our models for this metric.

In the next section we show the results for the machine learning models on the most important metrics. In the end we choose one model to implement in the chatbot.

5.5.1 Model comparison

The results of the machine learning models on the different scoring metrics are shown in figure 5.6 and the confusion matrices at a cut-off of 50% can be seen in figure 5.7.

As shown, the XGBoost model gives the best results on AUC, accuracy and precision (percentage of given answers that is correct), but the Random Forest model has the highest F-score. The F-scores of the Random Forest model and XGBoost model only differ by 0.001, and XGBoost performs much better on the other evaluation metrics. Therefore we consider the XGBoost model to be the best performing model. When looking at the confusion matrices for XGBoost and Random Forest we can see that the Random Forest model gives a lot more answers, but also a lot more wrong answers (which explains the lower precision). The F-score of the Random Forest model is high because the recall (percentage of answering opportunities taken) is very high. We conclude that the XGBoost model performs best on our dataset and therefore this model will be used in the chatbot implementation.

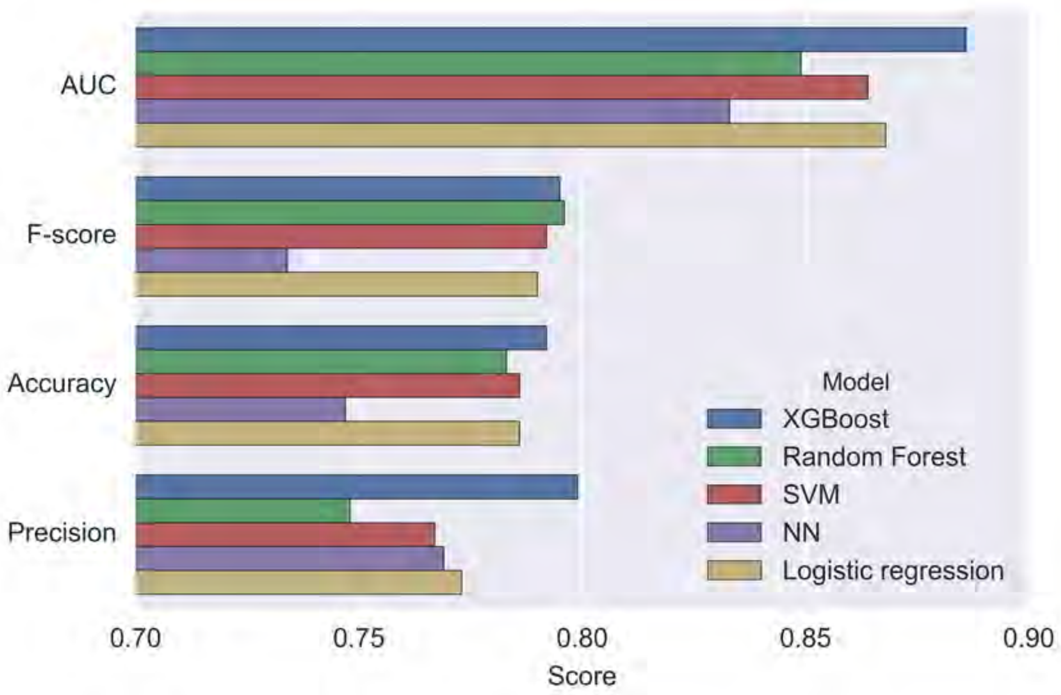


FIGURE 5.6: The results of all machine learning models on different evaluation metrics.

		prediction outcome	
		p	**n**
actual value	**p**	True positive	False negative
	n	False positive	True negative

(A) Confusion matrix

		prediction outcome	
		p	**n**
actual value	**p**	373	86
	n	106	357

(B) XGBoost

		prediction outcome	
		p	**n**
actual value	**p**	391	68
	n	132	331

(C) Random Forest

		prediction outcome	
		p	**n**
actual value	**p**	376	83
	n	114	349

(D) SVM

		prediction outcome	
		p	**n**
actual value	**p**	322	137
	n	96	367

(E) Neural Network

		prediction outcome	
		p	**n**
actual value	**p**	371	88
	n	109	354

(F) Logistic regression

FIGURE 5.7: The confusion matrices for all machine learning models at a confidence level of 50%.

Chapter 6

Chatbot implementation

In this chapter we implement the vector space model and the machine learning model in a chatbot application. The application is a simple GUI built on top of the models. In the process of creating this GUI some business decisions were made.

First we determined the chat flows, which describe the way in which a user is answered by the chatbot. Our chat flows are described in section 6.1. Then, we decided on the confidence levels needed to answer a question or ask a return question as described in section 6.2. Finally, we added an extra functionality to the chatbot making it improve its answer matching with every use. This is described in section 6.3. The results of the implemented models can be found in section 6.4.

6.1 Chat flows

A chat flow is the way in which a user is answered by the chatbot. In our application a user asks a financial question and, based on the characteristics of this question, he ends up in one of three chat flows as shown in figure 6.1.

If the user question is close to a question in the database the chatbot thinks that it "understands" the question, and the user goes into flow 1. In flow 1 the user is greeted by the bot, receives an answer to his question and the bot asks whether this answer is satisfying to the user. If the answer was satisfying the conversation stops there. If the user finds the answer incorrect, then his question is sent through to a human expert on the topic.

If the user question has some similarity with a question in the database the chatbot "maybe understands" the question, and the user goes into flow 2. In flow 2 the user is greeted by the bot, and the bot tries to formulate a yes/no return question. This return question is formed by one-by-one adding important financial terms (e.g.

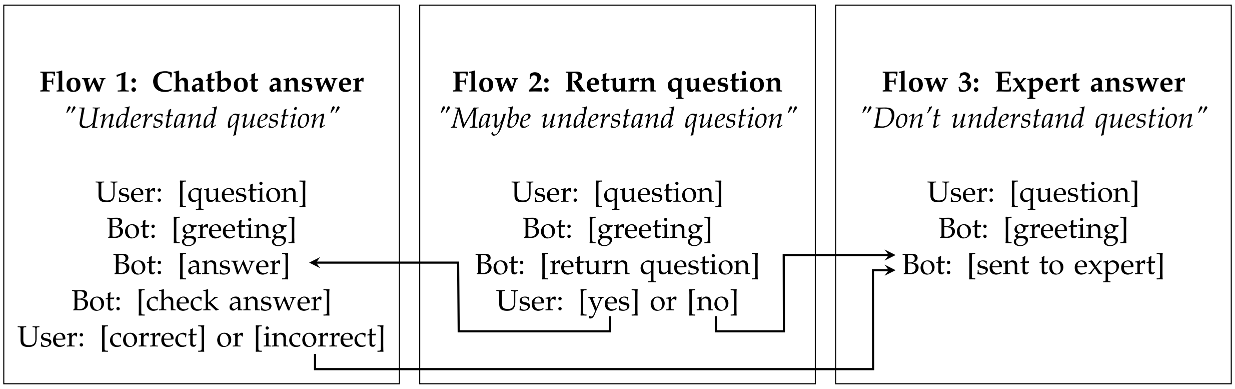


FIGURE 6.1: A schematic overview of the chatbot flow.

"aow" or "zzp") to the user question and then checking if it now matches a question in the database more closely. When a term is found that makes the user question match a database question closely, then the user is asked "Does your question have something to do with [term]?" This question is shown in a pop-up window and the user can click yes or no. If the user answers yes, then the answer to the closely matched question is given and the question is further handled like in flow 1. If the user answers no, then the original user question is sent through to a human expert.

If the user question cannot be matched (with a high enough score) to the questions in the database then the chatbot "doesn't understand" the question, and the user goes into flow 3. In flow 3 the user is greeted by the bot and the question is then sent through to a human expert.

6.2 Confidence levels

The scores, or confidence levels, at which point a user is sent through to flow 1, 2 or 3 need to be determined. The goal is to answer as many questions as possible but not to answer a too high percentage of those questions wrongly. To set these confidence levels, we use a validation set for which we know whether two questions match or not and for which we have the cosine similarity and XGBoost score between the questions. We could not use our test set here, as then we would tune our application to the test set and the measured performance would be higher than it actually is.

The validation set was ran through our chatbot model and the chatbot of the respective model selected the best possible answer. After this procedure we manually labelled these answers as matching or non-matching. The vector space model chatbot found a match for 33% of the questions and the machine learning model chatbot found a match for 38% of the questions in our validation set.

6.2.1 Vector space model

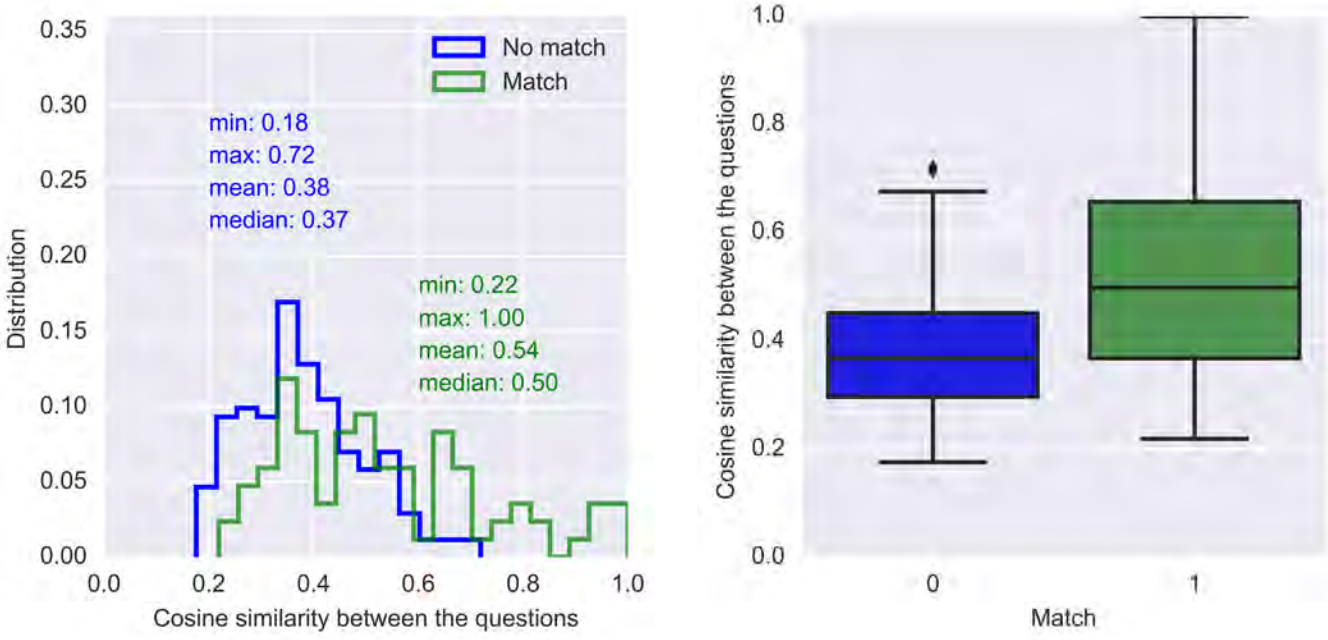
In the vector space model we measure the cosine similarity between two questions to predict if they match or not. To find the confidence level at which two questions match, maybe match or don't match we have created the graphs in figure 6.2 and 6.3.

In the histogram in figure 6.2, we see that non-matching questions have cosine similarities between 0.18 and 0.72, and that the histogram overlaps quite much with the one for matching questions. The matching questions have cosine similarities between 0.22 and 1, and their distribution is more towards higher cosine similarities.

