Hochschule Darmstadt
- Department of Computer Science -

Intelligent Bot

an interactive man-machine communication system for incoming international students

for attainment of the academic degree of

Bachelor of Science (B.Sc.)

Presented by

Emmanuel Günther

Advisor: Prof. Dr. Bettina Harriehausen-Mühlbauer
Co-advisor: Dr. Hrafn Loftsson

Declaration of authorship

I certify that the work presented here is, to the best of my knowledge and belief, original and the result of my own investigations, except as acknowledged, and has not been submitted, either in part or whole, for a degree at this or any other university.

Dreieich, February 7, 2013

Signature

Abstract

This work deals with the topic Natural Language Processing (NLP). The task is to create a bot program, which returns an intelligent answer to the questions asked. Incentive for this work is the idea of creating an information system that can be used by exchange students to get answers to unresolved questions or more information on the host institution.

In preparation for this work, a program was developed that uses tokenization, spelling correction, keyword identification, the creation of base forms of the keywords and the identification of possible synonyms for working with a given question. Then it compares the keywords and synonyms with the keywords of questions and their answers of a previously given database. This comparison is done by weighting the input question with the questions in the database under usage of the Vector Space Model and returning the answer with the highest weighting.

It is also ensure that if an appropriate answer is not found within the database by using the previously mentioned method of weighting, suitable information will be sought on the internet. This is done by looking for pages and subpages of a pre-defined address, which were indexed before and will be searched for keywords.

This paper describes the implementation and use of technologies for these aspects of the program and concentrates more precisely on the pros and cons of these in comparison to previously applied technologies and alternative approaches. Furthermore the results of user testing are described.

Abstract

Diese Arbeit behandelt das Thema Natural Language Processing (NLP). Die Aufgabe besteht darin, ein Bot Programm zu erstellen, welches auf gestellte Fragen eine intelligente Antwort zurückliefert. Anreiz für diese Arbeit gibt die Überlegung, ein Informationssystem zu kreieren, welches von Austauschstudenten genutzt werden kann, um ungeklärte Fragen beantwortet oder weitere Informationen zur Gasthochschule zu bekommen.

Im Vorfeld dieser Arbeit wurde ein Programm entwickelt, welches mit Hilfe von Tokenisierung, Korrektur von Schreibfehlern, Keywortidentifikation, der Bildung von Basisformen der Wörter und der Erkennung möglicher Synonyme eine gestellte Frage bearbeitet. Es vergleicht diese Keywörter und Synonyme dann mit Keywörtern von Fragen und deren Antworten in einer zuvor gegebenen Datenbank. Dieser Vergleich wird durch Gewichtung der Ausgangsfrage mit den Fragen in der Datenbank unter Verwendung des Vector Space Models durchgeführt und die Antwort mit der höchsten Gewichtung zurückgegeben.

Es wird auch dafür gesorgt, falls die Datenbank keine treffende Antwort auf die Frage durch die zuvor genannte Gewichtung findet, dass außerhalb des Programmes nach einer möglichen Hilfestellung oder Information im Internet gesucht wird. Dies geschieht durch die Suche nach Seiten und Unterseiten einer bestimmten, vorher angegebenen Adresse, welche zuvor indiziert wurden und eine Suche nach Keywörtern durchgeführt wird.

Diese Arbeit beschreibt die Umsetzung und die Nutzung von Technologien für diese einzelnen Teilbereiche des Programms und schildert genauer, welche Vor- und Nachteile es bei diesen gibt im Vergleich zu den zuvor eingesetzten Technologien und Alternativen. Des Weiteren werden auch die Ergebnisse eines Benutzertests beschrieben.

Contents

Figures

Tables

1 Introduction

In most international offices the staff receives a lot of e-mails from incoming students and potential students which want to study abroad with questions regarding university related issues. Most of these questions range from how to apply at the university to offered courses and questions concerning the life in and around the university. Some members of the faculty answer these e-mails by using predefined text blocks, which they copy in email responses. This method saves time, but it is still very time consuming, because of the amount of incoming e-mails every day. The process of answering those e-mails takes up to several hours a day.

The same problem of answering e-mails exists at the international office of the department of computer science at the University of Applied Science in Darmstadt. The international representatives get many e-mails from exchange students with questions concerning university affairs and information they would like to have before arriving. In most cases many incoming exchange students ask the same questions or the same kind of questions.

More and more systems were introduced in the last years to the market providing a direct electronic answer to those who have questions. These systems give users the possibility to type in a question and provide an answer or advise the user about how to find the needed information.

For the problem of answering a lot of e-mails from students or potential students that kind of a system would be very helpful and could prevent faculty staff members of replying to so many emails. In order for this system to be able to give answers on questions from incoming students, several goals are defined.

The first goal for the system is to be easily integratable and usable on every university environment worldwide. In order to achieve this goal, the system should be implemented in a generalized way to easily provide the necessary data and configuration parameters - e.g. prede-

fined answers and the name of the database. The second goal is to get a wider range of available information; therefore the system should not only have a database to search for information and give an answer to the user. A second source will be available within the program as well - the biggest database on earth - the internet.

The general purpose of the system is to make having to write emails to university personnel unnecessary in most cases.

The starting point for developing the program was a previous development done by a student group project of the NLP master course and background information on other systems working in the problem area.

2 Background

The project system of the NLP master course was a bot, which could answer questions of incoming students and potential students that want to study abroad. This bot is a Java Servlet program and runs on an Apache Tomcat web server. The bot first processes the given question in the field of natural language processing. It tokenizes the question into single words and the unnecessary words are filtered out by using a list of "stop words". The remaining words are then taken as the keywords for the question.

The keywords are then run through a lemmatization and synonyms are created. The keywords and synonyms are ranked against the keywords in the database using the vector space model. The answer of the thus found best matching question in the database is then returned to the user if it has a value higher than a previously defined threshold.

If no question was found in the database, the system performs a web search on specified web pages using the SearchBlox web service. If the SearchBlox web search returns no results, the user is provided with the means of sending an email directly to a previously defined staff member.

One other system is the Jabberwacky AI which is using contextual pattern matching techniques to find the most appropriate response on a given text of a user. The system stores all questions and user responses in a self-learning database. The Jabberwacky AI does not use any other external source of information but its own database.

"The 'general AI' of Jabberwacky stores everything everyone has ever said, and finds the most appropriate thing to say using contextual pattern matching techniques. In speaking to you it uses just that learnt material, and borrows a little bit of your intelligence as it learns more. With no hard-coded rules, it relies entirely on the principles of feedback." [Jab]

The self-learning function of the Jabberwacky AI needs a lot of time to learn enough in order to answer questions appropriately. But there is another bot on the internet. Her name is called "Alice". This bot uses the Artificial Intelligence Markup Language (AIML) as its database including pattern-based knowledge.

"The Artificial Intelligence Markup Language is a derivative of XML (Extensible Markup Language) [...]. Its goal is to enable pattern-based, stimulus-response knowledge content to be served, received and processed on the Web and offline in the manner that is presently possible with HTML and XML." [Wal]

"Alice" also has a self-learning function which is not really useful for the design of an intelligent question and answer bot because it can't use users to get information and saving them to its database. The users which will using the program want to get more information they do not know. Most of the other bots are using pattern matching for matching the given question or answer to a response in the database. The vector space model gets better results than pattern matching and is better suitable for differently worded questions with the same answers. That is why the vector space model is the better suited solution for the planned program.

New design ideas for developing a system for giving information and answers to exchange students are described in the next chapter.

3 Design

With the new design, based on the group project of the natural language processing course, two major changes are introduced to the initial system. The first one is changing the keyword extraction. The NLP group used an extraction of "stop words" from the list of word tokens. The “stop word” list contains only English words and is thus only applicable if the question and answer system is used in English. In order to design a generalized system that can be run on a wider range of systems, the extraction of keywords should not be based on a hardcoded deletion of stop words. There is a much better way of getting keywords in a more general way then just matching them against a stop words list.

