HTW Dresden - University of Applied Sciences

Faculty of Computer Science and Mathematics

Department of Artificial Intelligence

Master Thesis

Course of Studies: Computer Science

Topic:

Question Answering with Unstructured Data

Alexander Bresk

Dresden, July 21, 2015

Thesis Supervisor : Prof. Dr.-Ing. habil. Hans-Joachim Böhme

Contents

Abstract

With the use of unstructured text documents a question answering system for the German language is designed, developed and tested. Beside this, a novel approach for a feature extractor for answer type detection and crucial parts of a question answering system are, for the first time, applied to the German language. Parts of the system are evaluated to find the best parameters and configuration for a successful deployment. The system offers a text annotation pipeline that can be used for a wide variety of natural language processing tasks.

Acronyms

AMS	Annotation Management System
ANN	Artificial Neural Network
API	Application Programming Interface
AR	Answer Representation
ARFF	Attribute-Relation File Format
ATD	Answer Type Detection / Detector
BSON	Binary JSON
CLI	Command Line Interface
CRF	Custom Random Fields
CSV	Comma Separated Value
CoNLL	Conference on Computational Natural Language Learning
DATD	Detailed Answer Type Detection / Detector
DOM	Document Object Model
FFRDC	Federally Founded Research Development Center
GSON	Google JSON Library
HGC	Huge German Corpus
HTML	Hyper Text Markup Language
HTW	Hochschule für Technik und Wirtschaft
IDF	Inverse Document Frequency
IP	Internet Protocol
IR	Information Retrieval
JSON	JavaScript Object Notation
KB	Knowledge Base
LOC	Location
MISC	Miscellaneous
MIT	Massachusetts Institute of Technology
NE	Named Entity
NER	Named Entity Recognition

NLG	Natural Language Generation
NLP	Natural Language Processing
ORG	Organization
PER	Person
PHP	Hypertext Preprocessor
POS	Part of Speech
QA	Question Answering
SERP	Search Engine Result Page
STTS	Stuttgart Tübingen Tag Set
TF	Term Frequency
TREC	Text REtrieval Conference
URL	Uniform Resource Locator
VSM	Vector Space Model
XML	EXtensible Markup Language

Glossary

Answer Candidate A string/character sequence that was originally extracted from a text to possibly contain the correct answer to a given question.

Answer Extraction Desribes the process of extracting answer candidates from the text.

Answer Filter These filters are checking a set of answer candidates according to plausibility, meaningfulness and type.

Answer Representation A representation of an answer describes the display options and type of announcement. This can be as list, as simple fact or as complex paragraph.

Answer Type The answer type of a question is estimated during question answering process. This type reveals which type of answer the given question requires. To realize this task, a specific taxonomy is used.

ARFF File An ARFF file contains feature vectors and their labels. The data format was developed to train and represent statistical models.

Cognitive Systems The realization of psychology research into computer models is called cognitive systems. These systems use complex models to link information and compute on complex data structures.

Deep Question Parsing This term is used to point out that the parsing of a question includes semantic and syntactic processing as well as logical reasoning on the question or query.

Extractor In this work an extractor is used to locate or extract information from text. The purpose of extraction depends on the specific extractor.

Feature Detector/Extractor This part of the system realizes the transformation from text to feature vector. This transformation follows rules that were previously described and are later applied on each input text. In general, the feature detector transforms raw input to a point in feature space.

Feature Vector Describes a vector that contains all characteristics of a given problem, called features. The n-dimensional feature vector can be projected into a n-dimensional feature space.

Information Retrieval Information retrieval describes the task of finding relevant information in a set of documents.

Knowledge Base Describes a database that contains structured information. This information pairs are representing a specific kind of knowledge.

Named Entity Recognizer A program that finds named entities in text data. Examples for named entities could be a location, person or an organization.

Natural Language Processing Describes an area of research in artificial intelligence and machine learning. The trained models are working on natural language or on human language data.

N-Grams Describes a pair of consecutive tokens or elements that are combined to a gram (or cluster). For example, the word *die* can be seen as 3-gram.

Part of Speech / POS POS represents the grammar of a given text (sentences or question). The grammar is denoted with specific tags (like ADJ for adjectives).

Plausibility/Sanity-Checks These checks are performed to evaluate which answer candidate is plausible to the question.

Search Engine Result Page / SERP This page is provided from the search engine and it contains relevant documents according to the query.

Semantic Frame A semantic frame is a semantic interpretation of something (in this thesis a word or a group of words. Each word can correspond to a number of semantic frames.

Stemmer A stemmer tries to break down a word to the stem. For this, the stemmer uses language-specific algorithms.

Structured Data In contrast to unstructured data, this type of data can be used to perform structured search requests.

Tagger A tagger walks through a set of data and attaches tags (often called labels) on the data set. Taggers can perform different tasks.

Text Normalization Describes the process of making text comparable. After the input text was normalized (pruned, stemmed and enriched with word relations) it can be compared to other input text or to a set of similar normalized text documents.

Unstructured Data Can be seen as a synonym for a simple text document that just contain sentences or language-specific constructs.

List of Figures

1. Introduction

In modern times data has more value than ever. Companies operate on large scale computer systems to evaluate tons of data sets for smarter decisions. Researchers and data scientists are querying large unstructured data sets to find answers, trends and mine new knowledge out of these sets. But a few more fields of application exist, for instance a robot that answer questions that were asked by humans. All of the mentioned and even more tasks can be solved by question answering. QA describes the task from taking a raw bunch of text (a question), process it and end up with providing the right answer. This task is very complex and consists of a high number of algorithms, statistical models and an efficient code base. One of the main challenges in natural language processing is the problem to make text computable. But text isn't text at all.
This thesis provides the first factoid question answering system for the German language. Furthermore it provides a new approach to answer type classification, introduces answer representation and brings in combined data sources for the German language to complete the task.

1.1. Motivation

Question answering is one of the current research fields in the area of natural language processing. Thereby, the research community is limited to English-based systems to participate at well known conferences, for instance the TREC[1]. The review of all known question answering systems leads to the conclusion that no German-based system operating on unstructured data exists, which was one of the main motivations for this thesis. Another main motivation was to get a deep understanding of all natural language processing tasks that are important to create a question answering system. This complex task and the way to create such a system from the scratch leads to a lot of byproducts and spin-offs that can be used for other natural language processing tasks.

[1] Text REtrieval Conference

These tasks are part-of-speech tagging, named entity recognition, semantic annotation and text normalization to name just a few. Another motivation was to create a text processing pipeline that is able to get a deep understanding of the text input to use it for a wide variety of natural language processing tasks.

1.2. Aims

This section describes the goals of the thesis. This will be done in two steps. The first step describes the goals for a practical use of the thesis outcomes and the second part describes the focus during development of the Question Answering system. Below the goals for the practical use are listed:

Proof for German QA in unstructured data sets The systems that are constructed to work on an English corpus are already working. This thesis should provide a baseline for German QA on unstructured data. Furthermore no test sets exists to measure a classifier or system output against another German system. One of the byproducts of this thesis are sets to train answer type and answer representation classifier as well as test set to evaluate the question answering process.

Part of a human machine interaction system The HTW Dresden (University of Applied Sciences) has a project in cooperation with a museum located in Dresden. The university developed a robot that explains the exhibition and guides the visitors through it. Beside a small talk the robot shall also be able to answer technical questions about the exhibition.

Replacing Frequently Asked Questions FAQs on conventional web sites that provide information for customer can be replaced with an input field that searches in company knowledge to provide the correct answer or a comprehensive description.

Querying Text Documents using Natural Language Mining in text data is one of the most demanded tasks of companies and organizations. This can help researchers or employees to get a comprehension of a term exploration that finds related locations, persons, organizations, dates, weights, distance, semantic relations and more. To realize this the system that was created for this thesis builds an ideal framework.

Querying the Word Wide Web using Natural Language All tasks that were mentioned in the aim above, can also be realized online using the knowledge of the world wide web. Furthermore the open domain QA, which is a huge research interest in the NLP community, can be tackled using German language models. The world wide web also has the advantage that systems with low disk space can use the power of the network to produce an answer or a term exploration result.

These goals for the practical use describe a wide range of applications. The system was designed to easily cover all of these aims where the focuses and aims of development can be described as follows:

Serial pipeline Each step shall be done after another for two main reasons. The first reason is to easily switch modules on and off for experiments and the second reason is the quicker understanding of what the system does during the process. For an experimental system this architecture seems to be ideal. This can be parallelized in the future.

Speed The system was designed to do all the time consuming work at start up. This leads to smart time management which results in faster response times and a better performance during QA process. Furthermore all database are loaded into fast Java structures and were mapped for reverse lookup in hash maps.

Client Server Architecture Another important aim was to develop the system as client/ server application to ensure the operation on thin clients that have not enough memory or disk space to execute the complete system. For instance the robot of the HTW Dresden (named *August der Smarte*) needs memory and disk space to handle other tasks like navigation, localization and people detection/tracking. The robot will use this system as a client and connects to the server to use the lightweight API.

Fast configurable A fast configuration is desirable if a lot of parameters can be changed. For this the system has a centralized and well documented configuration file. One of the main reason for formulating this aim is the fast configuration during experimentation stage. The configuration file contains numeric parameters that change the behavior of the system as well as paths that are pointing to a database or model.

Easy Serializable Serialization is a well-known problem to all developers and scientists. JSON (Javascript Object Notation) provides a elegant and slim structure to represent data. This system was designed with the requirement to serialize every object and system state in an easy way. This is necessary for the text annotation, easy debugging and to realize a lean API that consists of just two Java classes which can be serialized to JSON and loaded into an object in an easy way.

Confidence of Question & Answer Question Answering needs a lot of measures to decide which answer is correct. To give the end-user a metric, the system should provide confidences about how well the question was understood and how accurate and valuable the answer seems.

1.3. Related Work

After the definition of the goals and aims of this thesis the research of related work tried to find similar systems and researchers with mutual interests. The issue of comparison with other systems is grounded by two essential problems:

- Returning of related passages instead of facts
- The target language and the associated mechanisms and test sets

All reviewed systems that are described in this section match at least one of this problems. Therefore this section deals with the systems that are the closest to the system that will be introduced and described in this thesis. All systems are explained with their most important features in short. Further considerations are beyond the scope of this thesis. To get advanced knowledge about one of these approaches consult the cited sources.

1.3.1. LogAnswer

LogAnswer was organized by Prof. H. Helbig (FernUniversität Hagen) as well as Prof. U. Furbach (Universität Koblenz-Landau) and is currently the only system that provides Question Answering for German language. The system was introduced [19] in 2008, inspired by other rule-based QA and Natural Language Processing systems like PowerAnswer [33], Senso [37] and DORIS [4]. Since then presented on a few conferences like CLEF[2] and QA4MRE[3].

The system uses a corpus of unstructured data and annotates it with the WOCADI [22] parser to a MultiNet [23] representation of the data. MultiNet is a multi-layered

[2] Conference and Labs of the Evaluation Forum

[3] Question Answering for Machine Reading Evaluation

semantic network that holds knowledge and can be queried easily. The knowledge is stored in the IRSAW retrieval module. This will be used to retrieve specific passages that are related to the answer. The query is decomposed into a logical representation a so-called *conjunctive list of query literals* [19]. After passage retrieval and ranking the answer will be selected. For these logical tasks the system uses the MultiNet Prover [23] as well as the E-KRHyper Prover [35]. Figure 1.1 provides a short overview of the system, as it was provided in 2008.

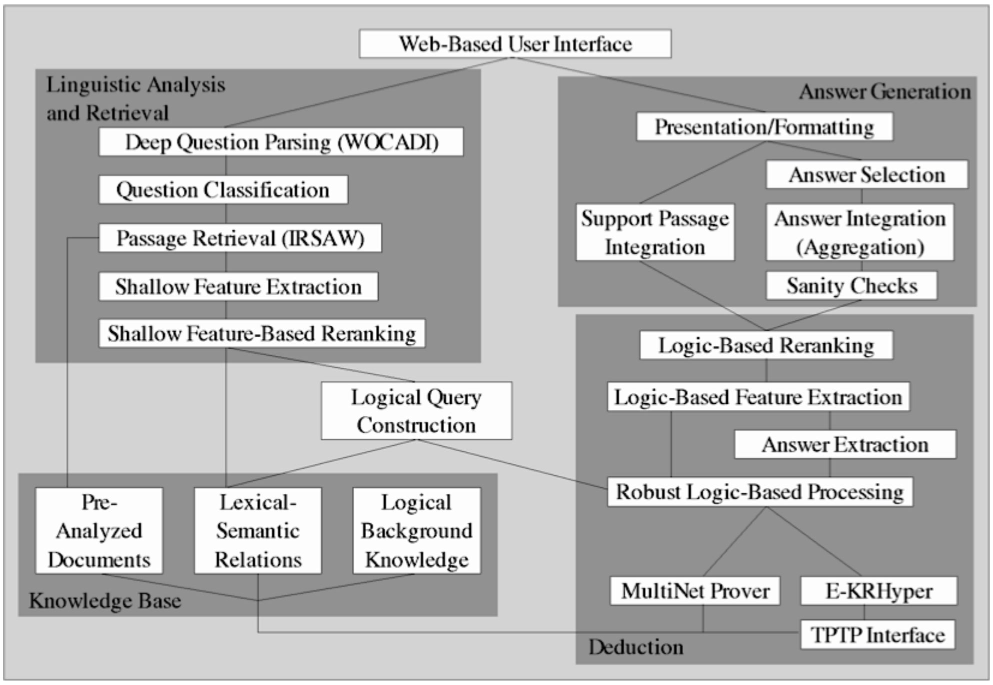

Web-Based User Interface
Linguistic Analysis and Retrieval
Deep Question Parsing (WOCADI)
Question Classification
Passage Retrieval (IRSAW)
Shallow Feature Extraction
Shallow Feature-Based Reranking
Answer Generation
Presentation/Formatting
Answer Selection
Support Passage Integration
Answer Integration (Aggregation)
Sanity Checks
Logic-Based Reranking
Logic-Based Feature Extraction
Answer Extraction
Logical Query Construction
Robust Logic-Based Processing
Pre-Analyzed Documents
Lexical-Semantic Relations
Logical Background Knowledge
Knowledge Base
MultiNet Prover
E-KRHyper
TPTP Interface
Deduction

Figure 1.1.: System architecture of LogAnswer system proposed in [19]. The system consists of four main parts: Linguistic Analysis and Retrieval, Knowledge Base, Deduction and Answer Generation.

According to [19] the following four conceptional steps in the QA pipeline exist. The first step deals with linguistic analysis which transforms the question into a MultiNet representation and does the classification of the question type. The next step then uses the transformed representation to analyze documents that possibly contain the answer. Therefore the knowledge base is requested with found lexical and logical relations. The constructed query and the extracted facts from the knowledge base are then heading into the deduction process. This process contains logical provers that are based on MultiNet mechanisms. After that the proven facts are heading into the process of answer generation. Before the presentation of the answer, the candidates were checked by the sanity criterion and formatted for presentation.

In contrast to the system presented in this thesis, the question classification is limited to two answer types (just definition and factual). The chiAQuery system, introduced in this thesis, uses more classes than just two. Another difference between these two systems is that LogAnswer just returns the passage that contains the answer instead of returning the asked fact, list or paragraph. The advantage of using decomposed rules to represent a question can be the understanding of temporal reasoning, their descriptions and to understand deep semantic relations in general.
The web interface of LogAnswer[4] can be used for queries against the German Wikipedia corpus. The follow up project of LogAnswer is named RatioLog[5].

1.3.2. Ephyra

This system was first developed by Nico Schlaefer and proposed at TREC 2006 [39] as well as TREC 2007 [40]. It works for english language and classifies into 44 question classes. For this classification the system uses lexical features (n-grams), syntactic features (W-word, verbs and more) and semantic features (using WordNet [14] relations). Another classification determines between factoid, list or other questions which can be compared to the Answer Representation type (see section 5.6) of the system described in this thesis. After this the question will be split into focus, property and context. The system contains handwritten-rules that are, depending on the found answer type, searching in the text to find the correct patterns. Question-answer pairs are learned by the system using these rules. The system also uses a pipeline for text annotation containing the Named Entity Tagger of the OpenNLP toolkit[6] and stores the annotated data in the Indri IR engine[7]. Figure 1.2 provides a short overview of the Ephyra system with all sub-systems for processing factoid and list questions. As the figure points out the process from question to answer can be explained in three parts: Query Generation, Search and Answer Selection. Ephyra itself can be seen as a framework which can be extended with generators, miners and filters. The system was renamed into openEphyra and an open-source version of the system can be downloaded on the project page[8].

[4] The web interface of LogAnswer http://www.loganswer.de/

[5] Project page http://ratiolog.uni-koblenz.de/

[6] Can be downloaded on http://opennlp.sourceforge.net/

[7] Project page of Indri IR http://www.lemurproject.org/

[8] Project page of openEphyra http://www.ephyra.info/

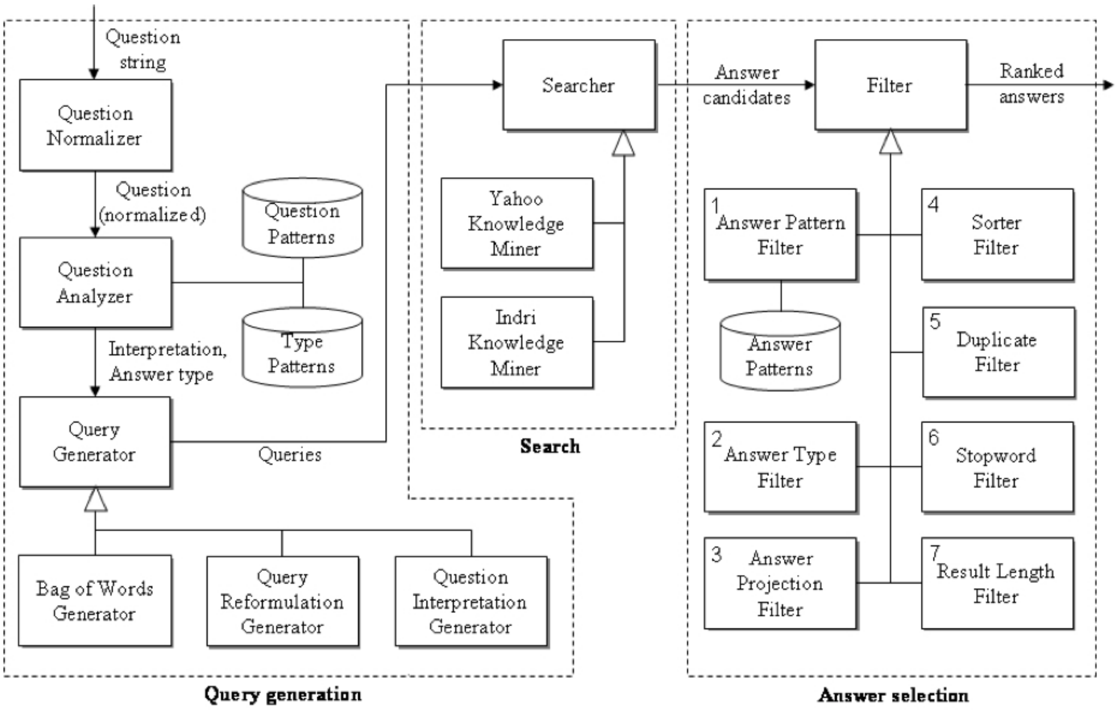


Figure 1.2.: Processing pipeline of Ephyra system for factoid and list questions proposed in [39]. The system is divided into three parts: Query generation, Search and Answer Selection.

1.3.3. QANUS

The system by Ng and Kan [34] that is correctly pronounced *KAY-NESS* was developed to provide a QA framework that combines routines that are often used by Question Answering systems. According to Ng and Kan the entry point for researchers that want to deal with Question Answering is very high. This was the main motivation for the development of QANUS.

The framework is organized as described in figure 1.3. Four main parts exist, that can be divided into:

- Information Source Preparation
- Question Processing
- Answer Retrieval
- Evaluation

All four of these parts have two conceptual sub layers which are *FrameworkController* and a *FrameworkEngine*. These parts can be extended using an own logic to control the behavior of the pipeline. As figure 1.3 shows, the process through the pipeline is quite straight forward.

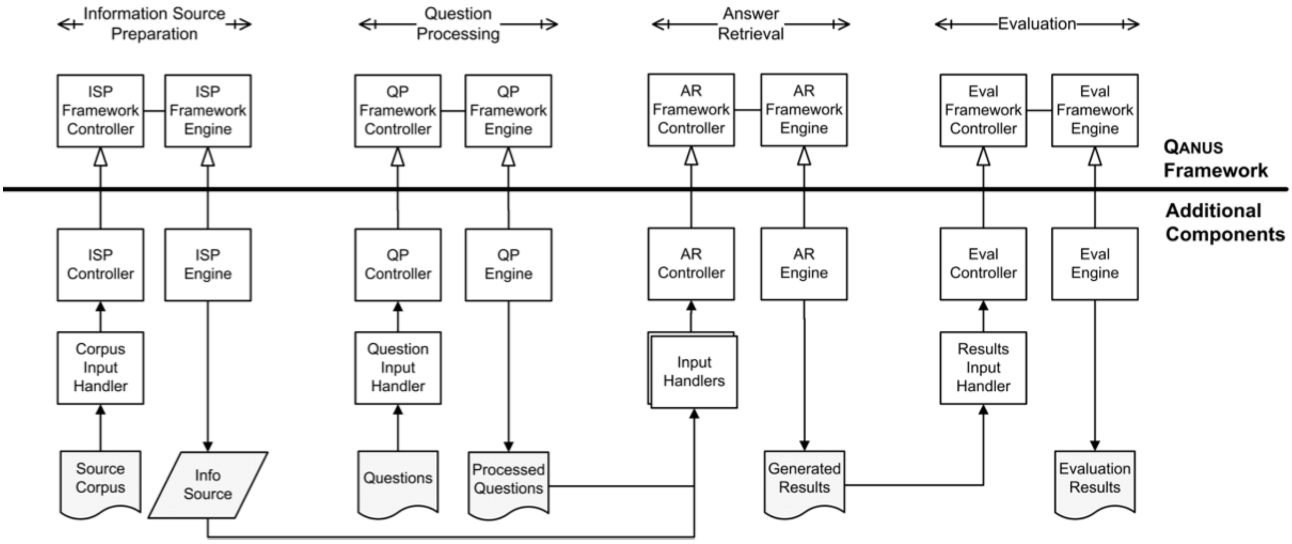


Figure 1.3.: System architecture of QANUS system introduced in [34]. The framework and the additional components are shown.

To show the quality and flexibility of the framework, Ng and Kan developed a Question Answering system called QA-SYS on top of the QANUS framework. The architecture of QA-SYS is shown in figure 1.4. All of the four main parts described above can be found in this figure with their appropriate and specific implementations. Below the four main parts and the specific implementations are listed:

Information Source Preparation They used the AQUAINT-2[9] data and provided an XML-parser for this task. After this the system build a Lucene (will be introduced in section 2.3.4) index on the data.

Question Processing The taxonomy and data from Li and Roth [26] is used in this approach. The work introduced in this work as well as QA-SYS are using the taxonomy proposed by Li and Roth. For classification the proposed classifier from Manning and Klein [30] is used.

[9] Can be downloaded on http://www.nist.gov/tac/data/forms/index.html

Answer Retrieval For retrieval of the correct answer the Lucene query returns passages that are likely to contain the correct answer. Therefore the Named Entity Recognizer proposed by Finkel et al. [15] is used and named entities are linked to the answer type taxonomy described earlier. Based on passage based heuristics and word counts the best answer candidate is selected.

Evaluation QA-SYS handles just factoid questions which leads to a simple measure for evaluation. The evaluation is done using the *accuracy* measure which is defined as *Number of Correct Answers* divided by the the *Number of All Questions*.

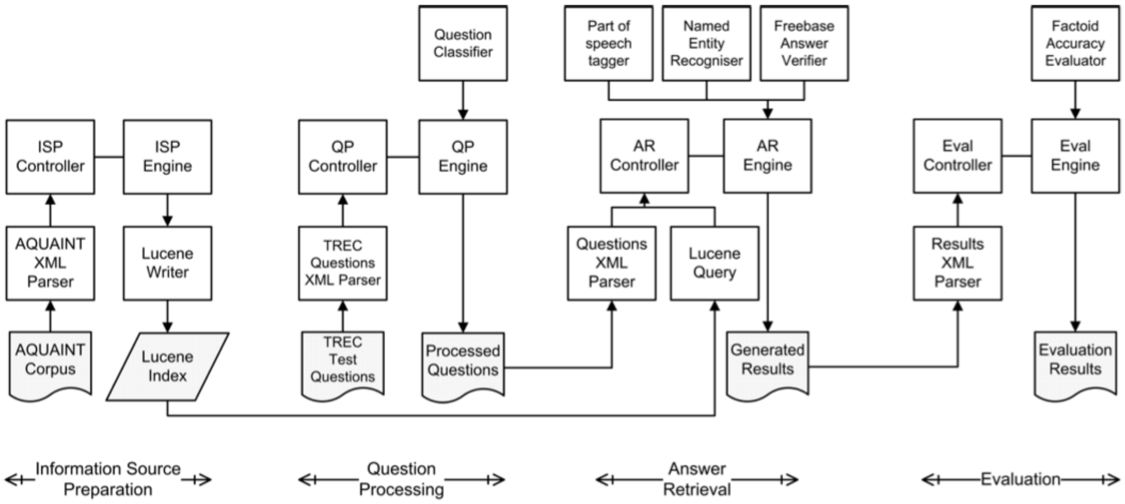


Figure 1.4.: System architecture of QA-SYS system built on top of QANUS [34]. The conceptional framework was already introduced in figure 1.3.

Another system that was built on top of the QANUS framework is iceQA from Geirsson [18]. This system is the first question answering that handles the icelandic language. Geirsson used IceNLP to annotate data from icelandic web services for evaluation. The processing pipeline is similar to the handling of QA-SYS as well as the question classification which is done using the Li and Roth [26] taxonomy.
These two examples of an implementation of QANUS show the practicality of this framework.

1.3.4. Qanda

This system was firstly described in 2002 by Mardis et al. [32] as TREC competitor to other attending systems. Qanda was developed at MITRE[10] and is based on an Annotation Management System which is in turn based on a semantic tagger, annotator and mapper for text called Catalyst[11]. These frameworks ensure an abstraction layer to build rich Question Answering systems.

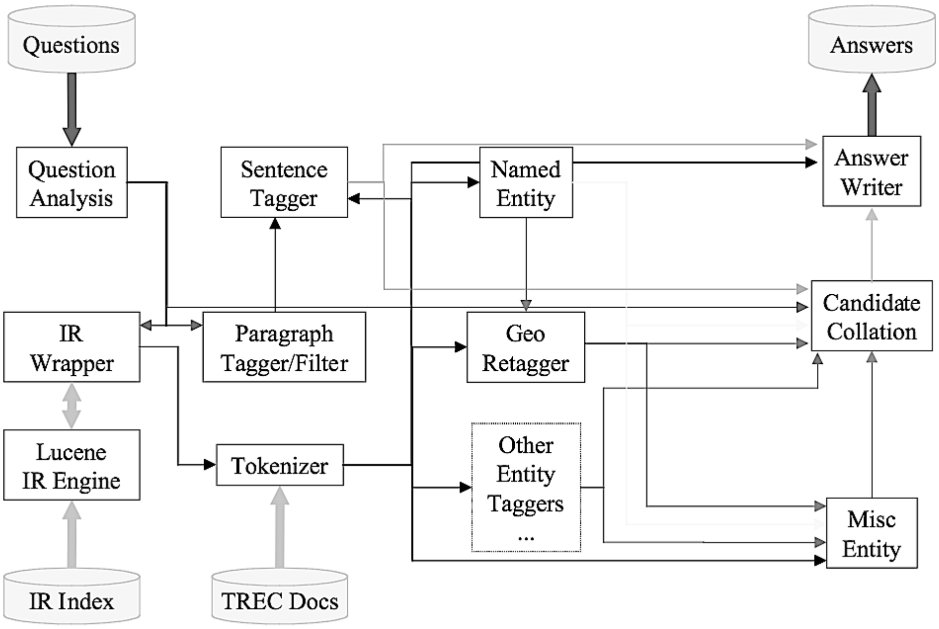


Figure 1.5.: System architecture of Qanda as it was introduced in 2002 by Burger and Mardis [32]. The figure is specialized to show the straight way from question to answer.

The system deals with factoid question as well as with 'definition' questions. The following list describes the main components of the system that are also shown in figure 1.5 as conceptional overview:

Document Processing Here the documents undergo a deep analysis of structure and semantics. A POS-tagger, NE-recognizer and a tokenizers are used as well as the extraction of temporal expressions.

[10] MITRE Research is a not-for-profit organization that is devoted to federally founded research and a FFRDC of the United States.

[11] A project that received funding from the European Union's Seventh Framework Programme for research. *n°6611188.* Project page http://catalyst-fp7.eu/

Question Analysis The question is chunked and parsed to find meaning and need of the given question. The meaning of the question is expression in the following terms: anchor, property, name, answer restriction, event, salient entity, geographical restriction, temporal restriction and superlative.

IR Wrappers The wrapper for Information Retrieval is provided in the AMS system. Qanda uses Lucene (will be introduced in section 2.3.4) for coarse document search. The queries are processed using the phrase operator and the weights that the Lucene system provides.

Passage Processing Qanda applies the log-IDF measure on all retrieved documents to find the most related passages in the result set. The most related passages are then forwarded to the rest of the Question Answering pipeline.

Fixed Repertoire Taggers These taggers find named entities like locations and are able to specify them as cities or states. This is done using fixed word lists. The taggers are also able to find animals, metals and a few more categories.