In the box plot in figure 6.2, we see that the difference in cosine similarities between non-matching and matching questions is not significant as the inter-quartile ranges of the box plots overlap. Since the medians of the box plots do not overlap with the interquartile ranges of the other box plot it is likely that the cosine similarities of matching and non-matching questions do differ.

Vector space model confidence levels

To find the right cosine similarities to use as confidence levels (or cut-off points) in the chatbot application we look at the cumulative distribution in figure 6.3. This figure shows that from about a cosine similarity of 0.7 there are no more non-matching questions. However choosing 0.7 as the confidence level for answering a question would also mean that we answer less than 20% of the questions to which we do have a correct answer. Looking at the non-matching cumulative distribution we see



(A) Histogram

(B) Box plot

FIGURE 6.2: Charts displaying the distribution of the cosine similarity for matching and non-matching questions.

that at a cosine similarity of 0.6 we have already cut-off more than 95% of the non-matching questions. At a cosine similarity of 0.6 we answer more than 30% of the matching questions. This sounds like a good balance but since only 33% of questions match we would still end up answering very few questions. At a cut-off of 0.45 we would answer 50% of matching questions and 20% of non-matching questions. We think this is a good balance between right and wrong answers and therefore set a cosine similarity of 0.45 as the confidence level to go into flow 1.

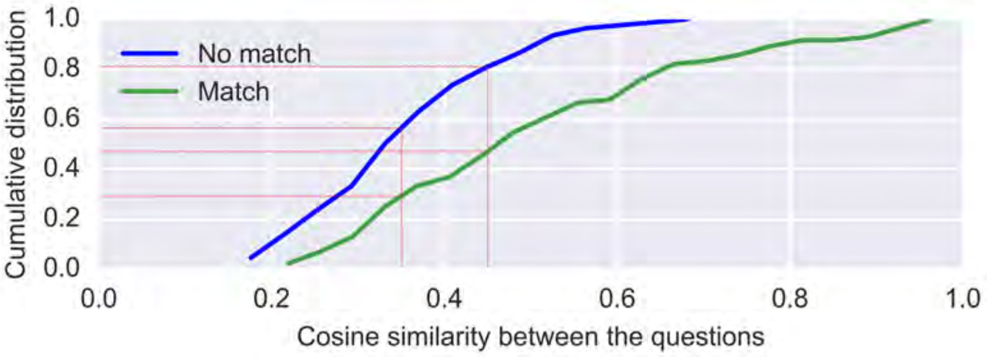


FIGURE 6.3: A chart displaying the cumulative distribution of the cosine similarity for matching and non-matching questions.

Some questions might be a bit unclearly posed or phrased very differently from questions in the dataset. To give these questions a second chance at matching we ask the return question in flow 2. To set the minimal cosine similarity at which a question can go into flow 2, we look at figure 6.3 again. At a cosine similarity of

0.35 we have already filtered out 60% of non-matching questions, and only 30% of matching questions. Setting the confidence level for getting into flow 2 at a cosine similarity between 0.35 and 0.45 means that this gives us the potential to still answer the 20% of matching questions that falls between these cosine similarities correctly. Therefore we think that a minimum confidence level of 0.35 for the return question is appropriate.

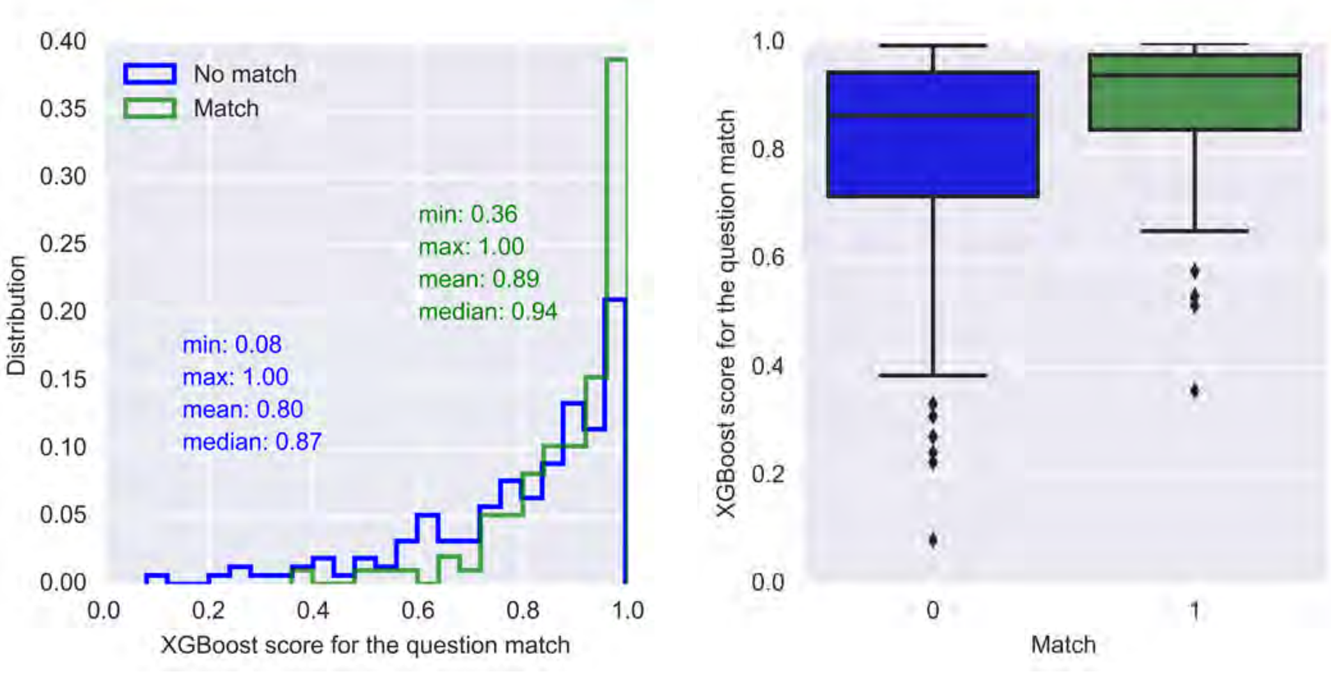
If the confidence level is below 0.35 the user will go into flow 3, and receive an expert answer.

6.2.2 Machine learning model

In the machine learning model we estimate the matching probability of two questions using an XGBoost model. To find the confidence level at which two questions match, maybe match or don't match we have created the graphs in figure 6.4 and 6.5.

In the histogram in figure 6.4, we see that non-matching questions have an average XGBoost score of 0.80, and that the histograms for non-matching and matching questions overlap for many XGBoost scores. The matching questions have an average XGBoost score of 0.89, and this group is distributed more towards high XGBoost scores.

In the box plot in figure 6.4, we see that the difference in XGBoost score between non-matching and matching questions is not significant as the inter-quartile ranges of the box plots overlap. Also the medians overlap with the inter-quartile ranges of the other box plot meaning that it is likely that matching and non-matching questions have similar XGBoost scores.



(A) Histogram

(B) Box plot

FIGURE 6.4: Charts displaying the distribution of the XGBoost score for matching and non-matching questions.

Machine learning model confidence levels

To find the right XGBoost scores to use as confidence levels (or cut-off points) in the chatbot application we look at the cumulative distribution in figure 6.5. This figure shows again that the differences between non-matching and matching questions in XGBoost score is small. At an XGBoost score of 0.92 and higher the matching question graph is steeper than the non-matching question graph. This means that there are more matching than non-matching questions here (percentually). At an XGBoost score of 0.92 we answer 40% of matching questions and 20% of non-matching questions. We think that this is a good balance between right and wrong answers and therefore set an XGBoost score of 0.92 as the confidence level to go into flow 1.

Some questions might be a bit unclearly posed or phrased very differently from questions in the dataset. To give these questions a second chance at matching we ask the return question in flow 2. To set the minimal XGBoost score at which a question can go into flow 2, we look at figure 6.5 again. What stands out is that the slopes of non-matching and matching questions between 0.6 and 0.92 are very similar. This means that the percentage of non-matching and matching questions between these XGBoost scores are similar. Because of this similarity asking a return question is garantueed to lead to the answer "No" in at least 50% of cases. This is the reason for deciding not to implement the return question for the machine learning model.

If the confidence level is below 0.92 the user will go into flow 3, and receive an expert answer.

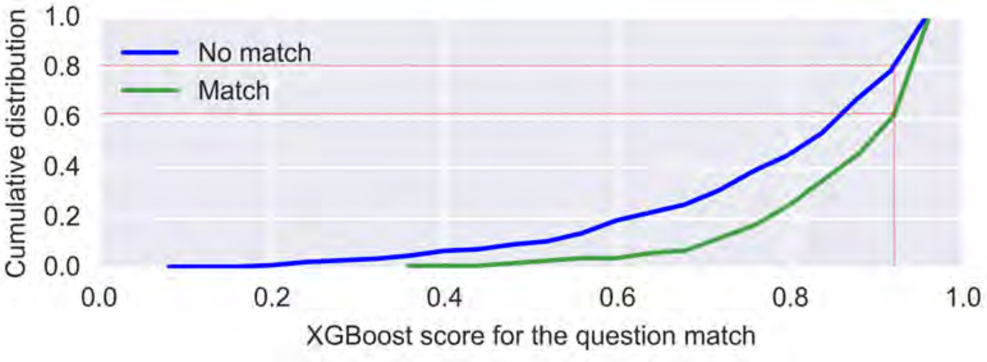


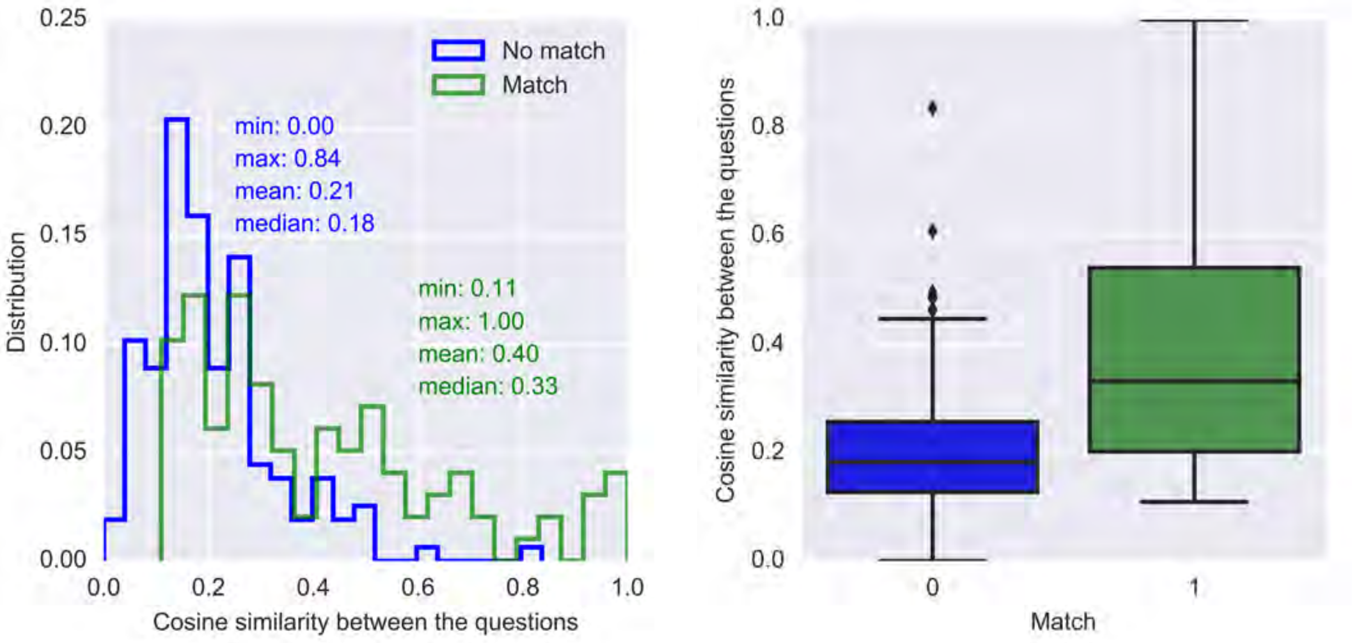
FIGURE 6.5: A chart displaying the cumulative distribution of the XGBoost score for matching and non-matching questions.