With using a part-of-speech tagger (POS Tagger) the system analyzes the question the user sends and marks all words with their matching tag. It is then easy to delete all unnecessary tags as keywords and only the useful and meaningful keywords remain in the keyword pool. The used API for pos tagging is the POS Tagger from the Stanford Natural Language Processing Group. It uses the Penn Treebank English POS tag set for tagging the words. For a generalized system the pos tagger has also the advantage of also being able to work wiith a wider range of laguages such as Englisch, Arabic, Chinese and German.

The SearchBlox web service that the initial system uses in not applicable for an usage with a generalized multilingual system. The web service also has a limitaion of the free requests per day and is only configurable through the web page of the service. It would be a better to have all configuration parameters in one place and not scattered in many places.

In order for the program to work as intended, we need a service that has the same web searching capabilities as SearchBlox. The SearchBlox web services uses the Apache Lucene API for indexing and searching web pages.

The second major change to the initial design is the integration of the Apache Lucene API

directly into the system, so a predefined wep page can be searched with it. To get all the data from a web site, the system gets also a web crawler to mine the data from the web site and find links to all of the other pages of that web site. In order for the system to get the information within a website, a web crawler is used to mine the data and find links to all pages within the given website. A second Java Servlet is included that updates the system with the information gathered from data mining and indexing the web page. PDF documents can also be indexed and searched by the crawler thus allowing the gathering of a broader range of information.

To avoid inputting settings within different files and services, only one configuration file will be created. This text file is contained within the system and is read upon the start of the servlet. The configuration includes all settings of the program – for example the URL of the web page, the document file formats mined by the web crawler and predefined answers.

This new system design approach enables better integrating options within a wider range of different environments compared to the system developed by the project group of the natural language processing master course.

The new system design and architecture is described in detail within the next chapter of this paper.

4 Architecture

The following flowchart shows the system architecture. All requests are processed in the order defined by this architecture.

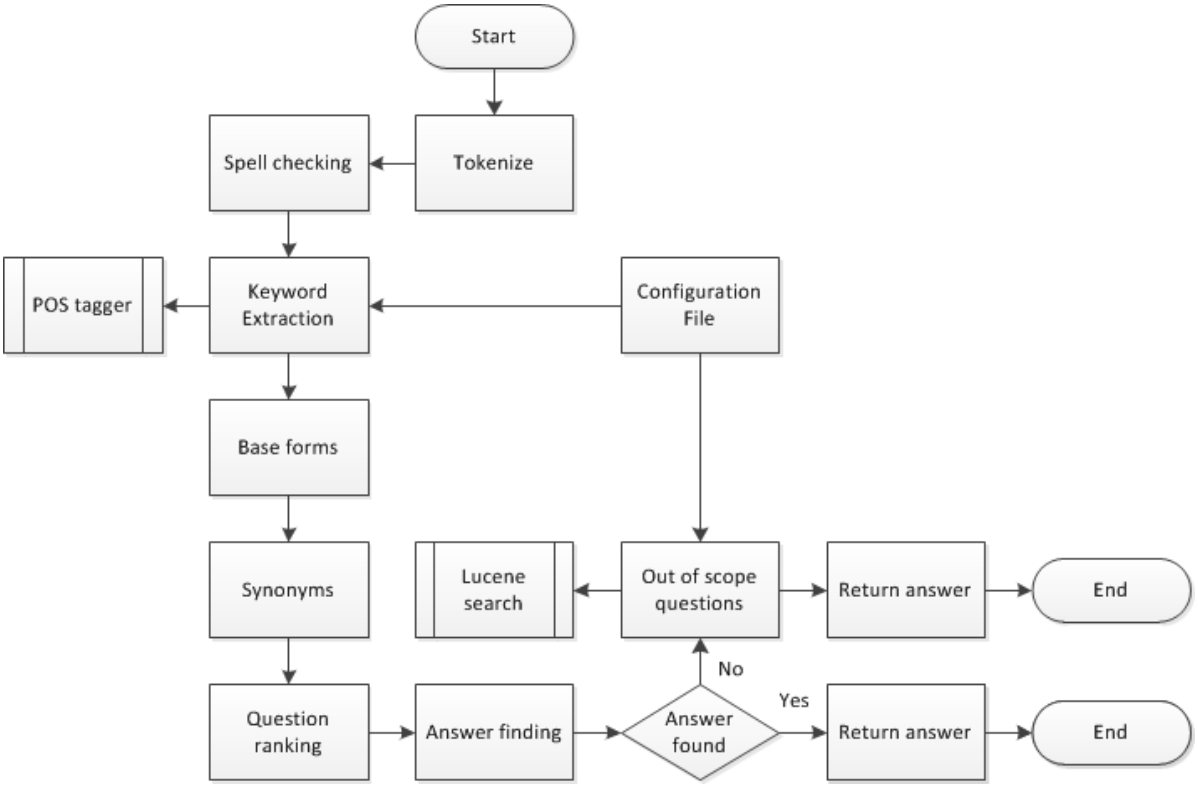

Figure 4.1: System Architecture Flowchart

The working process of the system starts with the tokenization of the question.

4.1 Tokenizer

A Tokenizer is a program that disassembles plain text into logical sequences, which are called tokens. The Tokenizer gets the question as a string and breaks it into a word list. In general a Tokenizer is program to disassemble plain text into logical sequences which are called tokens.

The tokenization process removes all punctuation marks as those are not relevant for the question rank. Then the question string is split into single word tokens which are listed in a Java ArrayList. All word tokens are then transformed into lowercase words, so the word list contains only lowercase tokens and thus avoiding token repetitions because of case differences. The thus created word token list is then forwarded to the spell checker.

4.2 Spell checker

The spell checker used for the implementation uses a list of all words from the database containing questions and answers. When a spell check for a word is done, the complete list is searched for the most similar word. This search method is done using Levenshtein distance and Keyboard Proximity for measuring similarities.

The Levenshtein distance is a string metric that calculates the differences between two words. To calculate the distance, the single-character edits that are required to get to the second word from the first one are counted. A smaller distance number means that the words are more similar.

The Keyboard Proximity method is based on the fact that a large percentage of spelling mistakes are typographical errors that occur because of typing slips on the keyboard. The input words are tested by changing single characters with ones that are placed nearby on the keyboard.

By using these two methods, a search is performed for the most similar word in the database of all single words, contained in the question and answer database.

If a word is found, the word token and the found word are converted into lowercase letters; if both words are equal then the user has inputted the word in the right spelling. In this case the word is wrapped in the "<plain>" and "</plain>" tags. If the two words don't match after

the conversion in lowercase, the found most similar word is wrapped in the "<suggestion>" and "</suggestion>" tags to indicate that the method has generated a suggestion for the word.

After getting these strings the spell checker uses the word in case the tags are "<plan>" and "</plain>" and also uses the word wrapped with the tags "<suggestion>" and "</suggestion>".

The now correct spelled word tokens are forwarded to the keyword extraction with the pos tagger.

4.3 Keyword extraction

A keyword extraction is made because to get a better impact on the ranking process without using unnecessary word tokens as keywords.

The keyword extraction is using a pos tagger developed by the Stanford Natural Language Processing Group. A pos tagger is in general a software which reads a text in any language and assigns and marks the words or tokens with their part of speech. For example noun, verb, adjective. The pos tagger from the Stanford Natural Language Processing Group is a log-linear part of speech tagger which uses the Penn Treebank POS tag set.

For working with the pos tagger the word token list is rebuilt into a string. For building this string the single word tokens of the list are appended to a word string. The pos tagger could only tag words within one string. The method to work with a list of word tokens wasn't implemented yet.

The returned string is now splitted into single word objects. To do this the words and their tag name are saved to a list of word objects where a word object contains the word string and the tag name. From this list all the word objects are removed which contain a not useful tag for a keyword. The list of not useful tags were defined with a list of stop words. Therefor a list of stop words was tagged by the pos tagger before. The list of removable word tags consists of tags which was more than one time in the tagged stop word list.