Numeric Taggers Beside the fixed repertoire taggers the numeric taggers are used to find numeric values and their corresponding measures or units to determine the semantic of the numeric value, which can be: currency, weight and so on.

Overlap Here WordNet [14] comes into account to find overlapping features between the question and the retrieved passages with their specific tags. Some of these features are named: Context IDF Overlap, Context Unigram Overlap, Context Bigram Overlap, Context Question Restriction Overlap, Context Salient Overlap and Context Synonym Overlap. A few other features extract but for a short summary the features mentioned here are sufficient.

Answer Collection and Ranking With all information from the Overlap-process the possible answers are collected and ranked using the features mentioned above.

Answer Selection After the ranking the selection algorithm decides which answer will be the final one. Therefore the factoid and list answers are just chosen from ranking but the questions that ask for a definition or a paragraph need to be preprocessed before printing.

The components described above are from Bayer and Burger [7] where the participation of Qanda in TREC track[12] was discussed.

[12] Text REtrieval Conference Open-Domain Question Answering competition

1.3.5. AskMSR

The AskMSR (Ask Microsoft Research) project was developed by Banko et al. [2] and [6] in the Microsoft Research Labs in Redmond. The system overview is shown in figure 1.6 with all components that the basic system had. The components are shortly explained below:

Rewrite Query The asked question will be rewritten by morphological similarities that the system found in its database. For the example in figure 1.6 the type of question is a location and the question is transformed into multiple versions that ask for a location. The queries are then sent to a search engine (in this case Google Search).

Mine N-Grams From the queries that were sent to the search engine the 1-, 2- and 3-grams are counted and ranked by their occurrence. For this only the summaries of the search engine result pages (SERPs) are used.

Filter N-Grams These ranked n-grams are then mapped to the expected answer type with a few handwritten rules and filters.

Tile N-Grams After the filtering the n-grams are tiled together to construct an answer from these rankings and processing.

The description of the system components points out that this system is only able to solve factoid questions. Because of the n-gram mining structure of the system no processing of 'description' or 'explain' question was intended. The ARANEA system which is a complete re-implementation of AskMSR was proposed by Lin [28]. Lin tackles

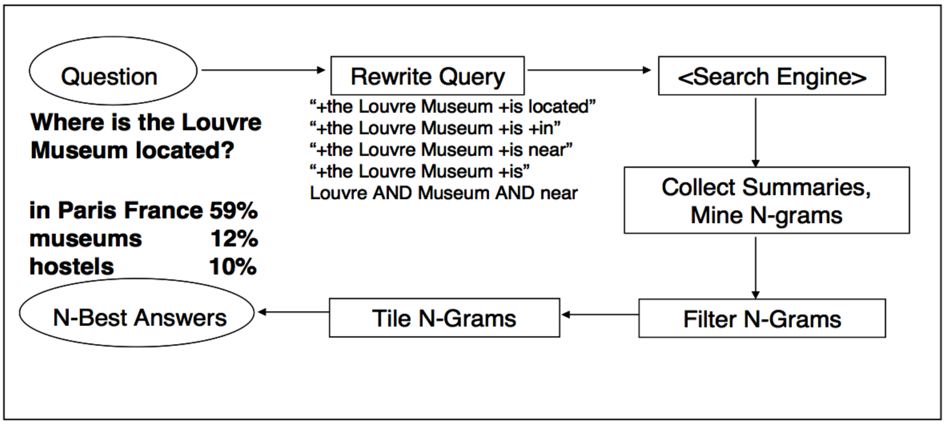

Question
Where is the Louvre Museum located?
in Paris France 59%
museums 12%
hostels 10%
N-Best Answers
Rewrite Query
"+the Louvre Museum +is located"
"+the Louvre Museum +is +in"
"+the Louvre Museum +is near"
"+the Louvre Museum +is"
Louvre AND Museum AND near
<Search Engine>
Collect Summaries, Mine N-grams
Filter N-Grams
Tile N-Grams

Figure 1.6.: System architecture of AskMSR as proposed in Banko et al. [6]. The n-gram approach is presented here.

the redundancy-based approach again and described his system more modular than AskMSR. Furthermore his system provides a better control of experiments caused by its architecture. In addition Lin claims his work as a guideline for future researchers dealing with Question Answering.

1.3.6. QuASE

Sun et al. [41] proposed a system that is named QuASE (Question Answering via Semantic Enrichment) and uses semantic resources to find the correct answer. The researchers are claiming with three main contributions which consist of the new QA framework, the new Answer Type Checking Model and Extensive Experimental Evaluation. All of these three contributions are new in their field and a step towards a rich Question Answering system. They are using Freebase[13] as a semantic knowledge base (KB) to find semantic relations in the question and to answer them. For linking text passages to entities an Entity Linking Tool [10] is used. Below the Question Answering pipeline of QuASE will be described in short:

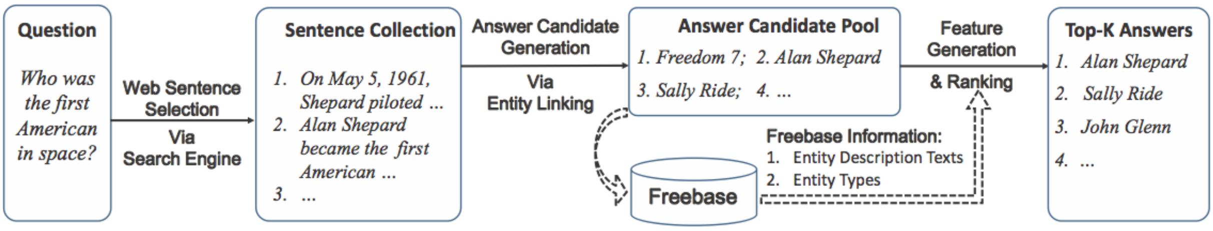


Figure 1.7.: The system framework as introduced in Sun et al. [41]. The system uses infromation from a structured database (Freebase) to find the most likely answers.

Web Sentence Selection via Search Engine The question acts as a query for a commercial search engine (Google Search in this case). The best 50 documents are processed. This processing calculates a cosine similarity of word count vectors. Therefore every sentence in the documents is transformed in a word vector representation w_s. The question is also transformed in a word vector representation w_q and the cosine similarity is calculated between the question and the sentences $similarity = cos(w_q, w_s)$. The sentences that deviates the most are discarded.

[13] Freebase will be no longer maintained and was merged into the Wikidata project.

Answer Candidate Generation via Entity Linking Here the entity linking comes into account that tries to find entities in Freebase that are related to the question and the answer.

Feature Generation and Ranking After entity linking the similarity scores are calculated and ranked. For this a bunch of features are used: counts, frequency, textual relevance on question and answer candidate sides.

The researchers also propose a new *Word to Answer Type Model* that calculates the probability $P(t|w)$ of an observing Freebase type t when word w exists in the question. After a detailed research of related works this system can be seen as one of the most modern systems in Question Answering.

1.4. Conclusion of Related Work

The systems, discussed in section 1.3, are chosen because of their similarity to the system developed in this thesis. As mentioned earlier there is no system available which tackles exactly the same field of research. Therefore the most similar systems were chosen for the English language.
After studying these systems, a bunch of conclusions can be drawn. The use of the answer type taxonomy from Li and Roth [26] is used in nearly all systems. Therefore this taxonomy can be seen as a state-of-the-art and was also implemented in this system. The concept of proving facts as presented in [19] was not inherited in the system, because the work to maintain the MultiNet including all analysis applications was beyond the scope of this work, but should be tackled in further experiments. Nearly all of the explored related systems are using a Part-of-Speech tagger as well as a recognizer that finds named entities. Therefore these parts were also implemented in this system to handle the grammar of a sentences and find relevant organizations, persons or locations. The Ephyra system [39] also uses the Indri IR engine for a coarse search. This idea, of a fast coarse search, was adapted by the system presented here which uses Lucene search engine. The choice of using Lucene was driven by the slim and fast architecture of the Lucene system. In contrast to Ephyra, the Qanda system [32] uses Lucene and a mechanism that is called numeric taggers. Beside the extraction of named entities, this system extracts also numeric values and their corresponding measures. This idea was adopted for the presented system and implemented into semantic extractors. The AskMSR system [2] brought the concept of redundancy to the presented system. AskMSR uses a

redundancy approach to rank the correct answer. The idea behind this is that words that occur frequently, are possibly the correct answers. In the system that is presented here, the score of similar answer candidates are accumulated to reach a higher scoring. The latest system of all presented is the QuASE system of Sun et al. [41]. This system has an enhanced approach to handle answer type classification. Instead of a rigid taxonomy, a graph in a semantic network was used to estimate the answer type. Each node in the graph has a probability for each answer type. This paper was released after the work on this system was started. Thus, the findings of this work couldn't be tested, but the reflections of the paper together with revealing experiments were added to the further work section.

1.5. Outline of the Thesis

After this introduction to the topic, the thesis is structured as follows:

Chapter 2, Technical Base Provides an overview of used techniques, databases and software package to create the Question Answering system.

Chapter 3, The chiAQuery System Introduces the systems architecture, explains the starting options as well as features and defines the directory structure.

Chapter 4, Text Annotation Explains the procedure of annotating unstructured documents and the ChiaqueryKnowledge module that takes care of this process.

Chapter 5, Input Processing The way from a question as text input to a question with semantic interpretation will be explained in this chapter.

Chapter 6, Query Processing All procedures that were done to the semantic interpreted input to find answer candidates will be introduced here.

Chapter 7, Answer Selection Explains the approach of selecting an answer or a list of answers and how they will be displayed.

Chapter 8, Experiments & Evaluation The experiments will be introduced and evaluated in this chapter. Furthermore the competitiveness to other systems will be compared.

Chapter 9, Conclusion & Further Work This thesis can not cover all aspects and optimizations of this wide field. Therefore a conclusion of the current system and further work tasks as well as idea lists will be presented in this chapter.

2. Technical Base

In this chapter the technical base of the master thesis will be introduced and explained. The used notations are explained as well as all measures to evaluate the performance of the system. Used machine learning techniques are introduced and motivated. Furthermore the used software packages are explained and motivated. These packages are very important for the success of this work, because they cover a lot of aspects for instance pre-processing. The used data sources are also named and analyzed in this section.

2.1. Notations & Measures

In this section all used notations and measures are explained. Notations are very important to have consistent pronunciations and symbols when defining an internal and external representations of the system. Also unified performance measures are important to have comparable outcomes of the system. This section also provides an introduction to all measures used in this thesis.

2.1.1. Answer Type Detection

To process the answer to a given question the system has to determine what the question is asking for. This task is done by a module called *Answer Type Detector*. The detector calculates the confidences of which class is the most likely one. For this the detectors first feed a feature extractor with the question. This extractor performs the transformation from an asked question as string to a n-dimensional feature vector which can be seen in equation 2.1. It finds several characteristics in the string and maps this to the feature space.

$$\text{“}Wann\ starb\ Konrad\ Zuse?\text{”} \longmapsto \begin{pmatrix} f_1 \\ \vdots \\ f_n \end{pmatrix} \tag{2.1}$$

The answer detection then learns the mapping from the feature space (provided by the extractor) to the classes which the system distinguishes. These classes and their notations are introduced in figure A.8.

```
ENTITY with sub class ANIMAL results in ENT_ANIMAL as class name
```

As can be seen in this figure the sub classes are denoted with the three first letters of the main class followed by an underscore as well as the name of the sub class.

2.1.2. Answer Representation

Follwing the detection of an answer type the system also needs to define the way of representing the answer. For this reason an extractor finds significant features in the input string and returns a vector representation as already shown in equation 2.1. The detector of answer representation then uses the extracted feature vector to learn the mapping to the three classes that can be seen in figure 2.1.

Class Name	**Description**
FACT	A question asks for one fact and nothing more. An example for this answer representation is: 'Wer war der erste Mann auf dem Mond?'
LIST	The question requires more than one fact. This list delivers the most likely facts of the system. An appropriate example is: 'Welche Menschen waren zuerst auf dem Mond?'
PARAGRAPH	In contrast to list and fact this answer representation calls for a bunch of sentences or a story. For example: 'Wer war Neil Armstrong?'

Figure 2.1.: Answer representation types of the Question Answering system

2.1.3. Performance Measures

Following the performance measures used in this work are defined and explained in short. Machine learning systems need a lot measures to be evaluated accurate and do cover all expects of failure and faults that can possibly occur during the run of the system. Beside the theoretically explanation of the measures and metrics, this section also provides a practical example: medical cancer detection. Let's assume a classifier exists, that receive a set of features that are significant for detecting cancer and tries to classify the feature set into two classes: CD (cancer detected) and NCD (no cancer detected).

Accuracy

Accuracy is one of the simplest measures for evaluating systems. Equation 2.2 shows how the accuracy works (the '#' stands for 'number of').

$$ACC = \frac{\#\, correctly\, classified\, samples}{\#\, all\, samples} \tag{2.2}$$

This measure returns a percentage value of how many samples of all samples were classified or rated correctly. To measure a overall performance the accuracy measure can be used. Applied to the cancer detection example, accuracy points out that the percentage of all correctly (patients with non cancer to non-cancer class as well as patients with cancer to cancer class) classified samples divided by all patients that underwent the cancer test.

Positives & Negatives

To measure positive and negative answers of a system, four measures were developed to categorize data in four classes. For binary classification where just positives, denoted with P, and negatives, denoted with N, exist. All of these are described below:

True Positive (TP) This means all samples that were positive (P) and classified as positive.

True Negative (TN) True negatives describe all samples that were negative (N) and classified as negatives.

False Positive (FP) This class contains all samples that were wrongly classified as positives.

False Negative (FN) False negatives contain all samples that were wrongly classified as negatives.

In statistical hypothesis testing, FP and FN are often named type-I-error and type-II-error because both are describing a rejection of a null hypothesis. FP describes a rejection of a true null hypothesis with the detection of an incident that is not present (type-I-error). On the other hand FN means the rejection of a false null hypothesis with no detection of an incident that is present.
The cancer detection example can be applied as follows:

TP Contains all patients that were classified with CD and truly suffer from cancer.

TN Contains all patients that were classified with NCD and truly not suffer from cancer.

FP Contains all patients that were classified with CD, but don't suffer from cancer.

FN Contains all patients that were classified with NCD, but actually suffer from cancer.

The four terms TP, TN, FP and FN are used in the following sections to explain more complex measures.

Sensitivity & Specificity

With the definition of the four variations of positives and negatives two measures can be specified.

$$SEN = \frac{TP}{P} = \frac{TP}{TP + FN} \tag{2.3}$$

The sensitivity SEN also called the true positive rate (TPR) is defined in equation 2.3. It specifies the proportion of correctly classified positives (P) to the false negative rate (FNR).

$$SPC = \frac{TN}{N} = \frac{TN}{TN + FP} \tag{2.4}$$

Specificity SPC or true negative rate (TNR) defines the correctly classified negatives (N) divided by the false positive rate (FPR). This measure is denoted in equation 2.4. These both measures can be applied to cancer detection example. Specificity predicates how good a test performs on avoiding false alarm. The goal is to maximize the TN (correctly classified NCD) and the minimization of the FP (wrong classified CD). In contrast, sensitivity also known as recall tries to maximize the detection of positives. For this the sensitivity tries to maximize the number of TP (correctly classified CD) and minimize the number of FPs (wrongly classified CD).

Precision & Recall

Beside sensitivity and specificity there are more measures to describe the performance of a system. To be exact it is just the precision which is a new measure. The recall measure was formerly introduced in equation 2.3.

$$PR = \frac{TP}{TP + FP} \tag{2.5}$$

Precision also named positive predictive value is defined by all correct classified positives (P) divided to all positive test outcomes. The measure is denoted in equation 2.5.

$$RE = SEN = \frac{TP}{P} = \frac{TP}{TP + FN} \tag{2.6}$$

Recall also called sensitivity (eq. 2.3) defines the correct classified positives (P) divided by the sum of all samples that are positives by condition (all samples that are labeled as positives). The precision can be seen as the relevance score. Applied to the cancer detection sample this test tries to maximize the TP (correctly classified CD) and minimize the FP (wrongly classified CD). To sum up, the test determines how many CDs really suffer from cancer.

F-Measure

Chinchor [9] proposed the F-Measure at the Fourth Message Understanding Conference in 1992. It can be seen as the harmonic mean of precision (PR, eq. 2.5) and recall (or sensitivity, RE, eq. 2.6).

$$F = \frac{2\,PR\,RE}{PR + RE} \tag{2.7}$$

This measure is widely used to evaluate machine learning systems.

ROC Space

The Receiver-Operator-Characteristics area is often used to evaluate and optimize processes in signal discovery. The visualization shows the efficiency in dependence of the error rate. The application of ROC is to find a optimal value of a parameter, in most cases, for a two classes problem. In machine learning and information retrieval the ROC is, for instance, applied to measure the quality of a classifier.

For this the TP (true positive rate) and FP (false positive rate) values of the process need to be estimated. With this both measures the curve or just a specific point can be drawn. If the curve approaches to a diagonal the outcome of the process seems to be a random process. The area under the curve can be seen as the quality of the process. The larger the area the higher is the quality of the process. Figure 2.2 shows an example ROC space with classifier results and there true positive rate to false positive rate measure. This example shows just one data point (a true positive rate with the corresponding false positive rate). Applied to the cancer detection example the ROC space depicts the correctly classified CD (patients who really suffer from cancer) on the

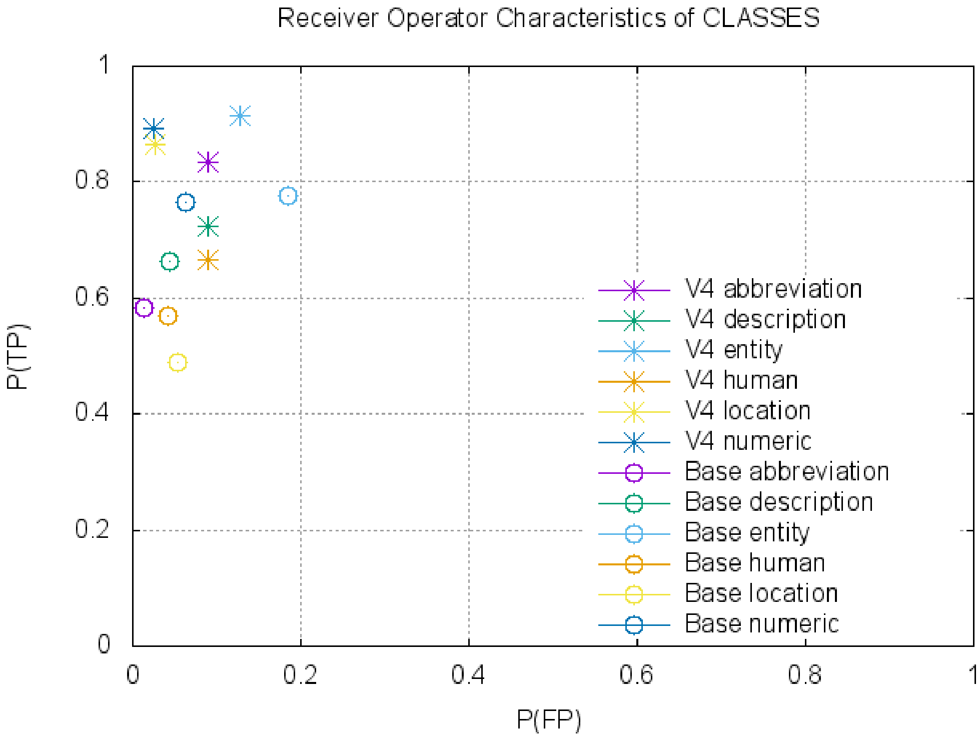


Figure 2.2.: An example ROC space that shows the true positive rate on the y-axis and the false positive rate on the x-axis.

y-axis and the wrongly classified CD (patients who are classified as 'suffer from cancer' but the don't) on the x-axis.

2.2. Machine Learning Techniques

For this work a few machine learning techniques are used to train statistical models. In this section these techniques are shortly introduced to get a comprehensive understanding of the models. The motivation is not to fully explain all of these techniques in detail with the corresponding theory. Here, just the basics are presented. For further and deeper explanations follow the cited works.

2.2.1. Naive Bayes Classifier

The Naive Bayes classifier is one of the simplest classifier that are based on the frequency table concept. The classifier itself can be defined as a function b that projects a n

dimensional vector to a class C.

$$b : \mathbb{R}^n \to C \tag{2.8}$$

The classifier solves the problem in equation 2.8 by using the Bayes' theorem which describes the probability of an event based on another condition. The probability of an event is denoted with $P(A)$ which points out the probability of event A to occur. The probability of occurrence of an event B under the condition that event A also occurs is denoted with $P(A|B)$.

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \tag{2.9}$$

Equation 2.9 presents the Bayes theorem that is used to calculate the confidence values for the classes. For this calculation a possible formula could be to calculate $P(C_1|F_1)$. This points out the probability of class C_1 when feature F_1 occurs. According to equation 2.9, the event A equals the event that a class (in this case class oneC_1) occurs and the event B equals the occurrence of feature F (here feature one F_1). For this classifier a few different implementation exists, for instance: Gaussian naive Bayes, Multinomial naive Bayes and some more. To understand the theorem equation 2.10 explains the conceptional meaning of the probabilities.

$$posterior = \frac{likelihood \times prior}{evidence} \tag{2.10}$$

The implementation of this classifier can be seen as data-driven analysis using a histogram of appearing features. The appearance of each event (class or feature) will be counted as well as the combined appearance of them. After that the theorem can be applied on the histograms.

This classifier is often used in Natural Language tasks for instance:

- Spam Detection
- Sentiment Analysis
- Text Classification

Beside the tasks mentioned above there are a lot of other tasks in the area of machine learning that are using Naive Bayes as a baseline method.

2.2.2. Multilayer Perceptron

This classifier is motivated by the biological model of the human brain. The brain consists of a huge number of brain cells (called neurons) that have connections to other cells through synapses. The brain works with electric potentials and stimulis. If a neuron gets stimulated with an electric stimuli it fires and propagates the potential through the parts of the neural network. One way how the brain learns is to adjust the synapse (and the synaptic cleft). The more important the information is (and the event of remembering was triggered) the thicker the synapse and the layer of myelin around the synapse get. To bring this model to the computer the neurons are used as calculation nodes and the weights (synapse) between neurons are adjusted during training time. The procedure of adjustment is called back propagation where the output of the net is compared to the correct output and then the weights are adjusted to match the output using a step range variable or learning rate (often denoted as η). The multilayer perceptron is a special case of an artificial neural network which works in a feed-forward manner. This means that each neuron from a layer is connected to all neurons from the next layer. This principle is also illustrated in figure 2.3. The figure provides a visualization of a multilayer perceptron where $X \in \{x_1, ..., x_n\}$ denotes the feature space (input, the properties to classify), $w_{i,j}$ the weight from neuron i to neuron j and $y \in \{y_1, ..., y_m\}$ denotes the classes to classify in. To provide a comprehensive example,

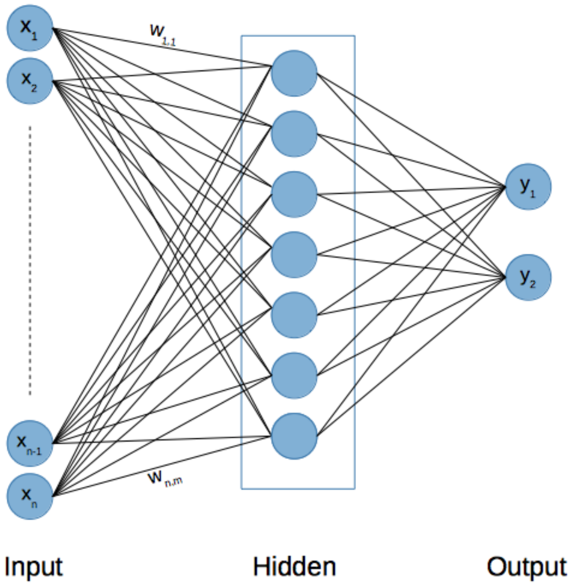


Figure 2.3.: Visualization of an mulitlayer perceptron. The feature vector is applied on the input layer and forward propagated to the output layer.

the classification of questions will be explained in short. After a visual inspection into the data (a set of questions) maybe the idea that the words 'Where' or 'Wo' point to a possible class 'location' comes up. This can be seen as a typical feature and the appearance of this word seems to be significant for the 'location' class. Therefore this feature can be added to the feature vector (for instance in a binary manner, appears or not). For each possible class features need to be found and added to the vector. This procedure is called feature extraction. After the engineering of each feature, the transformation from question string to feature vector can be made. The neural network learns the representation from feature vector to the class label (in this case 'location'). During the training of the neural net, the algorithm tries to minimize the error (delta) between the wanted output and the calculated ouput. When the neural net is used for classification tasks, then the output of the net contains the confidences (a real number in the range $[0, 1]$) for each class with a given feature vector.
An artificial neural network is trained with a back propagation algorithm. This algorithm works in the following steps during training time:

1. An input pattern $\{x_1, ..., x_n\}$ will be forward propagated through the net.

2. After that the produced output will be compared with the correct output.

3. The difference (delta) can be seen as the error and the algorithm propagates back though the net and adjusts every weight concerning to its influence to the error value.

2.3. Software & Frameworks

On the following pages the software that was used to complete the work in this thesis will be introduced. Question answering is a strong research area of Natural Language Processing wherefore a wide range of reliable frameworks is necessary. The system that is described here needs a fast and reliable pipeline to train statistical model, this pipeline will be realized using KNIME and Wek3. Both applications are described in sections 2.3.1 and 2.3.2. To realize a complex system that answer questions a reliable framework is needed that is able to cover all necessary and basic tasks of modern question answering systems. For this the thesis uses Stanford NLP Core which contains Part-of-Speech taggers and models that find named entities (like persons, locations and organizations) in german text. Stanford NLP Core will be introduced in section 2.3.3. A coarse search on indexed knowledge is realized with the Apache Lucene system. More details on this systems are provided in 2.3.4. Especially to tackle the open domain question answering task, a system that scrapes and handles web documents is needed. For the presented system the two applications JSoup and Palladian are used and introduced in section 2.3.5. At last, the serialization of the results and documents is an important feature. The system solves this task using the GSON library which will be introduced in section 2.3.6.

2.3.1. KNIME - Konstanz Information Miner

To have a fast pipeline to prototype experiments and to test features KNIME[1] can be used. KNIME was developed by the University of Konstanz in cooperation with software developers from silicon valley and first applied [43] in the area of biology and chemistry The work flow can be seen as tool or processing chain where each processing step has input and output and additional steps can be added using a graphical UI. Furthermore KNIME provides a rich tool set with a lot of functionality that can be parameterized, executed and easily substituted with other processes that the tool chain needs. For this thesis KNIME was used for:

- Evaluate the quality of features with machine learning models and visualize results
- Write *arff* files to train models and other I/O tasks
- Explore questions, feature vectors and other data
- Transform, filter and convert data

KNIME was released under GPL license.

2.3.2. Weka3

The Waikato Environment for Knowledge Analysis [20] was introduced in 1997 by Cunningham [11]. Since then the framework was developed to classify, cluster, analyze and visualize data sets. The fields of application are very wide. To train statistical models for this thesis Weka was used to easily generate a model from a given *.arff* file. The framework offers all common used machine learning techniques with all needed parameters for these tasks. Furthermore it is possible to integrate the Weka environment into the code of the software that was developed during the work on this thesis. This integration can

[1] Project page http://www.knime.org/

Tag	Meaning
I-LOC	location
I-ORG	organization or company
I-PER	person or human-like object
I-MISC	other entity

Figure 2.4.: List of all named entity classes of the Stanford Core NLP tagger

simplify the analysis of data and training of models which then leads to a one-click task to train more accurate models on new data or to rearrange an experimental stage.
The Weka3 framework was released under GNU General Public License.

2.3.3. Stanford NLP Core

The Stanford NLP Core [31] package is developed and maintained by the NLP Group[2] of the Stanford University. The software contains POS taggers, named entity recognizers, annotators, tokenizers, sentiment analyzers, regex tools and a bunch of more tools for multiple languages. This packages is often used for complex NLP tasks to reduce the complexity of text.
In this thesis only the Part-of-Speech tagger as well as the named entity recognizer are used. The Stanford NLP Core is licensed under GPL[3] version 3, some parts of it are still licensed under version 2 or version 2+.

Named Entity Recognizer

The system proposed in this thesis works with a model that was trained on the HGC (Huge German Corpus) corpus from CoNLL[4] 2003. The data consists of 175 million tokens from German news-wire data that was clustered into 600 clusters. The recognizer (based on a CRF model) was, beside others, described in [42] and the trained model was proposed in [13]. The model was released under a simple GPL license. The provided classes of the Named Entity Recognizer are shown in figure 2.4.

[2] Web site of the Group: http://nlp.stanford.edu/

[3] GNU General Public License

[4] Conference on Computational Natural Language Learning which took place in Edmonton at May 31 and June 1.

Part Of Speech Tagger

The POS Tagger finds grammatical parts in the input text is implemented as a Maxium-Entropy Model (log-linear) and was proposed by Toutanova in [44] and [45]. The trained model is based on the NEGRA [5]corpus that was tagged with the STTS tag set [38] which consists of 54 different tags. These tags are listed in figure A.9. Stanford Core NLP provides a few trained models but the work in this thesis uses the *german-fast* tagger which is significantly faster and provides the best speed to accuracy ratio for the needs of this work.

