VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

INTELIGENTNÍ DOTAZOVÁNÍ WIKIPEDIE PRO MOBILNÍ ZAŘÍZENÍ NA PLATFORMĚ ANDROID

BAKALÁŘSKÁ PRÁCE
BACHELOR‘S THESIS

AUTOR PRÁCE ANDREJ KOVÁČ
AUTHOR

BRNO 2015

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

INTELIGENTNÍ DOTAZOVÁNÍ WIKIPEDIE PRO MOBILNÍ ZAŘÍZENÍ NA PLATFORMĚ ANDROID

QUERY ANSWERING OVER WIKIPEDIA FOR MOBILE DEVICES ON THE ANDROID PLATFORM

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE AUTHOR	ANDREJ KOVÁČ
VEDOUCÍ PRÁCE SUPERVISOR	Doc. RNDr. PAVEL SMRŽ, Ph.D.

BRNO 2015

Abstrakt

Tato bakalářská práce se zabývá vývojem systému pro inteligentní dotazování Wikipedie pro mobilní zařízení s operačním systémem Android. Tato technická zpráva dále popisuje teoretické znalosti úzce související s tématem a dále je popsána implementace serverového systému a klientské aplikace. Část zprávy obsahuje testování výsledného systému a v závěru je nastíněn potencionální budoucí vývoj.

Abstract

This bachelor thesis deals with the development of a system for query answering over Wikipedia for mobile devices running Android operating system. In this technical report theoretical knowledge related to this topic is described as well as the implementation process of a server system and client side application. Part of this thesis is dedicated to testing of the system and in the final part the potential for future development is drafted.

Klíčová slova

Inteligentní dotazování, NLP, python, QANTA, Android

Keywords

Query answering, natural language processing, python, QANTA, Android

Citace

Andrej Kováč: Query Answering over Wikipedia for mobile devices on Android platform, bakalářská práce, Brno, FIT VUT v Brně, rok 2015

Query Answering over Wikipedia for Mobile Devices on the Android Platform

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Doc. RNDr. Pavla Smrže, Ph.D.
Další informace mi poskytl doktorand Mohit Iyyer z Univerzity Maryland
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……………………
Andrej Kováč
20. května 2015

Poděkování

Tímto bych rád poděkoval vedoucímu mé bakalářské práce, panu Doc. RNDr. Pavlu Smržovi, Ph.D., za jeho odbornou pomoc, cenné rady, ochotu a čas, který mi poskytl při tvorbě této práce. Dál bych chtěl poděkovat doktorandovi Mohit Iyyerovi za jeho pomoc při adapadaptování systému QANTA k mojí práci.

Contents

1 Introduction

Wikipedia is the biggest freely available encyclopedia on the Internet, it is used by millions of people around the world every day, with hundreds of articles edited, created and deleted on a daily basis. It is the sixth most widely used website in the world with an estimate of 475 000 monthly unique visitors [**1**] and the largest free information source of the world. This collaborative effort to collect all of humanity's knowledge in one place is undoubtedly becoming more and more intertwined with the way we search for knowledge on the internet, more often than not being the first stop.

Android operating system is the most widely spread OS for mobile devices in the world with 76.6% global market share in Q4 of 2014. [**2**] With vast amount of tutorials and a complete development studio available for this platform it was an easy choice as the target platform.

The goal of this thesis is to develop an application that allows natural language queries over Wikipedia on mobile devices in an intuitive and easy way. Mobile devices are not designed nor powerful enough for handling computationally intensive tasks such as query answering. QANTA a recursive neural network model for paragraph length factoid question answering is adapted to process the natural language queries and a system is built around it that automates the process of extracting data from the Wikipedia database dump and training of the model itself.

This thesis is divided into seven chapters. The second chapter is the introduction into the state of the art, introduction to Natural Language Processing and Machine Learning and QANTA itself. Furthermore existing question answering systems are looked at and analyzed, following is an evaluation of the usability of Google's speech recognition engine.

The third chapter provides an introduction to the employed technologies that were used to implement this system, while in the fourth the design of the server and the android application is described.

Subsequently in the fifth chapter the process of implementation can be found followed by the testing and evaluation in the sixth chapter.

I draw my conclusions in the final, seventh chapter.