The complete stop word list:

able, about, across, after, all, almost, also, am, among, an, and, any, are, as, at, be, because, been, but, by, can, cannot, could, dear, did, do, does, either, else, ever, every, for, from, get, got, had, has, have, he, her, hers, him, his, how, however, i, if, in, into, is, it, its, just, least, let, like, likely, may, me, might, most, must, my, neither, no, nor, not, of, off, often, on, only, or, other, our, own, rather, said, say, says, she, should, since, so, some, than, that, the, their, them, then, there, these, they, this, tis, to, too, twas, us, wants, was, we, were, which, while, who, will, with, would, yet, you, your

The result for the list of removable word tags are :

- CC
- IN
- DT
- RB
- VBP
- VB
- MD
- VBZ
- VBD
- PRP
- PRP$
- WRB
- VB
- LS

A complete list of the Penn Treebank tag set is found in the appendix.

Example of the keyword extraction:

Initial question: What is the deadline for foreign students?

Question after POS tagging: What_WP is_VBZ the_DT deadline_NN for_IN foreign_JJ students_NNS

Resulting keywords after removing not useful tags: What, deadline, foreign, students.

The remaining keywords are forwarded to the Lemmatizer.

4.4 Lemmatizer

A Lemmatizer is a program which generates the lemma of a word. A lemma is called the lexical root or the base form of a given word.

"Lemmatization is the process of reducing an inflected spelling to its lexical root or lemma form. The lemma form is the base form or head word form you would find in a dictionary." [Mor]

In the English language the base form for a verb is defined as the simple infinitive form of the verb. An example for this is the gerund form "striking" and the past form "struck" which are both forms of the lemma "to strike". For a noun the base form is the singular form of the word. An example for this is the plural word "mice" which is a form of the lemma "mouse".

In English the most spellings can be lemmatized without using any different rules than the regular rules of the English grammar. But there are irregular forms which don't follow these regular rules. Therefor has to be a special handling. For example the word "axes" which has four answers possible as a lemma. The possible lemmas are the singular noun "axe", the singular noun "axis", the verb "to ax" or the verb "to axe". In this case the part of speech information of the word has to be known to distinguish more complicated examples.

This system is using the English Lemmatizer of the "MorphAdorner" project for getting the word lemmas. The Lemmatizer searches first with a spelling pair which includes the word and the NUPOS part of speech tag. If there is a matching entry found in the word lexicon the lemma will be returned as specified. If there is no matching entry in the word lexicon the

English Lemmatizer will use a combination of irregular forms and grammar rules to determine the lemma of the given word.

For example for the word striking the pair is (striking, vvg). The spelling pair is not found in the word lexicon. Then the NUPOS tag is converted to one of the following major word classes.

Major word classes used by the lemmatizer:

- adjective
- adverb
- compound
- conjunction
- infinitive-to
- noun, plural
- noun, possessive
- preposition
- pronoun
- verb

The example (striking, vvg) with the NUPOS gerund tag vvg maps to the verb word class. The lemmatizer now checks the spelling pair (striking, verb) against the list of irregular forms. If there is no matching with the irregular forms list the lemmatizer uses rules of detachment. The example is matching the rule for "CVCing CVCe" with says that our word is matching a consonant which is followed by a vowel, followed by a consonant and followed by "ing" at the end of the word. Therefor the replacement is to keep the consonant, the vowel and the second consonant but replace the "ing" with an "e".

The result for the example striking is that striking is lemmatized to strike.

If all words of the previous extracted keywords are lemmatized in this process the lemmas of all words are forwarded to determine their synonyms.

4.5 Synonym resolution

In language some words have synonym words which have the same meaning or assertion but they are different words. Therefor the system is using the Java API for WordNet Searching which is using the WordNet database to retrieve the synonyms and their tagged definitions. For getting a ranking weight for the synonyms the WordNet stores for this reason a tag counter.

The ranking weight is defined as follows:

Be r a rank with a given word w for which pair a set of n definitions d is found. The formula $D := \{(d_i, t_i)\}$ is the assignment to a tag counter of all definitions where t is the tag counter number of the corresponding definition. $S := \{(s_j, d_i)\}$ describes all applicable synonyms s which are corresponding to these definitions and finally the value t_{max} be the highest tag counter which is found in D.

Then the resulting rank for synonyms is calculated as follows:

$$R := \{(s_j, r_i) \,|\, (s_j, d_i) \in S \ \wedge \ r_i = r \cdot \left(\tfrac{t_i}{t_{max}}\right)^x\}, \quad x \in [0, 1]$$

The ranking r_i of a synonym s_j is calculated by multiplying the old rank r of the given word with a factor between 0 and 1. The tag counter of the synonyms meaning and the highest tag counter are building these factor between 0 and 1. The factor for decreasing towards 0 is adjusted with the exponent x.

For $x = 1$ the formula $r_i = r \cdot \left(\tfrac{t_i}{t_{max}}\right)^1 = r \cdot \tfrac{t_i}{t_{max}}$. Choosing $x = 0$ will lead to $r_i = r$, since $r \cdot \left(\tfrac{t_i}{t_{max}}\right)^0 = r \cdot 1 = r$. So if the new ranking weights r_i are to stay close to the original ranking weight r, then a value close to 0 will be appropiate for x.

As result a HashMap with the word as key and the weight as value is returned from the synonym resolution process. These HashMap is than forwarded to the question ranking process.

4.6 Question ranking

The question ranking process is using the vector space model to represent the question and answer documents as a vector. The similarity between the vector of the list of keywords and

synonyms of the given question and the vector of keywords from the questions for one answer are calculated with the cosine similarity.

"In the **vector space model** of information retrieval, documents and queries are represented as vectors of features representing the terms (words) that occur within the collection" [JM08]

In general the vector space model has the following representation of a document vector and a query vector:

document vector: $\vec{d_j} = (w_{1,j}, w_{2,j}, w_{3,j}, ..., w_{n,j})$

query vector: $\vec{q} = (w_{1,q}, w_{2,q}, w_{3,q}, ..., w_{n,q})$

The formula for calculating the cosine similarity the question ranking process is using is defined as follows:

$$\text{similarity} = \cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} (A_i)^2} \times \sqrt{\sum_{i=1}^{n} (B_i)^2}}$$

Table 4.1: Cosine Similarity Formular [ABC+]

A and B are two vectors, representing the query (A) and one document (B), where each dimension is a frequency of one term.

But the raw frequency is not sufficient to distinguish the documents and the process for ranking is using the inverse document frequency as factor to the base vector.

"The second factor is used to give a higher weight to words that only occur in a few documents. Terms that are limited to a few documents are useful for discriminating those documents from the rest of the collection, while terms that occur frequently across the entire collection aren't as helpful." [JM08]

Therefor the formula for calculating the inverse document frequency is as follows:

$$idf_i = \log \frac{N}{n_i}$$

Table 4.2: Inverse Document Frequency [ABC+]

In that fomular the number of documents is N and the number of documents where the term i occurs is n_i .

The whole cosine similarity formula including the inverse document frequency is as follows:

$$\text{similarity} = \frac{\sum_{i=1}^{n} A_i \times B_i \times idf_i^2}{\sqrt{\sum_{i=1}^{n} (A_i \times idf_i)^2} \times \sqrt{\sum_{i=1}^{n} (B_i \times idf_i)^2}}$$

Table 4.3: Cosine Similarity Complete Formular [ABC+]

The similarity of the documents is the probability for one answer to be related to the query. This process of calculating the ranking of an answer to the given question is a list of answer and probability pairs which is forwarded to the find answer process.

4.7 Find answer

The list of answer and probability pairs is now processed to find the best matching answer with analyzing the probability value to each answer. For the probability value is a predefined threshold set which defines a minimum value for the probability of an answer. The threshold is set to a value of 0.5 . With this threshold value the system has a minimum probability. The process is searching in the list of answer and probability pairs for the highest probability and returns this answer as result assumptive the threshold value is not undershot.

If the process finds no answer which has a probability higher than the minimum threshold the initial question will be forwarded to the out of scope web search.

4.8 Web search

The web search is using the Apache Lucene API for indexing and searching. The keywords for searching on the web site are extracted on the same way than described in section 4.3. After that if the list of keywords size is smaller than 4 the keywords will be appended to one string. If the list of keywords size is greater than 3 the keywords will group to a size of two in a single string. For example if the list of keywords has a size of 6 the generated keyword pairs in one string will be 1 and 2, 2 and 3, 3 and 4, etc. All of the keywords are separated with one whitespace.