2.3.4. Apache Lucene

Lucene is a project of the Apache Software Foundation and often used by internet and other information technolgy companies for term search in big data sets. Lucene itselfs creates an index based on the occuring words and is able, based on these statistics, to find the most related documents. For Question Answering tasks Lucene can provide the coarse document search to filter all irrelevant documents that have no connection to the question asked. In the software that was developed for this thesis, Lucene was used with the *GermanAnalyzer* and some parts of the system are using the *GermanStemmer* as well. Both parts of Lucene are described below:

Search using GermanAnalyzer Lucene uses analyzers to analyze documents as well as the incoming query. To get more accuracy out of the data the analyzer of the target language shall be used. This analyzer adds the language-specific *Standard-Tokenizer*, *StandardFilter*, *LowercaseFilter*, *StopFilter* and *GermanStemFilter* to the processing pipeline.

Stemming using GermanStemmer The stemmer is based on a proposed stemmer from Caumanns [8]. Stemming means the reduction from a given word to its stem (root word). To put similar words in the same bucket during processing can help to find relations and also reduces complexity of data. For index structures a stemmer can be applied, where the stem can be used as a key which points to a list of words or a list of sentence ids that contain this or similar words with the same stem.

Apache is known for using the Vector Space Model and the TF-IDF measure. This software was released under Apache License 2.0.

2.3.5. JSoup and Palladian

Two frameworks for web crawling and scraping are JSoup[5] and Palladian[6]. JSoup was developed by Jonathan Hedley with the ability to scrape and parse HTML files from files or strings, find and extract DOM elements, manipulate HTML elements and tidy up the HTML code. This project uses JSoup as one of two frameworks that are tested for the optimal web scraping technique for Question Answering on websites. JSoup was released under the MIT license.

Palladian was developed at the TU Dresden and released under Apache License 2.0. It provides text classification, language detection, keyword extraction, content extraction, web search, web crawling, web feed detection and sentence splitting to name just a few. For this work Palladian was used to evaluate the readability performance and the performance of the main content extractor against JSoup.

2.3.6. GSON - Google JSON

Google developed GSON[7] as a JSON library to extend the Java Generics. GSON is able to convert Java object into JSON objects and backwards. This can be very helpful if you want to print objects and serialize them in a readable and easy manner.

This project uses GSON to:

- Serialize annotated documents (readable and JSON compliant)
- Serialize system states to disk
- Debugging and printing of complex structures in CLI
- Realize an easy API for server/client architecture

The software was released under Apache License 2.0.

2.4. Data Sources

To train models and to provide semantics to enrich documents and queries a lot of data is needed. This section deals with all sources that are used to create semantic information and connect words and meanings with each other. Also models need to be

[5] Project page http://jsoup.org

[6] Project page http://palladian.ws

[7] Project page https://code.google.com/p/google-gson/

trained and therefore a huge set of data was annotated for this. A source that contains synonyms and other linguistic relations are needed to extend a possible query to the system. Therefore the GermaNet was acquired and will be explained in section 2.4.1. Another important task is to find semantic frames and relations in words. To tackle this, the SALSA project is used. It contains the largest database of word to semantic frame mapping for the German language and will be described in section 2.4.2. Classifiers need a lot of training and testing data. For this task no data exists, so data was created and labeled only for this work (section 2.4.3). The presented system was originally created to answer closed domain questions on a given data set. This data set will be described in section 2.4.4.

2.4.1. GermaNet

The German counterpart to the WordNet [14] is called GermaNet [21] [24] and contains lexical-semantics of German verbs, nouns and adjectives. The GermaNet was developd by the University of Tübingen and available under Academic Research License, Research and Development License and Commercial license. For this work the data was acquired using the Academic Research license. For the current version 10.0 the content of the GermaNet can be seen in figure 2.5. For this thesis the GermaNet is used to find semantic relations in documents and incoming queries. After this the data can be extended or semantic concepts can be applied to the data.

Parts	#
Lexical units	131814
Conceptual relations	114619
Lexical relations	4199
Split compounds	54572
Interlingual Index records	28567
Wiktionary descriptions	29535

Figure 2.5.: GermaNet statistics of version 10.0

2.4.2. SALSA

The SALSA[8] project [12] of the University of Saarbrücken provides a large and frame-based net for German language. The data can be used to detect semantic frames in text

[8] Saarbrücken Lexical Semantics Acquisition Project

with rich semantic and syntactic properties and examples. The english counterpart is the FrameNet and SALSA follows the internal structure of FrameNet as well.
For the Question Answering system developed in this thesis the uses SALSA to find semantic frames in the documents and the incoming questions.

2.4.3. DATD & AR Data

This data was annotated just for this thesis to train the models for the:

- Answer Type Detector which is denoted with DATD and
- Answer Representation which is denoted with AR.

The annotators of this data are asked to write down questions for each class of the class set that can be seen in A.8. Furthermore the task of the annotators includes 4 up to 5 questions for each specific sub class with a given annotation schema. These data sets can be seen in figure 2.6. The data sets are used in chapter 8 for experiments and evaluation of the proposed system.

name	#	description
DATD-1	1543	Recorded by annotators for DATD task
DATD-2	733	Based on randomly chosen samples from DATD-1. Questions are modified for testing process.
AR	502	Recorded by annotators with AR-specific labels

Figure 2.6.: Self annotated data sets used in this thesis

The annotation follows the structure that is shown below:

```
class#subclass#question
```

Applied to an example the annotation for an *ENTITY* sample (answer type classes are explained in section 2.1.1) that relates to the sub class *PLANT* looks like:

```
ENT#ENT_PLANT#Welche Pflanzen wachsen in Südamerika?
```

All annotators are listed in A.3.

2.4.4. TSDD Data

This data set represents the content from the *Technische Sammlung* in Dresden. The robot of the HTW Dresden was build to serve and guide visitors through the exhibition. This vehicle was one of the target systems for the software system developed from this thesis. To answer factoid questions, the robot should retrieve information in a closed data set and then try to answer the question.

The data consists of texts from the exhibition and related material grabbed from slides and web pages. Currently the set includes 21 documents. The data consists of simple text files that contain text and sentences.

3. The chiAQuery System

In this chapter all crucial structures and necessary surroundings of the introduced system, called chiAQuery, are explained. This starts by an system overview, continues with all features, options to start the systems and ends up with all needed and acquired databases to create a question answering system. The detailed overview of the system can be seen in figure A.7.

3.1. System Overview

This section provides a coarse overview of the system and explains all necessary terms. The system was developed to perform question answering tasks on a set of unstructured text documents. As can be seen in figure 3.1 the system parts can be divided in four main parts. These for main parts are also shown in detail in the system overview figure A.7.

Before the processing of any input can be started, the system needs documents for annotation. This can be seen as the step to process unstructured documents to structured documents. The sources can be local documents as well as web documents, which then get another preprocessing workflow to separate the main content from the rest of the web page. The process of text annotation will be explained in detail in chapter 4, Text Annotation. After the system indexed the provided documents to its knowledge the first query can reach the system. For this procedure the process of Input Annotation was established. The workflow is similar to the Text Annotation procedure with the reason to have comparable texts that, in the best case, can match with a part of an annotated document. Chapter 5 describes the procedure with more depth.

The given input and the given knowledge are then processed. Queries are generated out of the input and the relevant parts of the knowledge are extracted. The resulting documents in the knowledge are then used to traverse through them using query plans and find answer candidates. Query Processing will be explained in more detail in chapter 6. Towards the answer candidate creation the candidates are sorted out, filtered and the

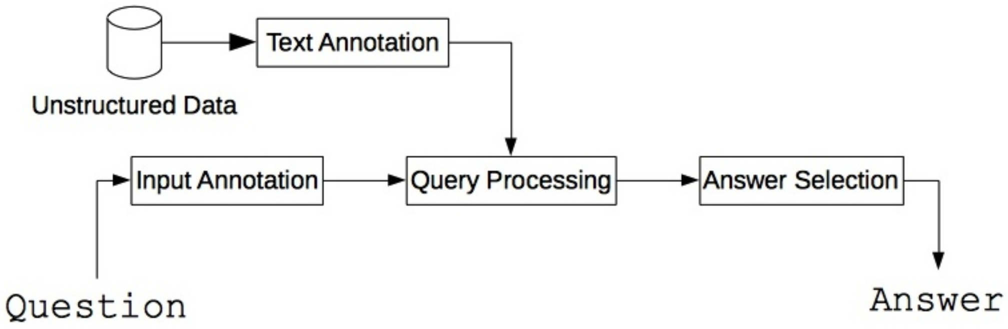


Figure 3.1.: Coarse overview of the Question Answering process

scores are compared to find the most likely answer. The answer is then set and the estimated display options are chosen for a correct answer representation. More insights of the Answer Selection process are shown in chapter 7.

3.1.1. Request & Response

The system was designed to be flexible. The Question Answering component can be seen as one functionality and the system can be easily extended with other features, for instance Text Analysis (find all relevant locations, persons and organizations relating to the input) or Term Exploration (explore the semantic relations of a term). To realize this, a request to response structure was established to:

- Encapsulate all internal objects from the outside of the system
- Easily parse an incoming request to the server and
- Provide a standardized response from the server.

The request and response handling is not only used in client/server mode. It also takes place when the system is started on the command line. The command line interface (CLI) also sends a request to the system and get a returning response.

For the client/server mode the Request as well as the Response object are serialized as JSON objects to ensure an easy handling and a fast rejection if the incoming Request is not well-formed.

A Request object consists of the following attributes that must be set to handle the request successfully:

get This defines the used branch of the system. Currently the only registered branch is the *qa* or *questionanswering* branch.

As can be seen in figure 3.2 the Request and Response objects are always at the beginning and at the end of a process through the system.

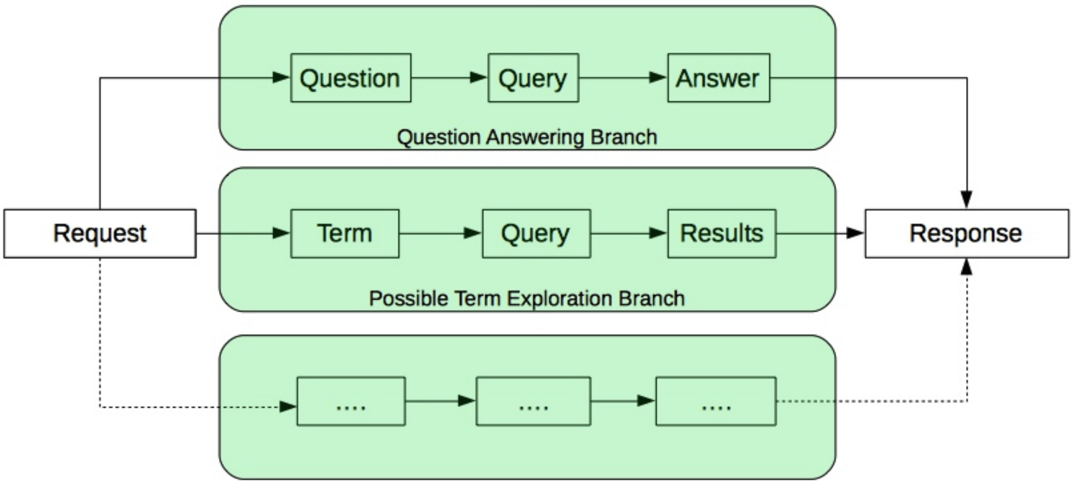


Figure 3.2.: Request to response structure of the system to handle all incoming queries. This figure is focused on the QA branch.

operation The operation points out which operation should be shall be called in the chosen branch. For the *qa* branch the operation *question* predicates that this requests brings in a question. Another operation for the branch of Question Answering can be another input type or a *stats* operation that

query Defines the query to the system. The query depends on the *get* and *operation* parameter.

options The option values can be set to commit special parameters to the system to control the behavior. The *qa* branch uses this options to set the source of Question Answering (*web* or *local*).

A Response object is defined using the following attributes:

processed This attribute points out whether the request was processed or not. If it was not processed a failure in request formulation can be a possible reason. Another reason can be a missing module or branch as well as a operation that was not defined.

message The response message contains the message of the system. For the *qa* branch it should contain the answer, if an answer was found.

```
{
  "query": "Wann kamen in der DDR die ersten Heimcomputer in den Handel?",
  "operation": "question",
  "get": "qa",
  "options": {
    "source": "web"
  }
}
```

Figure 3.3.: A sample request to the QA branch of chiAQuery system. Displayed with JSON format

responseType If the type of response equals the operation (that was defined in the request) the system understands the request and provided an appropriate response.

finalRepresentation The representation of the response predicates how the response must be interpreted. For the *qa* branch, the representations are defined in section 2.1.2.

responseObject Contains the object that the system responded to the request.

```
{
  "processed": true,
  "message": "1984",
  "responseType": "QUESTION",
  "finalRepresentation": "FACT",
  "responseObject": <complete response of QA process>
}
```

Figure 3.4.: A sample response from the QA branch of chiAQuery system. Displayed with JSON format.

Figures 3.3 and 3.4 show a request and the corresponding response of the system. In this example the *qa* branch of the system is requested. The question comes in with the Request object as well as all necessary parameters and the Response object returns the answer. Both objects are shown in their corresponding JSON representations.

3.1.2. Question, Query & Answer

Three classes that are important to understand the systems behavior during the Question Answering task are *Question*, *Query* and *Answer*. Chapters 4, 5, 6 and 7 are dealing with these terms. All three terms are conceptual classes of the system and designed to express the processing steps of the current request. These three terms are explained in detail below:

Question This object holds the incoming question. During the process of Input Processing (explained in chapter 5) all modules that are used to normalize, analyze, detect and tag the question are read and write the given *Question* object. It also contains the answer representation and the estimated answer type.

Query The structure holds the queries that are created to request the Lucene search system (theoretically explained in section 2.3.4 and practically shown in section 6.2) and their scores. The *Query* objects also holds the generated query plans for the question answering process (explained in chapter 6).

Answer An answer is created during the Answer Selection process (explained in chapter 7). It contains the answer string as well as the final estimated representation type. A list of the most likely answer candidates is also given. If an exceptional question was detected, then an attribute called *exceptionalAnswer* points out that the answer was created during an exceptional answer procedure. The approach of exceptional questions is explained in section 3.1.4.

To sum up this section a few summarizing words about these three classes (and concepts) are needed. The *Question* object contains all the information retrieved about the incoming question. It also contains the *Answer* object. The *Query* object contains all information that the system needs to guide from an incoming question to an outgoing answer.

3.1.3. Knowledge

During the QA process the system needs knowledge to search for answers. This knowledge is represented by an object called *ChiaqueryKnowledge*. Figure 3.5 shows the interaction of the module with its environment. *ChiaqueryKnowledge* is used to:

- Annotate unstructured documents
- Index documents into systems knowledge
- Perform web search with given terms and given search engines
- Index web documents on the fly

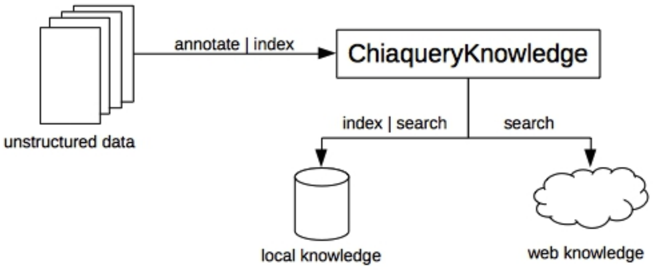


Figure 3.5.: Visualization of the *ChiaqueryKnowledge* module. Unstructured documents are indexed in into the local knowledge. During execution time the module is able to lookup the web for relevant documents.

- Writes to the document cache for web documents
- Handling of Apache Lucene searcher

The knowledge module can be seen as a contact point for the system to reach the indexed data. During the process of QA this module is used to firstly perform the Lucene search that returns interesting documents and to secondly open and read the annotated files that are corresponding to the hits of the Lucene system. This leads to two separate indexes. The first index reads the raw unstructured data. The measures for document ranking are Lucene specific. Generated queries from the system are used to get relevant documents from this first index. Subsequently, the second index comes into play. All retrieved and ranked documents from the first index (Lucene index) are opened and converted to Java objects that the system can use for ranking sentences, terms and furthermore possible answer candidates.

3.1.4. Exceptional Question Handling

The system is flexible and extensible in the way it answers questions. To establish a new routine to handle a certain type of questions the system can be advanced by a new exception question routine. Figure 3.6 points the conceptional attempt of the exception handling. The exception handling has two main parts that must be changed if a new answer routine shall be established. These two were explained below.

ExceptionalQuestionDetector

The detector finds patterns in the question and decides whether it is an exceptional question or not. These patterns are simple text patterns and matcher that are ordered

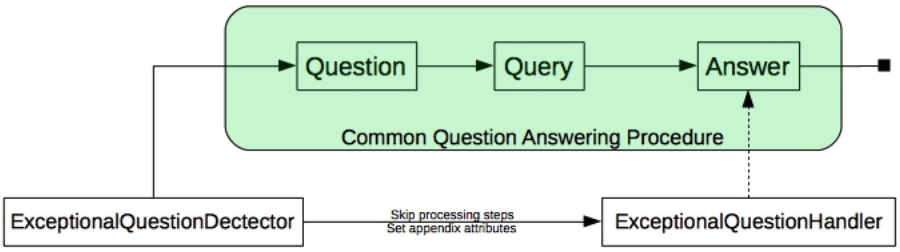

Question
Query
Answer
Common Question Answering Procedure
ExceptionalQuestionDectector
Skip processing steps
Set appendix attributes
ExceptionalQuestionHandler

Figure 3.6.: Visual explanation of the Exceptional Question Handling. Exceptional questions are skipping the common QA procedure and can be answered using the handler.

by priority. If a pattern was found the detector sets the *exceptionalQuestion* attribute of the Question object to a different value than *NO_EXCEPTION*. These values are also defined in the ExceptionalQuestionDetector. Another attribute that the detector uses it the *appendix* attribute of the Question object. It is used to transport important data from the ExceptionalQuestionDetector to the ExceptionalQuestionHandler that is needed to process the handler function. This part of the system also defines which processing parts of the common QA system are not needed and can be skipped.

The detector can be easily extended using a custom routine to find exceptional patterns in the question and a value need to be defined to identify the type of exceptional question in the ExceptionalQuestionHandler.

ExceptionalQuestionHandler

After the detection of an exceptional question and the conditional execution of all parts of the QA system the ExceptionalQuestionHandler will be called. If the attribute *exceptionalQuestion* contains a different value than *NO_EXCEPTION* the handler tries to find the correct function for the exceptional question and processes the values in the *appendix* attribute of the Question object.

After that the result of the handler function will be set as the correct answer for this question request.

Example of Exceptional Questions

The current system contains the first implementation of an exceptional question. These exceptional questions are containing a computational calculation task for the system.

For this task the PHP expression parser[1] of the author was rewritten in Java and used. Therefore the rules for an exceptional treatment are as follows:

- Question starts with ”Was ist” followed by a computable expression
- Question starts with ”Wieviel ist” followed by a computable expression

An example for this can be seen below:

```
Was ist 4*5/2?
```

The ExceptionalQuestionDetector detects this form of question and sets the necessary flag for exceptional questions and it also manipulates the array of variables to skip processing steps. If the common QA pipeline was passed, the ExceptionalQuestionHandler handles all questions that are flagged as exceptional. The handler contains functions that then handle the question in a certain way. For the example, the Ipsum parser parses the incoming expression and returns the result. The resulting answer can also be set in the handler with the certain output that the exceptional type requires.

3.1.5. ChiaqueryHelper

During the development of the system, a lot tasks are recurring each time a feature was established. To have one proven routine for these tasks, the *ChiaqueryHelper* was developed. This class holds all facilities that are often needed, simplify the development and manage simple operations. It was implemented as a static class and can be called system-wide without instantiating as an object.

The helper takes care of the following tasks:

- Read separated files to lists
- Read separated files to hash maps
- Read JSON serialized objects
- Write and print JSON serialized objects
- Produce the MD5 string of a string input
- Print help texts

[1] Project page of the Ipsum parser in PHP http://www.cip-labs.net/projects/ipsum/

- Write to debug or log files (or both, it is called *tell*)

The helper class is often used to read databases and models, files for the feature detector and to serialize the current system state. The use of Google GSON brings a high degree of flexibility to the system.

3.2. Features & Starting Options

In this section all starting options of the system are explained in detail. These options are useful to start the system in the correct mode and to use all of the given options, for instance to train new models or to search the local index.

3.2.1. Question Answering

The command line switch to start in Question Answering mode is *qa*. This switch starts the Question Answering branch of the system with the default database (which is local). If you want to use web sources, the system needs to be started with *qa web*. The explicit switch to use the local sources is *qa local*.
In both cases the *ChiaqueryQA* module is loaded and the *ChiaqueryQACLI* (command line interface) shows a prompt that is awaiting the input as question.

Another command line switch is the *qa benchmark* option. This starts the *ChiaqueryQABenchmark* module and also awaits a file name that provides an annotated benchmark file. This annotation is presented in section 8.1.3.

Below all four options to start the QA branch of the system are shown:

```
java -jar caqy.jar qa
java -jar caqy.jar qa web
java -jar caqy.jar qa local
java -jar caqy.jar qa benchmark <benchmark_file>
```

3.2.2. Client Server Architecture

One requirement of the system (defined in section 1.2) was to create a client-server architecture to execute the system on small clients that have not enough capacities to reading and writing the documents as well as calculating the correct response with regard to the

speed.
The communication between client and server works with just 2 classes. The clients sends a request to the server. The server responds. This simple communication was realized with the use of two classes: Request and Response. These both classes were explained in section 3.1.1. These classes were initialized for a request or response and serialized as string using the GSON library (introduced in section 2.3.6). The receiver of the message uses the GSON library to deserialize the message and to transform it to an object. This simplifies the communication between client and server and breaks the learning effort for the API (Application Programming Interface) down to the understanding of two classes.

The command line switch to start the application as client or server are introduced below. To start the system as the server the declaration of a port is necessary:

```
java -jar caqy.jar server 27025
```

Starting the system as client requires the host (as IP address or host name) and the port where the server is listening:

```
java -jar caqy.jar client localhost 27025
```

The named port must be unused from other programs on the server machine.

3.2.3. Indexing & Search

To use a local knowledge for question answering the system needs unstructured documents to index them. The location of these files can be set in the config file of the system. To start indexing all files the command line switch *index* is used. To search in the local knowledge or to retrieve documents that are relevant to a given term the command line switch *search* is used. After all the work is done or another knowledge base should be indexed the command line switch *clearIndex* can be used.

```
java -jar caqy.jar index
java -jar caqy.jar search "Zuse Z3"
java -jar caqy.jar clearIndex
```

3.2.4. Feature Detectors for Model Training

To have a fast workflow of data input and feature vector output this command line switch was developed. The data samples can easily read and the corresponding feature vectors were printed to use them for creating an ARFF file. After that this file can be used to train models (artificial neural network, support vector machine, random forest and so on).

Currently the chiAQuery system offers two different types of feature detectors for the training. The first is the answer representation (AR) and the second is the answer type detection (AT). The feature detectors of both of them can be called using the command line switch *features* in combination with the feature detector: *v2AT*, *v3AT*, *v4AT* and *v3AR*.

```
java -jar caqy.jar features v2AT <annotated file>
java -jar caqy.jar features v3AT <annotated file>
java -jar cagy.jar features v4AT <annotated file>
java -jar caqy.jar features v4AT_<classes> <annotated file>
java -jar caqy.jar features v3AR <annotated file>
```

The procedure and notation for an annotated file is described in section 8.1.1. For all v4AT variants the previous and initially execution of the Bayes training is necessary. Therefore, the annotated files for the 6 master classes and the sub classes must be provided. The following commands are necessary to train the Bayes-based feature extractor.

```
java -jar caqy.jar trainBayes databases/datd/sets/classes.phrase
java -jar caqy.jar trainBayes databases/datd/sets/abb.phrase ABBREVIATION
java -jar caqy.jar trainBayes databases/datd/sets/des.phrase DESCRIPTION
java -jar caqy.jar trainBayes databases/datd/sets/ent.phrase ENTITY
java -jar caqy.jar trainBayes databases/datd/sets/hum.phrase HUMAN
java -jar caqy.jar trainBayes databases/datd/sets/loc.phrase LOCATION
java -jar caqy.jar trainBayes databases/datd/sets/num.phrase NUMERIC
```

Above the parameter to start the process of feature detector/extractor training can be seen. This process produces JSON-like representations of the Naive Bayes model. Below an example application for the feature detection is illustrated. This process produces a comma seperated value file that contains the feature vectors and the corresponding class of phrase from the phrase file.

```
java -jar caqy.jar features v4AT databases/classes.phrase > models/classes.csv
java -jar caqy.jar features v4AT_ABBREVIATION databases/abb.phrase > models/abb.csv
java -jar caqy.jar features v4AT_DESCRIPTION databases/des.phrase > models/des.csv
java -jar caqy.jar features v4AT_ENTITY databases/ent.phrase > models/ent.csv
java -jar caqy.jar features v4AT_HUMAN databases/hum.phrase > models/hum.csv
java -jar caqy.jar features v4AT_LOCATION databases/loc.phrase > models/loc.csv
java -jar caqy.jar features v4AT_NUMERIC databases/num.phrase > models/num.csv
```

3.2.5. Annotation

To have the text annotation procedure (described in chapter 4) easy testable and usable as a stand alone application the command line switch *annotate* was developed. With this option, an unstructured document can be transformed into the same JSON representation that the system uses to do the Question Answering task.

```
java -jar caqy.jar annotate <text file>
```

3.2.6. Help

The *help* command line switch displays the help menu of the system. If the system did not start correctly or another command line switch shall be looked up, this switch can be used.

```
java -jar caqy.jar help
```

3.3. Created Databases & Models

The task of Question Answering requires a lot databases and models that extract information from text and transform text to syntactic and semantic interpretations. All of these databases and models are listed in this section and all basics are explained to get an understanding about their functionality and use for the system created for this thesis.

3.3.1. SemMap

SemMap was created for this thesis. The origin of SemMap is the SALSA project that was explained in section 2.4.2. During the development of the thesis some problems of the SALSA network were identified:

- Duplicated frame names
- Sparse members in frames and
- The graph format was to slow for the requirements of the system

After noticing these problems the idea to realize an own semantic network was born. For this the semantic frames and their members were used to create a JSON-like representation of the data. A semantic frame groups words that have a similar semantic meaning

Attribute	#
Frames	265
Words	1893
Words per Frame before Enrichment	5.24
Words per Frame after Enrichment	7.14

Figure 3.7.: Statistics of the current SemMap version implemented in the system

or intention under a corresponding semantic label. In contrast to SALSA, some frame names were reduced to avoid duplicates. To enrich the words of the SALSA data set, all synonyms found in GermaNet for each word were added. The enrichment with GermaNet synonyms and the cleaned SALSA structure result in the currently most complete semantic frame database for the german language. One of the further development goals of SemMap is the training of frame correlations between each frames. Figure 3.7 shows the number of words and frames as well as the word to frame ratios. With SemMap the application is able to find semantic relations in the text to get a deeper understanding of the input text.

3.3.2. SynMap

The creation of SynMap was caused by the complexity of GermaNet and its loading time. GermaNet was just loaded once and during the execution held in a hash map which provided more performance in speed than the GermaNet implementatioin with jGraph[2]. The loading times are compared in figure 3.8. Beside this the words are split into three different files. These files are divided into adjectives, verbs and nouns. The numbers of words for all of these three files are illustrated in figure 3.9. The quantity of data from GermaNet was seamlessly assigned to SynMap. It is important to mention that just the synonym (synset) relation of the GermaNet relation was used. 'Syn' in SynMap stands for synonym and therefore it contains just synonyms of words. The benefit of this structure is the division in these three parts and the easy handling using a JSON parser.

Database	**Loading Time**
GermaNet	40.0 sec
SynMap	0.3 sec

Figure 3.8.: Comparison between SynMap and GermaNet speed

[2] A Java implementation for manipulating and querying through a graph structure.

Database	# Elements
adjectives.json	4634
nouns.json	31603
verbs.json	3899

Figure 3.9.: Content of SynMap files

3.3.3. Verb Database

This database is used for the *VerbNormalizer* which tries to normalize the verb and get to know the tense (temporal form) of the verb. Therefore a bunch of often used verbs are annotated. The annotation is shown below:

```
...
waren;sein;PAST
sind;sein;PRESENT
werden;sein;FUTURE
...
```

The verb is followed by the normalized form and the tense of the verb. With this annotation the information of the verb changed from its activity to its activity plus the time of this activity. This is very helpful if the question asks for a date in the past, present or future. For this data set the often used verbs are annotated. At the moment the database consists of 669 annotated verb forms.

3.3.4. Pronoun Database

Having objects (or subjects) in a question can cause the use of a pronoun. These pronouns betray the secret whether the object is male, female or neuter. The pronoun database contains the often used pronouns with the corresponding gender. The annotation is shown below:

```
...
ihre,FEMALE
seine,MALE
es,OBJECT
...
```

At the moment of writing this database contains 10 entries. The mechanism behind this database is that if one of these words occur in the text, the named entities that appeared previously in the input are labeled with a possible gender flag.

3.3.5. Acronym Database

Acronyms are very hard to understand. Using a single token like the dollar sign has a semantic meaning which must be decoded to work with it. Beside single tokens an acronym can also be a few letters that stand for a longer word. This database covers a few of them and contains, at the moment of writing, just 5 entries. The problem is that the number of abbreviations and acronyms are too large and a lot of them are ambiguous with different meanings in different fields. For this, the system has this interface to feed the database with acronym knowledge about a specific topic.

```
...
usw.,und so weiter
NLP,Neurolinguistische Programmierung
...
```

The annotation can be seen above. The acronym is followed by its longer version. If a word from the database was found in this database, the long version is set to the vector that contains alternative forms of this word which helps during search and query execution.