6.2.3 Combination model

In the previous sections we found that the vector space model chatbot found a match for 33% of the questions and the machine learning model chatbot found a match for 38% of the questions, when answering every question with its closest match. We also found that the cosine similarity seems to be able to better differentiate between matching and non-matching questions than the XGBoost score. This can be seen when comparing the box plots in figure 6.2(B) and 6.4(B). Therefore we have decided to test a model that selects the closest match based on XGBoost score, and decides which flow a question belongs to using cosine similarity. To find the confidence level at which two questions match, maybe match or don't match we have created the graphs in figure 6.6 and 6.7.

In the histogram in figure 6.6, we see that non-matching questions have an average cosine similarity of 0.21. The matching questions have an average cosine similarity of 0.40, and this group is distributed more towards higher cosine similarities.

In the box plot in figure 6.6, we see that the difference in cosine similarity between non-matching and matching questions is not significant as the inter-quartile ranges of the box plots overlap. However, the medians of the box plots do not overlap with the interquartile ranges of the other box plot which indicates that it is likely that the cosine similarities of matching and non-matching questions do differ.



(A) Histogram

(B) Box plot

FIGURE 6.6: Charts displaying the distribution of the cosine similarity for matching and non-matching questions in the XGBoost model.

Combination model confidence levels

To find the right XGBoost scores to use as confidence levels (or cut-off points) in the chatbot application we look at the cumulative distribution in figure 6.7. At a cosine similarity of 0.4 about 95% of the non-matching questions are cut-off and we answer 40% of the matching questions. This leads to a high percentage of correct answers but also to not answering many questions. To answer more questions we can implement the return question flow on questions with a cosine similarity between 0.25 and 0.4. At a cosine similarity of 0.25 we cut-off 80% of non-matching questions and we keep 60% of matching questions.

Other possibly good scenarios, where we answer more questions, would be to set the confidence level at 0.25 or 0.3 for getting an immediate chatbot answer. At 0.25 we cut-off 80% of non-matching questions and we keep 60% of matching questions. This would lead to a lower percentage of correct answers than at 0.4 but it would also lead to giving many more answers. A confidence level of 0.3 would be somewhere in between. What the correct balance is between how many questions to answer and what percentage of questions to answer is a business decision. A company wants the chatbot to be of use, but also to answer a lot correctly.

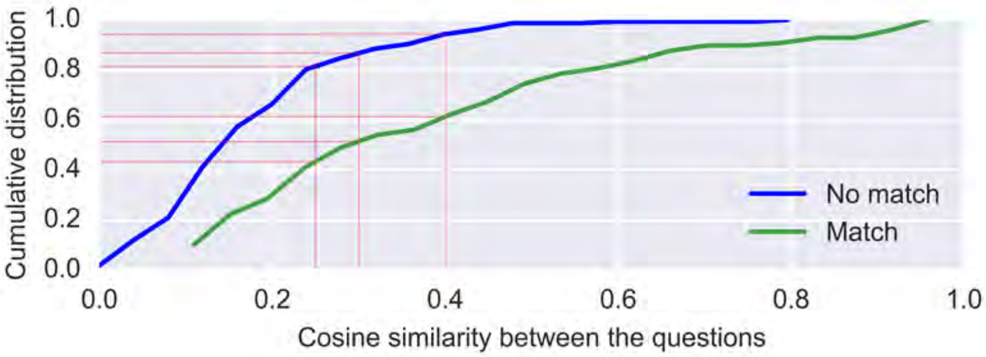


FIGURE 6.7: A chart displaying the cumulative distribution of the cosine similarity for matching and non-matching questions in the XGBoost model.

6.3 Dynamic learning

To increase the amount of questions that the chatbot can answer we extend the question-answer dataset with questions that are correctly answered by the chatbot. When the chatbot gives an answer to the user it also asks whether this answer was satisfactory, and if the user presses the green smiley this will lead to the user question and the given answer being added to the question-answer dataset.

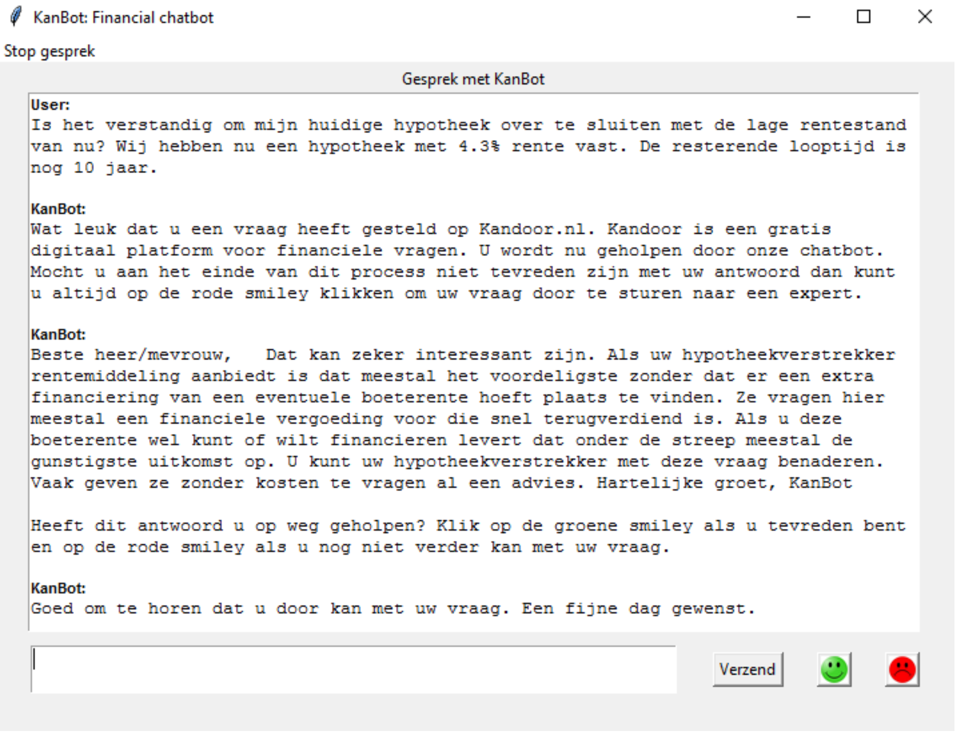
In the case of the machine learning chatbot the combination of the user question and the matched question could also be added as a positive or negative match in the Q2Q dataset. This would lead to a process of dynamic learning. This is not applied in this chatbot prototype as we use the already trained model in the chatbot and to apply this we would need to retrain the model at certain time intervals. Applying the retraining of the chatbot overnight would be a good functionality for a real (non-prototype) implementation of the chatbot.

6.4 Results

In this section we give the results of the chatbot implementation. First, in section 6.4.1 we shows some examples of answers given by the chatbot. Then in section 6.4.2, 6.4.3 and 6.4.4 we give the results for the vector space model, machine learning model and combination model chatbot implementation, respectively.

6.4.1 Answer examples

The features of the chatbot described in the beginning of this chapter, like the return question and dynamic learning are all implemented in our vector space model chatbot. Therefore we show some examples of answers this chatbot gives. Examples of the machine learning chatbot are very similar. Figure 6.8, is an example of a user question that is sent to "Flow 1: Chatbot answer". This user question is immediately answered by the chatbot, and the chatbot asks whether the answer was satisfactory. In this case we pressed the green smiley as the answer was correct and helpful.



User:
Is het verstandig om mijn huidige hypotheek over te sluiten met de lage rentestand van nu? Wij hebben nu een hypotheek met 4.3% rente vast. De resterende looptijd is nog 10 jaar.

KanBot:
Wat leuk dat u een vraag heeft gesteld op Kandoor.nl. Kandoor is een gratis digitaal platform voor financiele vragen. U wordt nu geholpen door onze chatbot. Mocht u aan het einde van dit process niet tevreden zijn met uw antwoord dan kunt u altijd op de rode smiley klikken om uw vraag door te sturen naar een expert.

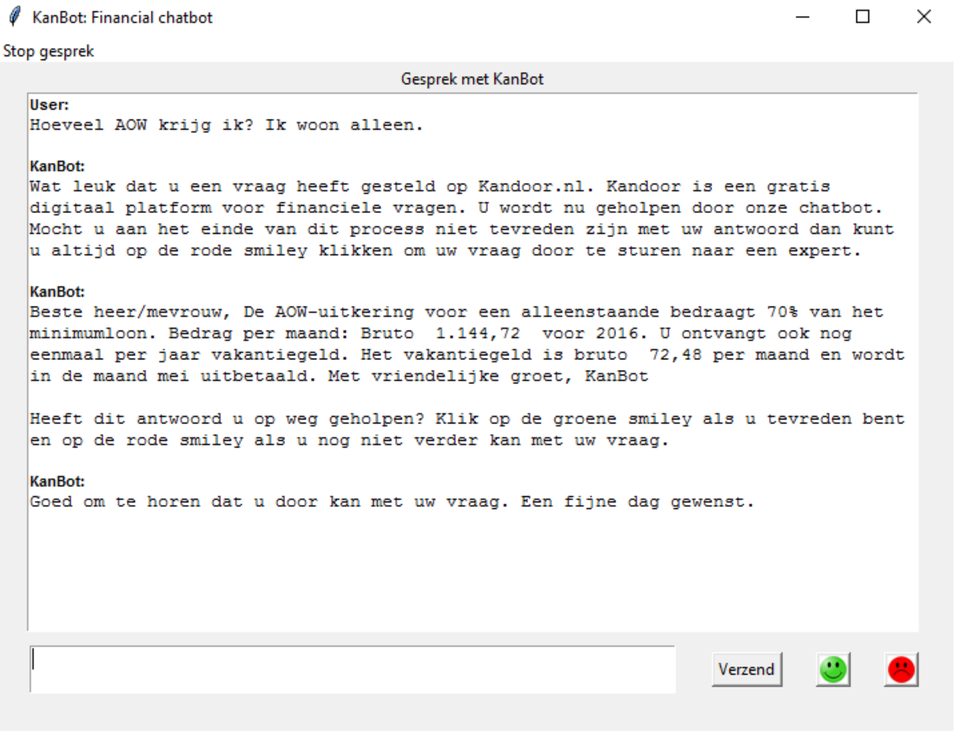
KanBot:
Beste heer/mevrouw,   Dat kan zeker interessant zijn. Als uw hypotheekverstrekker rentemiddeling aanbiedt is dat meestal het voordeligste zonder dat er een extra financiering van een eventuele boeterente hoeft plaats te vinden. Ze vragen hier meestal een financiele vergoeding voor die snel terugverdiend is. Als u deze boeterente wel kunt of wilt financieren levert dat onder de streep meestal de gunstigste uitkomst op. U kunt uw hypotheekverstrekker met deze vraag benaderen. Vaak geven ze zonder kosten te vragen al een advies. Hartelijke groet, KanBot

Heeft dit antwoord u op weg geholpen? Klik op de groene smiley als u tevreden bent en op de rode smiley als u nog niet verder kan met uw vraag.