Then will be searched for every entry in the list of keywords. The search is executed on every link saved in a list of strings which were found with the UpdateEngine. The result of every search of one link is saved in a list of URL objects which contain the URL as a string and the weight. If this list contains already a link the weight will be updated to the new weight value but only if the new weight value is higher than the old weight value.

The result is then sorted after the weight from higher to lower values and a predefined number of links beginning with the highest weight value are returned to the user. The number of links which are returned to the user is configured in the configuration file. For more information see section 4.10.

To update the data for searching the system has a second Servlet called the "UpdateEngine".

4.9 Update web search data

The system has implemented a second Servlet for updating the data of the web search. This second servelt is using a Web Crawler to mine the data of a web page and search for new links on a web site. The data is indexed and saved with the Apache Lucene API.

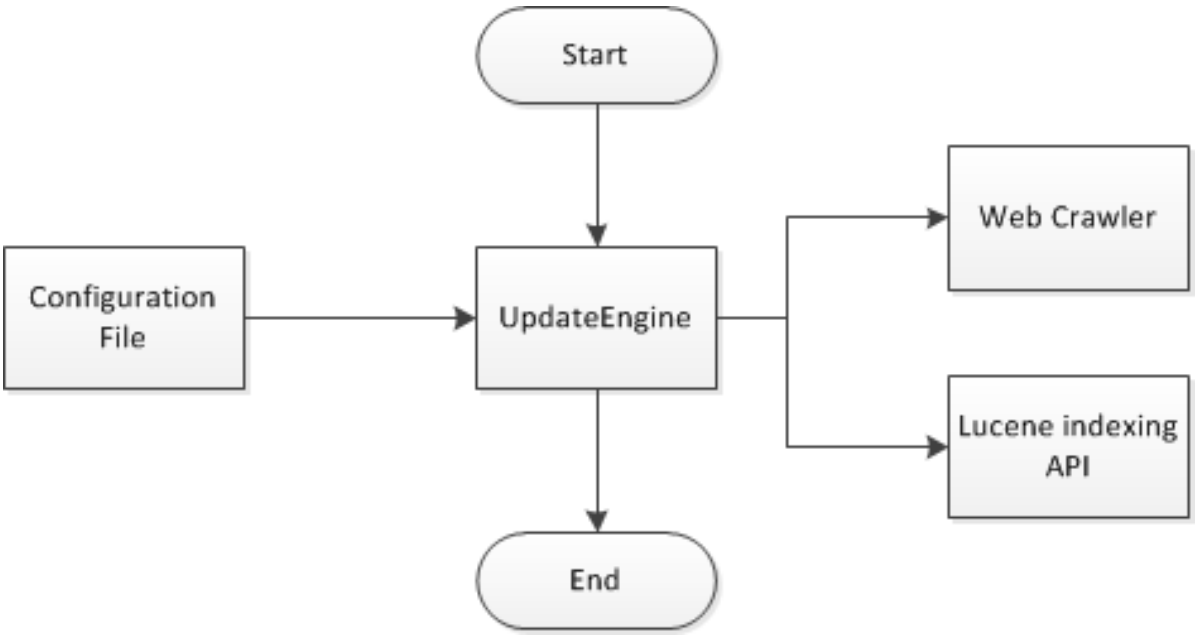

Figure 4.2: Flowchart of the architecture of the UpdateEngine

"Lucene is a high performance, scalable Information Retrieval (IR) library. It lets add indexing and searching capabilities to your applications." [MHG10]

The Web Crawler is getting the web site URL from the configuration file where it shall search on. The Web Crawler can search for different URL styles which are also configured in the configuration file.

The different URL styles are:

- htm/html: URLs which are ending with an htm or html file ending
- php: URLs which are ending with a php file ending
- pdf: URLs which are ending with a pdf file ending to include pdf files into the search process
- "/": URLs which are ending with a slash sign or have no ending like the endings above

To let the Web Crawler working politely there is also a validation of every url which is searched for in the robot.txt file. To disallow the Web Crawler searching on a web page url the robot.txt file has to contain these lines:

- a line with a specified user agent name, the two values are "student-Bot" or "*"
- url entries after "Disallow:"

The text data of every found web page is forwarded to the Apache Lucene API to index the data and save this data to an index. The data for every found web page has to be converted to a different format to search rapidly and don't have to search with a slow sequential scanning process.

"To search large amounts of text quickly, you must first index that text and convert it into a format that will let you search it rapidly, eliminating the slow sequential scanning process. This conversion process is called indexing, and its output is called an index." [MHG10]

The index data of the update process is saving into a temporary folder named "indexFiles2" to let the system be still working while the update process is running. When the update process finished the indexing the folder "indexFiles" with the previous index data is deleted and the temporary folder is renamed into the name of this folder.

The index data is saved in a file to prevent the system to search every time the Servlet is started or restarted. The web search is using these index files. A text file containing all the links the Web Crawler has found to reload all the link names saved in the index data after restarting the Servlet.

All the configuration parameters for the web search and indexing are set in the configuration file which is described in the next section.

4.10 Configuration file

The student-Bot also has a configuration file in which the main configuration options could be set. The configuration file is an easy way to configure the main configuration options without having to change the code and to easily adjust the system to fit into different environments. When the Servlet is starting the configuration file is loaded and changes to the configuration file will only be changed in the student-Bot program if the Servlet will be restarted.

The configuration file contains the following options that could be set:

databaseFilename

For this value the filename of the question and answer database can be set.

websiteURL

With this value the address of the website for the Web Crawler can be set. If you set more than one address the addresses must be separated with an "';"'.

linkEndings

Here you can set the link endings of the address or files for the Web Crawler which you want to add to the web search.

maxNumberDisplayed

Here you can set the number of displayed results of the web search to the user.

enableRobot

Here you can set if the Web Crawler is using the robot.txt file to work more polite or if he is ignoring it and search for everything.

debugWebcrawler

Here you can set if you want to have the debug output of the Web Crawler.

tagDictionary

Here you can set the name of the tag dictionary file for the POS tagger.

websearchAnswer

Here you can set the answer which is provided before the link results of the web search.

lastGivenAnswer

Here you can set the last answer which will be given to a user if no answer will be found in the database and the web search result is empty.

The values for "tagDictionary", "websiteURL", "databaseFilename" and "lastGivenAnswer" must be set and for "linkEndings" a minimum of one. Otherwise the system will not work.

```
#######################################################
#         student-Bot configuration file              #
#                   version 1.0                       #
#                                                     #
#######################################################

# IMPORTANT NOTICE: If you set more that one value, you must separate them
#                   with a semicolon (;).

# set the database filename of the question and answers
databaseFilename="questions_h-da.json"

# set the website url where the webcrawler should search on
websiteURLs="http://en.ru.is"

# set your linkstyle here and file endings
# htm   = *.htm/*.html
# php   = *.php
# pdf   = *.pdf
# /     = link ends with an /
linkEndings="/;"

# maximume number of results of the web search displayed to the user
maxNumberDisplayed="4"

# enable or disable the robot.txt to let the webcrawler behave more polite
enableRobot="true"

# debug webcrawler
debugWebcrawler="false"

# set the tag dictionary file
tagDictionary="english-bidirectional-distsim.tagger"

# set the answer which is provided before the resulting links of the web search
# are shown
websearchAnswer="Hi, I found no information in my database but I found some information on the following websites:<br />"

# set the last answer if no answer is found in the database and no links on the
# webpage (html tags are allowed)
```

Figure 4.3: Configuration File Example

4.11 Database

For the data of the questions and answer database the file type is set to a JSON file type. For reading the file the Jackson Java JSON Processor library is used. The structure is a List<Map<String, Object» where the key for the answer is "answer" and for the questions is "questions". The answer is saved as Map<String, String> object and the questions as a Map<String, List<String» object. The data for one answer is saved in a QuestionAndAnswer object containing the answer as string and the questions as List<String>.