3.3.6. Filter Database

To filter wrongly annotated answers from the set of answer candidates the filter database contains words, that are very unlikely to be a possible candidate. The list contains 223 elements and can be easily extended. This database has no special annotation. The words are in form of a list. If one of these words occur in an answer candidate, the candidate will be dropped.

3.4. Directory Structure

databases This folder contains all databases, lists, sources, experimental documents and data that exists in plan text format and was used for test, evaluation or during the execution of the system.

docs Documentations, task lists, change logs, example output and the special explanations are grouped under this folder.

documents Here, all text documents are located that serve as a knowledge source to the system. The standard path to index documents is built into the system.

knowledge This directory holds the index from the data from the documents/ directory. It contains the Lucene index and the annotated documents index.

logs The query log and the system log can be accessed here.

models POS models, NER models and all other statistical or trained models can be found here. It also contains the results and classifier output from training, evaluation and testing.

scripts The system needs some scripts for text format transformation and a few more. These scripts can be seen here.

4. Text Annotation

Here and in the following sections of this chapter the steps to annotate a plain text document are described. If a unstructured document (that means plain text) reaches the system, a few steps to structure the document are necessary. All of these steps will be described in this chapter. At first the document needs to be preprocessed. Preprocessing means that the text will be cut into sentences. Furthermore the preprocessor normalizes letters and token. This part will be described in section 4.1. The Part-of-Speech tagging is necessary to estimate the most likely grammar parts of the sentences which are later used to find targets like verbs, adjectives or nouns. The tagger will be described in section 4.2. Beside the tagging of the grammar, named entities are another important part of the sentence that needs to be estimated. Named entities contain locations, persons and organizations. The important task to find these entities in the text will be introduced in section 4.3. As already mentioned, normalization is crucial to make text comparable. Normalizers that deal with verbs (find the correct tense), symbols (find the long form of a symbol), acronyms (find the long form of an acronym), pronouns (estimate the gender and replace pronouns) and numbers (find written-out numbers and transform them) are described in section 4.4. Finding synonyms of words is important to extend a query with similar words that might help to find the right answer. SynMap was implemented for this task and the corresponding processor will be described in section 4.5. A text contains a lot of semantics. To analyze the text and to find these semantic frames, the SemMap processor tries to find frames. This part of the system will be described in section 4.6. Beside the semantic frames a text can also contain numbers and digits that have semantic meaning. These parts are extracted wit the mechanism described in section 4.7. After all of these mechanisms, the document needs fast index structures to easily request a large document. More detail on these structures is provided in section 4.8. To put this annotation step in the context to the whole system consult figure A.7 for more information.

4.1. Document Preprocessor

The processing of an unstructured document needs a reliable preprocessing. The document must be divided into sentences or syntactical elements that can be used for further operations on the text. At this a lot of problems can occure, for instance if the document is not well-formed or consists just of word groups without a grammatical separator. The separation of sentences will be covered by a module called *ChiaqueryParser*. After the parser divided the document into sentences the handling of inconsistencies can be done using the *DocumentPreprocessor*.
The preprocessor covers the following tasks:

- Normalize umlauts like ä, ö and ü
- Replace brackets, symbols and other unusable tokens
- Classify input into REQUEST, STATEMENT or QUESTION
- Create a stemmed version of the input using Lucene GermanStemmer
- Create a *processed* sentences that is normalized and consists of lowercase letters

Input	Konrad	Zuse	ist	der	Erfinder	des	Computers.
Output	konrad	zus	ist	der	erfi	des	compu.

Figure 4.1.: An input sentence after DocumentPreprocessor

Figure 4.1 shows an input sentence before and after the run through the preprocessor. The focus of the illustration is on the stemming which is very important to find indexed or similar words during the search process later.

4.2. Part-of-Speech Tagger

Part-of-Speech is a very important part of a Natural Language Processing pipeline. It represents the grammatical view on the data and likewise the first step to understand the text semantically. The output of such a tagger is a set of tags, each tag for each word. The tags provide the information of the word type, for instance verb, noun, adjective and a few more. The set of tags is illustrated in figure A.9. The grammatical representation of the input is provided by the Stanford NLP Core POS-Tagger (explained in section 2.3.3). The tagger annotates the incoming sentences of the documents and add the tags to the sentence structure as well as to the word representation itself. The tagger's output can be seen in figure 4.2. The tagger implementation used for this system has problems if a sentence is not well-formed. Therefore the document needs an accurate pre-processing procedure so that the tagger can work properly.

Input	Konrad	Zuse	ist	der	Erfinder	des	Computers.
Output	NE	NE	VAFIN	ART	NN	ART	NN

Figure 4.2.: An input sentence and its corresponding POS tags according to the tag set illustrated in figure A.9

4.3. Named Entity Recognition

Finding named entities in the text is also covered by the Stanford NLP Core NE-Recognizer which was already introduced in section 2.3.3. NE recognition tries to find locations, persons, organizations and other entities in the text. This is important to find answers that relate to such a type of entity. If the question asks for a person it is very likely that the answer was found during the annotation process and marked as person. The same procedure also works for other types of entities. The POS tagger tries also to find Named Entities but it doesn't segment into various types of entities. Figure 4.3 provides an example of a tagged input sentences. The person was correctly identified as a person. *Berlin* as city and capital of germany was also correctly identified as a location.

Input	Konrad	Zuse	wurde	in	Berlin	geboren.
Output	I-PER	I-PER	O	O	I-LOC	O

Figure 4.3.: An input sentence and its corresponding NE tags according to the set of possible NE tags illustrated in figure 2.4

4.4. Text Normalization

One of the important procedures for a successful question answering system is the normalization of the text. Verbs, nouns, numbers and a few more parts of the text can be expressed in several ways. The system needs a normalization module that creates an uniform representation of the text to make it comparable. This section deals with the normalization methods used for this system.

4.4.1. Verb Normalizer

Verbs in sentences or questions can appear in different modes. Firstly, they can occur with a tense and secondly, the formulation of the verb. The conjugation is subject to future work. To cover the first aspect the verb normalizer was developed. This normalizer loads a database (at system startup) that consists of verbs with their three tenses:

- PRESENT
- PAST
- FUTURE

If the normalizer finds the verb in the database, it can set the tense of the question and also set the present tense of the verb. This information is very important. If the answer type detector (described in section 5.5) detects that the question is asking for a date, this information helps to specify the statement to: *The question is asking for a date in the past.* This normalizer uses the database that was introduced in section 3.3.3. The second point regarding the formulation of the verb will explained in section 4.5. If a verb can't be found in the database maybe a synonym of the verb can be found with the another mechanism and then the correct tense can be determined.

Input	Konrad	Zuse	wurde	in	Berlin	geboren.
Output	O	O	ist/PAST	O	O	gebären/PAST

Figure 4.4.: An input sentence and its corresponding verb normalizer output

4.4.2. Symbol Normalizer

Some text documents or texts using symbols or special characters to point out a certain circumstance. These symbols are readable but difficult to understand for a computer system. The main problem of using symbols it that they often hide semantic relations. Consider the symbol $ or the symbol @. The first means dollar but the semantic relation is that it also means a semantic relation, namely the number before the symbol in the currency named 'dollar'. The second symbol could point out, that it consists in a mail address but it could also be part of a statement 'AC/DC @ Dresden' which means that the band AC/DC is playing a concert at/in Dresden. To tackle this problem of symbols, the symbol normalizer was established. The normalizer covers all simple symbols and

Input	Die	Rechenmaschine	kostete	1000$.
Output	O	O	O	Dollar

Figure 4.5.: An input sentence and its corresponding symbol normalizer output

can be extended easily. All symbols that were found and normalized are added to the *alternativeForms* of the word in the sentence. A simple example is illustrated in figure 4.5.

4.4.3. Acronym Normalizer

A problem similar to symbols is the use of acronyms. Acronyms cover a lot of semantics that a computer system need to understand when it tries to compare a text against another text. Two often used significant examples are 'zB' for 'zum Beispiel' or the 'FC' for 'Fussballclub'. The second example shows the obvious semantic enrichment of the acronym resolution. If a computer looks at 'FC' it can not catch the semantics behind it. But if it looks at 'Fussballverein' it can find a lot of synonyms like 'Ballsport', 'Organisation' and 'Verein' which are all very pivotal to find the right answer. Of course, the ability to find such synonyms is related to the used semantic network. In this work SynMap and SemMap are used to cover this task. The most challenging problem of this normalizer is the resolution of ambiguous acronyms. The example in figure 4.6 shows this ambiguity problem. In this case the normalizer adds all found long versions of the acronym to the dataset. In most cases the ambiguities are solvable. If the question consists of more information than just the acronym (some topic related information) the coarse search with all possible long versions will automatically find more relevant documents.

Input	Für	NLP	wird	die	Methode	oft	benutzt.
Output	O	Neurolinguistische Programmierung Natural Language Processing	O	O	O	O	O

Figure 4.6.: An input sentence and its corresponding acronym normalizer output

The acronym database, explained in section 3.3.5, can be easily extended. The database exists in plain text format and the insertion format is explained in the documentation. All found and normalized acronyms are added to the *alternativeForms* structure of the processed word and used during the process of query creation.

```
Wer war Marilyn Monroe und wann komponierte sie ihre tollen Lieder?

  "m_genders": {
    "Marilyn Monroe": "FEMALE"
  },
```

Figure 4.7.: Procedure of pronoun normalizer with an example

4.4.4. Pronoun Normalizer

Often, one question asks for more than one fact or the question asks for two semantically different things. Motivated by the laziness of the user, the use of pronoun is very common for this type of questions. A pronoun sets the reference to a named entity or noun that was written before. The interesting thing of this behavior called 'laziness' is that the user reveals and information that hidden from the system else.

Any given pronoun contains the gender of the named entity that were used previously in the same question. The system has to put together a pronoun (for example: 'er') with the named entity used in this question (for example: 'Konrad Zuse'). The previously found named entity will be replaced with the pronoun and the gender will be added to the *genders* structure of the sentence. Figure 4.7 introduces the output of the normalizer using an example question.

4.4.5. Number Normalizer

A large number of the questions asked to a Question Answering system are asking for numbers. The problem is that these numbers often not consist in a form that the system can process them. A number or digit as alphabetic value is the best example. If the digit hides behind 'zwei' or 'neunzehnhundertachtunddreißig' the system needs a mechanism that transfers these strings into processable digits. This was the motivation to develop a number normalizer.

Input	Die	Rechenmaschine	kostete	tausend	Dollar.
Output	O	O	O	1000	O

Figure 4.8.: An input sentence and its corresponding symbol normalizer output

The normalizer works as a left to right look ahead parser that tries to find symbols (for instance 'zwei', 'drei', etc.), fill words (e.g. 'und') and factors for multiplication (e.g.

'tausend', 'hundert', etc.). A short example of the output is provided in figure 4.8.

4.5. SynMap Processor

Synonyms are very important for the search of verbs, nouns and adjectives. All of these words have words with a similar meaning that in most cases are more likely to occur in the target text. These synonyms are used during coarse document search (section 6.2), web search (section 6.3) and for the creation of the query plans (section 6.1.2). This is done by going though each word and trying to find a possible synonym for it. The database that is used was introduced in section 3.3.2. An example can be seen in figure 4.9. Here synonyms were found for two words in the sentence. All the found synonyms are stored in the the annotated document to speed up the execution time and prevent the system from searching for synonyms again.

Input	Der	Computer	arbeitet	mit	Software.
Output	O	Rechner	O	O	Programm-Code
	O	Rechenanlage	O	O	Softwareprogramm

Figure 4.9.: An input sentence and its corresponding SynMap output

4.6. SemMap Processor

The SemMap Processor solves the task of semantic annotation. The database of SemMap was described in section 3.3.1. The processor goes through the text and looks up the semantic frame of each word. This can be very helpful because semantic similarities often point out facts that are likely to reach a specific topic. Semantic annotation is a crucial part of understanding a text and especially a sentence or question. During the question answering process the semantic frames are a criteria of comparison and help to find the right answer candidates. The found semantic frames are added to the *frames* structures of the word itself and on sentence level. Figure 4.10 illustrates the found semantic frames in two example sentences. A possible next step could be to combine semantic frames and find similarities between semantic frames to find correlations between sentences and questions.

Input	Der	Computer	heißt	Z3	und	Zuse	war	von	ihm	überzeugt.
Output	O	O	Being_named	O	O	O	O	O	O	Certainty

Input	Er	bemühte	sich,	beanspruchte	Patente	&	schätzte	die	Chancen	hoch	ein.
Output	O	Attempt	O	Claim_ownership	O	O	Likelihood	O	O	Assessing	O
	O	Using	O	O	O	O	Judgement	O	O	Judgement	O
	O	Self_motion	O	O	O	O	O	O	O	Rewards	O
	O	Statement	O	O	O	O	O	O	O	Awareness	O

Figure 4.10.: Input sentences and their corresponding SemMap output

4.7. Semantic Extractors

Beside named entities (for instance locations, persons and organizations) which are also semantic extractions there are a plenty of more possible extractions in a text. This section deals with numeric extractions and how they are treated from the system to find answers.

All explained extractors refer to a class of the numeric class of the answer type taxonomy that the system uses. This taxonomy was already explained in 2.1.1 and the full taxonomy is illustrated in A.8. The idea to convey the taxonomy of the numeric class straight to the extraction types is very feasible. When the answer type detector estimates a member of the numeric class (a subclass) to be the most likely answer type, the system can first try to apply an extracted numeric value (including its unit) from the most relevant document and the most relevant sentence. This minimizes the computational effort of the answer finding process because extracted text can be indexed and the mapping from answer type to extractor happened in a straight forward manner. Certainly, not all variations of semantic formulations can be covered. Some formulations are uncommon and therefore not considered during the conception of an extractor. For instance a new unit for representing a currency value. If the system does not know this currency (and its symbols that represent a currency value) the system can't detect this value and label it as a member to the money class. Subsequently all current semantic extractors are introduced in short:

Count Extracts the amount of something using keywords like 'Anzahl', 'Stück' and a few more.

Date Date formats are extracted. Supported formats are using dots and hyphen.

Distance Routes between way points are extracted here. Common units are used for this task like 'km', 'm' and others.

Money Currencies are extracted by the money extractor using known symbols and labels.

Order Rankings and hierarchies are extracted here. Keywords are 'Platz', 'Platzierung' and many more.

Percent Percentage values also extracted. This extractor finds shares, rates, dues and percents.

Period Time during two events is extracted with the period extractor. The extractor expects a time.

Temperature Values that expose temperature values are extracted with the most common units.

Time The corresponding mapping is equal to the date extractor. The extractor is specialized on time statements.

Volume & Size The extractor is specialized in finding area sizes and capacity of bodies.

Weight Statements that cover weight values and severity are found by this extractor together with the most common units.

4.8. Index Structures for Annotated Documents

All the mechanisms that were previously described in this chapter are important to explore the data and to enhance the possibility to find an answer. Part-of-speech tagging improves the possibility to find the right target, NE recognition enhances the possibility to find persons and many more. But more data requires more computational power. Therefore smart index structures must be found to ensure the performance during answer finding. Therefore, every document owns index structures covering all necessary data that was created during document preprocessing.
All types of structures are explained in this section.

- Locations, not stemmed
- Organizations, not stemmed
- Persons, not stemmed

- Entities, not stemmed
- Entities, stemmed
- Verbs, stemmed
- Adjectives, stemmed
- Nouns, stemmed
- Semantic frames

All the parts that are mentioned above exist in the head of the document with an index. The structure of the index contains of the stemmed or not stemmed word that is realized as a hash and points to a list of sentence numbers. The sentence numbers out indicate the hash key exists in the corresponding sentences of the document.

$$'berlin' \longrightarrow \{0, 4, 7, 11\} \tag{4.1}$$

As equation 4.1 illustrates that entity 'Berlin' occurs in four sentences with the sentence IDs 0, 4, 7, 11. This index structure has a huge advantage for finding relevant sentence. The algorithm that finds relevant sentences can operate on a lot of sets and can find intersections of sets that include the same sentence ID. With this behavior, finding relevant sentence in a large document is fast and more scaleable then seeking through the whole text.

The introduced indexes can be found on top of the document and they are also covering the whole document. Other index structures are settled on sentence level and help to easily find possible targets (which are often named entities or part-of-speech tags). These indexes are:

- Semantic frames that the sentence contains
- Part-of-Speech tags as an array to rapidly find possible targets
- Named entities as array to also find possible targets promptly
- Words as an array to extract corresponding candidates
- Containing nouns as array
- Containing adjectives as array

- Containing verbs as array

The sequence of using all of these indexes during the answer finding process can be considered as follows:

1. Find relevant sentences using the indexes on the document level

2. Using intersections of the indexes to rate relevant sentences

3. Find targets in relevant sentences using the indexes on sentences level

4. Extract answer candidates using indexes on sentence level

These indexes are used and explained in more detail in the following chapters. It is important to understand that these indexes are part of every annotated document, so that each document can be processed independently. Other indexing concepts try to centralize specific index structures to access them very quickly (and also to avoid overhead). For this work no centralized concept was chosen, because every document should stand alone. An annotated document should be parse from another system, if needed and therefore provide all needed data. This approach uses more disk space but at the moment of processing just one file needs to be opened up, whereas the centralized approach opens a few index files and looks up the indexes in these files.

5. Input Processing

This chapter explains the pipeline from an input question to a query that the proposed system is able to use for the answer finding process. Parts of this pipeline were already described in chapter 4. In this chapter this pipeline will be explained for especially questions. To see the context of the input processing part consult figure A.7 which provides an overview of the pipeline. This first step of the pipeline is the preprocessing of the question. An incoming question needs to be normalized and stemmed to be comparable with the documents indexed in chapter 5. This preprocessor will be explained in section 5.1. After the preprocessing two important models come into account. These models first estimate the grammar of the question and try find possible named entities (like persons, organizations and locations) in the question. This is important to disassemble the question into nouns, verbs and adjectives as well as to lookup found entities. These two parts are described in sections 5.2 and 5.3. After that the next annotation step is more complex. It contains the search for synonyms and semantics in the question. Synonyms are later used to enhance the question and reformulate it. Semantics are important to understand the question in a deeper way. Therefore semantic frames are extracted. This annotation step is described in more detail in section 5.4. Afterwards, the answer type needs to be estimated. The answer type describes the required type (this could be a location, a weight or a company) of the question and is important to find the correct answer in the documents. The answer type detector will be explained in section 5.5. Another important part is the classifaction of the correct representation type. The question could require a fact, a list of facts or a paragraph (description). This needs to be estimated before the system starts to query the knowledge base. Answer representation will be described in more detail in section 5.6.

5.1. Question Preprocessor

The preprocessing module takes care of the questions equals the *DocumentPreprocessor* that was introduced in section 4.1. Questions are a specialized form of a sentence and

Input	Wo	wurde	die	Z4	übergeben?
Stemmed	wo	wurd	die	z4	ubergeb
Processed	wo	wurde	die	z4	uebergeben

Figure 5.1.: Question after preprocessing step

therefore questions need a special treatment to be preprocessed correctly. Beside other things, the preprocessor mainly takes care of:

- Normalize umlauts (ä, ö and ü)
- Replace brackets, symbols and other unusable tokens
- Create a stemmed version of the input using Lucene GermanStemmer
- Create a *processed* question that is normalized and consists of lowercase letters

After the application of this module, the question can be seen as normalized.

As figure 5.1 shows the question was stemmed and the umlauts are removed or replaced with corresponding substitutes.

5.2. POS Tagger

With the POS (Part-of-Speech) tagger the question is analyzed and the most likely tags are estimated. The model used for this task comes from Stanford NLP Core (motivated in section 2.3.3).This step is important to get to know the grammatical structure of the question. Question structures are often similar. Most factoid questions ask for an action (verb) in connection with a subject and an object (nouns or entities). The part that describes the action executed on a subject is represented by the adjectives. Other characteristics of a question are the succession of appearance of POS tags and the occurrence of a specific tag in general. In figure 5.2 two examples are presented.

Input	Wo	wurde	die	Z4	übergeben?
Output	PWAV	VAFIN	ART	NN	VVINF

Input	Welche	Fahrzeuge	wurden	in	Zuffenhausen	hergestellt?
Output	PWAT	NN	VAFIN	APPR	NE	VVPP

Figure 5.2.: Questions after POS tagging using the tag set depicted in figure A.9

The structure of tags is hierarchical. If a tag begins with a ’V’ it expresses a verb

’VINF’ means full and infinite. To know this is very important for the handling of these tags. Because the used tagger operates on a statistical model. In some cases the tagger fails to estimate the type of the verb. To provide a short example: The verb ’V’ was correctly estimated, but the type ’AFIN’ (auxiliary, finite) was not correctly estimated and instead the system estimated ’VPP’ (full, participle perfect). To ensure fault tolerance, the system presented here, works, in most cases, just with the simple estimation of the POS type (’V’ for verb, ’N’ for nouns and named entities and ’A’ for adjectives) and other tags that have not such a deep structure (for instance ’CARD’ for cardinal number or digit in general). The single exception is the answer type detector, where the statistical feature extractor works with the detailed form of the tags. In that case, a wrong estimation is not a big deal, because the feature extractor learns on a huge data set and appears to be robust against single incorrect estimations. The full tag set is presented in figure A.9.

5.3. Named Entity Recognition

Finding named entities can be crucial for the processing of a question. Therefore NE-recognition is applied to the incoming question using the Named Entity Recognizer of the Stanford NLP Core (introduced in section 2.3.3). Basically an entity, in nearly all cases, is a noun. If the system detects a named entity, then this piece of information can be set on top of the basic information that this word (or term) is a noun. The second example in figure 5.3 points out how important it is to understand the semantic meaning of a noun. If this question is considered with just the POS tags (explained in section 5.2) it is hard to find out, what the question wants to know. With using the NE representation it becomes much clearer. The relation between ’Porsche AG’ as an organization and ’Zuffenhausen’ as a location appears.

Input	Wer	war	Konrad	Zuse?
Output	O	O	I-PER	I-PER

Input	Welche	Fahrzeuge	stellte	die	Porsche	AG	in	Zuffenhausen	her?
Output	O	O	O	O	I-ORG	I-ORG	O	I-LOC	O

Figure 5.3.: Questions after NE recognition using the tags depicted in figure 2.4

The system proposed in this thesis takes advantage of the found named entities during query generation and scoring of the answer candidates. Furthermore the appearance of a named entity is also part of the statistical feature detector, which will be explained in upcoming sections of this chapter.

5.4. Text Annotation for Questions

The annotation of text was already described in chapter 4. In this section the text annotation optimized for question will be explained. The mechanisms are quite similar to the text annotations used for documents. To have comparable structures this is very important. This section will explain all normalizers, analyzers, processors and extractors with question examples.

5.4.1. Question Normalization

Normalized text reduces complexity and helps to simplify the rule set that processes it. To achieve this normalization the question runs through a bunch of normalizers that transforms the incoming text. Below all of the systems normalizers are listed with a short description of their behaviors.

Verb Normalizer Verbs often occur not in the present tense and in a conjugated form. To normalize these verbs and to enhance the information of the verb to verb plus tense of verb this normalizer was built. The application of this normalizer is important to search in text data that have also normalized verbs. At times, the given question formulates a verb in the past and in contrast the given information base contains this verb in the present tense. After normalizing the position with the verb in present tense can also be found.

Acronym Normalizer Acronyms are shortened terms that, in most cases, include a lot of information. The task to extend the acronym to the correct term is very important to get more information about the question. It is important to know that one acronym can have different meanings. At the current point, the proposed system isn't able to distinguish correctly between the right and the wrong meaning. Anyhow the extension to explore a long version of an acronym in queries and search process is important.

Symbol Normalizer Like acronyms, symbols display the same problem for information retrieval. They are shorter than their long forms and therefore they are generally accepted for expression in the text. A symbol always has an implicit semantic -

for instance '$' expresses a currency (which again is the semantic relation of the number that appeared before the dollar sign). To uncover this semantic the normalizer replaces the sign with the right expression and adds terms to the alternative formulations of this token.

Pronoun Normalizer In some cases the question is more complex. In these cases the author of the question uses a pronoun ('er', 'sie', 'es', 'ihre' and more) to make a reference to a previously introduced entity (person or thing). This allows two conclusions. The first conclusion is that the current position of the text (the pronoun) can be replaced with the entity. The second is that it allows an inference of the gender. These two conclusions are added to the question after processing this normalizer.

Number Normalizer Numbers are an important part of a question. Often, year dates or a quantity of something is declared. The author of the question can use the numeric expression of the number or digit (e.g. '13') and the author can also choose the textual expression (e.g. 'dreizehn'). The numeric expression is not the problem here. This normalizer is specialized to find the numeric value to a given textual expression.

Input	Was	passierte	neunzehnhundertdreiundachtzig	in	der	DDR ?
Number Norm	O	O	1983	O	O	O
Acronym Norm	O	O	O	O	O	Deutsche Demokratische Republik

Input	Wer	war	Konrad	Zuse	&	wann	starb	er?
Pronoun Norm	O	O	O	O	O	O	O	Konrad Zuse
Verb Norm	O	sein/PAST	O	O	O	O	sterben/PAST	O

Input	Welches	System	kostet	1000$?
Symbol Norm	O	O	O	Dollar

Figure 5.4.: Questions after text normalization

Figure 5.4 shows three examples with their text normalization output. For each question all normalizers are applied. The figure just shows the relevant applications. After this normalizations further processing steps can be applied.

5.4.2. SemMap Processor

As aforementioned semantics are very important to understand the meaning and reason of a question. The SemMap processor applies its database to each word from the left to right. If a word was found in the database the corresponding frame is added to this word. A semantic frame can be understood as an unit of meaning that share different words together. Therefore semantic frames are an important feature to find similar units of meaning in the question as well as in the data source.
The semantic frame can help to answer the question, why the question was asked. Maybe the author of the question has the intention to buy a product or the author has other intention with the organization that he or she has mentioned in the question.

Input	Wie	heißt	der	Computer	von	Zuse?
Output	O	Being_named	O	O	O	O

Figure 5.5.: Questions after SemMap processing

The current data set with frames has not reached a size of a productive deployment. Section 3.3.1 illustrates the size of the database and the prospects for the future. Figure 5.5 shows the problem. For a simple question just one semantic frame was found. At the first view this seems poor, but for this simple question and for question processing, in most cases, one frame suffices for a match.

5.4.3. SynMap Processor

Synonyms are important to extend a question and to re-formulate it when the results for the initial question are bad. This part is provided by the SynMap database that was formerly introduced in section 3.3.2. Ordinary, the author of a question is not the author of the documents in the knowledge base. Different people use different formulations and this is one aspect that makes the task of understanding the question and matching answer candidates so hard. To tackle this problem, each word in the question is looked up in the database and if a synonym was found then it will be added to the word and considered during the processes of query generation and finding answer candidates. The example

Input	Welcher	Computer	arbeitet	mit	Software?
Output	O	Rechner	O	O	Programm-Code
	O	Rechenanlage	O	O	Softwareprogramm

Figure 5.6.: Questions after SynMap processing

in figure 5.6 shows synonyms for the two nouns in the incoming question. Beside nouns, the database also covers verbs and adjectives. With an improvement of the SynMap database could come more possibilities for paraphrases of the original question which leads to a larger set of results and information redundancies which are necessary to find the right answer.

5.4.4. Semantic Extractors

Extractors are used to find numerical values and their semantics in questions. If the question contains '1000 kg' then the semantic behind this term is the number '1000' as a weight value with the corresponding unit 'kg' which stands for kilograms. The problem that the semantic extraction tries to solve, is also tackled (for a few cases) with the acronym and symbol normalizer. These normalizers replace the unit in the text, but the extractors are doing one more step. They extract value and unit into a computable structure. To find such semantic-related numbers in the questions extractors exist that find the following things in the question: counts, dates, distances, money/currencies, orders, percentages, periods, temperatures, times, volumes, sizes and weights. They were formerly explained in section 4.7. The current system extracts these values but the bigger usage of them is, no doubt, on the side of document annotation.

For the sake of completeness the extractors are mentioned here but the current system just tries to find semantic extractions in the knowledge base to register them as answer candidates if the question are asking for such a type (weight, currency and others).

5.5. Answer Type Detection

The detection of the correct answer type is probably the most important action during a Question Answering process. If the system estimates the wrong answer type, the behavior of the system changes and the likelihood to find the correct answer decreases rapidly. This section describes the process of answer type detection and introduces the model that was used to build the classifier.

The taxonomy used for the classifier was established by Li and Roth in 2002 [26] and later extended in 2006 also by Li and Roth [27]. This taxonomy is illustrated in figure A.8. Because of the high quantity of classes the classifier is realized in a hierarchical manner. The first layer estimates the top-classes and the second layer estimates the sub-classes also named detailed answer type. For the top-classes just one model needs to be trained, but each top-class also has a model for their sub-classes.