KanBot:
Goed om te horen dat u door kan met uw vraag. Een fijne dag gewenst.

FIGURE 6.8: The chatbot screen with a good answer to a newly typed question.

Figure 6.9 and 6.10 are steps in "Flow 2: Return question". After the user asks a question that ends up in flow 2, the chatbot tries to add a word to the question that makes the user question get the score needed to be answered by the chatbot. In this case that word is "alleenstaande", as shown in figure 6.9, which has to do with getting a different amount of state pension if you live alone than if you live together. This question has something to do with "alleenstaande" and therefore we press "Yes". After pressing "Yes" the answer belonging to the matched question is given as shown in figure 6.10.



User:
Hoeveel AOW krijg ik? Ik woon alleen.

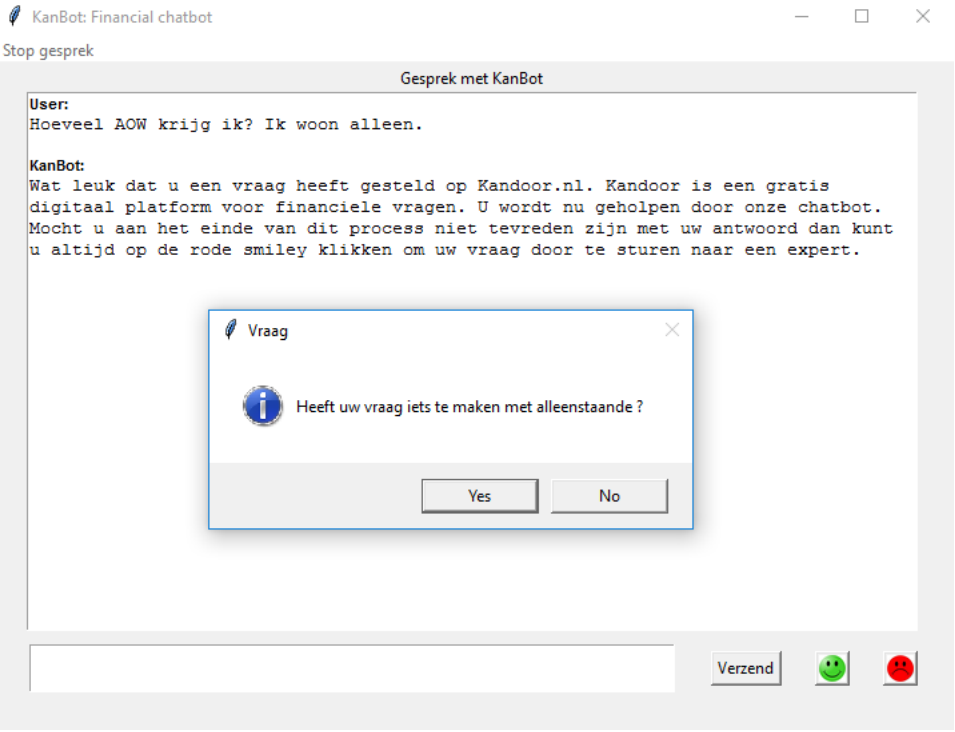
KanBot:
Wat leuk dat u een vraag heeft gesteld op Kandoor.nl. Kandoor is een gratis digitaal platform voor financiele vragen. U wordt nu geholpen door onze chatbot. Mocht u aan het einde van dit process niet tevreden zijn met uw antwoord dan kunt u altijd op de rode smiley klikken om uw vraag door te sturen naar een expert.

KanBot:
Beste heer/mevrouw, De AOW-uitkering voor een alleenstaande bedraagt 70% van het minimumloon. Bedrag per maand: Bruto 1.144,72 voor 2016. U ontvangt ook nog eenmaal per jaar vakantiegeld. Het vakantiegeld is bruto 72,48 per maand en wordt in de maand mei uitbetaald. Met vriendelijke groet, KanBot

Heeft dit antwoord u op weg geholpen? Klik op de groene smiley als u tevreden bent en op de rode smiley als u nog niet verder kan met uw vraag.

KanBot:
Goed om te horen dat u door kan met uw vraag. Een fijne dag gewenst.

FIGURE 6.10: The chatbot screen after correctly answering a question in which a return question was answered with "Yes".



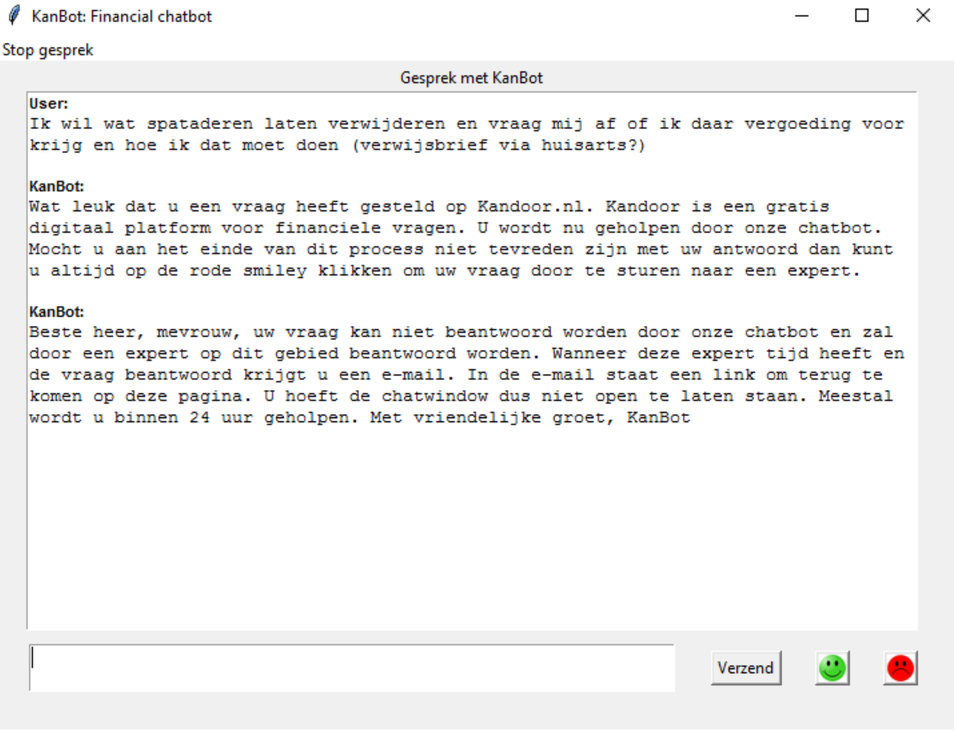
User:
Hoeveel AOW krijg ik? Ik woon alleen.

KanBot:
Wat leuk dat u een vraag heeft gesteld op Kandoor.nl. Kandoor is een gratis digitaal platform voor financiele vragen. U wordt nu geholpen door onze chatbot. Mocht u aan het einde van dit process niet tevreden zijn met uw antwoord dan kunt u altijd op de rode smiley klikken om uw vraag door te sturen naar een expert.

Heeft uw vraag iets te maken met alleenstaande ?

FIGURE 6.9: The chatbot asking a return question.

Finally, it can also happen that the chatbot does not find a high enough cosine similarity for a question to go into flow 1 or 2. The question then ends up in "Flow 3: Expert answer" as shown in figure 6.11. In this flow the user is greeted and then the question is sent through to a human expert.



User:
Ik wil wat spataderen laten verwijderen en vraag mij af of ik daar vergoeding voor krijg en hoe ik dat moet doen (verwijsbrief via huisarts?)

KanBot:
Wat leuk dat u een vraag heeft gesteld op Kandoor.nl. Kandoor is een gratis digitaal platform voor financiele vragen. U wordt nu geholpen door onze chatbot. Mocht u aan het einde van dit process niet tevreden zijn met uw antwoord dan kunt u altijd op de rode smiley klikken om uw vraag door te sturen naar een expert.

KanBot:
Beste heer, mevrouw, uw vraag kan niet beantwoord worden door onze chatbot en zal door een expert op dit gebied beantwoord worden. Wanneer deze expert tijd heeft en de vraag beantwoord krijgt u een e-mail. In de e-mail staat een link om terug te komen op deze pagina. U hoeft de chatwindow dus niet open te laten staan. Meestal wordt u binnen 24 uur geholpen. Met vriendelijke groet, KanBot

FIGURE 6.11: The chatbot screen with a question that cannot be answered by the chatbot and is sent through to an expert.

6.4.2 Vector space model implementation

In section 4.5, we already saw the results of the vector space model without boundaries. Meaning that we answered every question with the answer to the question with the highest cosine similarity. This led to answering 33% of the questions correctly.

The results of the implementation of the vector space model are shown in figure 6.12 and 6.13. In figure 6.12, we see that 38% of questions is immediately answered by the chatbot, for 29% we try to formulate a return question and 33% is immediately sent through to an expert. From the questions that are sent to flow 2, we cannot find a suitable return question for 31% of cases. For the user questions to which we do ask a return question, 29% of these questions are answered with "Yes" and 71% are answered with "No". In total this all means that 44% of questions are answered by the chatbot and of the given answers 55% is correct. The other 56% of questions is sent to experts.

In figure 6.13, we show the vector space model implementation results in a confusion matrix of sorts. On the top we have the cosine similarity scores, where questions with scores higher than 0.45 receive an answer and questions with lower scores do not. On the left axis we show if the correct answer to a question was selected (no matter if it was answered or not). The numbers in the confusion matrix including the return question add up to different numbers of correct/incorrect answers selected and different cosine score results, as asking the return question can improve cosine scores and alter the answer selected.