During reading the database and saving the QuestionAndAnswer objects the keywords for the questions are extracted as described in section 4.3. The keywords are also saved in the QuestionAndAnswer object to only generate them once on starting the program.

```
[
  {
    "answer": "The application deadline is July, 15th, for
       international applicants and September, 1st, for German
       applicants for the following start in September.",
    "questions": [
      "What is the deadline to apply?",
      "What is the deadline for foreign students?",
      "When do you need the application forms?",
      "When i could start my Application process?",
    ]
  },...
]
```

Figure 4.4: Database Structure Example

5 User evaluation

The user evaluation was conducted to test the system on giving right answers or information to the asking user and to find errors and mistakes in the logical behavior of the system.

5.1 Target User Group

The initial user group was international students. In this group are incoming students which are undergraduate and graduate students. It is not important whether they are only attending a single semester or a full year because the target user group for the system is all incoming students which are studying abroad.

5.2 User Evaluation Framework

5.2.1 Goal for User Evaluation Sessions

The primary outcome of the test sessions is to find errors and mistakes in the program and in general get to know whether users would trust the answers returned from the program. In connection with this trust the user has not to write an additional e-mail to somebody for answering their question. This would be the program main task to relieve the people which are answering many e-mails with questions from incoming students.

5.2.2 User Selection

The user selection criteria for the tests are based on the defined target user group (see 5.1). The number of users was at least 10 people to get a meaningful statement about the program.

5.2.3 Interview Process

The structure of the user evaluation process is a preset structured process across all interviews. The test user have to follow two tasks which are stated in the "User Testing Guideline" (see A.2).

5.2.4 Test Locations and Settings

The test settings for the entire test users are the same. All test users have to use a browser to open the web page of the program and ask their questions. The locations for this evaluation were the University of Applied Science in Darmstadt and the Rekyjavík University in Iceland.

5.3 Results of the User Evaluation

The result of the user evaluation is okay. Altogether the program answered 87 questions which were asked by the test users. From these questions the program answered 12 questions correct and 40 questions wrong. The rest of the total value of questions were questions which are not in the database and the program made a web search or answered the last answer and found no appropriate information at the web search.

correct answers	wrong answers	not known answers	total answers
12	40	35	87

Table 5.1: Analysed Answers Table

On this evaluation 10 people participated with the test and asked the program questions and filled in the questionnaire. All of the test users were between 20 and 30 years old and from many nationalities. Most of them mark their computer skills as good or very good.

Many of the test users asked questions about general purposes like chatting with a person and not asking a bot questions were they want information about. Another test user asked only keywords to let the program search every time on the web site for information.

In general purpose the statistic of answering the question if they want to receive information quickly. All of them want to receive information quickly from within 2-3 hours to within 4-7 days. But most of them want to receive them within 2-3 hours. On the question whether they would trust information received from a bot the result of the survey says that most of the test users would be neutral against this statement.

Also for writing an additional e-mail most of the test users said that they would write an additional e-mail to a person responsible for foreign matters after they get information from a bot. They said for the question if they would write still an e-mail if they got information from a bot that they would write that e-mail also.

Most of the test users were satisfied with the speed of answers the program gaves but nearly all of them were not really satisfied with the given answers. More details on the statistical results of the user evaluation are in chapter A.3 of the appendix.

With this result of the user evaluation the program and the database could be improved and the weak spots can be identified.

6 Conclusion

After analyzing the results of the user evaluation the program is working but has also its problems. One problem is that the program answers many questions wrong in the user evaluation. Nearly half of the asked questions were answered wrong. But this is depending on the database.

The database gives the possibility to specify more than one possible question for one answer. In the test scenario the database has some answers with more than one possible question and some with only one possible question. Therefor it could be that an answer with only one possible question has a higher similarity likelihood value than an answer with more than one question. This occurs by having less keywords in a single question answer than in an answer with more than one question.

Preventing the program before making those mistakes the minimum number of questions matching to an answer should be more than two questions in the database. All of the answers should be having at least this number of questions. That would prevent the ranking and similarity likelihood value not to be higher than an answer with only one question.

Also the test users asked many questions which were not part of the provided information in the database. To give a good database the provided information (questions and answers) should be wide ranged and at least the most common questions asked by incoming exchange students. Within those questions there should be also answers to questions which are not information about the university but information about public transportation and accommodation.

To have all the necessary information in the database there should be a survey conducted to determine the most needed information and questions an incoming exchange student has.

The second negative thing as result of the user evaluation is that the users were not really sure about trusting the information a bot program would return to them. In some cases this could

be a cultural problem and in other cases it could depend on how good the returned information are. To spread no unrealiable information the database of the program should be as good as possible. The information in the database could also be improved and more information added while the program is running. A self-learning variant of the information database is not possible for this environment.

The program has also its good features. It is providing information very quickly to the users. Most of the test users wanted to have the information quickly and not have to wait very long to get them. That is also an advantage for incoming exchange students to make their plans before going to the foreign country faster.

The combination of a static database of question and answers information and the information of a whole web page is very good. The test user gave good feedback on this functionality and were very satisfied with the information. But this depends on how good the provided information on the web page are.

All things considered the program "student-Bot" is working to answer questions and can relieve the staff of international offices in answering the questions of many incoming exchange students and potential students which want to study abroad.

Bibliography

[ABC+] Arnreich, Daniel, Bär, Björn, Contag, Daniel, Fahlbusch, Georg, Gimbel, Stephan, Greppmeier, Christian, Liepelt, Thomas, Luthmer, Arne, Neef, Tobias, Neupärtl, Michael, Noll, Walter, Oelmann, Simon, Rihm, David, Rudat, Sascha, Rühl, Ludwig Jean, Sturm, Alexander, Turba, Tobias, and Vogel, Jan: *Fbi bot.*

[Ali] *Alicebot.* `www.alicebot.org`. last visited on 22.01.2012.

[Jab] *Jabberwacky.* `http://www.jabberwacky.com`. last visited on 20.01.2012.

[JM08] Jurafsky, D. and Martin, J.H.: *Speech and language processing*, 2008.

[Luc] *Apache lucene api.* `http://lucene.apache.org/core/`. last visited on 20.01.2012.

[MHG10] McCandless, M., Hatcher, E., and Gospodnetić, O.: *Lucene in Action.* Manning Pubs Co Series. Manning, 2010, ISBN 9781933988177.

[Mor] *Morphadorner lemmatizer.* `http://morphadorner.northwestern.edu/morphadorner/lemmatizer/`. last visited on 17.01.2013.

[PDF] *Apache pdfbox java library.* `http://pdfbox.apache.org/`. last visited on 20.01.2012.

[TKMS03] Toutanova, Kristina, Klein, Dan, Manning, Christoph, and Singer, Yoram: *Feature-rich part-of-speech tagging with a cyclic dependency network.* `http://nlp.stanford.edu/~manning/papers/tagging.pdf`, 2003. last visited on 14.01.2013.

[Wal] Wallace, Richard: *Artificial intelligence markup language (aiml).* `http://www.alicebot.org/TR/2005/WD-aiml/`. last visited on 22.01.2012.

[Wora] *Wordnet - a lexical database for english.* `http://wordnet.princeton.edu`. last visited on 18.01.2012.

[Worb] *Java api for wordnet searching.* `http://lyle.smu.edu/~tspell/jaws/index.html`. last visited on 20.01.2012.

Abbreviations

API Application programming interface is a specification intended to be used as an interface by software components to communicate with each other.

Java Java is a object-oriented programming language developed by Sun Microsystems.

NLP Natural Language Processing.

pos part of speech.

Servlet Servlet is a Java programming language class used to extend the capabilities of a server.

URL Uniform Resource Locator is called the address of a web page on the internet.