2 State of the Art

In this chapter the topic of natural language processing, machine learning and text classification are introduced. Following this a look is taken at existing query answering systems and how they work. Subsequently some tests on Googles speech recognition engine are performed, accuracy is looked at, and then the usability is assessed from the viewpoint of an end user.

2.1 Natural Language Processing

Natural Language Processing (hereinafter NLP) is a field of computer science and artificial intelligence concerned with the interactions between computers and human (natural) languages. NLP is related to the area of human-computer interaction.

Commonly researched tasks from NLP that are related to this work are:

- Query answering: Given a human-language question determine its answer.
- Parsing: Given a human-language sentence convert it into a parse tree (grammatical analysis)
- Speech recognition: Given a human spoken audio recording, accurately return the contents of the recording in text form.

Modern NLP algorithms are based on machine learning.

2.1.1 Machine Learning and Supervised Learning

As defined by Encyclopedia Britannica: "Machine learning, in artificial intelligence (a subject within computer science), discipline concerned with the implementation of computer software that can learn autonomously. Expert systems and data mining programs are the most common applications for improving algorithms through the use of machine learning. Among the most common approaches are the use of artificial neural networks." [**3**]

Taking a look at a typical scenario of machine learning, we have an outcome measurement that we wish to predict based on a set of features. This outcome is usually quantitative (e.g. predict how well a stock is going to do on the stock-market) or categorical (e.g. recognize which number is written on a paper). Furthermore we have a set of data (the training set) in which we observe the outcome, and feature measurement for a set of objects. Using this data we can build a prediction model that is able to predict the outcome for previously unseen data. Based on if the model is "shown" the desired (correct) output for the data in the training set we can divide machine learning into un-supervised and supervised learning. In supervised learning the desired outcome (label, value) is paired with the input data and it is used to guide the learning process. [**4**]

The task of choosing the correct label for an input is called classification, supervised classification if the desired output is paired with the input. Each input is considered in isolation from all other inputs and the set of labels is defined in advance. During training, a feature extractor is used to convert each input to a feature set, which captures the basic information about given input that should be used to classify it. After this pairs of feature sets and labels are fed into the machine learning algorithm to generate a model. During prediction the same feature extractor is used to convert unseen inputs to feature sets. These feature sets are then fed into the model, which generates predicted labels. [**5**]

The framework of this process can be seen on Figure 1.

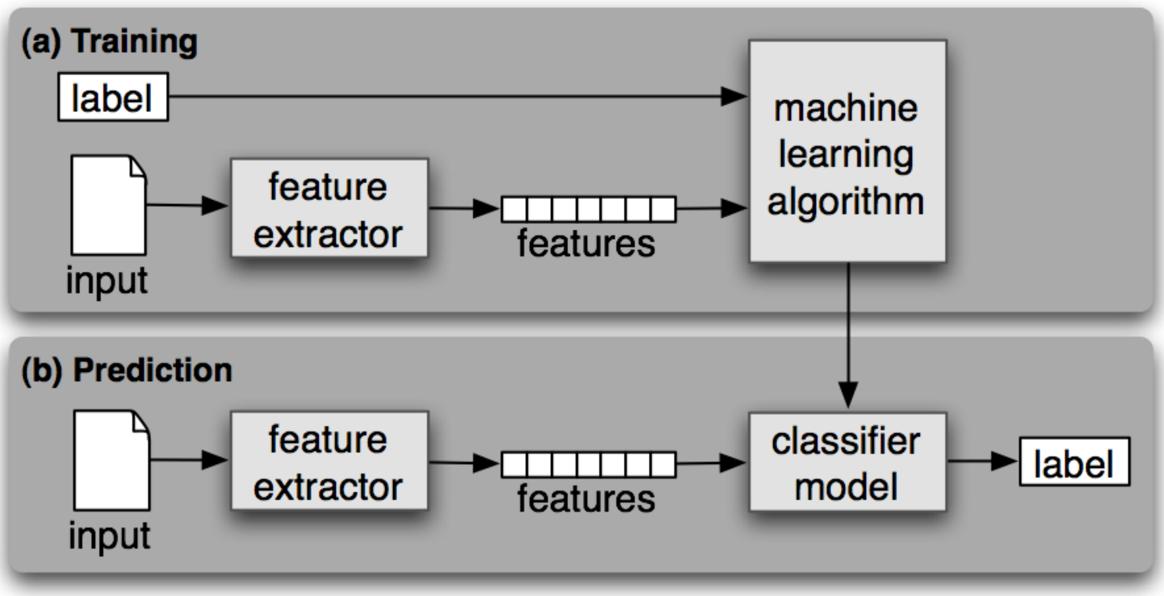


Figure 1 Supervised classification[1]