Figure 6.13a, shows that 61 questions were answered correctly, we did not answer 22 questions to which we did select the right answer (missed opportunity), we answered 50 questions incorrectly and 122 questions were correctly not answered. If we would not have asked the return question but just have cut-off the cosine values

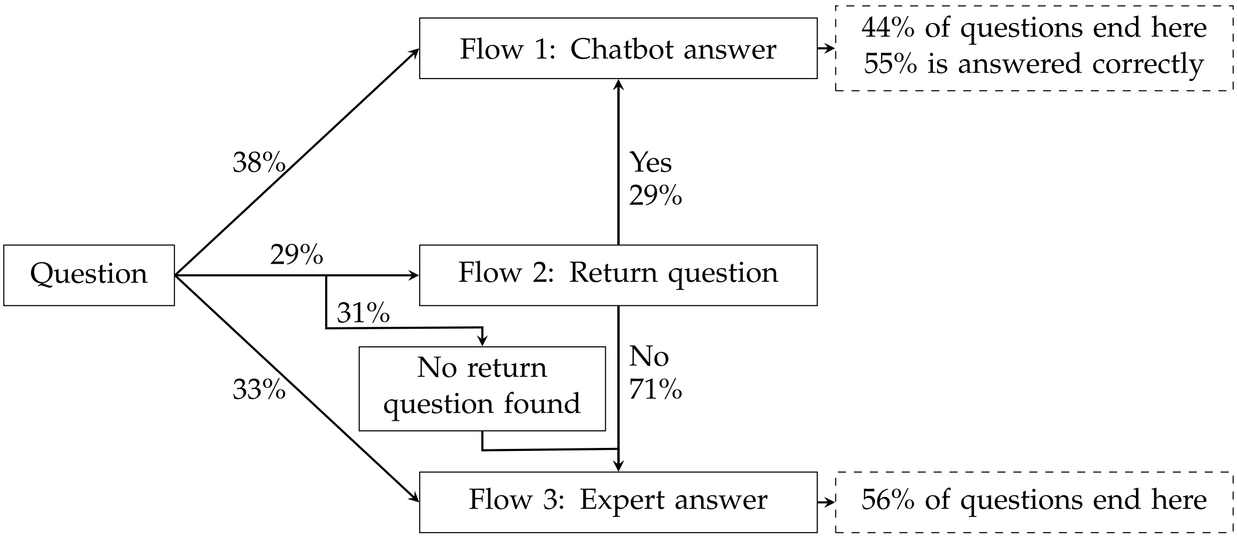


FIGURE 6.12: An overview of the results of the vector space model chatbot implementation.

at 0.45 we would have had the results in figure 6.13b. This means that we would have answered 52 questions correctly, we would not have answered 32 questions to which we did select the right answer (missed opportunities), we would have answered 44 questions incorrectly and 127 questions would correctly not have been answered. Comparing these two outcomes we see that the return question leads to answering more questions, but also to answering more questions wrongly. In percentages, if we calculate the precision, we see that the model including the return question answers 55% of questions correctly and the model excluding the return question answers 54% of questions correctly. Since the number of given answers is higher and the percentage of correct given answers is higher we prefer the vector space model implementation with the return question.

	$\cos \geq 0.45$	**$\cos < 0.45$**
Correct answer selected	61	22
Incorrect answer selected	50	122

(A) Including return question

	$\cos \geq 0.45$	**$\cos < 0.45$**
Correct answer selected	52	32
Incorrect answer selected	44	127

(B) Excluding return question

FIGURE 6.13: The confusion matrices for the implementation of the vector space model.

6.4.3 Machine learning model implementation

In section 5.5, we showed the results of the machine learning model on a question-to-question test set. In that section the objective was to predict whether two questions were the same or not. For the chatbot implementation this works a little differently. A user question is matched to all questions in our database and then the match with the highest XGBoost score is selected. The test set used contains 255 user questions and is the same one as for the vector space model. We give the results for the question match with the highest XGBoost score. Without any cut-off we correctly answer 38% of the questions.

The results of the implementation of the machine learning model are shown in figure 6.14 and 6.15. In figure 6.14, we see that 40% of questions is answered by the chatbot and 60% of questions is sent through to an expert. From the questions that are answered by the chatbot 52% is answered correctly.

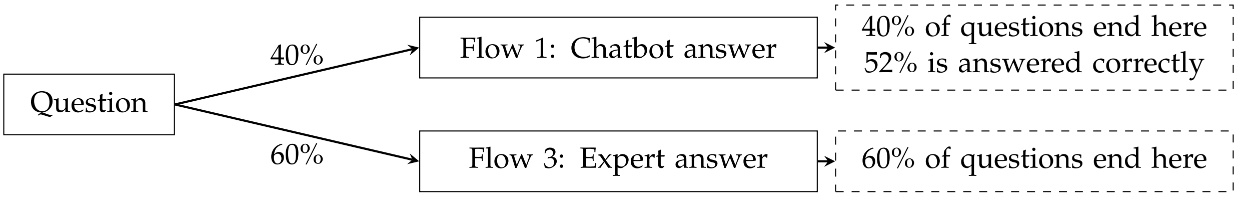

Question
40%
60%
Flow 1: Chatbot answer
Flow 3: Expert answer
40% of questions end here
52% is answered correctly
60% of questions end here

FIGURE 6.14: An overview of the results of the vector space model chatbot implementation.

In figure 6.15, we show the machine learning model implementation results in a confusion matrix of sorts. On the top we have the XGBoost scores, where questions with scores higher than or equal to 0.92 receive an answer and questions with lower scores do not. On the left axis we show if the correct answer to a question was selected (no matter if it was answered or not).

Figure 6.15, shows that 53 questions were answered correctly, we did not answer 45 questions to which we did select the right answer (missed opportunities), we answered 49 questions incorrectly and 108 questions were correctly not answered.

	$\mathbf{XGB \geq 0.92}$	$\mathbf{XGB < 0.92}$
Correct answer selected	53	45
Incorrect answer selected	49	108

FIGURE 6.15: The confusion matrix for the implementation of the machine learning model.

6.4.4 Combination model implementation

For the combination model implementation we decided to try multiple confidence levels to be able to make a good choice of confidence level. The results of the implementation of the combination model at all of these confidence levels are shown in figure 6.16, 6.17, 6.18, 6.19 and 6.20.

Comparing figures 6.16 to 6.19, we clearly see that a lower confidence level leads to a higher percentage of chatbot answers and a lower percentage of correct answers. Adding the return question also leads to a lower percentage of correct answers. When comparing figure 6.16 to figure 6.18, we see that at a confidence level of 0.3 without return question more questions are answered and we answer a higher percentage of questions correctly. Therefore the combination model at a confidence level of 0.3 is superior to the combination model at a confidence level of 0.4 with return question.

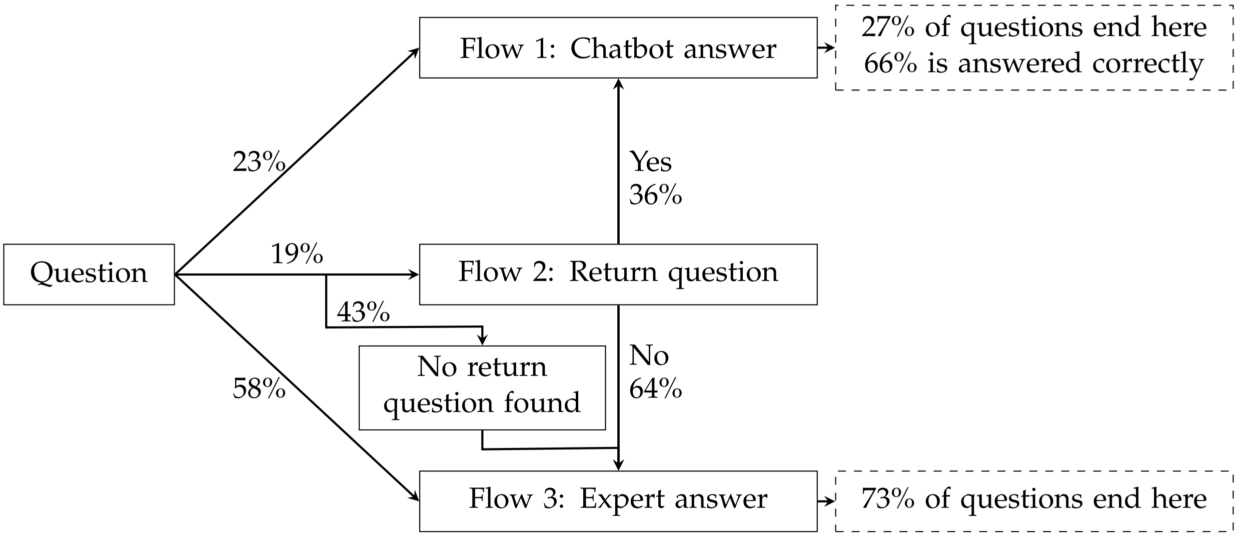


FIGURE 6.16: An overview of the results of the combination model chatbot implementation at a confidence level of 0.4 with a return question at a cosine similarity of 0.25.

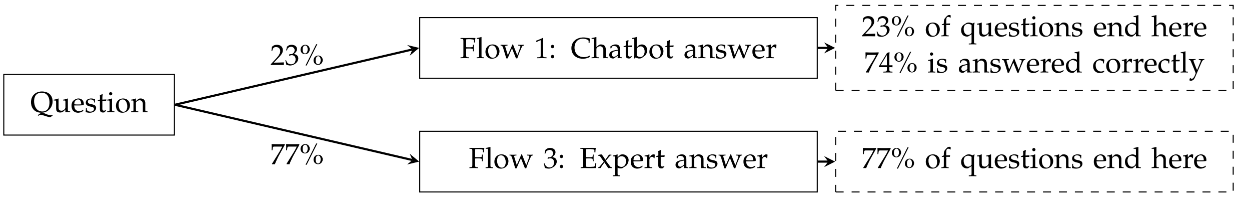

Question
23%
77%
Flow 1: Chatbot answer
Flow 3: Expert answer
23% of questions end here
74% is answered correctly
77% of questions end here

FIGURE 6.17: An overview of the results of the combination model chatbot implementation at a confidence level of 0.4 without return question.

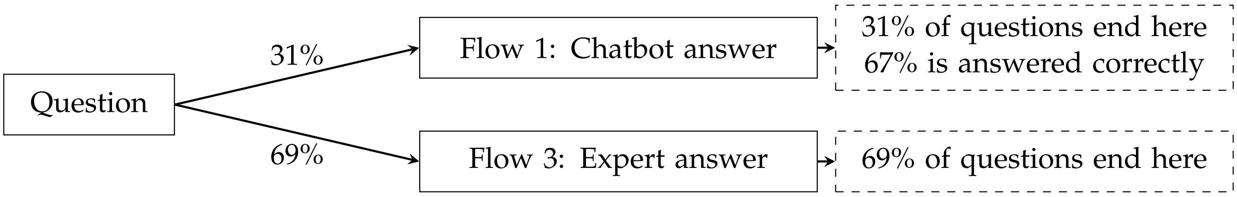

Question
31%
69%
Flow 1: Chatbot answer
Flow 3: Expert answer
31% of questions end here
67% is answered correctly
69% of questions end here

FIGURE 6.18: An overview of the results of the combination model chatbot implementation at a confidence level of 0.3 without return question.

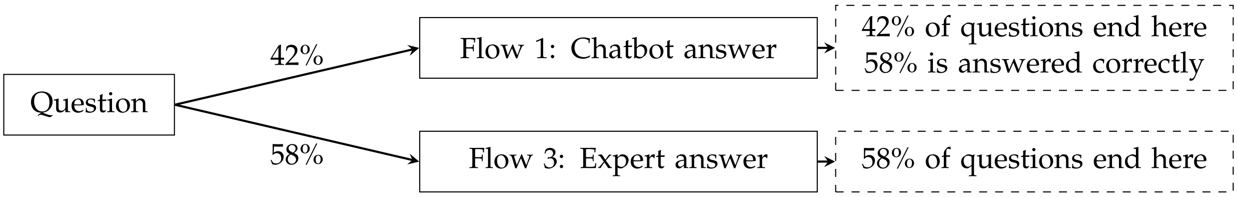

Question
42%
58%
Flow 1: Chatbot answer
Flow 3: Expert answer
42% of questions end here
58% is answered correctly
58% of questions end here

FIGURE 6.19: An overview of the results of the combination model chatbot implementation at a confidence level of 0.25 without return question.