Appendices

A.1 Appendix A: Penn Treebank Tagset

Tag	Description
CC	Coordinating conjunction e.g. and,but,or...
CD	Cardinal Number
DT	Determiner
EX	Existential there
FW	Foreign Word
IN	Preposition or subordinating conjunction
JJ	Adjective
JJR	Adjective, comparative
JJS	Adjective, superlative
LS	List Item Marker
MD	Modal e.g. can, could, might, may...
NN	Noun, singular or mass
NNP	Proper Noun, singular
NNPS	Proper Noun, plural
NNS	Noun, plural
PDT	Predeterminer e.g. all, both ... when they precede an article
POS	Possessive Ending e.g. Nouns ending in 's
PRP	Personal Pronoun e.g. I, me, you, he...
PRP$	Possessive Pronoun e.g. my, your, mine, yours...
RB	Adverb Most words that end in -ly as well as degree words like quite, too and very
RBR	Adverb, comparative Adverbs with the comparative ending -er, with a strictly comparative meaning.
RBS	Adverb, superlative
RP	Particle

Table A.1: Penn Treebank Tagset part 1

Tag	Description
SYM	Symbol Should be used for mathematical, scientific or technical symbols
TO	to
UH	Interjection e.g. uh, well, yes, my...
VB	Verb, base form subsumes imperatives, infinitives and subjunctives
VBD	Verb, past tense includes the conditional form of the verb to be
VBG	Verb, gerund or persent participle
VBN	Verb, past participle
VBP	Verb, non-3rd person singular present
VBZ	Verb, 3rd person singular present
WDT	Wh-determiner e.g. which, and that when it is used as a relative pronoun
WP	Wh-pronoun e.g. what, who, whom...
WP$	Possessive wh-pronoun
WRB	Wh-adverb e.g. how, where why

Table A.2: Penn Treebank Tagset part 2

A.2 Appendix B: User Testing Guidelines

User Evaluation
student-Bot

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACULTY OF COMPUTER SCIENCE

Today I would like to test with you my new program "student-Bot" which I developed in the project work part of my Bachelor thesis. The student-Bot is a program to answer questions of incoming exchange students and give information. Your work in this user evaluation is to solve the following tasks:

1. Ask the student-Bot 10 to 15 questions of your choice like you want to get information about the university you want to study abroad.

Address: http://holna.fbi.h-da.de:8080/student-Bot/

2. Fill out the questionnaire

Address:
https://docs.google.com/spreadsheet/viewform?formkey=dERoZUhJXzBnT3JPNC1OMEhHWVBKRGc6MQ#gid=0

Thank you for doing this evaluation.

Table A.3: Participation Information Sheet

A.3 Appendix C: Statistical results of user evaluation

A.3.1 Personal Information

Age

22 and lower	23	24	25	26 and higher	total
3	1	2	1	3	10

Table A.4: Test User Age Table

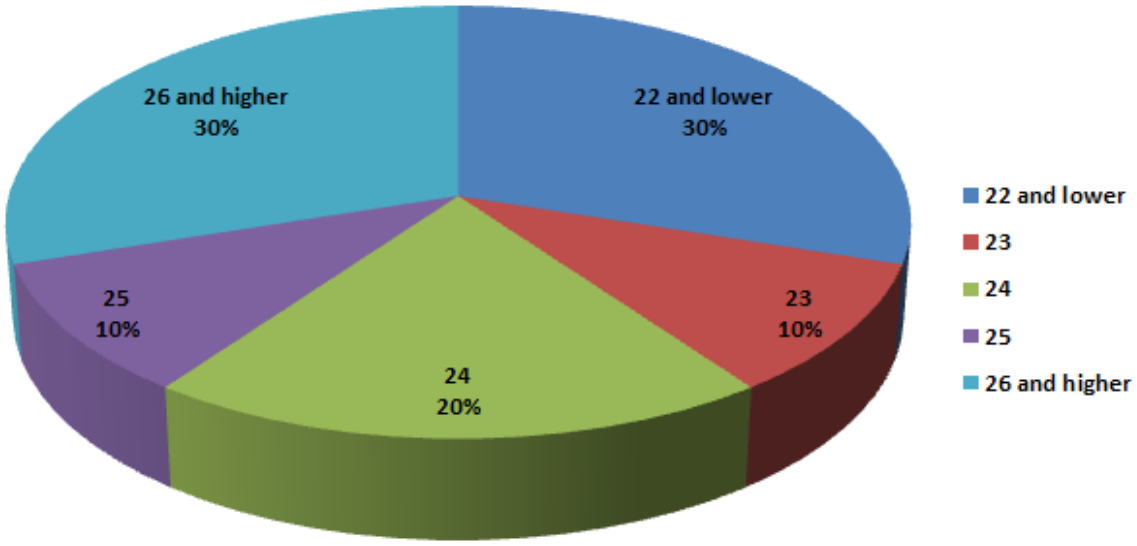

Figure A.1: Test User Age Diagram

Sex

male	female	total
8	2	10

Table A.5: Test User Sex Table

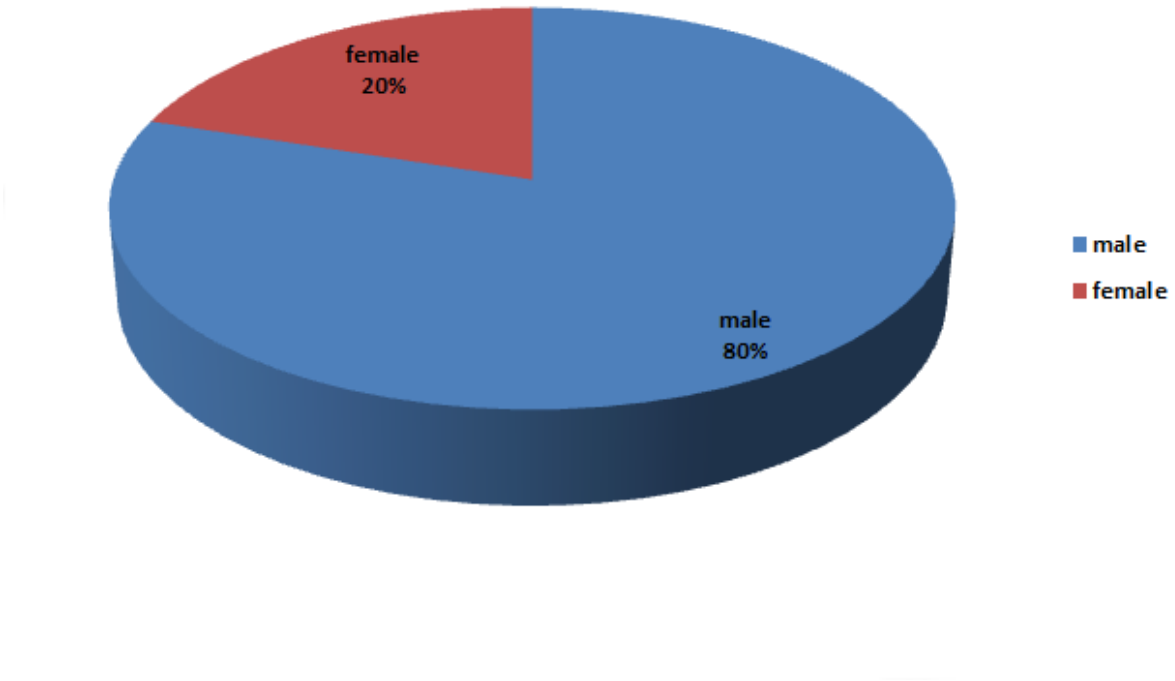

Figure A.2: Test User Sex Diagram

Nationality

Czech Republic	Finland	India	Indonesia	Netherlands	Pakistan	Spain	Vietnam	total
1	1	2	1	1	2	1	1	10

Table A.6: Test User Nationality Table

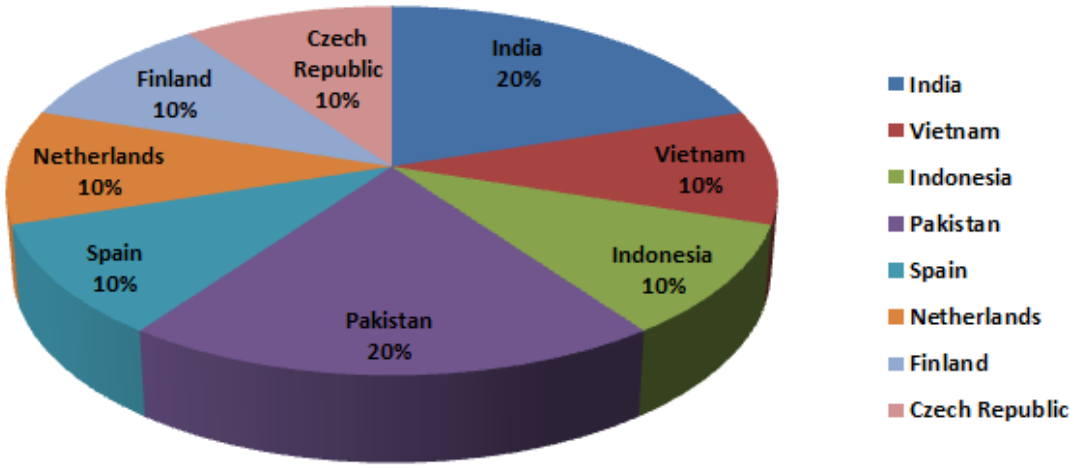

Figure A.3: Test User Nationality Diagram

How would you assess your own Computer Skills?

