Intelligent Student-Bot for an Interactive Question and Answer User Interface

Emmanuel Günther and Bettina Harriehausen-Mühlbauer

University of Applied Science, Department of Computer Science, Darmstadt, Germany
Emmanuel-Maurice.Guenther@stud.h-da.de,
Bettina.Harriehausen@h-da.de

Abstract. We face the task to minimize hours of email-answering for similar questions that were being asked by so-called incoming students, i.e. foreign students that would like to spend some time in our department. We wanted to create an information system that was more intelligent than previous BOTs working on the level of ELIZA, i.e. on the basis of a simple pattern-matching algorithm. Our paper describes the use and implementation of technologies for these steps and discusses pros and cons of our implementation in comparison to alternative solutions, including the vector-space-model, the cosine similarity and web search with n-grams.

Keywords: Intelligent Bot, vector-space-model, cosine similarity, web search, n-grams.

1 Introduction

In most international offices the staff receives a lot of e-mails from incoming students and potential students who want to study abroad with questions regarding university related issues. Some of these staff members are using pre-defined text blocks to answer the questions they have in many e-mails and send them as a response. But it takes the staff many hours to answer all the e-mails. We are seeking ways to minimize this time of email-answering for similar frequently asked questions. For handling this problem we created an information system to answer the questions directly therefore eliminating the need for incoming students to write an e-mail to the staff of an international office.

In order to be able to provide answers to questions from incoming students, several goals were defined. The first goal was to provide an easily integrated and usable system for different universities 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. The second goal was to get a wider range of available information. In order to achieve this, the system should not only have a static database, rather a second source should be available to search for information – the internet.

2 Background

One other system, the Jabberwacky AI[1], is using contextual pattern matching techniques to find an answer or a response text to a given text. Jabberwacky stores all questions and responses in a self-learning database, which stores everything everyone has given as an input. From this information, Jabberwacky finds the most appropriate response text. In order to have a sufficient database and intelligent answers to a given input can be computed, the self-learning database takes a very long time to be called "intelligent". Another system called "Alice"[2] is using the Artificial Intelligence Markup Language (short AIML)[3]. "Alice" has a self-learning database as well.

In order to build an information system for incoming students, a self-learning database is not usable because we need a static database which already has enough information stored to answer questions of incoming students.

A group project of the natural language processing course at the University of Applied Science in Darmstadt developed a program to answer questions. This program tokenizes a given question, checking the spelling of these tokens and is filtering out unnecessary tokens with a so-called "stop word" list to get a list of keywords, which are lemmatized and synonyms created. This list of keywords and synonyms is ranked against keywords of questions in the database using the vector space model. The resulting rank value is compared against a threshold value and if the rank value is higher and also the highest rank value of all rank values, the defined answer of this rank value is returned. If there is no higher rank value, the program starts a web search using the web service of SearchBlox [4].

The use of a third party web service includes the problem to achieve the goal of a generalized system which can easily be integrated and usable in university environments. Our design is based on these features.

3 Design

Our design includes the extraction of keywords, building synonyms and ranking them against the keywords of questions in the database. In order to accomplish a better keyword extraction we are using a part-of-speech tagger to tag all word tokens and delete unnecessary tags from this list. This is a preferred, more generalized approach to the stopword list, as it is not limited to one language only. We chose the part-of-speech tagger from the Natural Language Processing Group of Stanford University[5]. Our design is programmed as a Java Servlet.

For a generalized system an external web service for performing a web search is not optimally performant, as an internal API. That is why we integrated the index and search function directly into the system, using the Apache Lucene API to index and

[1] `http://www.jabberwacky.com/`

[2] `http://www.alicebot.org/`

[3] A derivative of XML (Extensible Markup Language).

[4] `http://www.searchblox.com/`

[5] `http://nlp.stanford.edu/software/tagger.shtml`

search web pages on the internet. We are also including the Apache PDFBox API into Lucene to search specifically for pdf files on web pages. In addition, a Web Crawler is included which searches a web site for all its pages. To easily update the web data, a second servlet is integrated into the design which calls the update mechanism of the system architecture.

Avoiding hardcoded parameters, we add a single configuration file for setting up all the necessary parameters. For example, the web page URL, the file endings searched for (e.g. html, pdf) and predefined replies. This system design approach enables easy integrating options within a wider range of different environments.

4 Architecture

The following flowchart shows the system architecture.

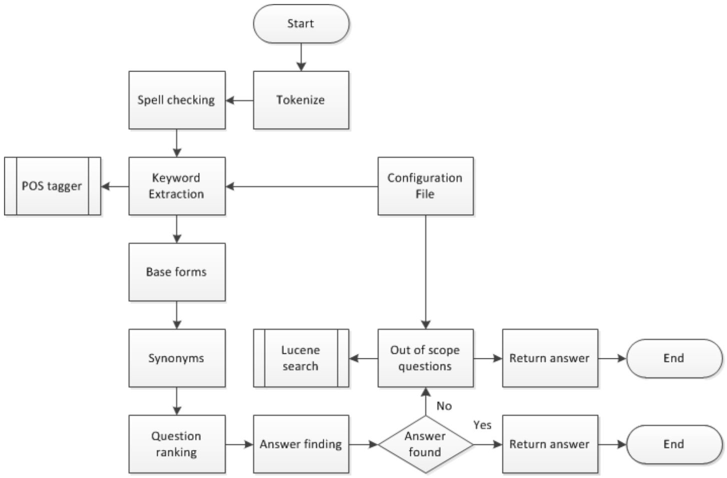

Fig. 1. System Architecture Flowchart

The working process of the system starts with the tokenization of the question. The Tokenizer splits plain text into logical sequences, which are called tokens, and removes punctuation marks. The word tokens are listed in a Java ArrayList and transformed into lowercase to avoid token repetitions because of case differences. This word token list is then forwarded to the spell checker.