In figure 6.20, we show the combination model implementation results in confusion matrices. On the top we have the cosine similarity scores, where questions with scores higher than 0.4 receive an answer and questions with lower scores do not. On the left axis we show if the correct answer to a question was selected (no matter if it was answered or not). The numbers in the confusion matrix including the return question add up to different numbers of correct/incorrect answers selected and different cosine score results, as asking the return question can improve cosine scores and alter the answer selected.

	$\cos \geq 0.4$	$\cos < 0.4$
Correct answer selected	45	47
Incorrect answer selected	23	140

(A) Including return question at 0.4

	$\cos \geq 0.4$	$\cos < 0.4$
Correct answer selected	43	55
Incorrect answer selected	15	142

(B) Excluding return question at 0.4

	$\cos \geq 0.3$	$\cos < 0.3$
Correct answer selected	53	45
Incorrect answer selected	26	131

(C) Excluding return question at 0.3

	$\cos \geq 0.25$	$\cos < 0.25$
Correct answer selected	62	36
Incorrect answer selected	45	112

(D) Excluding return question at 0.25

FIGURE 6.20: The confusion matrices for the implementation of the vector space model.

When comparing the confusion matrices in figure 6.20, we see that adding the return question in figure 6.20a leads to bad results compared to figure 6.20b. Adding the return question leads to two additional correct answers, but also 8 additional wrong answers. Therefore we prefer the combination model excluding the return question. When comparing figure 6.20b with figure 6.20c and 6.20d, it doesn't come down to a scientific decision but to a business decision. In figure 6.20b we answer only 58 questions in total and 15 of these are incorrect answers, in figure 6.20c we answer 79 questions in total of which 26 are incorrect answers and in figure 6.20d we answer 107 questions in total of which 45 are wrong answers.

We decide to choose the combination model excluding return question at a confidence level of 0.4. We choose this confidence level as lower confidence levels lead to more answers but also to even more wrong answers. For example, if the confidence level goes from 0.4 to 0.3, we give 10 more correct answers but also 11 more wrong answers.

6.5 Model comparisons

Figure 6.21 shows the results of all implemented models at all researched confidence levels.

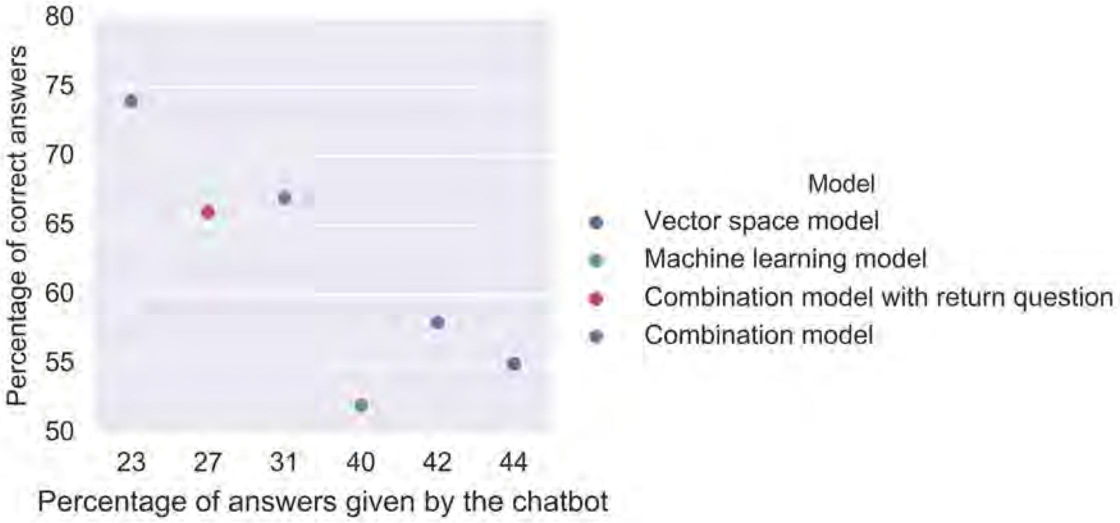


FIGURE 6.21: The results of the chatbot implementations of all models.

Comparing the results of the machine learning model with the results of the vector space model implementation, we see that we answer more questions and have a higher percentage of correctly answered questions using the vector space model. Therefore we find that the vector space model chatbot implementation performs better than the machine learning chatbot implementation at the selected confidence levels. We did not expect this finding as without any cut-off the vector space model answers 33% of questions correctly and the machine learning model answers 38% correctly. The difference is that the cosine similarity measure is a better measure for distinguishing between questions that we can and cannot answer than the XGBoost score. Therefore when implementing the confidence levels the vector space model performs better. This does not imply that the vector space model would perform better than implementations of the other machine learning models (like Random Forest), as the scores for these models might be better in distinguishing between questions that it can and cannot answer than the XGBoost score.

The combination model, that consists of using the XGBoost score for matching two questions and the cosine similarity to decide whether the chatbot should answer a question or not, performs similar to the vector space model looking at figure 6.21.

Here we see that if we would draw a line through the combination model points and extend it, we would reach to vector space model point. However, looking at figure 6.20b and 6.13a, we see that the vector space model implementation gives 18 correct answers more, but also 35 wrong answers more than the combination model at a confidence level of 0.4. We decide that the combination model with a confidence level of 0.4 is the best model for our business case.

Chapter 7

Conclusion & Discussion

The main objective of this graduation project was to build a chatbot to answer very complex questions, which often require even more complex answers, in a well defined domain (the financial domain). The focus was on only answering questions to which high-quality answers can be given. Three chatbot implementations based on different underlying models have been built and compared. The best performing chatbot answers 23% of user questions (the rest is sent through to human experts), and 74% of these questions are answered correctly. Success criteria for this graduation project were: a working chatbot system, use of NLP techniques, use of machine learning techniques and an evaluation of the chatbot system. These success criteria have all been met, but the answer quality of the chatbot is not very high.

We now discuss some reasons for the low chatbot answer quality. The first thing we notice is that there is a big gap between the results of the machine learning model on a Q2Q dataset and when implemented in a chatbot. This happens because in the implementation we select the question match with the highest XGBoost score for every test question and then evaluate. Our question-answer dataset is small and does not have an answer to many question variations and certainly not to very specific questions. When a question for which we do not have a matching question enters our chatbot we select the question with the highest XGBoost score for matching it. This means that we actually select the worst result (a non-matching question with a high matching score). If a question for which we do have a matching question enters our chatbot we also select the match with the highest XGBoost score, this is likely to be a good result. Now in total, if more than half of the questions do not have matching questions in our question-answer dataset it is logical that the machine learning chatbot implementation gives worse results than the model itself.

Another reason for the low chatbot answer quality is the specificity of the answers in the question-answer dataset. Since the question-answer dataset only contains answers given to a specific user question, sometimes the topic of the answer is correct but it is so specifically applied to a personal situation that it does not apply to the new user question.

We were surprised to find that the vector space model chatbot implementation gives better results than the machine learning model implementation. Without implementing a confidence level, that means that we answer every question with its closest match, the vector space model chatbot gave 33% correct answers and the machine learning model chatbot gave 38% correct answers. However, when implementing confidence levels it turns out that the cosine similarity measure is better at distinguishing between questions that the chatbot did and did not select the correct answer for. Due to this distinguishing capability the vector space model answers a bigger percentage of questions and answers a bigger percentage of these questions correctly. This does not imply that the vector space model implementation would give better results than all machine learning model implementations. Other models, like Random Forest, could have scores that are better than XGBoost at distinguishing between questions the model can and cannot answer in a chatbot framework.

The combination model, in which the machine learning model is used to select the best matching question and the cosine similarity is used to determine whether the chatbot gives an answer or not, is chosen as the best model. This choice was based on a business decision as the model is not obviously superior to the vector space model. The business decision was made while focusing on obtaining a high percentage of correct answers.

We have many recommendations for future research in this area as much is yet to be discovered.

1. Use a bigger question-answer dataset or a smaller domain
   We used a question-answer dataset containing 3153 questions after cleaning. We found that for the broad domain (financial domain) we had these 3153 did not cover many of the potential questions. For future research we advise to either extend the question-answer dataset or choose a smaller domain (e.g. only pension questions).

2. Create a model answer set
   Our chatbot gives back answers that were meant for a specific question. Often this leads to the answer not being appropriate for a new question in the same category. Therefore we advise to go through all the answers and rewrite them in a way that they can be applied to more questions.

3. Add more chat functionalities
   Our chatbot can answer a question, ask a yes/no return question and sent a question through to an expert. However you cannot have a back-and-forth conversation, as it works more like a question answering system. Therefore we advice to research how to integrate more chat functionalities in a chatbot meant for giving information.

4. Add hand-coded rules to the chatbot
   We wanted to research adding hand-coded rules to the chatbot as we can imagine that matching questions like "When will I get pension?" and "At what age do I get pension" is difficult for a cosine similarity or machine learning based chatbot. However when evaluating the test set we did not find many questions like this. Adding hand-coded rules might be very useful in a domain with less complicated and personalised questions.

5. Improve the Dutch NLP pipeline
   We created an NLP pipeline for Dutch, as there were no existing ones. In future work this pipeline can be improved and extended with more NLP techniques. An important point of improvement is the stemmer. We used the Dutch stemmer from the Python package Snowball. This word stemming improved our results, but the stemmer often transformed the words in a strange way. We advise to research the quality of this stemmer and/or build a dataset specific stemmer in the future.

6. Speed up the chatbot
   Our vector space model chatbot gives results in under a second, but the machine learning chatbot takes 15 seconds to answer on average (longer for longer questions). A similar version of this chatbot implemented in Scala takes under a second, so that is a solution. If you want to operate the chatbot in Python we

advise to parallel program the feature creation process for the machine learning chatbot.

7. Research ensemble learning
   When studying the machine learning models we used standard ensembling methods like XGBoost and Random Forest. However we did not try to ensemble our machine learning models themselves. Combinations of our machine learning models might give better results than one single model.

8. Research generative-based chatbots
   In our literature study we did some research on generative-based chatbots. We think that these are the chatbots of the future as they can give the impression that you are talking to a human being. However it is difficult to implement a generative-based chatbot on a specific domain as huge amounts of training data are needed.

Research done in the area of text mining is very time consuming as cleaning, creating and evaluating the data has to be done manually and consists of reading large amounts of text. Models applied on textual data take much time to evaluate. This is the case because checking if an answer to a question or a found document is correct has to be done manually, while if you are predicting a certain value checking if it is correct can be done computationally. Therefore we want to encourage others working on textual data to think about easy evaluation methods. In this thesis we built an evaluation framework, but we still had to manually read all selected answers and check there correctness.

The work done in this research adds to the body of work done in the area of chatbots. The main contributions are the NLP pipeline developed for Dutch questions which can easily be applied to text from other domains, applying the vector space model to question matching instead of document retrieval, researching the quality of different types of machine learning models on question matching and creating a new type of chatbot implementation using different chat flows including return questions.

Financial questions are highly complex and require reliable answers. Our success shows that even this type of questions can be handled by a chatbot. The NLP and machine learning techniques researched in this study can be applied to text mining problems in many domains, even domains with complex text data.

Appendix A

NLP: word lists

This appendix contains word lists which were used while applying NLP techniques to our dataset. Stop-word lists can be found in section A.1 and a compound word list can be found in section A.2.