The anatomy of the classifier is illustrated in figure 5.7. All components are described

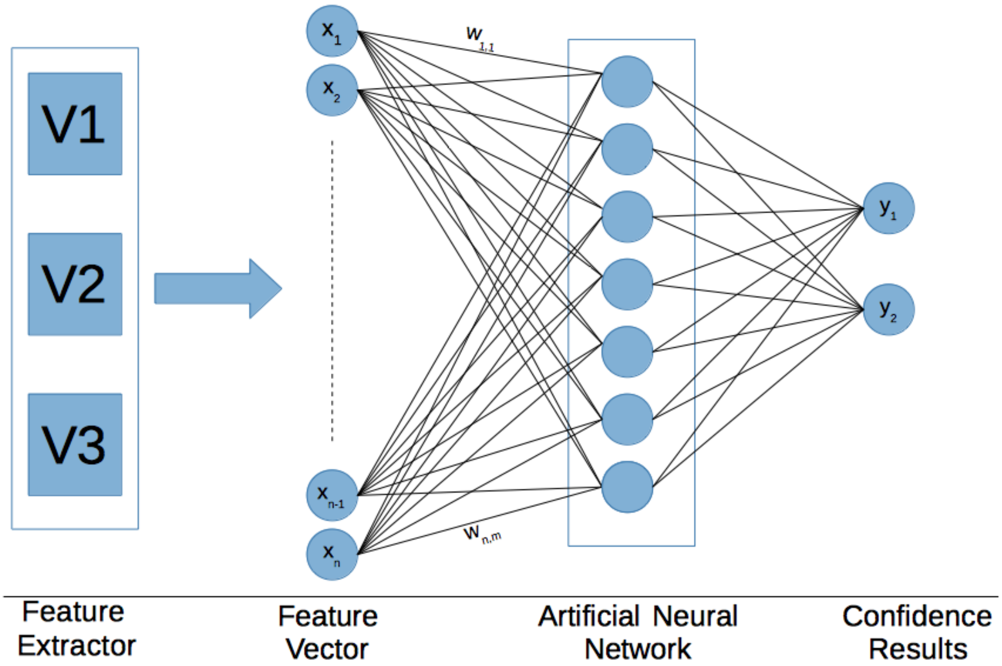


Figure 5.7.: The conceptual view of the answer type detector. The statistical feature extractor delivers the feature vectors for the multilayer perceptron which learns the coherencies of the features.

below:

Statistical Feature Extractor The feature extractor works on three levels of perception and uses the Naive Bayes for its decisions. It is important to know that the feature extractor must be previously trained using a labeled data set. The first perception level is called word level. N-gram models are trained that use the consecutive words as data. The second level takes care of the grammatical part of the question. The n-gram models use the consecutive POS tags as feature. The third level works with a combination of SynMap words, SemMap frames, tense of question and occurring named entity types as features to calculate the conditional probability for each class. The extractor creates a feature vector where each value of the vector is a number between 0 and 1 and reveals a specific confidence. Each class to estimate has a data dimension in each perception level. The detector needs data for training.

Artificial Neural Network For this task a Multilayer Perceptron with one hidden layer was used. The feature vector serves as input for the neural network that was previously learned using another data set than the feature extractor. This is very important to prevent the neural network from over-fitting. The network learns the mapping from the input vector to the classes. Since the features from all perception levels are put together in the net, the dependencies between each of the perception levels is learned by the neural network.

Confidence results After processing the neural network the model returns confidence values for each class. These values can be seen as a likelihood that a specific data point belongs to a specific class. These confidence values are later the foundation of decision and control the behavior of the system during the QA process.

For the extractor a model, container and trainer was developed that implement the Naive Bayes and were optimized to build and estimate n-gram models. Furthermore the neural net was trained with the Weka framework (introduced in section 2.3.2).

Classify Answer Type

The coarse estimation of the answer type needs the following top-classes:

- ABBREVIATION
- DESCRIPTION
- ENTITY
- HUMAN
- LOCATION
- NUMERIC

The classification result is shown in equation 5.1. For each class the result set contains a confidence value c.

$$C = \{c_{abb}, c_{des}, c_{ent}, c_{hum}, c_{loc}, c_{num}\} \tag{5.1}$$

These confidence values are decision-making for further classification. The system just uses the two highest valued classes and activates the sub-class models of these classes. The reason behind that is the computational cost. Selecting the two most likely classes is motivated by the fact that a narrow decision between two top-classes can be corrected because the sub-class confidence comes into account.

Classify Detailed Answer Type

Classifying the answer type, as explained in section 5.5, is the basis for this section. The system uses the two classes with the highest confidence and starts the detailed classifiers for each class. Each model has another previously trained statistical feature detector that is a specialist for a class. The advantage of doing so is that no dispensable features are used and that the classifier hasn't seen unnecessary (and confusing) data.

$$C_{hum} = \{cd_{group}, cd_{individual}, cd_{title}, cd_{description}\} \tag{5.2}$$

For example the HUMAN class, equation 5.2 points out the neural network returns a set of confidences *cd*. The confidence predicates the probability that this example belongs to a specific class.

Combined Answer Type Confidence

After the coarse classification and the detailed classification was done the confidences need to be compared and ranked to find out which is the most likely answer type.

$$CC = TC \cdot SC \tag{5.3}$$

The combined answer type confidence CC is calculated with a multiplication between the confidence of the top-class TC and its sub-class SC. The three classes with the highest combined confidences are set as final answer types. The estimation of these types is very crucial, because the estimated classes determine the choice of the query plans for the QA process.

5.6. Answer Representation Detection

The detection of the answer representation is another important step for the success of the QA process. All parts of the system can act correctly and possibly find a valid answer but the if the Answer Representation detector decides to display a *PARAGRAPH* instead of a *LIST* the valid answer will be transformed to a paragraph and the system fails to deliver the demanded list. The model decides between the following three classes:

- FACT
- LIST

- PARAGRAPH

The classifier is based on a neural network (Multilayer Perceptron) that was implemented with the Weka framework. The features for this task are engineered by hand. Some features that are most significant for this task are listed below:

- Prefixes and suffixes of nouns, verbs and adjectives
- The occurring W-word (or question word)
- Occurrence of umlauts
- Use of special Part-of-Speech combinations
- Use of named entities in the question

The list above is not complete. Just the most significant features are listed. At the moment the dimension of the vector that represents the features (called feature vector) has 100 dimensions. This means that all features that separate those three classes are coded in a vector with 100 dimensions. The output of the classifier are the confidence values for each class.

$$C_{representation} = \{c_{fact}, c_{list}, c_{paragraph}\} \tag{5.4}$$

These confidence values, indicated in equation 5.4, are ranked and the class with the highest confidence value (which equals to the highest likelihood) is picked and used for the answer representation. Experiments on this classifier are shown in section 8.2.2.

6. Query Processing

After explaining the annotation of documents and incoming questions this chapter deals with the steps that produce a query out of a processed question (for the context see figure A.7). Therefore a question needs alternative formulations and a plan to solve the question. The generation of this plan and how the plan is scored are the focus of this chapter. First the generation of queries, that are used to query the coarse document search using Lucene, will be described in section 6.1. These queries are used for the coarse search. This coarse search and the scoring are part of section 6.2. During Lucene will be used to query the local knowledge base, a search engine will be used to query against the knowledge in the internet. This mechanism of the system is described in section 6.3. After relevant documents were found (via Lucene or web search) the query processor comes into account. This processor uses the found documents to find the possible answer candidates in it. The processor is described in 6.4.

6.1. Query Generation

The Query Generation module consists of two main parts. The first one is the generation of queries that are used for Apache Lucene Searcher (section 6.1.1). These queries are generated from the question. The second part is the generation of query plans (section 6.1.2). These two procedures are explained in detail in the following sections.

6.1.1. Lucene Query Generation

Lucene queries are necessary to feed the Lucene searcher that comes into account in coarse document search which will be described in detail in section 6.2. In this system Lucene is used to retrieve all documents that are necessary and that are likely to contain the correct answer.

These queries are very important because the selection of words to formulate a query string are crucial to the success of the Lucene query. Therefore the generation algorithm

```
set(complete question)
set(all nouns)

for all nouns and their synonyms as n
  set(n)
    for all verbs and their synonyms as v
      set(n v)
      for all adjectives and their synonyms as a
        set(n v a)
      for all expansions and their synonyms as e
        set(n v e)
```

Figure 6.1.: Algorithm A to create Lucene queries

```
set(original question)
words = [][]
for each word and its expansions as s[] at index i
words[i] = s
setAllCombinations(words)
```

Figure 6.2.: Algorithm B to create Lucene queries

needs to be accurate to ensure the best results. The algorithm shown in figure 6.1 produces queries as noun-verb-adjective and noun-verb-expansion combinations with the goal to pick up the important parts of the questions, to make it short with containing just three parts. In contrast the algorithm presented in 6.2 returns the original question and all expansions of each word which leads to the combinations of all possible and found words in a question. The term 'expansion' means all found syonyms of the word that were found earlier during question normalizing and SynMap processing. The part of this pipeline was already explained in a previous section 5.4.
The 2 proposed algorithms are both delivering results and are also both implemented in the system. In the chapter that deals with experiments on the system, these both algorithms are tested and evaluated, see section 8.2.3.

6.1.2. Query Plan Creation

A query plan is very important for the system. This plan contains a lot of parts to find the correct answer. All of these parts as well as the priorities that a plan has during lifetime are explained in this section. Which query plans are activated is decided after the detection of the detailed answer type. Each of the answer types have specific query plans because each answer type refers to another semantic role or type of word. Furthermore, each answer type has certain targets to look for. For instance, if the question asks for a color it is very likely that the target refers to an adjective. To make sure that all these things are covered during the querying process, the plans were categorized by answer type.

Anatomy of the Query Plan

The plan consists of a few structures that have various intentions during the process of finding answer candidates. These structures are implemented as lists or arrays to quickly walk through them during execution time. The structures are described below:

id Identification number to trace back the plan. This is used to assign answer candidates to corresponding plans for scoring or just for debugging afterwards.

answerRepresentation Contains the desired Answer Representation of the system.

actions Verbs of the question and not part of the answer. This structure is implemented as an array.

focuses Nouns and entities of the question and not part of the answer. Focuses are implemented as an array.

target POS or NE tag of the target in the text. The system first need to match this criterion.

hints System-given words that are likely to appear in a candidate sentence, but not part of the answer. Implemented as array.

parts Tokens that are likely to be used in the answer string and can be part of the answer. Implemented as an array.

majorPriority This is defined by the first, second or third answer type. It contains an integer value.

minorPriority Defines the quality of the target within the major priority. This is implemented as integer value.

answerTypeConfidence Confidence of the primary detected answer type. This was already explained in section 5.5.

score Calculation of the score, is explained, in detail, in section 6.1.2.

story In some cases and if the query plan intends a paragraph as answer type, a story can be set to control the behavior of paragraph generation, see section 8.2.3.

Multiple plans exist for every answer type. The idea behind this is that a location can be detected by POS Tagger and the Named Entity Recognizer (both described in sections 5.2 and 5.3) even as 'I-LOC' which is a valid tag for a location. But the both taggers can fail and classify the location not as 'I-LOC' but rather as 'NE' or 'NN' which are the tags for a 'Named Entity' and a ' Normal Noun'. To also cover these cases more than one query plan with a corresponding *minorPriority* was developed. The priority points out how important the target is according to the accuracy of the tag. For example a 'I-LOC' tag is higher prioritized than a 'NE' when the query plan searches for a location.

Priority & Scoring of Query Plans

As already mentioned in this section, the query plan consists of two levels of priorities. For the scoring, the two priority levels and the formerly explained parts of the query plan are important. The first scoring measure is named $QPCS$ (Query Plan Component Score). For this each part of the plan needs a weight w, that reveals how important this part is. The occurrences of a specific part is also needed and denoted with o. The $QPCS$ can be seen as a linear combination of all attributes and their occurrences o and weights w with one exception which is the target. To the scoring value of the target, the minor priority MIP is multiplied to increase the score for more likely targetsand to decrease less likely targets. After calculating the $QPCS$ the overall query score can be calculated using the CC (answer type confidence, explained in equation 5.3) and the major priority MAJ of the query plan.

$$QPS = MAJ \cdot CC \cdot QPCS \tag{6.1}$$

This final score calculation of the query plan is shown in equation 6.1.

6.2. Coarse Document Search

After the generation of the Lucene queries (with the algorithms explained in section 6.1.1) follows the search via Lucene searcher. This section explains how the queries are used to retrieve relevant documents to work with and extract facts as well as sentences for a paragraph. Section 6.2.1 explains the workflow of applying queries to the searcher and section 6.2.2 describes the scoring of the queries and how the Lucene scoring fits in the scoring of the whole system.

6.2.1. Queries

The Lucene searcher takes queries to run these against the set of all indexed documents. This will be done with all the queries that were generated with the procedure explained in section 6.1.1.
The Lucene classes for this task named *IndexSearcher*, *GermanAnalyzer* and *QueryParser*. All documents were previously indexed from the ChiaqueryKnowledge module and exist in the index of the system. How this part of the system is structured was already explained in section 3.1.3.

6.2.2. Scoring

Lucene uses a few measures to determine the score of a document. The Lucene scoring determines the match between a query and a document. The core of the scoring function is the VSM (Vector Space Model) where every query and document can be presented in a high dimensional space with weighted vectors where each dimension is a specific term of the index space - and weighted by TF-IDF. The similarity between a document d and a query q is computed by the cosine similarity which is illustrated in equation 6.2.

$$cossim(q,d) = \frac{V(q) \cdot V(d)}{|V(q)||V(d)|} \tag{6.2}$$

For that the function V represents the vector for the text. Furthermore $V(q) \cdot V(d)$ and $|V(q)||V(d)|$ are presenting the dot product and euclidean norm of the vectors.
The complete calculation is far more complex. Beside the cosine similarity there are a few more components that are important for Lucene overall scoring function:

coord In some cases, a query is suited by a set of sub-queries. Each sub-query is applied to the Lucene searcher. To balance the different result sets and documents that may occur in more than one sub-query, the coordination factor exists.

boost At the time of applying the query to the Lucene searcher, each part of the query or sub-query can be boosted and has more weight during search. If a document fulfills the constraints for higher rated sub queries more than others, the document will be boosted with the given factor.

doclen The document length normalization factor scales the document by its length. Some documents are larger than others. To normalize the cosine similarity output the doclen factor is applied.

docboost At the time of indexing each document can be defined with a boost factor because it is more important than other documents. This factor is applied to boost the document score.

$$score(q, d) = coord(q, d)boost(q)cossim(q, d)doclen(d)docboost(d) \tag{6.3}$$

Equation 6.3 brings all the factors from the listing together. This is the function that is used by Lucene to score documents in conjunction to queries.

6.3. Web Document Search

Another option beside the local knowledge (section 6.2) is the search via search engines that return web documents. The advantage for this option is that the web always contains the latest information and a much wider range of knowledge. Section 6.3.1 deals with the selection of queries and the how they were delivered to the related module. After this the readability of the document will be discussed in section 6.3.2.

6.3.1. Queries

To request the search engines in a smart manner, just a few queries are selected for this procedure. The problem can be that the search engine blocks the system because the API limit was reached with just a pair of requests or answered questions. On the other hand the answer can suffer from sparse information that was provided from just a few queries. The accuracy and precision of leading search engine companies is the main thing, that the local system must rely on to successfully rate the results.
The system proposed in this thesis uses the at least three queries for each question, thus for each question answering process that was started.

6.3.2. Readability Techniques

The readability of the retrieved web documents is the crucial part here. The worst case for the system is a document, that the parser can not parse and the text annotation

pipeline fails to estimate Part-of-Speech tags, named entities, sentence boundaries and all other parts.
The problem is to get the main content out of the complete HTML file that was retrieved. When the web site is opened up in the browser, the visual inspection of the site can clearly determine which part of the content is the main part, which parts are listings, tables and how important they are. As a computer this task is harder to solve. Two frameworks are selected to tackle this problem. The first one is JSoup which was developed to manipulate the structure of a HTML document and provides an easy mechanism to download web documents. The second framework is Palladian which has provides more tools and mechanisms for content readability and text classification. These both are introduced in section 2.3.5 and evaluated in section 8.2.4.

6.4. Query Processor

The processing of the question that was transformed into a query is the main part of the QA process. It unites the query plans (section 6.1.2) with the indexed documents from the knowledge base (section 3.1.3) as well as retrieved documents from search engines (section 6.3). Therefore, some preparations are necessary which will be explained in section 6.4.1. The process to find answer candidates for paragraphs and facts (and lists which consist of n facts) will be described in sections 6.4.2 and 6.4.3. Afterwards, the creation of answer candidates will be discussed in section 6.4.4.

6.4.1. Build-Up

The relevant documents that were retrieved in the Lucene search process (6.2) are passed into the processor and need to be sorted. For this the documents will be re-arranged. The documents with the highest scores are on top of the list (descending order). Currently the system has a variable $QP\_MAX\_DOCUMENTS$ that decides the number of documents used for the process. By default this value is set to 10. So the list of documents will be cut off after the $QP\_MAX\_DOCUMENTS$ element. All of these documents are then opened up from the systems index and each query plan will be applied to each document. A query plan contains an answer representation (AR) attribute which decides the direction of the query plan. If the AR value equals to LIST or FACT, the procedure described in section 6.4.2 will be applied. Otherwise, the AR value equals PARAGRAPH, the procedure for paragraph generation will be administered. This procedure will be discussed in section 6.4.3. That the query processor is a crucial part of the system can be seen in figure 6.3. All generated query plans are processed on each document with the goal to generate answer candidates. These candidates are later filtered, sorted out and scored to receive a ranked and sorted list of possible answer hypotheses.

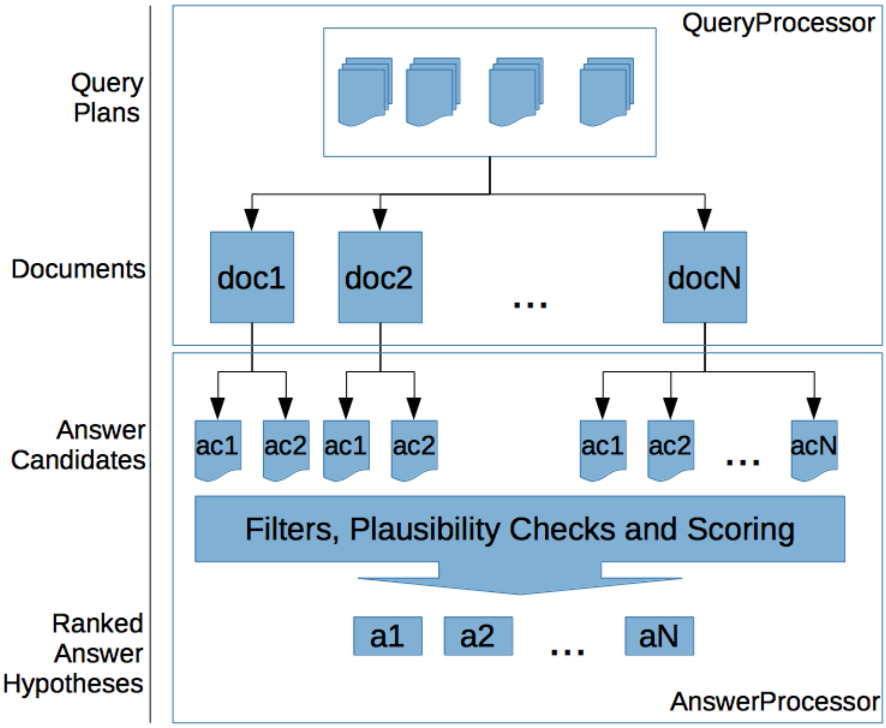


Figure 6.3.: Conceptual description of QueryProcessor module. Each query plan is applied to each document. After the generation of answer candidates, plausibility checks and scoring are applied.

6.4.2. Processing Facts

For the processing of facts index information of the annotated text document plays a significant role. This information was motivated and described in section 4.8 and is used to index verbs, nouns, entities and a few more important word types. These index structures are used for this part of the thesis to rapidly find relevant parts in large documents.

Finding Relevant Sentences

Relevant sentences are those which contain parts and expressions of the question as well as possible targets. To find the relevant sentences out of thousands of them the index

structures are used as sets. The attributes of a query plan (described in section 6.1.2) are applied and sentence that contain specific parts of the query plan are scored accordingly to the containing parts. For this, the system calculates intersections of the sets in the corresponding index.
As an example, if a question contains the verb 'bauen', then 'bauen' and its synonyms are stemmed and assigned to the query plan's *action*-part. The system then tries to lookup the index for the verb 'bauen' and its synonyms. For this example, we assume that sentences $\{4, 9\}$ contain this verb. Furthermore the system tries to find the *focus*-part of the query plan. In this example the found focus lies on the found entity 'Z4' which the system found in sentence $\{9\}$. To bring this example to an end, the intersection of the focus and the action-part ($\{9\} = \{4, 9\} \cap \{9\}$) leads us to the conclusion that sentence 9 is more important and therefore this sentence is scored with a higher value. If one part of the query plan was found the containing sentence is added to the found structure. All of these structures are introduced below:

actionFounds Contains all ids of sentences that contain a member of the query plans action list, where the verbs index of the document is used

focusFounds Contains all ids of sentences that contain a member of the query plans focus list, where the nouns and entities index of the document is used

hintFounds Contains all ids of sentences that contain a member of query plans hint and part list, where the nouns and entities index of the document is used

These 3 lists contain the relevant sentences ids. To find out which of these sentences are more important, an intersection is calculated for all variations (of the list) which leads to 7 lists after the operation (each list itself, two lists in combination and a list containing all three criteria).
The list that contains all three criteria named *bullsEye*.
After the placement of all sentences in the correct list (with different initial scoring), each sentence is processed to find the correct answer.

Finding Possible Candidates

Finding the correct target (which is also a part of the query plan) comes after the selection of relevant sentences. Therefore, the algorithm travels through the sentences from left to right and if the correct POS tag (or tags, in case of a target with more than

word) occurs the system adds this word to the list of answer candidates.
After this a bunch of scoring boosts are performed by the system:

- Add occurrence of action, focus and hint items
- Set the length of the target and the corresponding answer candidate
- Add occurrence of an item from the part list

Some plausibility checks are done afterwards. The checks deny possible candidates that contain words that are not allowed in an answer string. These checks include all words of the focus and hints structure as well as all words like 'eines', 'ihrer' and some other words that are wrongly classified by the POS tagger. The module that solves this task name *AnswerFilter* and loads a database that contains a word list of German words, that are often misclassified and not likely to be the answer.
If the answer candidate passes all of these stages, it will be added to the set of possible proven candidates.

6.4.3. Processing Paragraphs

In general, the generation of a paragraph can be divided into two approaches. The first is the extraction approach which selects words and phrases that are already exist in the text to create the paragraph. The second approach is named abstraction approach and it creates an internal semantic representation of the text and creates the paragraph using a NLG module which brings the semantic representation to a bunch of sentences. For the abstraction approach far more knowledge is needed, so this work is based on the extraction-based approach.
The system implements two different extraction-based algorithms. The first is specialized on building a paragraph out of the hints of a query plan and the second follows a story line to create the answer.

Hint-based Paragraph

This algorithm is used for answer types that provide a description, explain a reason or a term. For this, the sentences from all relevant documents that most accurately match the question are merged together to build a paragraph with a formerly determined number of sentences.

Story-based Paragraph

A story has a story line which consists of actions and focuses. This story is used to create more complex paragraphs. For instance the answer type HUM_DESCRIPTION, which expects a statement about birth, childhood, profession, family and optional death of a human individual. Each part of the story contains the actions (verbs), focuses (nouns and entities), the order in the paragraph and the number of sentences. This can be seen as a few sub queries that solve a different problem, in particular find sentences that match the requirements of a part of the story.

6.4.4. Answer Candidates

All of the above created answer candidates are then transported to the Answer Processor which will be described in chapter 7. The candidates are transported in a list of *AnswerCandidate* elements. Candidates of all query plans are combined in this list. The achieved scoring so far is attributed to each candidate in the list. Also all answer representation types (FACT, LIST and PARAGRAPH) are combined in this list.

7. Answer Selection & Presentation

To find the correct answer to a given question the extraction of candidates is very important. Another important problem to solve is the ranking of text, more precisely relevant text. This chapter deals with the ranking of possible answer candidates and how they will be presented to the questioner (consult figure A.7 for the context of this module). Section 7.1 deals with the most important module of the system regarding to answer selection and representation. Here the inappropriate answer candidates are filtered out and the candidates are ranked in terms of their relevance. After this was done, the answer representation will be applied to display the answer correctly.
At the end of the process the natural language generation comes into account. This module explains in a sentence how the system understood the question and which answer was retrieved. This additional part of the system is described in section 7.2.

7.1. Answer Processor

The answer processor gets the list of answer candidates (as explained in 6.4.4) and the option to filter answers (as section 7.1.1 will describe). This option can be seen as a boolean switch, that can take the two values: *true* and $false$. After this step the candidates can be registered into the answer process (section 7.1.2) and will undergo a final re-arrangement of the scoring. This answer process is interrupted if the primary estimated answer representation type is a PARAGRAPH. For this, the created paragraph will be used and displayed. This decision will be discussed in section 7.1.3.

7.1.1. Answer Filters

Answer filters are an additional feature. The answer filter used here has two functionalities that are necessary to succeed the Question Answering pipeline. The first one is to filter out words that are accidentally misclassified by POS or NER but chosen to be a valid answer candidate. This was already described and used in section 6.4.2 to filter misclassified words and avoid adding them to the answer candidates. This has the advantage of deleting wrong candidates early and to not let them all the way through the system. The second functionality is the filter by semantic criteria. The answer filter is also capable of filtering candidates that are not plausible to be a date, a weight or distance.

If the query plan contains the answer type NUM_DATE which is looking for date/time values then -1 is not a plausible answer. The answer filter takes care of this and can be called system-wide in a static manner.

7.1.2. Register & Rank Answer Candidates

After the filter process was done, the candidates that still exist can be registered to the answer candidate set of the final answer. For this, all candidates were transferred to a separate list. If a candidate string occurs twice, the scores were added to a counter for each string. This increases the likelihood to find a possible interesting candidate. The redundancy of information leads (in this case) to the significance of information. This rule can be applied of this level of information where all unnecessary words (stop words and irrelevant parts) are removed.

The accumulated scores of the combined candidates are then re-arranged with the highest value on top of the list. It is important to keep in mind that lists (containing a specific number of facts) need a threshold value for the elements of the list. Currently this is not implemented. If the question asks for a list of facts the system always returns 5 or 10 items. For further development a real threshold should be established to cut off the list correctly.

7.1.3. Apply Answer Representation

The answer representation, explained in section 5.6, comes into account now. During the Question Answering process the query plans traveled through documents, retrieved a massive amount of information and created paragraphs. To decide the information that can be displayed, the decision of the primary answer representation type is very important.

The answer representation type can come to the following conclusions:

FACT Chooses the answer candidate that has the highest score.

LIST Chooses a selection of the answer candidates with the highest score.

PARAGRAPH Decides to display the query plan that contains the story or paragraph with the highest score.

After the decision which answer and answer type will be represented, the candidate/s are assigned to the *answerString* which is the attribute that shows the final answer. This answer is provided using all semantic information to perform a Natural Language Generation that tries to build a sentence.

7.2. Natural Language Generation

After all decisions and the process that turns a raw text question and a generated query into an answer string, the answer needs to be displayed correctly and in human language. For this, the system has a natural language generator, that combines two core functionalities:

- Explain how the system understood the question
- Integrate the answer string into a well-formed sentences

These two features are shown with two example questions below:

```
Question: "Welche Maschinen baute Konrad Zuse?"
1) Gesucht ist nach N Produkten mit den Tätigkeiten 'baute', 'bauen'(bzw. 'baute', 'bauen')
in der Vergangenheit und der Person 'Konrad Zuse' .
2) Die gesuchten Produkte sind Z1, Z2, Z3, Z4.

Question: "Welche Firma stellt den iMac her?"
1) Gesucht ist nach 1 Organisation, Verein oder Firma mit der Tätigkeit
stellt (bzw. 'stellen', 'her') in der Gegenwart und dem Objekt 'iMac' .
2) Die gesuchte Organization, Verein oder Firma ist Apple Computer.
```

8. Experiments & Evaluation

The system that was introduced during the previous chapters undergoes experiments and will be evaluated in this chapter. Additionally, it depicts the experimental environment and the involved parts. The experiments are ordered by the position of the system under test in the entire processing pipeline. This thesis evaluate just chosen parts of the system, but there are a lot of other possibilities for experiments. Therefore, further experiments are presented in section 9.2.2.

8.1. Introduction

This section introduces the experimental environment and the workflow during the experiments. The most important aspects are covered here - all other aspects can be found in the sources directory or the documentation of the system.

8.1.1. Annotation for Model Training

The annotation of training data follows a simple principle. Each annotator that helped to collect data samples for two tasks (answer type detection and answer representation) follows the same constraints. A question is annotated with the top-class, sub-class and the question itself. Below an example is shown:

```
entity#body#Welcher Teil des Körpers beinhaltet das Limbische System?
```

Using a script this question is automatically processed and assigned to two data sets. The first set for the top-classes and the second set is the set for the 'entity' classifier.

```
Welcher Teil des Körpers beinhält das Limbische System?#ENTITY
Welcher Teil des Körpers beinhält das Limbische System?#ENT_BODY
```

The first question is used for the training of the classifier that decides the top-classes and the second question is used to decide the sub-classes of the 'entity' top-class. For the Answer Representation task the data is annotated in an analogous manner.

```
Welcher Teil des Körpers beinhält das Limbische System?#FACT
Wie definiert man Informatik?#PARAGRAPH
Welche Tiere können fliegen?#LIST
```

8.1.2. From Phrase to ARFF

In the previous section the annotation of a phrase was described. This section resumes the previous section and shows the way from the correctly annotated phrase to a feature vector that can be used for training models - in this system, artificial neural networks using the Weka3 framework.
The first step is to start the chiAQuery system with the following parameters:

```
java -jar caqy.jar features v4AT <phrase file>
```

The output of the system is a feature vector that the statistical feature extractor determined on all of the phrases. This vector (with the corresponding label) is expressed as displayed below:

```
0.2, 0.1, 0.0, 0.782, ... ,0.1,ENTITY
```

The last step is to generate a valid ARFF file out of this CSV file format. For this step, the KNIME toolbox was used with an easy tool chain that consists of reading CSV file and writing (if valid) to an ARFF file.