The spell checker uses a list of all words of the database containing questions and answers. The complete list is searched for the most similar word using the Levenshtein distance [6] and Keyboard Proximity [7]. The original word token and the found word token are then compared against each other. If they are equal nothing is done,

[6] Minimal number of character changes to transform one word into another.
[7] Hitting wrong key on the keyboard.

otherwise the original word token is replaced by the found word token. The corrected word token list is now forwarded to the keyword extraction with the POS tagger.

The keyword extraction is using a POS tagger, which is using the Penn Treebank POS tag set for tagging the word tokens. All word tokens having a tag of the removable tags are removed from the list and the remaining word tokens are then used as keywords. Removable word tags are: CC, IN, DT, RB, VBP, VB, MD, VBZ, VBD, PRP, PRP$, WRB, LS, these are the tags of conjunctions, determiner, adverbs, verbs in various tenses, modals, pronouns, wh-adverbs and a list item marker.

In the next step the Lemmatizer is generating a lemma for every word, which is called lexical root or the base form. The system is using the English Lemmatizer of the "MorphAdorner" project [8], which is using a combination of irregular forms and grammar rules to determine the lemma of a word token. The remaining lemmatized word tokens are then forwarded to the synonym resolution.

The Synonym resolution in our system is done by using the Java API for WordNet Searching [9] which uses the WordNet [10] database to retrieve synonyms. The synonym resolution process returns a HashMap with the word as key and a weighting value as value, which is for obtaining a different weighting for synonyms then for previous keywords. The HashMap is then forwarded to the question ranking.

The question ranking process in our system is using the vector space model to represent the question and answer documents as a vector. The similarity between the vector 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. The inverse document frequency is also included into the cosine similarity formula. In (1) A and B are two vectors, representing the query (A) and one document (B).

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

The similarity of the vectors is the probability for one answer to be related to the query. The result is a list of answer and probability pairs which is forwarded to the find answer process.

The list of answer and probability pairs is now processed to find the best matching answer and the probability of all answers is analyzed for these pairs. The probability has to be higher than a threshold value which is defined as 0.5 [11] and the highest value of all probabilities. The answer with the highest probability value above the threshold is returned to the user. If there is no probability value higher than the threshold value, the given question is forwarded to the web search.

The web search is performed on the basis of the previously extracted keywords of the given question. These keywords are concatenated in one list of single keywords n-grames, where n is the maximum number of keywords. This results in a list of search

[8] `http://morphadorner.northwestern.edu/morphadorner/lemmatizer/`

[9] `http://lyle.smu.edu/~tspell/jaws/index.html`

[10] `http://wordnet.princeton.edu`

[11] This value is set as a result of testing because most non matching answers are below this value and matching answers above.

tokens. All web pages identified are saved in a list with their weighting value, which is the percentage matched. If the weighting is higher than the previously stored weighting value for a specific URL, it is updated with every search performed on one search token. A number of web links, specified in the configuration file, is then returned to the user or if no web link is found, the last answer is returned. For indexing the web pages the system has a second servlet to create or update the web search data.

The configuration file includes: the possibility for setting up the database filename, the web site URL, the link endings of pages and documents, the maximum number of displayed link results, the possibility to enable or disable the robot.txt file, the name of the tag model for the POS tagger, the predefined answers displayed before the web links and the last answer if no result is found.

5 User Evaluation

Our program was evaluated at the University of Applied Science in Darmstadt, Germany and at the Reykjavík University in Iceland. The target user group was made up of incoming exchange students. We used a very small database with only a few frequently asked questions. The outcome of the evaluation was that 20 % of the test users would trust information of a BOT and 30 % would not. The program answered 87 questions in total of which 14 % were correctly answered, 46 % wrong and 40 % were unknown questions.

6 Conclusion

After analyzing the results of the user evaluation, our BOT can be used, but its answers are not totally reliable, which is mainly caused by the limited data in our database. To prevent the program from answering so many questions incorrectly, the database should have at least 2 to 3 related questions for a single answer and the dataset should be as large as possible. During the evaluation the test user asked many questions which were not part of the database, that led to a majority of the incorrect answers. However, the main problem is that not all of the test users actually trusted the answers returned by the BOT. We believe that this problem will be resolved by enhancing the database with more current data.

References

1. Jurafsky, D., Martin, J.H.: Speech and language processing (2008)
2. McCandless, M., Hatcher, E., and Gospodnetic, O.: Lucene in Action. Manning Pubs. Co. Series. Manning. (2010) ISBN 9781933988177
3. Toutanova, K., Klein, D., Manning, C., Singer, Y.: Feature-rich part-of-speech tagging with a cyclic dependency network (2003), `http://nlp.stanford.edu/~manning/papers/tagging.pdf`