A.1 Stop-word lists

The standard Dutch stop-word list built-in to the Python NLTK package contains:

- de
- en
- van
- ik
- te
- dat
- die
- in
- een
- hij
- het
- niet
- zijn
- is
- was
- op
- aan
- met
- als
- voor
- had
- er
- maar
- om
- hem
- dan
- zou
- of
- wat
- mijn
- men
- dit
- zo
- door
- over
- ze
- zich
- bij
- ook
- tot
- je
- mij
- uit
- der
- daar
- haar
- naar
- heb
- hoe
- heeft
- hebben
- deze
- u
- want
- nog
- zal
- me
- zij
- nu
- ge
- geen

- omdat
- iets
- worden
- toch
- al
- waren
- veel
- meer
- doen
- toen
- moet
- ben
- zonder
- kan
- hun
- dus
- alles
- onder
- ja
- eens
- hier
- wie
- werd
- altijd
- doch
- wordt
- wezen
- kunnen
- ons
- zelf
- tegen
- na
- reeds
- wil
- kon
- niets
- uw
- iemand
- geweest
- andere

Extra stop-words we added to this list, specifically for this dataset are:

- wel
- groeten
- vragen
- via
- mogelijk
- graag
- gaan
- weten
- informatie
- beste
- heer/mevrouw
- bent
- vraag
- kunt
- bedankt
- dank
- hallo
- krijgt
- goedenavond
- krijg
- kun
- wilt
- hebt
- gaat
- vriendelijke
- groet
- wij
- even
- we
- geachte
- hoi
- hi
- groetjes
- hoogachtend
- meneer
- mevrouw
- heer

A.2 Compound word list

The list with words to extract from compound words contains the following words:

- alleenstaand
- aow
- arbeidsongeschikt
- bank
- belast
- belegg
- bewijs
- bezwaar
- bijstand
- contact
- dekking
- erf
- huis
- huur
- hypotheek
- kind
- leeftijd
- levensloop
- lijfrente
- loon
- mantelzorg
- minimum
- nabestaand
- offerte
- ontslag
- ouderdom
- overbrugging
- overlijd
- overstap
- partner
- pensioen
- polis
- premie
- reis
- rekening
- rente
- samenleving
- schade
- scheid
- spaar
- studie
- telefon
- toeslag
- uitkering
- vakantie
- verzeker
- vrijwillig
- werkgever
- werklo
- werknemer
- woon
- ziek
- zorg
- zzp

A.3 Synonym list

The list of synonyms contains words in table A.1.

Model word	**Synonym 1**	**Synonym 2**	**Synonym 3**	**Synonym 4**	**Synonym 5**	**Synonym 6**
pensioen	pension	pesion	pesioen			
partner	man	vrouw				
pensioenopbouw	opbouw	bouw				
tijdens	terwijl	tegenlijk	tegelijk			
ontvang	ontvangen	ontving	ontvangt			
pensioenoverzicht	upo	pensioenbrief	uniform			
voorwaardelijk	voorwaardelijke					
waardeoverdracht	overdracht	overgedragen				
invloed	effect	gevolg	gevolgen	effecten		
recht	rechtmatig	wettig	aanspraak			
dekkingsgraad	beleidsdekkingsgraad	dekkinggraad				
gebeurt	gebeuren	gebeurde				
herplaatsingstoelage	hpt	herplaatsing				
buitenland	buiteland	emigratie	emigreren	belgie	duitsland	meer landen
informatie	info	kennisgeving	inlichting	mededeling	kennis	inzicht
ww	wwuitkering	werkloosheidsuitkering				
inkomen	loon	salaris	vergoeding	broodwinning		
invaliditeitspensioen	arbeidsongeschiktheidspensioen	ip				
betekent	betekend					
pensioenregeling	pensioenregels	pensioenreglement				
bijdrage	contributie	afdracht	premie			
ex	expartner	exman	exvrouw			
indexatie	indexering					
hoogte	hoog	hoeveel				
aow	overheidspensioen	sow				
loonheffingskorting	loonheffing	loonbelasting				
aanmelden	meld	meldt	aanvragen			
fiscale	fiscaal	belasting	belastingtechnisch	fiscus		
woon	gevestigd	verblijf	verblijven			
werken	gewerkt					
arbeidsongeschiktheid	arbeidsongeschikt					
overlijden	gestorven	overleden	overlijdt			
berekend	vastgesteld					
uitkering	toeslag	tegemoetkoming	toelage	bijstand		
werkgever	bedrijf	baas				
ingehouden	gekort	verlaagd	korten			
zorgverzekeringswet	zvw	volksverzekeringen				
verevening	verdeling					
anw	nabestaandenpensioen	nabestaandenuitkering	anwpensioen	overlijdensuitkering	anwuitkering	anwcompensatie
jaaropgave	jaaroverzicht					
wijzigen	veranderen	verandert	wijziging	verandering		

looptijd	duur	duurtijd	loopt			
pensioenpremie	premie					
bijsparen	inleggen					
pensioenfonds	fonds	abp	spw	pfzw	bpfb	meer pensioenfondsen
verlaging	korting	mindering				
werk	arbeid	job	baan			
65	pensioenrekenleeftijd	vijfenzestig	pensioenleeftijd	aowleeftijd	65ste	
diensttijd	werkgevers	werkverleden				
wao	arbeidsongeschiktheidsuitkering					
gehuwd	getrouwd					
aanvulling	aanvullen					
echtscheiding	scheiden	scheiding				
burgerservicenummer	bsn	sofi	sofinummer			
bijspaar	bijsparen	ruimte				
afkoop	uitbetaling	afkoopbedrag	afkopen			
deeltijd	deeltijdpensioen	vroegpensioen	keuzepensioen			
opgebouwde	opbouwen					
nabestaande	nabestaanden					
levensloopverlof	levensloop	levenslooppregeling				
iban	rekening	bankrekening	banknummer	rekeningnummer		
wia	ziekteuitkering					

TABLE A.1: Table of synonyms

Bibliography

[1] Alfred V Aho and Jeffrey D Ullman. *The theory of parsing, translation, and compiling*. Prentice-Hall, Inc., 1972.

[2] Tomoyosi Akiba, Katunobu Itou, and Atsushi Fujii. “Question Answering Using" Common Sense" and Utility Maximization Principle.” In: *NTCIR*. 2004.

[3] Amara D. Angelica. “How Watson Works: a conversation with Eric Brown, IBM Research Manager”. In: *Kurzweil Accelerating Intelligence* (2011). URL: `http://www.kurzweilai.net/how-watson-works-a-conversation-with-eric-brown-ibm-research-manager`.

[4] Ricardo Baeza-Yates and William Bruce Frakes. *Information retrieval: data structures & algorithms*. Prentice Hall, 1992.

[5] Satanjeev Banerjee and Alon Lavie. “METEOR: An automatic metric for MT evaluation with improved correlation with human judgments”. In: *Proceedings of the acl workshop on intrinsic and extrinsic evaluation measures for machine translation and/or summarization*. Vol. 29. 2005, pp. 65–72.

[6] Alistair Barr. *Google plans new, smarter messaging app*. 2015. URL: `https://www.wsj.com/article_email/google-plans-new-smarter-messaging-app-1450816899-lMyQjAxMTA1OTI5MjUyMDI5Wj` (visited on 01/24/2017).

[7] Yoshua Bengio et al. “A neural probabilistic language model”. In: *Journal of machine learning research* 3.Feb (2003), pp. 1137–1155.

[8] Adam Berger et al. “Bridging the lexical chasm: statistical approaches to answer-finding”. In: *Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval*. ACM. 2000, pp. 192–199.

[9] Eric Brill. “A simple rule-based part of speech tagger”. In: *Proceedings of the workshop on Speech and Natural Language*. Association for Computational Linguistics. 1992, pp. 112–116.

[10] Peter F Brown et al. “Class-based n-gram models of natural language”. In: *Computational linguistics* 18.4 (1992), pp. 467–479.

[11] Chris Buckley. *Implementation of the SMART information retrieval system*. Tech. rep. Cornell University, 1985.

[12] Robin D Burke et al. “Question answering from frequently asked question files: Experiences with the faq finder system”. In: *AI magazine* 18.2 (1997), p. 57.

[13] Claire Cardie et al. “Examining the role of statistical and linguistic knowledge sources in a general-knowledge question-answering system”. In: *Proceedings of the sixth conference on Applied natural language processing*. Association for Computational Linguistics. 2000, pp. 180–187.

[14] Stanley F Chen and Joshua Goodman. "An empirical study of smoothing techniques for language modeling". In: *Proceedings of the 34th annual meeting on Association for Computational Linguistics*. Association for Computational Linguistics. 1996, pp. 310–318.

[15] Kenneth Ward Church. "A stochastic parts program and noun phrase parser for unrestricted text". In: *Proceedings of the second conference on Applied natural language processing*. Association for Computational Linguistics. 1988, pp. 136–143.

[16] Mark Clark. *A Chatbot Framework*. 2016. URL: `http://info.contactsolutions.com/digital-engagement-blog/a-chatbot-framework` (visited on 02/01/2017).

[17] K. M. Colby. *Computer Model of Thought and Language*. Ed. by R. C. Schank. W. H. Freeman and Company, 1973.

[18] Greg Corrado. *Computer, respond to this email*. 2015. URL: `https://research.googleblog.com/2015/11/computer-respond-to-this-email.html` (visited on 02/16/2017).

[19] Doug Cutting et al. "A practical part-of-speech tagger". In: *Proceedings of the third conference on Applied natural language processing*. Association for Computational Linguistics. 1992, pp. 133–140.

[20] Simon Dennis et al. "Introduction to latent semantic analysis". In: *Slides from the tutorial given at the 25th Annual Meeting of the Cognitive Science Society, Boston*. 2003.

[21] Alan Descoins. *Chatbots and automated online assistants*. 2015. URL: `https://tryolabs.com/blog/2015/07/23/chatbots-and-automated-online-assistants/` (visited on 02/09/2017).

[22] Kjell Magne Fauske. *Example: Neural network*. 2006. URL: `http://www.texample.net/tikz/examples/neural-network/` (visited on 02/27/2017).

[23] David Ferrucci et al. "Building Watson: An overview of the DeepQA project". In: *AI magazine* 31.3 (2010), pp. 59–79.

[24] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. *The elements of statistical learning*. Vol. 1. Springer series in statistics Springer, Berlin, 2001.

[25] Shalini Ghosh et al. "Contextual LSTM (CLSTM) models for Large scale NLP tasks". In: *arXiv preprint arXiv:1602.06291* (2016).

[26] Ben Goertzel and Cassio Pennachin. *Artificial general intelligence*. Vol. 2. Springer, 2007.

[27] D. C. Gondek et al. "A framework for merging and ranking of answers in DeepQA". In: *IBM Journal of Research and Development* 56.3.4 (2012), pp. 14–1.

[28] Bert F. Green Jr. et al. "Baseball: An Automatic Question-answerer". In: *Papers Presented at the May 9-11, 1961, Western Joint IRE-AIEE-ACM Computer Conference*. IRE-AIEE-ACM '61 (Western). New York, NY, USA: ACM, 1961, pp. 219–224. DOI: `10.1145/1460690.1460714`. URL: `http://doi.acm.org/10.1145/1460690.1460714`.

[29] Seth Grimes. *Unstructured data and the 80 percent rule*. 2008. URL: `https://breakthroughanalysis.com/2008/08/01/unstructured-data-and-the-80-percent-rule/` (visited on 01/23/2017).