none	low	moderate	good	very good	total
0	1	0	7	2	10

Table A.7: Test User Computer Skills Table

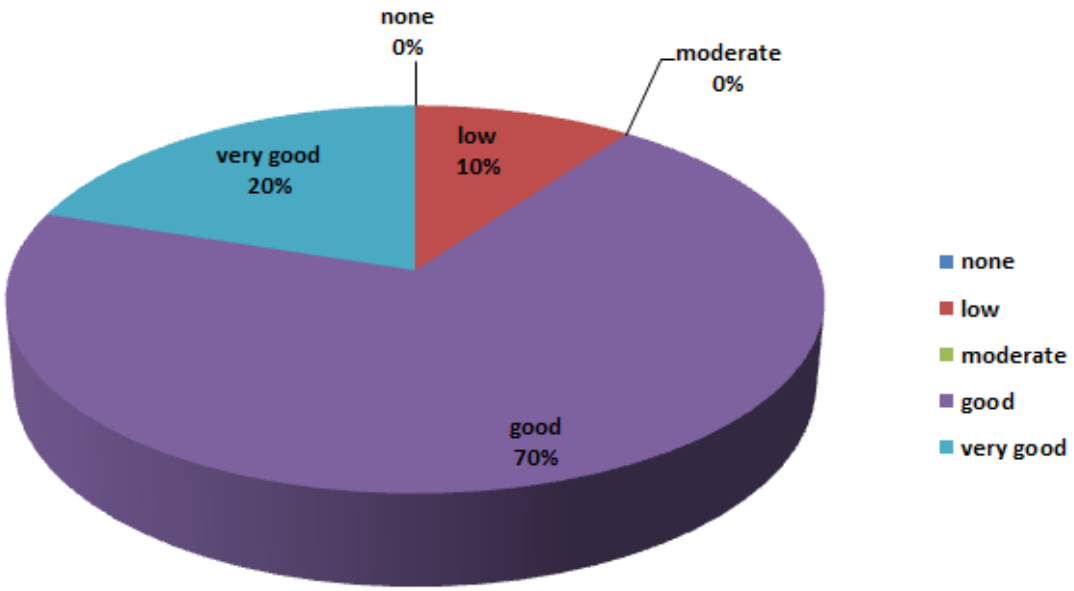

Figure A.4: Test User Computer Skills Diagram

A.3.2 General Information

I want to receive information quickly

within 2-3 hours	within 1 day	within 4-7 days	within 2 weeks	within 1 month	total
5	3	2	0	0	10

Table A.8: Quick Information Table

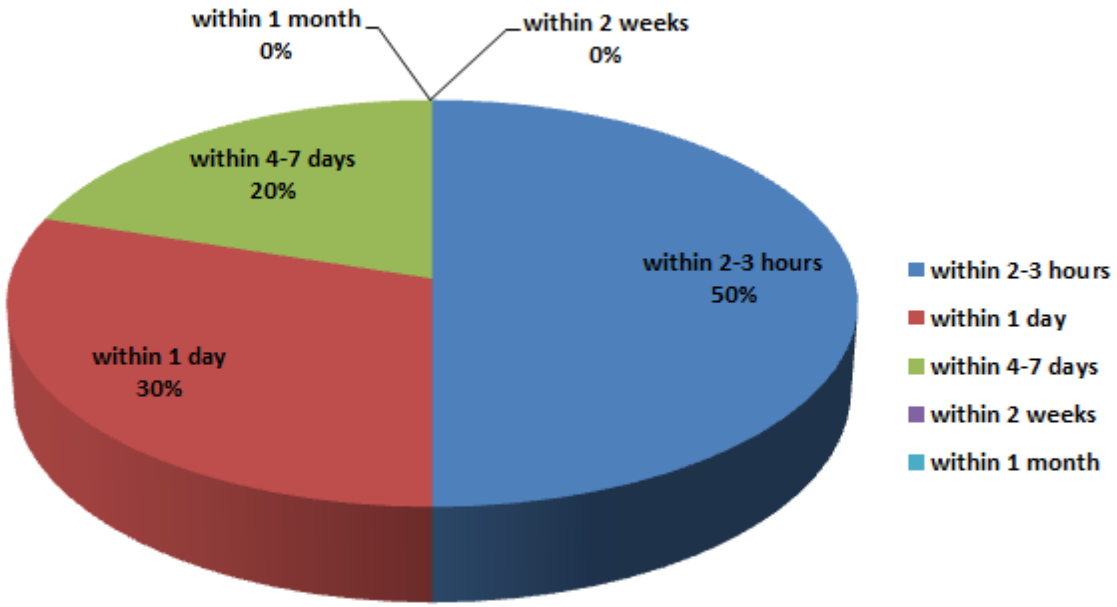

Figure A.5: Quick Information Diagram

I would trust a bot about information

1 - strongly disagree	2	3	4	5 - strongly agree	total
1	2	5	1	1	10

Table A.9: Trust Bot Information Table

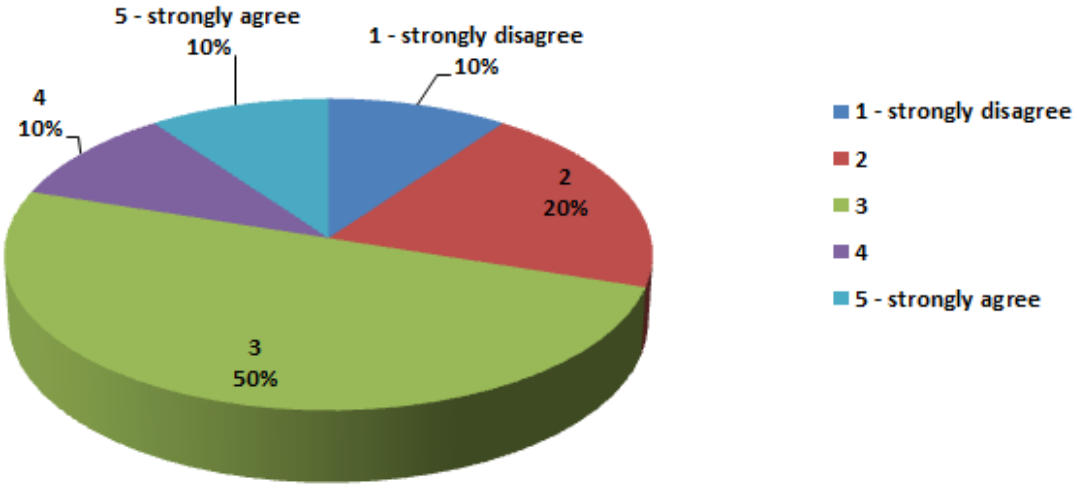

Figure A.6: Trust Bot Information Diagram

After getting an answer from a bot I would also an additional e-mail to a person responsable for foreign matters