8.1.3. ChiaqueryQABenchmark module

The module that was developed to evaluate QA performance takes a file with the question and some corresponding information.

```
<question>#<answer_type>#<detailed_answer_type>#<representation>#<answer>
Wann starb Konrad Zuse?#NUMERIC#NUM_DATE#FACT#1995
```

The response of the system contains a lot more information about the processing. After the question itself, the most likely answer types are responded (winner, second and third) as well as their sub classes. Next, the two (winner and second) answer representation are displayed, before the answer type confidences of the first two answer types are printed. In this example the system decides with a confidence of $99,9\%$ that the question asked for a 'date'. And the last statements are about question confidence QC and answer accuracy ACC.

```
Wann starb Konrad Zuse?#NUMERIC#ABBREVIATION#NUMERIC#NUM_DATE#
ABB_EXPRESSION#NUM_DISTANCE#FACT#PARAGRAPH#
atConfidence1:0.9990258902307149#atConfidence2:4.989532312230247E-4#
qc:0.2300568218279387#acc:64.34#
1941;1955;2010; ...
```

8.1.4. Error Classes

The evaluation of a QA system is not an easy task. The question: 'What is a good/excellent answer?' very is subjective and sentimental. Answers that expect a fact or a list of facts can be evaluated in a more objective manner, but this is not possible for entire paragraphs. Maybe all words of a given list appear in the paragraph (which is good), but a human can't understand the paragraph because the sentences are badly combined together. Not all of these problems are solved with the error classes, explained in figure 8.1, but the quantitative way of an evaluation is covered with that. In the field of Question Answering, the only method to evaluate subjective paragraph quality, is to have a number of subjects that all evaluate the same paragraph. After that, a mean value can be calculated about all statements of the subjects. Unfortunately, due to sparse time and resources, this method could not be applied in this thesis, although it would be the preferred way of evaluation. Note that answer sets used for the evaluation can also be incomplete or defective.

Class	Description
Excellent	For PARAGAPH, LIST and FACT the answer must contain all terms.
Good	For LIST and PARAGRAPH the answer must contain more than one term. For FACT the answer must appear in the top 5 ranked answers.
Sufficient	For LIST and PARAGRAPH one term must appear in the answer. If the representation is FACT, then the answer must appear in top 10 ranked answers.
Bad	No relevant term could be found in answer set. Only, if a possible answer could have been found in the set.

Figure 8.1.: Error classes for Question Answering

8.1.5. Experimental Data Sets

Here all created sets are introduced shortly. To have a broad range of questions, especially for the open domain question answering evaluation, some question sets are recorded

from different pages and sources. This was done to ensure the diversity of questions from different areas from general knowledge to knowledge of sport topics. The goal was to enhance the representative of the experiment with the use of different sets from several areas.
All of these sets are introduced and evaluated by quantity in this section.

Q50

This set was created using 50 modified questions from the DATD data, that was already explained in section 2.4.3. These questions were manually answered using web research and encyclopedia. This data set is the second largest that was used to evaluate the system built in this thesis.

Q100

Q100 was used for the content readability experiment and consists of 100 questions, randomly chosen to get responses from the search engine. For this set, no answers were needed, because the content readability experiment just needs documents to find the main content of the page.

Einbürgerungstest / Citizenship Test

The 'Einbürgerungstest' was acquired from a website[1] that offers free citizenship tests for the Federal Republic of Germany. 62 factoid questions were chosen and some of them modified to fit in the criteria of the system and its benchmark module.

Einstellungstest / Recruitment Test

The recruitment test set 'Einstellungstest' was acquired from an online portal[2] that asks questions for job applicants. This set consists of 24 questions from all categories that the portal offers.

[1] Source page of this data set: http://www.einbuergerungstest-online.eu/fragen

[2] Source portal of 'Einstellungstest' set: http://www.einstellungstest-fragen.de/category/allgemeinwissenstest/

Sport Quiz

To test the system also in the area of sport, the 'Sportquiz' set was created with questions from a sport page[3]. Just 9 questions matched the criteria of the system for a question and these are inserted into the set.

WWM

The German version of 'Who Wants to Be a Millionaire' is called 'Wer wird Millionär'. In this show the candidate must answer questions to win money and with the last and most valuable question the candidate wins 1 million euros / dollar. 15 out of 42 of the questions, that had 1 million euros / dollar of worth, are combined in the 'WWM' set[4].

TSDD

One possible use case of the system is the Technische Sammlung in Dresden, where the robot of the HTW cognitive robotics research group is located. For that, some documents (slides, texts and web paes) were acquired and preprocessed. The questions on that data set are restricted to the domain of the museum exhibits. The number of questions for the 'TSDD' set is 11 in total.

8.1.6. The Experimentation Environment

For all of the following experiments one machine was used for the tests. The machine is an Apple MacBook Professional that runs on 'OS X 10.9.5 (13F34) ', was manufactured in early 2011 and was set up with 16GB DDR3 (1333MHz) working memory (RAM). The data is stored on a SSD (Solid-State-Disk) of the type 'Samsung SSD 840 EVO 250GB Media' connected to the bus via SATA . The used processor named Intel Core i5 which runs on 2,3GHz.

8.2. Experiments

After the introduction to the mechanisms around the experiments this section deals with the experiments itself. The experiments are organized as they appear during the Question Answering pipeline.

[3] Source for sport questions: http://www.raetselstunde.de/quiz/sport-quiz.html

[4] Data acquisition from the site: http://www.bild.de/unterhaltung/tv/wer-wird-millionaer/feiert-15-jaehriges-bestehen-alle-42-millionenfragen-zum-nachquizzen-38179838.bild.html

8.2.1. Answer Type Detector

The answer type detection experiment tackles the part of the system that was described in section 5.5. In this experiment, the Weka framework (section 2.3.2) was used with the pipeline described in 8.1.2. At first, the features of the detector were hand-written rules combined with some statistical decisions. These features (Baseline approach) - the dimensions of the feature vector - are explained below:

Question words The appearance of the most significant question words (binary)

Known Entities The appearance of the most significant known entities for each class (numeric, cumulative by number occurrence)

Grammatical Footprint A distinctive transformation of the grammar into a n dimensional vector

Tense and Named Entities The occurrence of named entities and the tense (binary)

This baseline approach had the disadvantage that it was engineered by hand (manually) and the feature set was the same for each classifier, the main classifier and the classifier that determines the sub classes. Especially for sub classes that provide just 2 (abbreviation) or 4 (description) sub classes, the curse of dimensionality (also known as Hughes effect) comes into account. The number of data samples for some sub classes was too low related to the high dimensional feature vector. After noticing the bad performance of this approach, the idea of the statistical feature detector was established. These features were already explained in section 5.5. With this approach, each class that is determined using the classifier has a likelihood value on each perception layer. This means that the dimensionality of features increases with the number of classes to determine in a linear manner. In some cases as well as in this thesis, for a classifier that determines between a little set of classes, the training samples set is also smaller. With the new V4 perception approach, the classifier suffers less on the curse of dimensionality. Furthermore the some features of the baseline approach are encoded in the binary way. This leads to just two states of this feature $\{0, 1\}$. To take more use out of a single dimension, the enhanced approach uses all features as probability values with a continuous value range of $[0, 1]$. Figure 8.2 illustrates the comparison between baseline and V4 approach by overall accuracy of the main classifier (classes) and all sub classes.
The training needs to be well organized to prevent the system from overfitting. As figure 5.7 in section5.5 depicts, two parts of this system needs data for training. At first the

Class	Baseline Acc.	V4 Acc.
CLASSES	71.1%	85.9%
ABBREVIATION	74.6%	83.3%
DESCRIPTION	77.3%	83.7%
ENTITY	44.2%	89.1%
HUMAN	81.0%	93.3%
LOCATION	55.2 %	83.8%
NUMERIC	64.8%	88.7%

Figure 8.2.: Comparison of accuracy values between baseline classifier and V4 architecture.

statistical feature extractor needs to be trained. Therefore the data set 'DATD-1' was used. For the training of the neural network, the 'DATD-1' set was also applied for training. The network learned the mapping from the perception (V1 up to V3) and puts it together. After that 'DATD-2' was applied for testing the network. Both, the statistical feature extractor and the network are trained with the same set, because the network should learn the mapping for the same data set. The then used 'DATD-2' set is an unseen set to the trained classifier. The both data sets are described in more detail in section 2.4.3. The results of this experiment are shown in figures A.10, A.11, A.12, A.13, A.14, A.15 and A.16.
Below the ROC space for this experiment are illustrated in figure 8.3. Here the x-axis represents the false positive rate and the y-axis is denoted as true positive rate. The receiver operator characteristics for all the sub-class classifier can be seen in the appendix in figures A.1, A.2, A.3, A.4, A.5 and A.6. All ROC illustrations are following the same pattern. V4 data is displayed using a star and baseline data uses a circle. Furthermore, each class (for both classifiers) is colored with a unique color. Figure A.3 waives any legend, because of the high number of classes. Therefore, just stars and circles are drawn to compare the performance.

8.2.2. Answer Representation

The representation of the correct answer and the used features were already explained in section 5.6. For the task of Question Answering this task is very important because this classifier decides the form of representation and thereby which query plans are used as well as which plan was filtered out. For this experiment the Weka (section 2.3.2) framework with the explained experiment pipeline (section 8.1.2) was used to train a classifier for this need. Figure 8.4 show the results of the experiment, evaluated by

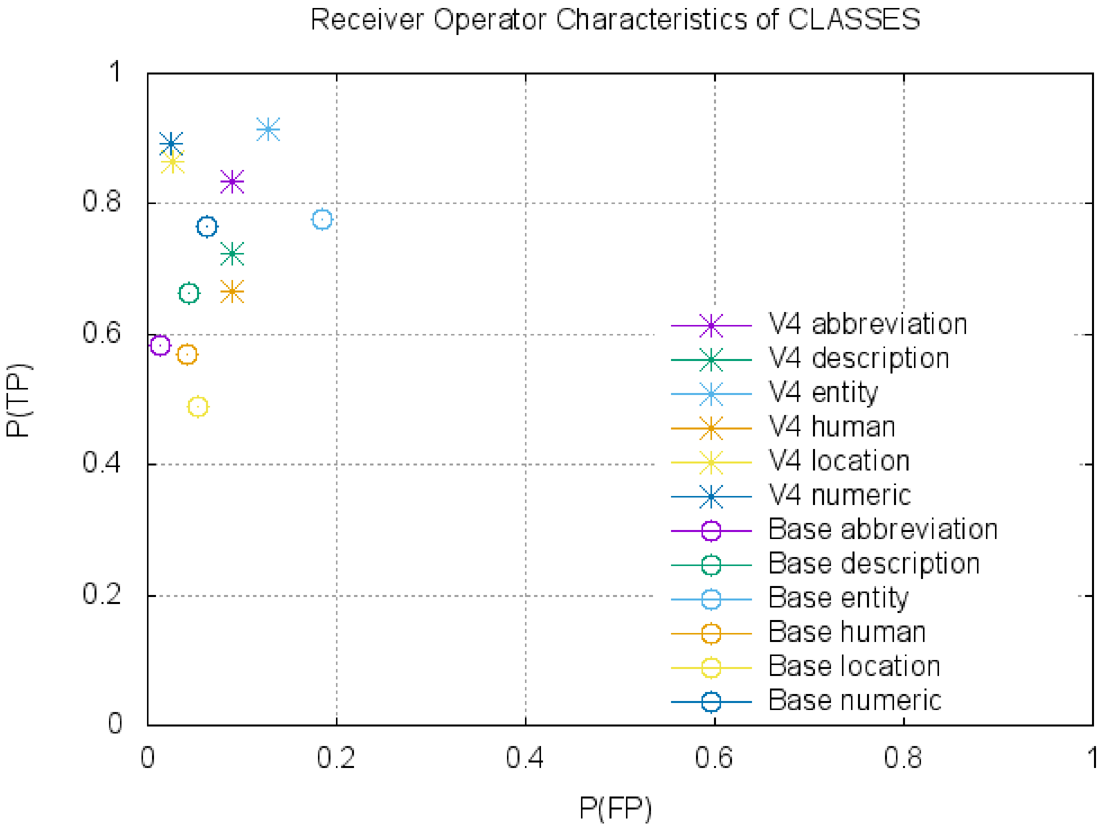


Figure 8.3.: ROC space of CLASSES experiment. Comparison between baseline and V4 classifier.

TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
0,774	0,105	0,801	0,774	0,787	0,675	0,906	0,855	FACT
0,849	0,073	0,844	0,849	0,846	0,775	0,950	0,897	PARAGRAPH
0,898	0,065	0,871	0,898	0,884	0,826	0,963	0,940	LIST
0,839	0,082	0,838	0,839	0,838	0,757	0,939	0,897	Weighted Avg.

Figure 8.4.: Results of the answer representation experiment

the performance measures explained in 2.1.3. The overall accuracy of the experiment was 83.865% where 502 samples were used for this experiment. The system was 10-fold cross validated. The confusion matrix, illustrated in figure 8.5 breaks down the recall

	as **FACT**	as **PARA**	as **LIST**
FACT	137	22	18
PARA	20	135	4
LIST	14	3	149

Figure 8.5.: Confusion matrix of answer representation experiment

values of each class. This figure shows how many 'FACT's are classified 'as LIST's for instance. The Recall value for the 'LIST' class is the best, followed by 'PARAGRAPH' and 'LIST'. For this experiment the configuration '-L 0.4 -M 0.15 -N 1500 -V 0 -S 0 -E 20 -H a -D' of the Weka Multilayer Perceptron was used. To complete this experiment the ROC space of the experiment is shown in figure 8.6. The x-axis represents the false positive rate and the y-axis is denoted as true positive rate.

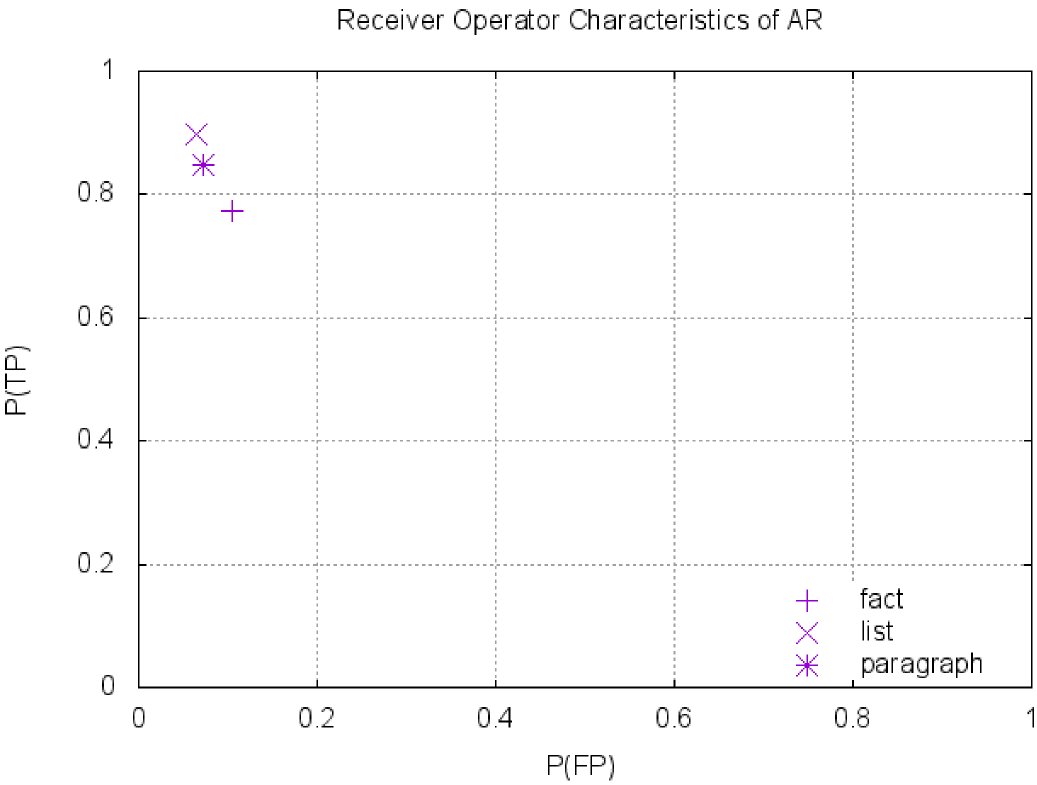


Figure 8.6.: ROC space of the Answer Representation classifier

8.2.3. Query Generation Algorithm

In this experiment, the two algorithms illustrated in figures 6.1 and 6.2 are tested against each other. Query generation is a crucial part of the system to find relevant documents (see section 6.1). Because a set of more and often reformulated queries perform better than just one query (or the question unprocessed as query), these two algorithms were developed. The experiment uses the 'TSDD' questions. Both algorithms are compared against each other in two criteria.
The algorithm that reaches a higher Lucene score shows that it is possible to find more related documents than the other algorithm, which is, for this experiment, the measure

of quality. Another criterion is the average score of all Lucene requests. For that the query scores, denoted as qs, for each query that a algorithm produces are added up and were later (eq. 8.1) divided by the number of produced queries N.

$$\frac{\sum_{n=0}^{N} qs_n}{N} \tag{8.1}$$

The results are shown in figure 8.7. The two criteria 'best' and 'average' score are listed and the two algorithms are compared. Algorithm B reaches, on both criteria, the more accurate results. 81.81% percent of the data reached the higher document score using algorithm B. Furthermore, algorithm B also reaches better results for the sum of qs to N ratio.

	Algorithm A	**Algorithm B**
Best Doc Score	18.18% (2/11)	81.81% (9/11)
Avg. Doc Score	36.36% (4/11)	63.63% (7/11)

Figure 8.7.: Result of Query Generation experiment

8.2.4. Content Readability Experiments

The readability experiment is important for the task of open domain question answering. This part of the system was already introduced in section 6.3.2. The problem here is to extract the correct part of a HTML page, in this case the correct part is the main content of the web page (without sidebars, headers, footers and other parts). Content readability is a subjective problem of perception and feeling. Therefore the data set Q100 (explained in section 8.1.5) was used and all documents that were retrieved by the systems were stored in systems cache. After that, the experimenter extracted each main part of each HTML document. Afterwards, each system used to ensure the readability, did the same on the documents. A script compares the results from the experimenter against the results from both systems. The 100 questions retrieved 997 documents (3 URLs could not be reached). The results are shown in figure 8.8. Here the quality of the output is measured in 3 categories. 'Correct' means the output of the system matches the string of the experimenter. 'Nearly' means that the returned string contains the main content, but also contains other tokens. A token means a letter or digit that is displayed on the web page after stripping all HTML tags and non-visible parts of a web page. This could be a part of the navigation bar that was wrongly classified as main

	JSoup	Palladian
Correct	391 / 39,2%	758 / 76,0%
Nearly	401 / 40,2%	198 / 19,9%
Wrong	205 / 20,6%	41 / 4,1 %

Figure 8.8.: Experimental results of the content readability experiment

content. To reach the 'Nearly' class the returned string could have 1 up to 200 wrongly classified tokens. The 'Wrong' class contains all documents that maybe contain the correct content but, and that is important, contains more than 200 other tokens that are not relevant.

8.2.5. GSON Performance

For the serialization of the annotated documents and their index structures a format called JSON was used. To handle the serialization with JSON the system used GSON (Googles library for JSON). GSON was already introduced in section 2.3.6. The focus of this experiment was on input and output speed as well as on file size of the annotated documents.

Two processes are depending from the values that are evaluated here. The first process is the indexing of documents (file size and write performance are evaluated here) which needs to be fast because the user don't want to wait if the documents are annotated during external index processor or when the program indexes the files from the web on the fly. The second process is the answering of a question (read performance will be evaluated here) itself. This process opens the files and uses them to find the correct answer quickly. For the file size criterion each annotated document file size was divided by the corresponding original file from the corpus. The resulting value points out how much times larger or smaller the annotated file is than the original file. For this calculation all documents from the TSDD (introduced in section 2.4.4) data set were annotated.

Criterion	Measured Value
File Size	38.3x
Avg Read (In)	0.127 sec
Avg Write (Out)	0.065 sec

Figure 8.9.: Results of GSON Experiment

To measure the write performance the indexing process was used. For all indexed TSDD documents the time from creating the JSON document up to the write operation was

captured. Similarly, the read performance used the same measure points for this experiment, but the involved process was another one. The system received 3 different questions and opened up the documents. The time for open up each document and convert the JSON-string into the write Java object was measured. Over each run the measured times were averaged.
The result of these experiment can be seen in figure 8.9. The file size increases very rapidly and is 38.3 times larger than the original document. The bad result of reading a document (0.127 seconds) is caused by time intensive mechanism of mapping complex JSON objects into valid Java objects. The write operation is relatively fast and was boosted with the fastest possibility in Java to write into a file. The time consuming part during the write is the convert process from Java object into JSON representation.

8.2.6. Closed Domain Question Answering

The first experiment that covers the overall system performance (from question to answer, all components from chapter 3 to 7) works on the TSDD data set (using 11 questions) that was already explained in section 8.1.5. This data set contains questions that could be answered, because the answer exist in the data set. Nonetheless the system needs a suitable formulation of the content to find the correct answer. To complete this experiment the benchmark module of the system was used (explained in section 8.1.3). As figure 8.10 illustrates the system was able to answer 27.27% correctly and 63.63% of the questions could be answered indirectly (contained the correct answer in the top 5 or top 10 answers). Beside the results of the correct answer, the system made decisions

Class	Percentage
Excellent	27.27% (3/11)
Good	36.36% (4/11)
Sufficient	27.27% (3/11)
Bad	9.1% (1/11)

Figure 8.10.: Performance of Closed Domain QA

	Percentage
ATD	72.72% (8/11)
ATD second	66.6% (2/3)
ARD	81.81% (9/11)
ARD second	100.0% (2/2)

Figure 8.11.: ATD and AR results of Closed Domain Question Answering

on a few more parts. These parts are listed with their accuracy in figure 8.11. Here the Answer Type Detector and the Answer Representation Detector as well as their second choices are shown. The percentage values of the second choices are calculated by the divison of all questions (samples) where the second choice of the system was correct through all samples that were classified wrong.

8.2.7. Open Domain Question Answering

This experiment evaluates the overall system performance (from question to answer, all components from chapter 3 to 7) for open domain question answering. The open domain refers to a lookup to relevant search engines. For this task some data sets were acquired and described in section 8.1.5. Out of this collection the sets 'Q50', 'Citizenship Test', 'Recruitment Test', 'Sport Quiz', 'WWM' are used. Together all of these sets provide 157 questions. The accuracy statistics of this experiment are illustrated in figure8.12. The system reaches 38/157 (24.2%) at 'excellent' class and in 129/157 of all cases (82.17%) the system maintains to provide an answer directly or indirectly. Figure 8.13 demonstrates the performance of the Answer Type Detector and Answer Representation Detector. The detector of the answer type reaches 73.25% accuracy and the correct representation of the answer was estimated in 128 of 157 cases (81.53).

Class	Percentage
Excellent	24.2% (38/157)
Good	32.48% (51/157)
Sufficient	25.48% (40/157)
Bad	17.84% (28/157)

Figure 8.12.: Performance of Open Domain QA

	Percentage
ATD	73.25% (115/157)
ATD second	64.29% (27/42)
ARD	81.53% (128/157)
ARD second	93.1% (27/29)

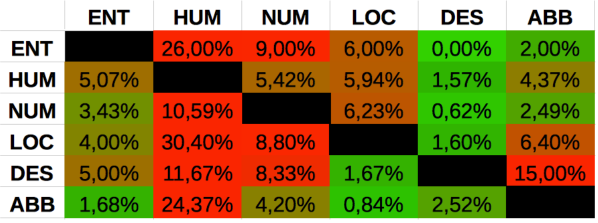
Figure 8.13.: ATD and AR results of Open Domain Question Answering

8.3. Evaluation

This section deals with the evaluation of the experiments and discusses their results. All section are listed in the same order as the experiments, which are ordered like their appearance in the system during the execution of the QA process.

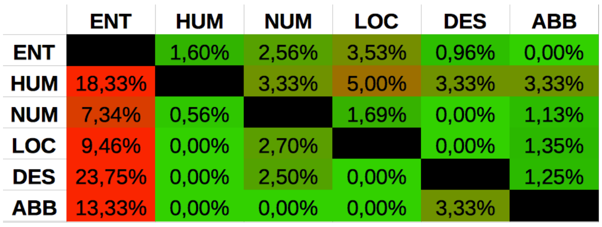
8.3.1. Answer Type Detector

The corresponding experiment was described in section 8.2.1. The comparison between the baseline and the V4 approach points out, that V4 offers more accurate results, especially when the classifier needs to determine between many classes.



	ENT	HUM	NUM	LOC	DES	ABB
ENT		26,00%	9,00%	6,00%	0,00%	2,00%
HUM	5,07%		5,42%	5,94%	1,57%	4,37%
NUM	3,43%	10,59%		6,23%	0,62%	2,49%
LOC	4,00%	30,40%	8,80%		1,60%	6,40%
DES	5,00%	11,67%	8,33%	1,67%		15,00%
ABB	1,68%	24,37%	4,20%	0,84%	2,52%	

Figure 8.14.: Baseline classifier confusion matrix. Classes HUM and NUM produce the most confusion.



	ENT	HUM	NUM	LOC	DES	ABB
ENT		1,60%	2,56%	3,53%	0,96%	0,00%
HUM	18,33%		3,33%	5,00%	3,33%	3,33%
NUM	7,34%	0,56%		1,69%	0,00%	1,13%
LOC	9,46%	0,00%	2,70%		0,00%	1,35%
DES	23,75%	0,00%	2,50%	0,00%		1,25%
ABB	13,33%	0,00%	0,00%	0,00%	3,33%	

Figure 8.15.: V4 classifier confusion matrix. The classification of class ENT can be seen as a problem and the improvement should be an object for further experiments.

Figures 8.14 and 8.15 illustrate the confusion matrices of both classifiers. These figures can be read from the left labels to the top labels and the corresponding cell. An example using figure 8.15: "The percentage of entities (ENT) that were wrong classified as locations (LOC) is 3.53%". The V4 confusion matrix can be seen as more accurate, but the classifier still struggles telling entities from non-entities. The entity class contains 22 sub classes and therefore this class has the highest diversity (see figure A.8 for the classes). Therefore the classifier had problems to properly separate the features. This problem should be tackled in further work with more detailed work on features for each class.

Improvements using V4

For instance the entity class which contain the most sub classes. Here the baseline approach (44.2%) is outperformed by the V4 approach (89.1%) by 44.9% in accuracy. After the visual inspection of the results it can be stated that the separate scaling of feature dimensions (each class has just one dimension in each perception layer) works well. The chosen n-gram features for each perception layer are more robust against mistyping of words or other formulations with the same stem or word root. Also the extension and inclusion of SemMap and SynMap (perception layer V3) bring more accurate estimation results.

Fast Extractor Training

Beside the improvements in accuracy, the new V4 approach also brings a flexible and fast possibility to train alternate class taxonomies. The extractor tears the question and the label apart and automatically learns the Bayes model. After the learning step the feature detector returns feature vectors to all phrases that hits the system. With these vectors, a new model can be learned. This approach has a lot similarities to the latest deep learning approaches with the possibility to easily extend the Bayes model with more particular features.

8.3.2. Answer Representation

The experiment on answer representation was already introduced in section 8.2.2. In the experiment the accuracy values as well as other measures show values that can be used as baseline results for further experiments. The detection of facts (accuracy 77.4%), lists (accuracy 89.8%) and paragraphs (accuracy 84.9%) works more than sufficient for the goals of this system. The accuracy values were extracted from the confusion matrix depicted in figure 8.5. The used features seem to be significant enough to tackle the answer representation task accurately. For further experiments more data is needed and an analysis of the significance of features should be done using GLVQ (Generalized Learning Vector Quantization) or similar methods.

8.3.3. Query Generation Algorithm

Algorithm B outperforms algorithm A in both criteria, this was already shown in the experiment in section 8.2.3. This means that 9 of 11 documents reached a higher document score with algorithm B and 7 of 11 documents reached a higher average document score with algorithm B. With this result, the decision to use algorithm B can be made. But, this experiment constitutes as a starting point for further evaluations. The chosen measures reflect the direct scoring of Lucene but not the the final system outcome using both algorithms. For further experiments, two parameters of the experiment need to be modified:

- More data is needed. The used TSDD set contains 11 questions. The problem is the orientation on closed domain, where the indexed data contains a few documents with a few questions. To conclude, the system needs more indexed data and more questions that can be answered using the system.
- Question Answering accuracy, maximum query plan scores and computational cost are more measures that can help to evaluate both algorithms. The current system uses two measures based on Lucene query score for evaluation. In further experiments the mentioned measures need to be applied to the experimental output.

8.3.4. Content Readability Experiments

The results of this experiment, described in section 8.2.4, are very interesting. The Palladian system outperforms JSoup by 36,8% in the best class 'correct'. This is very significant and leads to the conclusion that the algorithm[5] used by Palladian works well. It is based on heuristics about the names for main content layers, text length and link frequency. In contrast to that, JSoup extracts just the largest text value from all DOM elements that are div-layers.

This experiment has shown that Palladian appears to be the better choice for the system proposed in this thesis, because of two reasons:

1. The algorithm is specialized on this task and works with heuristics that are able to estimate, which makes the algorithm more robust in comparison to the algorithm used by JSoup.
2. The results are significant and point out that Palladian is the better choice. For both systems the classes 'Correct' and 'Nearly' are combined, then Palladian covers 95,9% of all documents and JSoup covers 79,4%. Here, the benefit for Palladian is obvious.

8.3.5. GSON Performance

The experiment, described in section 8.2.5, reveals that the current read and write performance, but especially the read performance must be increased for the processing

[5] The algorithm was a former implementation of https://www.readability.com and was used, among others, as plugin for the Mozilla Firefox Readability system.

of a high number of documents (for instance 100 selected documents). Furthermore the file size of an annotated file needs to be decreased to store more data more efficiently.