[30] Jiatao Gu et al. “Incorporating copying mechanism in sequence-to-sequence learning”. In: *arXiv preprint arXiv:1603.06393* (2016).

[31] Sanda M Harabagiu et al. “Falcon: Boosting knowledge for answer engines”. In: (2000).

[32] Peter A Heeman. “POS tags and decision trees for language modeling”. In: *Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora*. 1999, pp. 129–137.

[33] Eduard Hovy et al. “Toward semantics-based answer pinpointing”. In: *Proceedings of the first international conference on Human language technology research*. Association for Computational Linguistics. 2001, pp. 1–7.

[34] Jason L. Hutchens and Michael D. Alder. “Introducing MegaHAL”. In: *Proceedings of the Joint Conferences on New Methods in Language Processing and Computational Natural Language Learning*. NeMLaP3/CoNLL ’98. Stroudsburg, PA, USA: Association for Computational Linguistics, 1998, pp. 271–274. ISBN: 0-7258-0634-6. URL: `http://dl.acm.org/citation.cfm?id=1603899.1603945`.

[35] Sébastien Jean et al. “On Using Very Large Target Vocabulary for Neural Machine Translation”. In: *CoRR* abs/1412.2007 (2014). URL: `http://arxiv.org/abs/1412.2007`.

[36] Rudolf Kadlec, Martin Schmid, and Jan Kleindienst. “Improved deep learning baselines for ubuntu corpus dialogs”. In: *arXiv preprint arXiv:1510.03753* (2015).

[37] Anjuli Kannan et al. “Smart Reply: Automated Response Suggestion for Email”. In: *Proceedings of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD) (2016).* 2016. URL: `https://arxiv.org/pdf/1606.04870v1.pdf`.

[38] Harksoo Kim and Jungyun Seo. “Cluster-based faq retrieval using latent term weights”. In: *IEEE Intelligent Systems* 23.2 (2008), pp. 58–65.

[39] Kyoung jae Kim. *Financial time series forecasting using support vector machines*. 2003. URL: `https://www.slideshare.net/HouwLiongThe/time-series-forecasting-using-svm` (visited on 02/24/2017).

[40] Julian Kupiec. “MURAX: A robust linguistic approach for question answering using an on-line encyclopedia”. In: *Proceedings of the 16th annual international ACM SIGIR conference on Research and development in information retrieval*. ACM. 1993, pp. 181–190.

[41] Thomas K Landauer. *Latent semantic analysis*. Wiley Online Library, 2006.

[42] Jiwei Li et al. “A diversity-promoting objective function for neural conversation models”. In: *arXiv preprint arXiv:1510.03055* (2015).

[43] Jimmy Lin et al. “What Makes a Good Answer? The Role of Context in Question Answering”. In: *PROCEEDINGS OF INTERACT 2003*. 2003, pp. 25–32.

[44] Chia-Wei Liu et al. “How NOT to evaluate your dialogue system: An empirical study of unsupervised evaluation metrics for dialogue response generation”. In: *arXiv preprint arXiv:1603.08023* (2016).

[45] Ryan Lowe et al. “The ubuntu dialogue corpus: A large dataset for research in unstructured multi-turn dialogue systems”. In: *arXiv preprint arXiv:1506.08909* (2015).

[46] Minh-Thang Luong and Christopher D Manning. "Stanford neural machine translation systems for spoken language domains". In: *Proceedings of the International Workshop on Spoken Language Translation*. 2015.

[47] Minh-Thang Luong et al. "Addressing the rare word problem in neural machine translation". In: *arXiv preprint arXiv:1410.8206* (2014).

[48] Steven Lytinen and Noriko Tomuro. "The use of question types to match questions in FAQFinder". In: *AAAI Spring Symposium on Mining Answers from Texts and Knowledge Bases*. 2002, pp. 46–53.

[49] Christopher D Manning, Prabhakar Raghavan, Hinrich Schütze, et al. *Introduction to information retrieval*. Vol. 1. 1. Cambridge university press Cambridge, 2008.

[50] Tomas Mikolov, Wen-tau Yih, and Geoffrey Zweig. "Linguistic Regularities in Continuous Space Word Representations." In: *Hlt-naacl*. Vol. 13. 2013, pp. 746–751.

[51] Tomas Mikolov et al. "Distributed representations of words and phrases and their compositionality". In: *Advances in neural information processing systems*. 2013, pp. 3111–3119.

[52] Tomas Mikolov et al. "Efficient estimation of word representations in vector space". In: *arXiv preprint arXiv:1301.3781* (2013).

[53] George A Miller. "WordNet: a lexical database for English". In: *Communications of the ACM* 38.11 (1995), pp. 39–41.

[54] Amit Mishra and Sanjay Kumar Jain. "A survey on question answering systems with classification". In: *Journal of King Saud University-Computer and Information Sciences* 28.3 (2016), pp. 345–361.

[55] Catarina Moreira. "Learning to rank academic experts". PhD thesis. Master Thesis, Technical University of Lisbon, 2011.

[56] Martha Palmer, Daniel Gildea, and Nianwen Xue. "Semantic role labeling". In: *Synthesis Lectures on Human Language Technologies* 3.1 (2010), pp. 1–103.

[57] Kishore Papineni et al. "BLEU: a method for automatic evaluation of machine translation". In: *Proceedings of the 40th annual meeting on association for computational linguistics*. Association for Computational Linguistics. 2002, pp. 311–318.

[58] Nicole Perlroth. "Siri Was Born A Man And Other Things You Don't Know About Apple's New Personal Assistant". In: *Forbes* (Oct. 2011). URL: `http://www.forbes.com/sites/nicoleperlroth/2011/10/12/siri-was-born-a-man-and-other-things-you-dont-know-about-apples-new-personal-assistant/#666618592ed8`.

[59] Perseus. *Classification parameter optimization*. 2015. URL: `http://www.coxdocs.org/doku.php?id=perseus:user:activities:matrixprocessing:learning:classificationparameteroptimization&do=` (visited on 02/21/2017).

[60] Gerasimos Potamianos and Frederick Jelinek. "A study of n-gram and decision tree letter language modeling methods". In: *Speech Communication* 24.3 (1998), pp. 171–192.

[61] John Prager et al. "Question answering by predictive annotation". In: *Advances in Open Domain Question Answering*. Springer, 2008, pp. 307–347.

[62] M Ross Quillan. *Semantic memory*. Tech. rep. DTIC Document, 1966.

[63] John W Ratcliff and David E Metzener. "Pattern-matching-the gestalt approach". In: *Dr Dobbs Journal* 13.7 (1988), p. 46.

[64] Alan Ritter, Colin Cherry, and William B Dolan. "Data-driven response generation in social media". In: *Proceedings of the conference on empirical methods in natural language processing*. Association for Computational Linguistics. 2011, pp. 583–593.

[65] Ronald Rosenfeld. "Adaptive statistical language modeling: A maximum entropy approach". PhD thesis. Department of the Navy, Naval Research Laboratory, 2005.

[66] Gerard Salton. "The SMART retrieval systemexperiments in automatic document processing". In: (1971).

[67] Gerard Salton and Michael J McGill. "Introduction to modern information retrieval". In: (1986).

[68] Iulian V Serban et al. "Building end-to-end dialogue systems using generative hierarchical neural network models". In: *arXiv preprint arXiv:1507.04808* (2015).

[69] Lifeng Shang, Zhengdong Lu, and Hang Li. "Neural responding machine for short-text conversation". In: *arXiv preprint arXiv:1503.02364* (2015).

[70] Eriks Sneiders. "Automated faq answering: Continued experience with shallow language understanding". In: *Question Answering Systems. Papers from the 1999 AAAI Fall Symposium*. 1999, pp. 97–107.

[71] Alessandro Sordoni et al. "A neural network approach to context-sensitive generation of conversational responses". In: *arXiv preprint arXiv:1506.06714* (2015).

[72] Mark Stevenson and Yorick Wilks. "Word sense disambiguation". In: *The Oxford Handbook of Comp. Linguistics* (2003), pp. 249–265.

[73] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. "Sequence to sequence learning with neural networks". In: *Advances in neural information processing systems*. 2014, pp. 3104–3112.

[74] Noriko Tomuro. "Question terminology and representation for question type classification". In: *Terminology. International Journal of Theoretical and Applied Issues in Specialized Communication* 10.1 (2004), pp. 153–168.

[75] A. M. Turing. "Computing Machinery and Intelligence". In: *Mind*. 236th ed. Vol. 59. Oxford University Press, Oct. 1950.

[76] Oriol Vinyals and Quoc Le. "A neural conversational model". In: *arXiv preprint arXiv:1506.05869* (2015).

[77] Ellen M Voorhees and L Buckland. "Overview of the TREC 2003 Question Answering Track." In: *TREC*. Vol. 2003. 2003, pp. 54–68.

[78] Richard S. Wallace. *The Elements of AIML Style*. ALICE A. I. Foundation, Inc., 2003.

[79] Shaojun Wang et al. "Combining statistical language models via the latent maximum entropy principle". In: *Machine Learning* 60.1-3 (2005), pp. 229–250.

[80] Joseph Weizenbaum. "ELIZA - a computer program for the study of natural language communication between man and machine". In: *Communications of the ACM* 9.1 (Jan. 1966), pp. 36–45. URL: http://s3.amazonaws.com/academia.edu.documents/31085335/ElizaScript.pdf?AWSAccessKeyId=AKIAIWOWYYGZ2Y53UL3A&Expires=1486718310&Signature=xQgYrGh%2BBib4QkdXadoxjh2dLCA%3D&response-content-disposition=inline%3B%20filename%3DELIZA_a_computer_program_for_the_study_o.pdf.

[81] Steven D Whitehead. "Auto-FAQ: An experiment in cyberspace leveraging". In: *Computer Networks and ISDN Systems* 28.1-2 (1995), pp. 137–146.

[82] Terry Winograd. "Procedures as a representation for data in a computer program for understanding natural language". PhD thesis. Massachusetts Institute of Technology, Jan. 1971.

[83] Terry Winograd. *Understanding Natural Language*. New York: Academic Press, 1972.

[84] Wilson Wong. "Practical approach to knowledge-based question answering with natural language understanding and advanced reasoning". In: *arXiv preprint arXiv:0707.3559* (2007).

[85] W. A. Woods. "Progress in Natural Language Understanding: An Application to Lunar Geology". In: *Proceedings of the June 4-8, 1973, National Computer Conference and Exposition*. AFIPS '73. New York, NY, USA: ACM, 1973, pp. 441–450. DOI: 10.1145/1499586.1499695. URL: http://doi.acm.org/10.1145/1499586.1499695.

[86] Chung-Hsien Wu, Jui-Feng Yeh, and Ming-Jun Chen. "Domain-specific FAQ retrieval using independent aspects". In: *ACM Transactions on Asian Language Information Processing (TALIP)* 4.1 (2005), pp. 1–17.

[87] Jinxi Xu and W Bruce Croft. "Query expansion using local and global document analysis". In: *Proceedings of the 19th annual international ACM SIGIR conference on Research and development in information retrieval*. ACM. 1996, pp. 4–11.

[88] Kaisheng Yao, Geoffrey Zweig, and Baolin Peng. "Attention with intention for a neural network conversation model". In: *arXiv preprint arXiv:1510.08565* (2015).

[89] Victor Zue et al. "PEGASUS: A spoken dialogue interface for on-line air travel planning". In: *Speech Communication* 15.3-4 (1994), pp. 331–340.