1 - strongly disagree	2	3	4	5 - strongly agree	total
0	2	3	2	3	10

Table A.10: Write Additional E-Mail Table

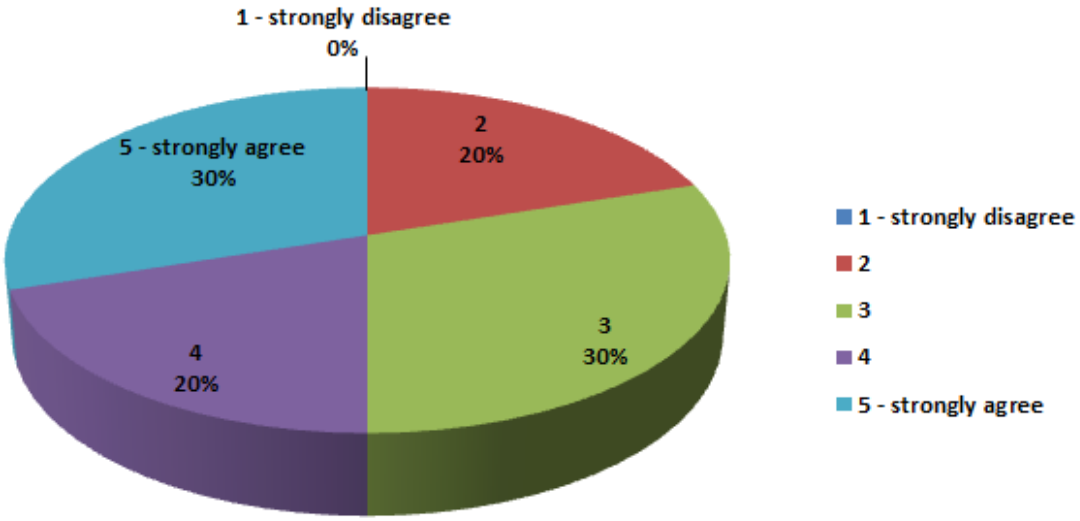

Figure A.7: Write Additional E-Mail Diagram

After getting an answer from a bot I would still write an e-mail to a person

1 - strongly disagree	2	3	4	5 - strongly agree	total
0	3	5	0	2	10

Table A.11: Write Still E-Mail Table

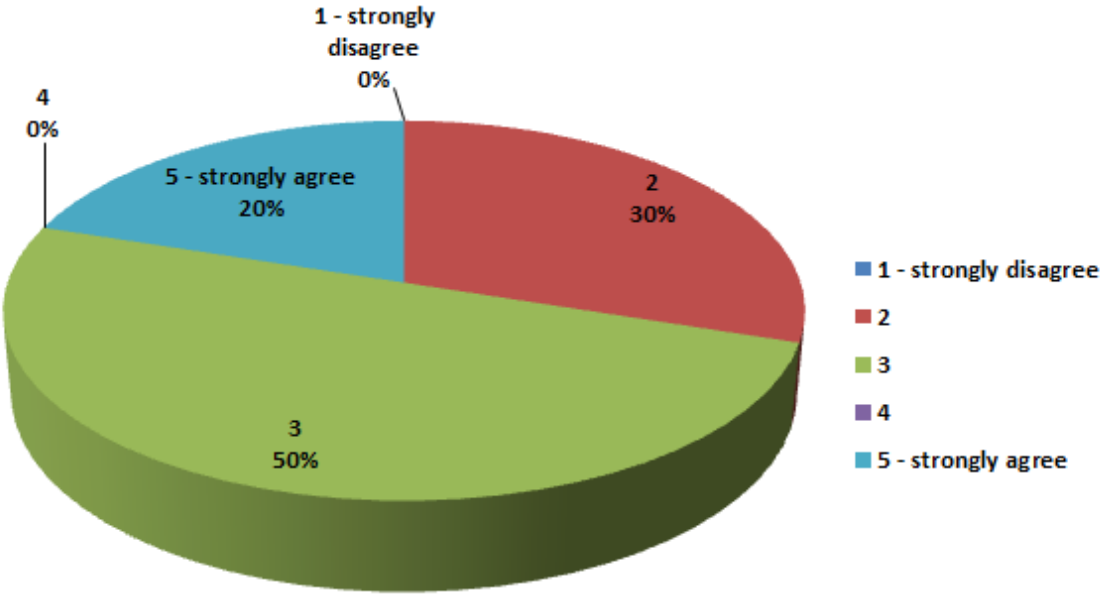

Figure A.8: Write Still E-Mail Diagram

A.3.3 student-Bot

The bot answers quickly

1 - strongly disagree	2	3	4	5 - strongly agree	total
1	1	2	3	3	10

Table A.12: Answer Quickly Table

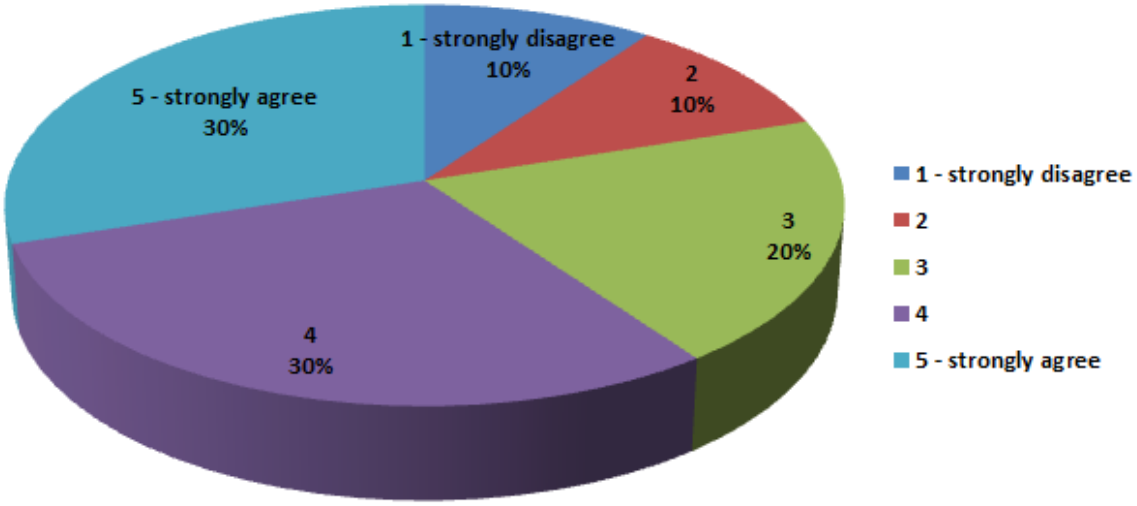

Figure A.9: Answer Quickly Diagram

How satisfied were you with the returned answers or information?

1 - strongly not satisfied	2	3	4	5 - strongly satisfied	total
3	3	4	0	0	10

Table A.13: Satisfy Information Table

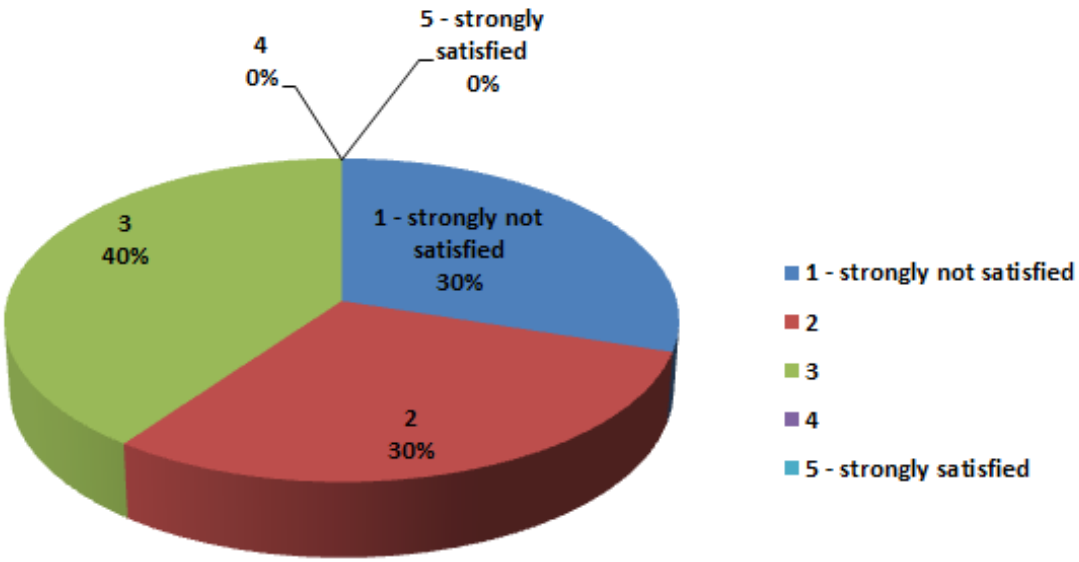

Figure A.10: Satisfy Information Diagram