Performance

The current average duration of a document read (0.127 seconds) can not be scaled out to 100 documents without an acceptable increase of time consumption. Solutions are a smaller structure of objects to speed up the read process or to use the binary JSON (BSON) for this. Binary objects don't need to be parsed from a string, they are already in a computer readable format. Currently, Google GSON doesn't provide a BSON implementation but it is planned for the future. For this further experiment another library can be tested against Google GSON and the JSON versus BSON test of each library can deliver interesting results, too. For the addressed experiments the libraries Boon[6] and Jackson[7] are proposed. JSON was initially considered because of the number of available parsers for a huge number of different languages and thereby the possibilities to work with the different outputs of the system. These outputs can be API results, system states, databases and the annotated documents.

File Size

The file size of an annotated document ends up with 38.3 times more size than the original file. This value is too high and can be a huge factor when scaling out the system. Sure the scaled system should provide more machine power and disk space but the value itself should be optimized in further experiments. To achieve this the use of BSON is proposed. The binary representation can be compressed and the file size can be reduced by method. To minimize the file size not depending from the used method, the annotated data can be distributed with a smart mechanism that stores indexed data in nodes that were loaded during starting up of the system. The remaining sentences of the document can be stored in the document that can be lazy-loaded when needed. Another improvement can be to reconsider the indexed structure and to optimize this structure.

[6] Project page https://github.com/RichardHightower/boon

[7] Project page https://github.com/FasterXML/jackson

8.3.6. Closed Domain Question Answering

As the first experiment that covers and evaluates the complete system performance (question to answer), the results, presented in section 8.2.6, can be seen as a baseline for further experiments. The system responded correctly in 3 of 11 cases (27.27%). In 7 of 11 cases (63.63%) the system response contained the correct answer in the top 5 or top 10 of the top ranked answers. This leads to 10 of 11 cases (90.9%) where the answer was provided by the system directly or indirectly. To improve the quality of the experiment, two conditions must be modified:

- More questions are needed that cover all classes of the Answer Type taxonomy and cover all possible knowledge that the data set contains.
- The number of data files (that are locally indexed) needs to be increased. The more unstructured knowledge exist in the index, the more questions can be formulated.

All in all, the results can be interpreted as a success for the first baseline experiment on a field with no direct competitors. Systems that were previously mentioned in section 1.3 are reaching approximately the same area of quality.

8.3.7. Open Domain Question Answering

Open domain question answering is one of the most challenging tasks in natural language processing. Therefore the experiment, described in section 8.2.7, can be seen as the most important and most complex experiment of this thesis. The complexity comes with the test of all components of the system that were touched during the run through the QA pipeline. Regarding this, the experiments evaluated the 'overall system performance'.

The system found the correct answer in 38 of 157 possible cases (24.2%) and the indirect answer (correct answer exists in top 5 or top 10 ranked answers) in 91 of 157 possible cases (57.96%). This result looks like the overall performance of systems that were formerly introduced in section 1.3. The system can't be compared on a scientific level, caused by the different language they cover and the test sets they are using, but the accuracy of the systems is very similar.

One of the most crucial components, the Answer Type Detector, found 115 of 157 correct answer types (73.25%). Also the Answer Representation classifier found 128 of 157 correct representations (81.53%). These experimental results can be seen as a baseline for further experiments.

Beside these results some problems are mentioned that also belong to an outcome of this experiment:

- Some questions had no corresponding Answer Type in the taxonomy that the system currently uses (Li and Roth [26]). During experiments the question was labeled with the answer type that can be seen as a semantic neighbor of the question.
- The formulation of some questions was not suitable for this system. This led to a reformulation of the questions.
- Some questions showed a too complex structure for the current system. These questions were trimmed to a factoid level or the part after a possible conjunction was pruned.

These problems were converted to tasks and ideas that should be used for further experiments and system design. Among others, these ideas are registered in section 9.2. After the open domain question answering experiment and the corresponding results the comparison to systems, based on English language, should be made. Sure these systems operate on another language, using other search engines as well as semantic networks. And the question of meaningfulness comes into account. How reasonable is a comparison between 'Apples' and 'Äpfel'? The immediate comparison should not be made, but the order of magnitude of the presented results is similar to the systems introduced in section 1.3. This can be seen as the main conclusion of this experiment which serves as a baseline experiment for factoid question answering system based on German language.

9. Conclusion & Further Work

After all experiments were done, this chapter provides the final conclusion as well as some further work ideas, that should be done to improve the performance of the system and helps to evaluate it. Therefore section 9.1 offers the final conclusion after development, testing and evaluating the system in the previous chapters. After the conclusion was drawn, section 9.2 introduces ideas and tasks for the current system, further experiments and ideas for a future system that uses advanced technologies to tackle the question answering task.

9.1. Conclusion

This section provides a roundup of the developed system, the experiments and evaluation that was done in this thesis. The system was designed to be fast, easily serializable, process requests in a serial manner and serve in a client/server environment. An experimental pipeline was created to evaluate all relevant components of the QA system. The acquired data sets helped, to evaluate and test the developed system. These recorded data sets are very helpful for the German QA community to evaluate systems on similar (and thereby comparable) data. To this time there exists, to the best of our knowledge, no system like the one proposed in this thesis. The enrichment of SynMap as well as SemMap can also be promoted and used for further work and projects in the field of Question Answering. As a baseline approach this work provides the first results for Question Answering using unstructured data operating on German language.

9.2. Further Work

This section deals with all things that could not be covered in this thesis, but should be addressed in the future. Improvements on the current state of the system are covered as well as further experiments on the current systems. Furthermore, features for a new or extended system are described. These ideas, suggestions and conceptions result from the development of the current system and from reading state-of-the-art publications that provided promising features and experiments.

9.2.1. Current System Improvements

Here, the improvements based on the current system are described. The system can be, due to its modular structure, extended easily. Some of the improvements, presented below, can be implemented fast, some need a bit more development or research.
These are the suggested improvements:

Parallelization of tasks This feature was already considered during the development of the first version, but regarding to the experimental character and linear execution of the system, postponed to further work. Some tasks, like the normalizers or a lot of parts of text annotation pipeline can be parallelized and pushed together afterwards. Furthermore the search on the system with Lucene can be separated into more than one task.

Robust Web Retriever Beside the use of Palladian for separating the main content of a web page from other uninteresting parts, the system needs an algorithm that segment sentences and words correctly as well as erase unnecessary tokens. This could push the performance of the Open Domain QA system, because the POS tagger will produce more accurate output.

Regular Expressions for Semantics At the moment the most semantic extractors are looking for keywords or the extractor decides with if-else conditions. To be more flexible, regular expressions can be used to find more complex expressions. This is also a good way to reduce code.

Glue Normalization Glues are a part of a question. If a question contains a 'und' word or another conjunction the system should split these glues and treat them separately. For this system no complex questions are assumed, but for further development the existing pipeline could be applied to question glues and with a more complex question, the answer representation can also be more complex. Glue normalization was already considered and partly implemented in the current system.

Improve Answer Type Taxonomy At the moment, the taxonomy from Li and Roth [26][27] is used. This taxonomy should be changed especially when the system will be deployed to a closed domain scenario. The reduction of classes can bring, in most cases, the reduction of complexity.

Lucene Customization Lucene is used to retrieve documents that may contain the right answer. To improve the handling of Lucene, the custom scoring as well as the phrase operator should be established, tested and evaluated. A related system, introduced in section 1.3.4, are currently working with this enhancements of Lucene.

Using Knowledge Bases The approach works on unstructured data, but structured data can be used to find similarities in a semantic network to classify questions and for a richer annotation of documents. If the index of the annotated document shows which sentences contain vehicles, animals, actors and other labels, this could be very powerful and after a semantic answer type detection, a mapping could answer a lot factoid questions. Also the relationship between types can be predicted and correlations can be found. For this, Wikidata[1] is suggested. A similar approach was suggested from Bordes et al. in [3].

From Closed to Open Domain If the system is deployed in a closed-domain scenario, the system should still have the possibility to answer questions to the web and process the retrieved documents. Currently the web request have a high latency, because all documents were indexed on-the-fly which could take more time then an average user wants to wait. Web sources should be considered only in cases where the local document sources fail to find an answer with a sufficient score.

Deeper Analysis and Reasoning Currently, the system passes through the pipelines, described in chapters 4, 5 and 6. This annotation solves a lot of things around the task of Question Answering, but reasoning and deep analysis of the question isn't implemented at the moment. Qanda[32], for instance, generates criteria like temporal restriction, geographical restriction, salient entity, event, answer restrictions and some more. With this, the system can produce a deeper analysis and reasoning which can help to provide better answers.

SemMap with correlations The implemented semantic network, named SemMap (section 3.3.1), contains semantic frames that match given words. Another feature could be to learn the correlations between semantic frames. Some frames significantly often occur together and other frames don't. So with a learned significance

[1] Web page of Wikidata project: https://www.wikidata.org

of correlation the SemMap could be enriched and the system could suggest frames that are most likely to appear. Strong correlation could refer to a frame that often occurs in the same sentence and weak correlations are correlations that appear in the proximity of the specific (pivotal) sentence.

Enrichment of verb database The database that finds verbs with the corresponding tense could be extended using the verbs from the SynMap database (section 3.3.2). The second solution, the other way round, could be to enrich the SynMap with all tenses of verbs and to use the SynMap database also for the verb normalization step.

Integrate verb tense for answer finding Especially for finding correct dates, the tense of the containing verb is a valuable hint to find the correct answer. The current system doesn't support this feature, but it can be crucial when the question requires temporal restrictions.

Rank web documents Because Google (or other search engines) provide just a ranking of documents and no score the system could provide an own ranking. This can be easily done using a cosine similarity and a dictionary approach. Each word and all synonyms, variations and semantic neighbors could have a representative in a n-dimensional vector. Also for each document such a vector is computed. Cosine similarity then computes the affinity of these vectors to each other.

Variation of search engines The current system uses Google search engine with an algorithm that find URLs in the result page. For further development, the use of a Google API or the API from other search engines is recommended.

Thresholding for list elements At the moment, the result list (if the answer representation requires a list) contains 10 elements (not more, not less). A threshold should be established to cut off irrelevant items from the list.

9.2.2. Further Experiments

Basic experiments were done in this thesis to evaluate the most important aspects. For a significant improvement of the overall system and its parts, this section provides experiments that should be done in the future.

Evaluate JSON/BSON libraries In this thesis Google GSON was used to generate JSON files. The experiment of GSON performance leads to the conclusion that a faster library must be found to ensure the performance on a scaled system. Also the possibility to use binary JSON (called BSON) should be evaluated. In some cases this can be a few times faster and the file size can be reduced using compression mechanisms.

Record data for classifiers Because of the limited time and resources the recorded data for the trained models was very sparse. This can be changed in further experiments with the use of other data sources or the help of annotators.

Evaluate TF-IDF relation to answer The term frequency with inverse document frequency (TD-IDF) is a well-known measure in the field of information retrieval. If a term occurs too often in a document, the algorithm ranks down the word. This can hurt the performance of a QA system, when the question requires a phrase, term, an adjective or words that are often unimportant for a search engine or retrieval system. For this, the influence of TF-IDF should be evaluated and furthermore a more accurate measure should be established.

Query Plan Characteristics The query plan contains of parts that need to be found in the text. Each of these parts has a specific weight, which controls the significance of each part in scoring. These weights should be evaluated and the best parameter set should be found. The suggestions for these is to randomly assign each part a weight between $0-1$ and try to optimize this vector regarding to the accuracy of correct questions.

Explore Parameters The configuration file offers the systems parameters. To ensure the best configuration of these parameters further experiments are needed that use variations of the parameters to explore robustness and effect on the systems outcome.

Semantic Extractors The system uses semantic extractors to find named entities or numbers with semantic relations. To find out how important these extractors are, this feature should be tested against a system configuration that works completely without these extractors.

Lucene experiments Lucene was used to find documents that most likely contain the correct answer. For its work, Lucene uses measures like the cosine similarity or the overall Lucene score. An interesting question is, how significant is the correlation between Lucene score and correct answer. The goal could be a scale that can provide a statement of how likely the system find a correct answer according to a given Lucene score.

Correlation of System Indicators The system uses two indicators to express its confidence on the actions - *QC* (question confidence) and *ACC* (answer accuracy). An experiment that evaluates the quality and threshold of these measures could be very important to let the system produce a scale and statement for its actions.

ANN vs. Cosine Similarity The current system uses an ANN to determine answer types and answer representations. The V4 approach delivers a feature vector based on statistical values. Furthermore, the cosine similarity appears to be a fast and often used measure to compare vectors of high dimensional vector spaces. An experiment could evaluate the speed as well as the accuracy measures of both approaches on equal test sets.

9.2.3. Future System Ideas

The current system based on a text annotation pipeline that annotates the text and after that applies all found relations on the previously found documents. Down the road, a system that achieves more accurate results should base on a more complex framework or system, that brings in more cognitive and semantic capacity. The following collection of ideas, rudiments and basic approaches can be applied and implemented on a system, that, based on all implications of the current system, tries to eliminate the weaknesses of the current state.

Semantic Annotation Parser Similar to LogAnswer (introduced in section 1.3.1) the question should be transformed into a semantic network. LogAnswer uses the WOCADI parser to represent the question in MultiNet. To bring this concept to the state-of-the-art, networks like Wikidata, SynMap, SemMap, SALSA or GermaNet should be used. Otherwise a new semantic knowledge base should be implemented to match closed domain scenarios in a more accurate manner.

Agile Answer Type Detection/Taxonomy The taxonomy of the answer type should be more agile. This means, the number of classes should increase and match the classes and types of a semantic knowledge base. The QuASE system (explained in section 1.3.6) uses Freebase to tackle this task, but Wikidata should be the better choice (Freebase was merged into Wikidata). With this agile taxonomy comes more complexity but on the other hand more possibilities and a more complete mapping of possible answer classes.

Semantic Feature Learning In section 3.2 of Sun et al. [41] semantic features are proposed with a significant improvement of overall QA performance. This features, a question term to answer term mapping, are learned in a statistical model that predicates which answer type is likely. This approach can also be used for a few more tasks in this system.

Use of Cognitive Systems A system that brings in cognitive capabilities and simulates different structures of the human brain can be a very important addition for a future QA system. The field of Cognitive Computing and Systems is rapidly growing and active at the moment. A few systems are already proposed, also on QA systems [16]. With the use of a cognitive system like OpenCog[2] the behavioral structure of thinking is changed. Other possible systems are ACT-R[3], PRS[4] or architectural parts of IBM Watson[5].

Integration of a UIMA system Unstructured Information Management represented by systems like Apache UIMA[6] or Catalyst[7] is an important task in Question Answering (or in general handling unstructured data sources). Apache UIMA also allows to spread the data, support the scalability and contains many useful tools. This system could serve as foundation for a successful system, that will be scaled out to a high number of machines.

Handling unconventional questions The current system handles just factoid questions that are formulated in a straight forward manner. An improvement can be to handle questions like '*Was gehört nicht zu X?*' where the possible answers are suggested and the system has to decide between $\{A, B, C, D\}$ as answers. Beside this variations, a few other types of requests are possible.

QA on images An approach that is currently pursued by the computer vision community is the possibility to answer questions on images. Malinowski et al. [29], Antol

[2] Project page: http://opencog.org/

[3] Adaptive Control of Thought - Rational, specialized on memory and rendering of atomic modules of cogitations.

[4] Procedural Reasoning Systems, which follows the belief ⟶ desire ⟶ intention model.

[5] Watson was the first computer that beats a human individual at the famous U.S. television show *Jeopardy!*. The system was developed by Dan Ferrucci as project manager of the IBM team.

[6] Project page: https://uima.apache.org/

[7] Project page: http://catalyst-fp7.eu/

et al. [1], Gao et al. [17] and Ren et al. [36] are all approaches that address this task. Karpathy and Li [25] proposed a system based on deep learning, that generates images descriptions. Together with these approaches a robot (in general a computer program) can learn about its environment and answer questions about it. Also searching on a large image database is a possible use case.

Bibliography

[1] ANTOL, S., AGRAWAL, A., LU, J., MITCHELL, M., BATRA, D., ZITNICK, C. L. and PARIKH, D. VQA: Visual Question Answering. *CoRR*, abs/1505.00468, 2015. URL `http://arxiv.org/abs/1505.00468`.

[2] BANKO, M., BRILL, E., DUMAIS, S. and LIN, J. AskMSR: Question Answering Using the Worldwide Web. In *Proceedings of 2002 AAAI Spring Symposium on Mining Answers from Texts and Knowledge Bases*, 2002.

[3] BORDES, A., WESTON, J.,COLLOBERT, R. and BENGIO, Y. Learning Structured Embeddings of Knowledge Bases. August 2011.

[4] BOS, J. Underspecification, resolution and inference for discourse representation structures. pages 117–124, 2001.

[5] BRANTS, T., HENDRIKS, R., KRAMP, S., KRENN, B., PREIS, C., WOJCIECH, S. and USZKOREIT, H. Das NEGRA-Annotationsschema. Negra project report, Universität des Saarlandes, Computerlinguistik, Saarbrücken, Germany, 1997.

[6] BRILL, E., DUMAIS, S. and BANKO, M. An analysis of the askmsr question-answering system. In *In Proceedings of 2002 Conference on Empirical Methods in Natural Language Processing (EMNLP*, pages 257–264, 2002.

[7] BURGER, J. D. and BAYER, S. MITRE's Qanda at TREC-14.

[8] CAUMANNS, J. A Fast and Simple Stemming Algorithm for German Words. 1999.

[9] CHINCHOR, N. MUC-4 Evaluation Metrics. *in Proc. of the Fourth Message Understanding Conference*, pages 22–29, 1992.

[10] CUCERZAN, S. and SIL, A. The MSR Systems for Entity Linking and Temporal Slot Filling at TAC 2013. In *Text Analysis Conference*, 2013.

[11] CUNNINGHAM, S.J. Dataset cataloging metadata for machine learning applications and research. In *Proc International Artificial Intelligence and Statistics Workshop*, Ft. Lauderdale, Florida, 1997.

[12] ERK, K., KOWALSKI, A. and PINKAL, M. A Corpus Resource for Lexical Semantics. *Proceedings of the Fifth International Workshop on Computational Semantics (IWCS), Tilburg*, 2003.

[13] FARUQUI, M. and PADÓ, S. Training and Evaluating a German Named Entity Recognizer with Semantic Generalization. In *Proceedings of KONVENS 2010*, pages 129–133, Saarbrücken, Germany, 2010.

[14] FELLBAUM, C. *WordNet: An electronic lexical database.* The MIT Press, 1998.

[15] FINKEL, J., GRENAGER, T. and MANNING, C. Incorporating non-local information into information extraction systems by gibbs sampling. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, ACL '05, pages 363–370, Stroudsburg, PA, USA, 2005. Association for Computational Linguistics. doi: 10.3115/1219840.1219885. URL `http://dx.doi.org/10.3115/1219840.1219885`.

[16] FURBACH, U., SCHON, C. and STOLZENBURG, F. Cognitive Systems and Question Answering. *CoRR*, abs/1411.4825, 2014. URL `http://arxiv.org/abs/1411.4825`.

[17] GAO, H., MAO, J., ZHOU, J., HUANG, Z., WANG, L and XU, W. Are You Talking to a Machine? Dataset and Methods for Multilingual Image Question Answering. *CoRR*, abs/1505.05612, 2015. URL `http://arxiv.org/abs/1505.05612`.

[18] GEIRSSON, Ó. P.l. IceQA: Developing a question answering system for Icelandic.

[19] GLÖCKNER, I. and HELBIG, H. LogAnswer - A Deduction-Based Question Answering System. August 2008.

[20] HALL, M., FRANK, E., HOLMES, G., PFAHRINGER, B., REUTEMANN, P., WITTEN, I. The WEKA Data Mining Software: An Update. *ACM SIGKDD explorations newsletter*, 11(1):10–18, 2009.

[21] HAMP, B. and FELDWEG, H. GermaNet - a Lexical-Semantic Net for German. *Proceedings of the ACL workshop Automatic Information Extraction and Building of Lexical Semantic Resources for NLP Applications. Madrid*, 1997.

[22] HARTRUMPF, S. *Hybrid Disambiguation in Natural Language Analysis.* Der Andere Verlag, Osnabrück, 2003.

[23] HELBIG, H. *Knowledge Representation and the Semantics of Natural Language.* Springer Science & Business Media, 2006.

[24] HENRICH, V. and HINRICHS, E. GernEdiT - The GermaNet Editing Tool. *Proceedings of the Seventh Conference on International Language Resources and Evaluation (LREC 2010)*, pages 2228–2235, May 2010.

[25] KARPATHY, A. and LI, F.-F. Deep Visual-Semantic Alignments for Generating Image Descriptions. *CoRR*, abs/1412.2306, 2014. URL `http://arxiv.org/abs/1412.2306`.

[26] LI, X. and ROTH, D. Learning Question Classifiers. In *Proceedings of the 19th International Conference on Computational Linguistics - Volume 1*, COLING '02, pages 1–7, Stroudsburg, PA, USA, 2002. Association for Computational Linguistics. doi: 10.3115/1072228.1072378. URL `http://dx.doi.org/10.3115/1072228.1072378`.

[27] LI, X. and ROTH, D. Learning Question Classifiers: The Role of Semantic Information. *Nat. Lang. Eng.*, 12(3):229–249, September 2006. ISSN 1351-3249. doi: 10.1017/S1351324905003955. URL `http://dx.doi.org/10.1017/S1351324905003955`.

[28] LIN, J. An Exploration of the Principles Underlying Redundancy-based Factoid Question Answering. *ACM Trans. Inf. Syst.*, 25(2), April 2007. ISSN 1046-8188. doi: 10.1145/1229179.1229180. URL `http://doi.acm.org/10.1145/1229179.1229180`.

[29] MALINOWSKI, M., ROHRBACH, M. and FRITZ, M. Ask Your Neurons: A Neural-based Approach to Answering Questions about Images. *CoRR*, abs/1505.01121, 2015. URL `http://arxiv.org/abs/1505.01121`.

[30] MANNING, C. and KLEIN, D. Optimization, maxent models, and conditional estimation without magic. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology: Tutorials - Volume 5*, NAACL-Tutorials '03, pages 8–8, Stroudsburg, PA, USA, 2003. Association for Computational Linguistics. doi: 10.3115/1075168.1075176. URL `http://dx.doi.org/10.3115/1075168.1075176`.

[31] MANNING, C. D., SURDEANU, M., BAUER, J., FINKEL, J. BETHARD, S. J. and MCCLOSKY, D. The Stanford CoreNLP natural language processing toolkit. In *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 55–60. Association for Computational Linguistics, 2014. URL `http://www.aclweb.org/anthology/P/P14/P14-5010`.

[32] MARDIS, S., BURGER, J., ANAND, P., ANDERSON, D., GRIFFITH, J., LIGHT, M., MCHENRY, C., MORGAN, A. and PONTE, J. Qanda and the Catalyst Architecture. In *Proceedings of The Tenth Text REtrieval Conference, TREC 2001, Gaithersburg, Maryland, USA, November 13-16, 2001*, 2001. URL `http://trec.nist.gov/pubs/trec10/papers/MITRE-trecX-1.pdf`.

[33] MOLDOVAN, D., BOWDEN, M. and TATU, M. A temporally-enhanced PowerAnswer in TREC 2006. In *Proceedings of the Fifteenth Text REtrieval Conference, TREC 2006, Gaithersburg, Maryland, USA, November 14-17, 2006*. Proc. of TREC-2006, 2006.

[34] NG, J. P. and KAN, M. Y. QANUS: An Open-source Question-Answering Platform. *CoRR*, abs/1501.00311, 2015. URL `http://arxiv.org/abs/1501.00311`.

[35] PELZER, B., WERNHARD, C. System Description: E-KRHyper. In: Automated Deduction. 2007.

[36] REN, M, KIROS, R. and ZEMEL, R. S. Image Question Answering: A Visual Semantic Embedding Model and a New Dataset. *CoRR*, abs/1505.02074, 2015. URL `http://arxiv.org/abs/1505.02074`.

[37] SAISAS, J. and QUARESMA, .P. The Senso question answering approach to Portuguese QA@CLEF-2007. In *Working Notes for CLEF 2007 Workshop co-located with the 11th European Conference on Digital Libraries (ECDL 2007), Budapest, Hungary, September 19-21, 2007.* Working Notes for the CLEF 2007 Workshops, 2007.

[38] SCHILLER, A., TEUFEL, S., STÖCKERT, C. and THIELEN, C. Vorläufige Guidelines für das Tagging deutscher Textcorpora mit STTS. Technical report, Universität Stuttgart, Institut für maschinelle Sprachverarbeitung, and Seminar für Sprachwissenschaft, Universität Tübingen, 1995.

[39] SCHLAEFER, N., GIESELMANN, P. and SAUTTER, G. The EPHYRA QA System at TREC 2006. 2006. URL `http://www.cs.cmu.edu/~nico/pubs/trec2006_schlaefer.pdf`.

[40] SCHLAEFER, N., KO, J., BETTERIDGE, J., SAUTTER, G., PATHAK, M. and NYBERG, E. Semantic Extensions of the EPHYRA QA System for TREC 2007, 2007.

[41] SUN, H., MA, H., YI, W., TSAI, C., LIU, J. and CHANG, M. Open domain question answering via semantic enrichment. In *Proceedings of the companion publication of the 24th international conference on World Wide Web*. ACM – Association for Computing Machinery, May 2015. URL `http://research.microsoft.com/apps/pubs/default.aspx?id=241399`.

[42] SUTTON, C. and MCCALLUM, A. An Introduction to Conditional Random Fields for Relational Learning. *Introduction to statistical relational learning*, pages 93–128, 2006.

[43] TIWARI, A. and SEKHAR, A. Review Article: Workflow Based Framework for Life Science Informatics. *Comput. Biol. Chem.*, 31(5-6):305–319, October 2007. ISSN 1476-9271. doi: 10.1016/j.compbiolchem.2007.08.009. URL `http://dx.doi.org/10.1016/j.compbiolchem.2007.08.009`.

[44] TOUTANOVA, K. and MANNING, C. D. Enriching the Knowledge Sources Used in a Maximum Entropy Part-of-Speech Tagger. *In Proceedings of the Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora (EMNLP/VLC-2000)*, pages 63–70, 2000.

[45] TOUTANOVA, K., MANNING, C. D. and SINGER, Y. Feature-Rich Part-of-Speech Tagging with a Cyclic Dependency Network. *In Proceedings of HLT-NAACL 2003*, pages 252–259, 2003.

A. Appendix

A.1. Figures

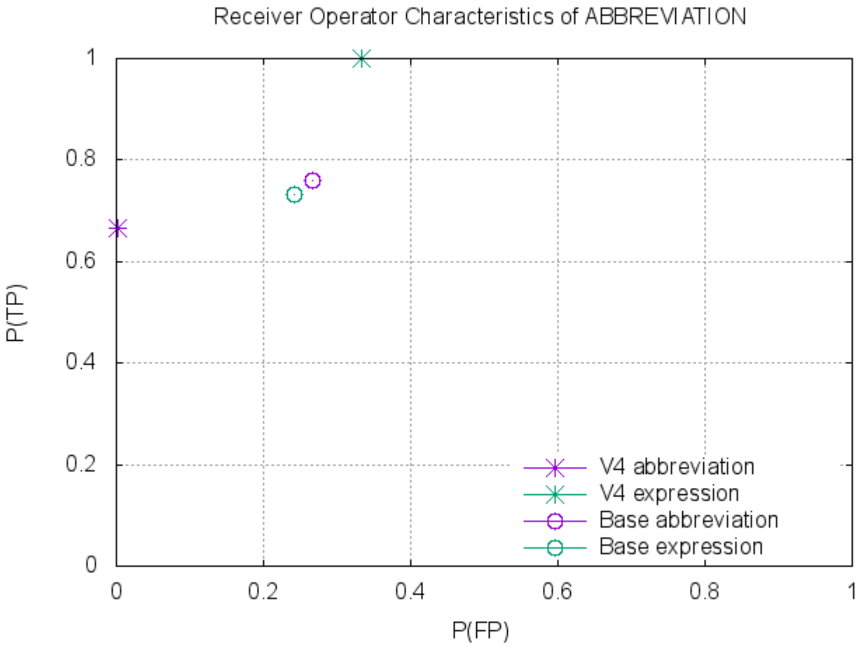


Figure A.1.: ROC space of ABBREVIATION class. Comparison between baseline and V4 classifier.

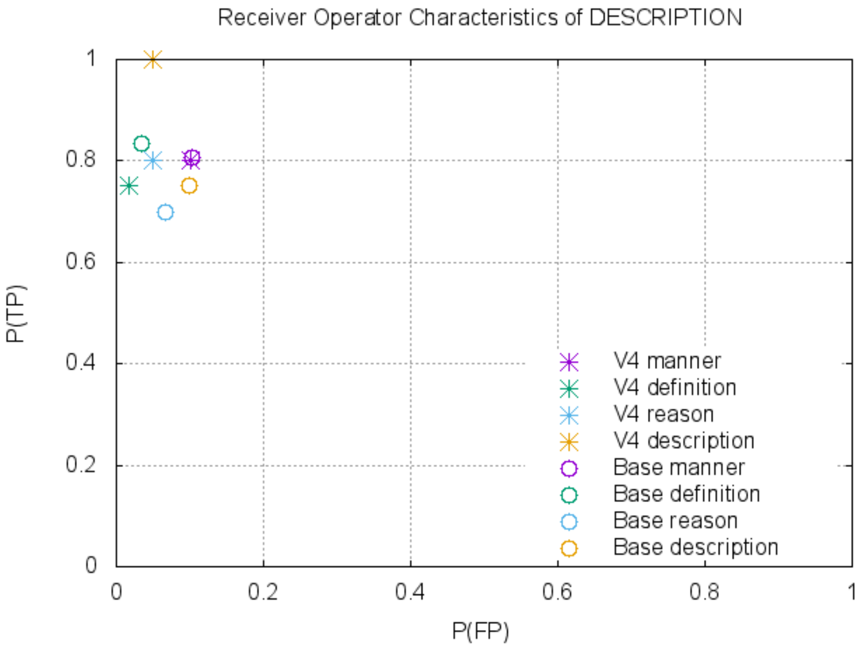


Figure A.2.: ROC space of DESCRIPTION class. Comparison between baseline and V4 classifier.

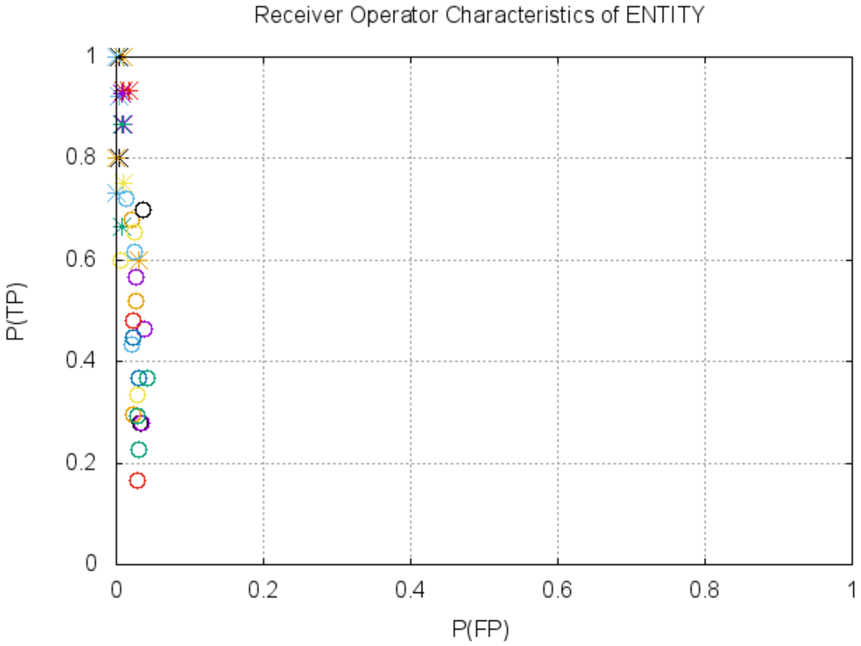


Figure A.3.: ROC space of ENTITY class. Comparison between baseline and V4 classifier. The legend was waived caused by the lack of space. Note that stars depict a V4 result and circles illustrate a baseline result. Same colors correlate to same classes.

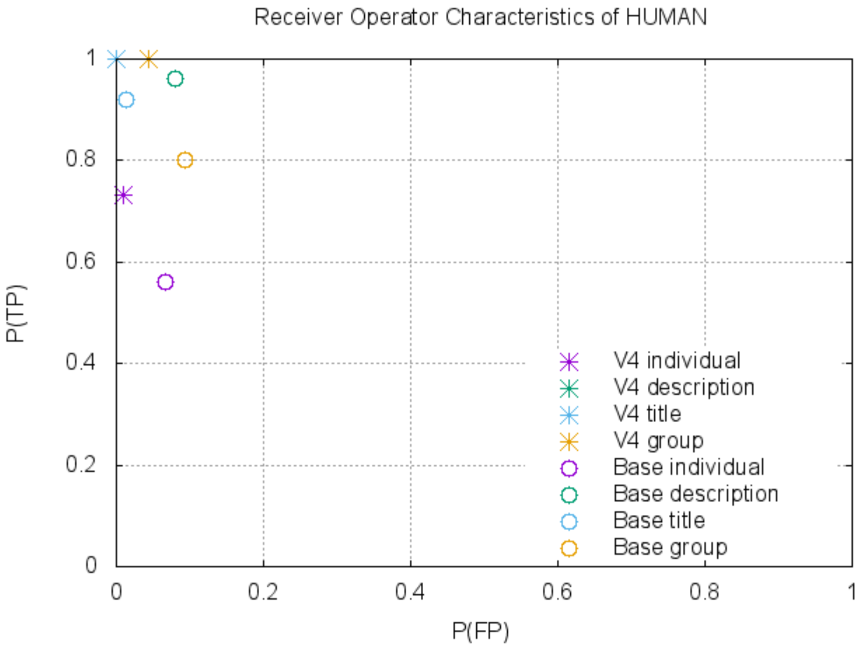


Figure A.4.: ROC space of HUMAN class. Comparison between baseline and V4 classifier.

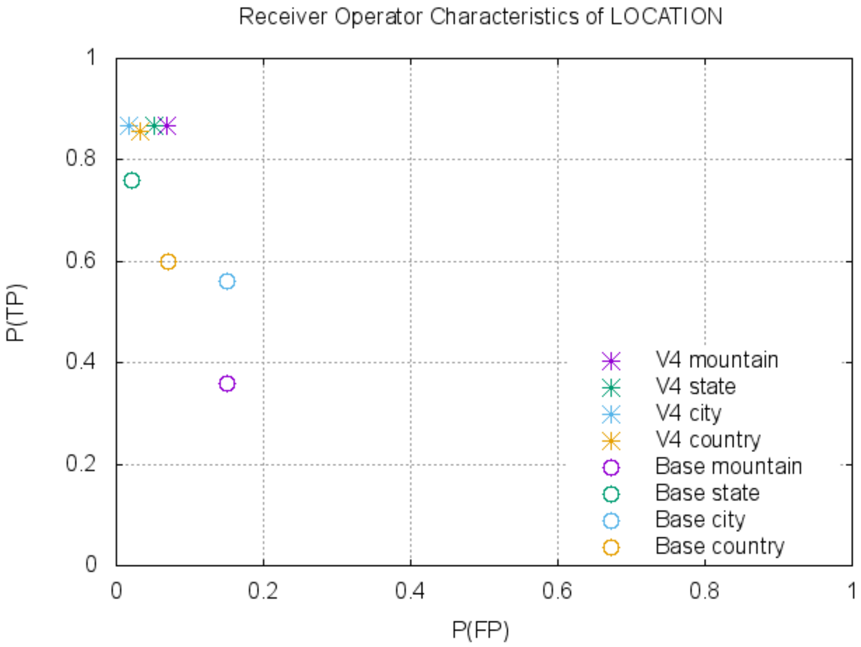


Figure A.5.: ROC space of LOCATION class. Comparison between baseline and V4 classifier.

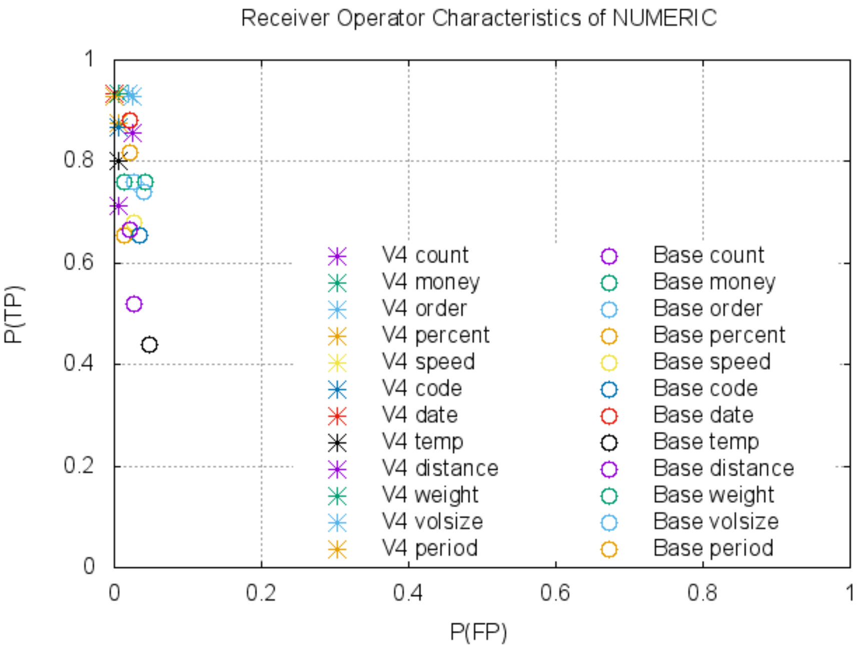


Figure A.6.: ROC space of NUMERIC class. Comparison between baseline and V4 classifier.

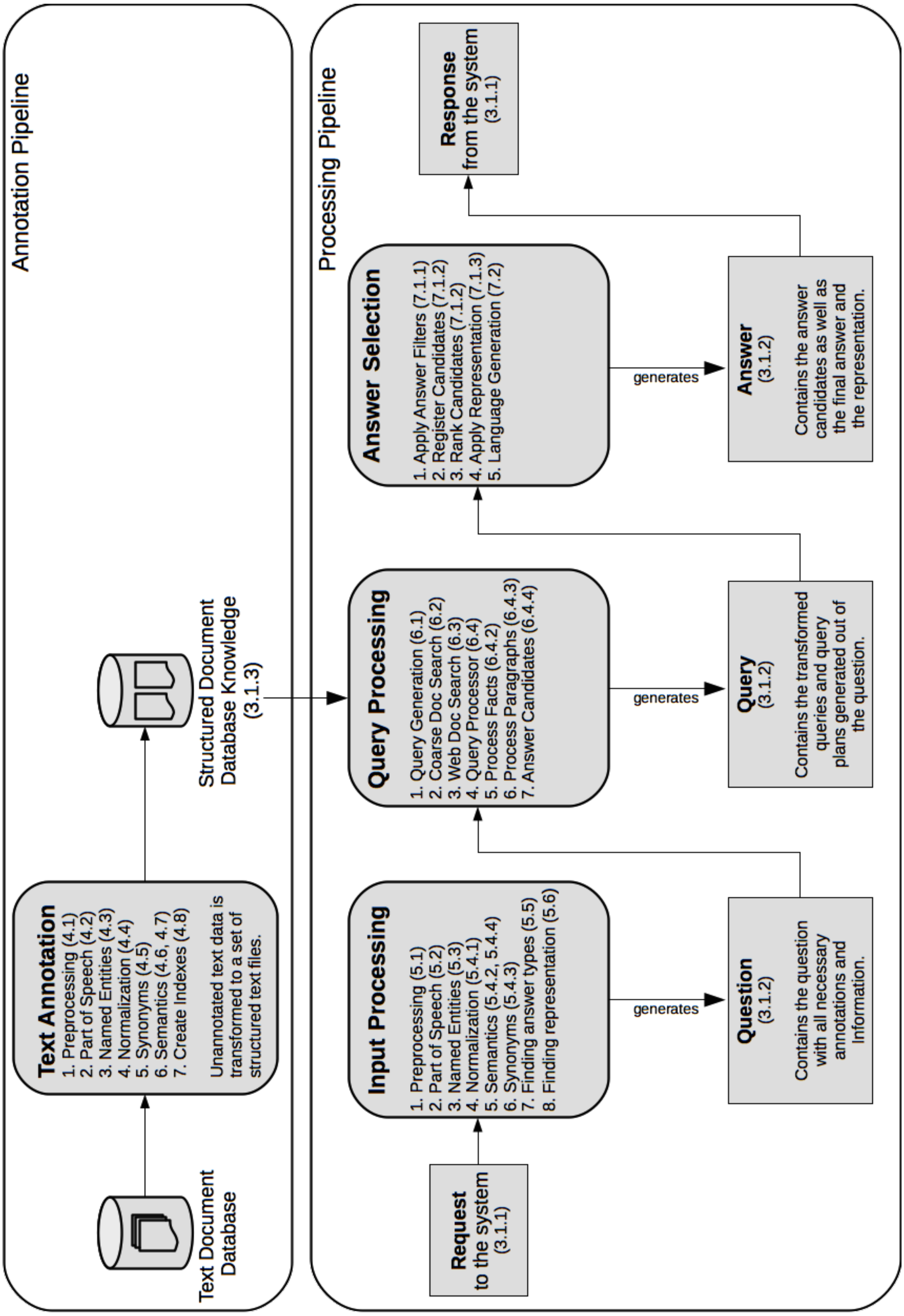

Annotation Pipeline
Text Document Database
Text Annotation
1. Preprocessing (4.1)
2. Part of Speech (4.2)
3. Named Entities (4.3)
4. Normalization (4.4)
5. Synonyms (4.5)
6. Semantics (4.6, 4.7)
7. Create Indexes (4.8)
Unannotated text data is transformed to a set of structured text files.
Structured Document Database Knowledge (3.1.3)
Processing Pipeline
Request to the system (3.1.1)
Input Processing
1. Preprocessing (5.1)
2. Part of Speech (5.2)
3. Named Entities (5.3)
4. Normalization (5.4.1)
5. Semantics (5.4.2, 5.4.4)
6. Synonyms (5.4.3)
7. Finding answer types (5.5)
8. Finding representation (5.6)
generates
Question (3.1.2)
Contains the question with all necessary annotations and information.
Query Processing
1. Query Generation (6.1)
2. Coarse Doc Search (6.2)
3. Web Doc Search (6.3)
4. Query Processor (6.4)
5. Process Facts (6.4.2)
6. Process Paragraphs (6.4.3)
7. Answer Candidates (6.4.4)
generates
Query (3.1.2)
Contains the transformed queries and query plans generated out of the question.
Answer Selection
1. Apply Answer Filters (7.1.1)
2. Register Candidates (7.1.2)
3. Rank Candidates (7.1.2)
4. Apply Representation (7.1.3)
5. Language Generation (7.2)
generates
Answer (3.1.2)
Contains the answer candidates as well as the final answer and the representation.
Response from the system (3.1.1)

Figure A.7.: System overview of the chiQAuery system. The figure shows the annotation pipeline as well as the processing pipeline of the system.

A.2. Tables

Questions classes that asks for...		
ABBREVIATION	ABB_ABBREVIATION	abbreviation of an expression
	ABB_EXPRESSION	Asks for an expression of an abbreviation
DESCRIPTION	DES_DEFINITION	definition of a term or expression
	DES_DESCRIPTION	description of a term or expression
	DES_MANNER	behavior, manner or the ’how is it’
	DES_REASON	reason, cause or the ’why is it’
ENTITY	ENT_ANIMAL	animal, pet or a creature
	ENT_BODY	body parts or the body itself
	ENT_COLOR	colors, tinges or colorings
	ENT_CREATIVE	music, portraits or in general art
	ENT_CURRENCY	currencies and units that describe money
	ENT_MEDICINE	diseases and medicine
	ENT_EVENT	events, meetings and scheduled things
	ENT_FOOD	food, recipes and ingredients
	ENT_INSTRUMENT	instruments or tools
	ENT_LANG	languages and ways of speaking
	ENT_LETTER	single tokens and members of an alphabet
	ENT_OTHER	all entities that are not covered from other classes
	ENT_PLANT	plants, trees and members of the vegetation
	ENT_PRODUCT	products or outcomes of the industry in general
	ENT_RELIGION	spiritual drifts and religions
	ENT_SPORT	sports and physical activities that are related to sport
	ENT_SUBSTANCE	liquid substances that are chemically or biologically created
	ENT_SYMBOL	symbols, heraldic figures and badges
	ENT_TECHNIQUE	techniques, strategies and tactics
	ENT_TERM	significant or prominent term
	ENT_VEHICLE	cars, plains, boats and other things that can move on
	ENT_WORD	signifcant word or a part of a term
HUMAN	HUM_GROUP	organization, company, society or a group
	HUM_INDIVIDUAL	the name of person
	HUM_TITLE	title, working description or rank of a person
	HUM_DESCRIPTION	detailed description (of the life) of a person
LOCATION	LOC_CITY	town, city or municipal
	LOC_COUNTRY	a country
	LOC_MOUNTAIN	rock, mountain or rock mass
	LOC_OTHER	location that is not covered by all other classes
	LOC_STATE	state or federal state that is part of a country
NUMERIC	NUM_CODE	postal code or codes that contain digits
	NUM_COUNT	amount or quantity of something
	NUM_DATE	date or time
	NUM_DISTANCE	distance between entities
	NUM_MONEY	value of money in a certain currency
	NUM_ORDER	ranked result
	NUM_OTHER	numeric values that are not covered by other classes
	NUM_PERIOD	duration or cycle
	NUM_PERCENT	percentage or proportion
	NUM_SPEED	speed value, velocity or acceleration
	NUM_TEMP	temperature or related measures
	NUM_VOLSIZE	volumes, capacities and sizes
	NUM_WEIGHT	weight

Figure A.8.: All classes that are used to classify a question

Tag	Description	Examples
ADJA	attributives Adjektiv	[das] große [Haus]
ADJD	adverbiales oder prädikatives Adjektiv	[er fährt] schnell, [er ist] schnell
ADV	Adverb	schon, bald, doch
APPR	Präposition; Zirkumposition	links in [der Stadt], ohne [mich]
APPRART	Präposition mit Artikel	im [Haus], zur [Sache]
APPO	Postposition	[ihm] zufolge, [der Sache] wegen
APZR	Zirkumposition	rechts [von jetzt] an
ART	bestimmter oder unbestimmter Artikel	der, die, das, ein, eine
CARD	Kardinalzahl	zwei [Männer], [im Jahre] 1994
FM	Fremdsprachliches Material	[Er hat das mit “] A big fish [” übersetzt]
ITJ	Interjektion	mhm, ach, tja
KOUI	unterordnende Konjunktion mit “zu” und Infinitiv	um [zu leben], anstatt [zu fragen]
KOUS	unterordnende Konjunktion mit Satz	weil, dass, damit, wenn, ob
KON	nebenordnende Konjunktion	und, oder, aber
KOKOM	Vergleichskonjunktion	als, wie
NN	normales Nomen	Tisch, Herr, [das] Reisen
NE	Eigennamen	Hans, Hamburg, HSV
PDS	substituierendes Demonstrativpronomen	dieser, jener
PDAT	attribuierendes Demonstrativpronomen	jener [Mensch]
PIS	substituierendes Indefinitpronomen	keiner, viele, man, niemand
PIAT	attribuierendes Indefinitpronomen ohne Determiner	kein [Mensch], irgendein [Glas]
PIDAT	attribuierendes Indefinitpronomen mit Determiner	[ein] wenig [Wasser], [die] beiden [Brüder]
PPER	irreflexives Personalpronomen	ich, er, ihm, mich, dir
PPOSS	substituierendes Possessivpronomen	meins, deiner
PPOSAT	attribuierendes Possessivpronomen	mein [Buch], deine [Mutter]
PRELS	substituierendes Relativpronomen	[der Hund ,] der
PRELAT	attribuierendes Relativpronomen	[der Mann ,] dessen [Hund]
PRF	reflexives Personalpronomen	sich, einander, dich, mir
PWS	substituierendes Interrogativpronomen	wer, was
PWAT	attribuierendes Interrogativpronomen	welche[Farbe], wessen [Hut]
PWAV	adverbiales Interrogativ- oder Relativpronomen	warum, wo, wann, worüber, wobei
PAV	Pronominaladverb	dafür, dabei, deswegen, trotzdem
PTKZU	’zu’ vor Infinitiv	zu [gehen]
PTKNEG	Negationspartikel	nicht
PTKVZ	abgetrennter Verbzusatz	[er kommt] an, [er fährt] Rad
PTKANT	Antwortpartikel	ja, nein, danke, bitte
PTKA	Partikel bei Adjektiv oder Adverb	am [schönsten], zu [schnell]
TRUNC	Kompositions-Erstglied	An- [und Abreise]
VVFIN	finites Verb, voll	[du] gehst, [wir] kommen [an]
VVIMP	Imperativ, voll	komm [!]
VVINF	Infinitiv, voll	gehen, ankommen
VVIZU	Infinitiv mit “zu”, voll	anzukommen, loszulassen
VVPP	Partizip Perfekt, voll	gegangen, angekommen
VAFIN	finites Verb, aux	[du] bist, [wir] werden
VAIMP	Imperativ, aux	sei [ruhig !]
VAINF	Infinitiv, aux	werden, sein
VAPP	Partizip Perfekt, aux	gewesen
VMFIN	finites Verb, modal	dürfen
VMINF	Infinitiv, modal	wollen
VMPP	Partizip Perfekt, modal	gekonnt, [er hat gehen] können
XY	Nichtwort, Sonderzeichen enthaltend	3:7, H2O, D2XW3
$	Komma	,
$.	Satzbeendende Interpunktion	. ? ! ; :
$(	sonstige Satzzeichen; satzintern	- [,]()

Figure A.9.: Overview of STTS tag set used for Part Of Speech tagging

TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
0,913	0,128	0,841	0,913	0,876	0,779	0,958	0,945	ENTITY
0,667	0,009	0,870	0,667	0,755	0,744	0,958	0,763	HUMAN
0,893	0,025	0,919	0,893	0,905	0,876	0,982	0,954	NUMERIC
0,865	0,026	0,790	0,865	0,826	0,806	0,962	0,890	LOCATION
0,725	0,009	0,906	0,725	0,806	0,791	0,968	0,871	DESCRIPTION
0,833	0,009	0,806	0,833	0,820	0,812	0,987	0,895	ABBREVIATION
0,859	0,065	0,863	0,859	0,858	0,805	0,966	0,916	Weighted Avg.

Figure A.10.: Results of the CLASSES classifier

TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
0,667	0,000	1,000	0,667	0,800	0,707	1,000	1,000	ABB_ABBREVIATION
1,000	0,333	0,750	1,000	0,857	0,707	1,000	1,000	ABB_EXPRESSION
0,833	0,167	0,875	0,833	0,829	0,707	1,000	1,000	Weighted Avg.

Figure A.11.: Results of the the ABBREVIATION classifier

TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
0,800	0,100	0,727	0,800	0,762	0,679	0,961	0,911	DES_MANNER
0,750	0,017	0,938	0,750	0,833	0,794	0,990	0,971	DES_DEFINITION
0,800	0,050	0,842	0,800	0,821	0,763	0,977	0,944	DES_REASON
1,000	0,050	0,870	1,000	0,930	0,909	1,000	1,000	DES_DESCRIPTION
0,838	0,054	0,844	0,838	0,836	0,786	0,982	0,957	Weighted Avg.

Figure A.12.: Results of the the DESCRIPTION classifier

TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
1,000	0,000	1,000	1,000	1,000	1,000	1,000	1,000	ENT_MEDICINE
0,000	0,003	0,000	0,000	0,000	0,000	?	?	ENT_OTHER
0,667	0,007	0,833	0,667	0,741	0,734	0,979	0,813	ENT_PRODUCT
0,733	0,000	1,000	0,733	0,846	0,851	0,996	0,944	ENT_TERM
0,600	0,003	0,900	0,600	0,720	0,725	0,964	0,809	ENT_WORD
1,000	0,010	0,833	1,000	0,909	0,908	1,000	0,992	ENT_SYMBOL
0,933	0,010	0,824	0,933	0,875	0,870	0,967	0,939	ENT_ANIMAL
0,933	0,010	0,824	0,933	0,875	0,870	0,998	0,979	ENT_BODY
0,800	0,003	0,923	0,800	0,857	0,853	0,938	0,870	ENT_PLANT
0,867	0,010	0,813	0,867	0,839	0,831	0,994	0,908	ENT_FOOD
0,867	0,007	0,867	0,867	0,867	0,860	0,998	0,965	ENT_COLOR
0,923	0,003	0,923	0,923	0,923	0,920	0,999	0,982	ENT_INSTRUMENT
0,800	0,000	1,000	0,800	0,889	0,890	0,997	0,957	ENT_VEHICLE
1,000	0,000	1,000	1,000	1,000	1,000	1,000	1,000	ENT_LANG
1,000	0,000	1,000	1,000	1,000	1,000	1,000	1,000	ENT_EVENT
0,933	0,017	0,737	0,933	0,824	0,820	0,982	0,890	ENT_SPORT
1,000	0,003	0,938	1,000	0,968	0,967	1,000	1,000	ENT_CURRENCY
0,929	0,007	0,867	0,929	0,897	0,892	0,999	0,984	ENT_CREATIVE
1,000	0,000	1,000	1,000	1,000	1,000	1,000	1,000	ENT_RELIGION
1,000	0,000	1,000	1,000	1,000	1,000	1,000	1,000	ENT_SUBSTANCE
1,000	0,010	0,833	1,000	0,909	0,908	0,999	0,981	ENT_LETTER
0,750	0,010	0,800	0,750	0,774	0,763	0,993	0,905	ENT_TECHNIQUE
0,891	0,005	0,900	0,891	0,890	0,888	0,991	0,948	Weighted Avg.

Figure A.13.: Results of the the ENTITY classifier

TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
0,733	0,000	1,000	0,733	0,846	0,821	1,000	1,000	HUM_INDIVIDUAL
1,000	0,044	0,882	1,000	0,938	0,918	1,000	1,000	HUM_DESCRIPTION
1,000	0,000	1,000	1,000	1,000	1,000	1,000	1,000	HUM_TITLE
1,000	0,044	0,882	1,000	0,938	0,918	0,999	0,996	HUM_GROUP
0,933	0,022	0,941	0,933	0,930	0,914	1,000	0,999	Weighted Avg.

Figure A.14.: Results of the the HUMAN classifier

TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
0,867	0,068	0,765	0,867	0,813	0,763	0,983	0,950	LOC_MOUNTAIN
0,867	0,051	0,813	0,867	0,839	0,797	0,975	0,884	LOC_STATE
0,867	0,017	0,929	0,867	0,897	0,872	0,998	0,991	LOC_CITY
0,733	0,034	0,846	0,733	0,786	0,739	0,959	0,897	LOC_OTHER
0,857	0,033	0,857	0,857	0,857	0,824	0,973	0,934	LOC_COUNTRY
0,838	0,041	0,842	0,838	0,838	0,799	0,978	0,931	Weighted Avg.

Figure A.15.: Results of the the LOCATION classifier

TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
0,714	0,006	0,909	0,714	0,800	0,792	0,987	0,908	NUM_COUNT
0,933	0,000	1,000	0,933	0,966	0,963	0,998	0,986	NUM_MONEY
0,000	0,017	0,000	0,000	0,000	0,000	?	?	NUM_OTHER
0,933	0,019	0,824	0,933	0,875	0,865	0,997	0,977	NUM_ORDER
0,875	0,006	0,933	0,875	0,903	0,895	0,998	0,979	NUM_PERCENT
0,933	0,006	0,933	0,933	0,933	0,927	0,998	0,976	NUM_SPEED
0,867	0,006	0,929	0,867	0,897	0,888	0,999	0,991	NUM_CODE
0,933	0,000	1,000	0,933	0,966	0,963	1,000	0,996	NUM_DATE
0,800	0,006	0,923	0,800	0,857	0,847	0,989	0,945	NUM_TEMP
0,857	0,025	0,750	0,857	0,800	0,784	0,984	0,928	NUM_DISTANCE
0,933	0,006	0,933	0,933	0,933	0,927	0,998	0,981	NUM_WEIGHT
0,929	0,025	0,765	0,929	0,839	0,828	0,982	0,918	NUM_VOLSIZE
0,929	0,000	1,000	0,929	0,963	0,961	1,000	0,995	NUM_PERIOD
0,887	0,009	0,910	0,887	0,895	0,888	0,994	0,966	Weighted Avg.

Figure A.16.: Results of the the NUMERIC classifier

A.3. Acknowledgements

It is a pleasure to thank those who made this thesis possible. At first, I would like to thank Prof. Dr.-Ing. Hans-Joachim Böhme (Thesis Supervisor) for the possibility of writing about a topic that fascinates me from the beginning and let me restless during the whole time of writing. I also thank him for guidance and support from the initial to the final state of this thesis.

I would like to show my gratitude to Dipl.-Ing. (FH) Johannes Fonfara (Supervisor during Research Seminar) who supported me and always took time for my questions and ideas. Thanks to Dr. Sven Hellbach, Dipl.-Inf. (FH) Peter Poschmann and all other colleagues at HTW Dresden (Research Group for Cognitive Robotics) for interesting and valuable talks as well as explanations of relevant topics.

Furthermore, I would like to thank Ariana Lippitsch, Johannes Fonfara, Marilyn Wenk, Andre Presser, Felix Wolff and Hermann Fliegel for the time they took to record data for this work and thereby helped me to develop the system. At least, my respect goes to my family who always supported me during my studies and gave me the freedom to grow.

To put my credit for all of you in one sentence, thank you for the interesting time and the enjoyable collaboration.

B. Derived Theses

The main findings of this master thesis are listed below:

- The encoding of members of a feature vector to a continuous space between $[0, 1]$ increases the possibilities of encoding in comparison to binary features. With this the number of dimensions can be decreased.

- The use of the V4 approach for Answer Type Detection can be seen as more robust against the baseline approach that uses manually engineered features. For that, each layer (word, grammar and semantic) uses n-gram models and Bayesian statistics.

- Lucene and StanfordNLP Core (both as the main parts) can be used to build a Question Answering system for the German Language.

- The basic experiments that measure the overall performance of the system was able to correctly answer 27.27% of the closed domain questions and 24.2% of the open domain questions.

- The created data sets match scientifically approved taxonomies like the Answer Type taxonomy proposed by Li and Roth[26]. Therefore these sets can serve, as the first for German language, as evaluation spot for the scientific community.

- The developed, tested and evaluated system can be extended using the proposed further experiments and ideas from chapter 9.

Statement of Authorship

Ich versichere, dass ich die vorliegende Masterarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Dresden, den 21. Juli 2015

(Alexander Bresk)