The first step in creating a classifier is deciding which features of the input should be used to represent it and how to encode these features. Next the training corpus and the division of said corpus needs to be decided upon. [**5**]

"A corpus can be defined as a collection of machine-readable authentic texts (including transcripts of spoken data) which is sampled to be representative of a particular natural language or language variety. Corpora provide a material basis and a test bed for building NLP systems." [**6**]

First a development set has to be chosen, containing the corpus data for creating the model, this development set is then subdivided into training and dev-test sets. As the name would suggest the training set is used to train the model while the dev-test set is used to perform error analysis and tuning of the model. [**5**]

It is important that the set of labels is the same for both the train and dev-test. If we were to have to use a dev-test set where provided label is not part of the set of labels for the training set, the

[1] Taken from: http://www.nltk.org/book/ch06.html

model trained on the training set would never be able to predict that label correctly and the tuning of the model could (most likely would) deform the model.

Once the dev-test was used to tune the model, it no-longer provides an accurate representation of how well the model would perform on new (previously unseen) inputs, which is why keeping a set of unused, separate data for testing is recommended. This test set should provide more reliable information about the accuracy of the model. [**5**]

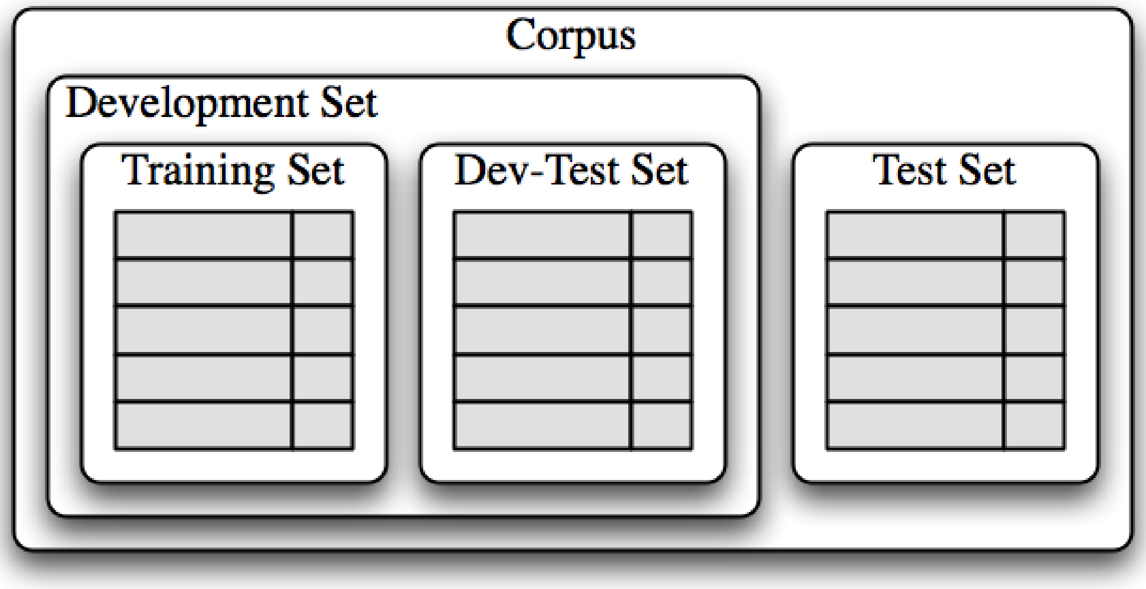


Figure 2 Example of dividing corpus[2]

2.2 Query Answering

This section was adopted from [**7**].

Traditional query answering (hereinafter QA/S for systems) systems generally cover a limited number of corpus, implement a single processing stream or technique that performs three standard steps sequentially question analysis, search and answer selection. These systems utilize a limited number of techniques and therefore are heavy limited when it comes to answering complex queries posed by the user.

Recently, many practical systems have been developed, that tackle this issue, and these implementations vary in many ways, either by employing additional processing modules or using multiple data sources or providing multimedia content-rich answers and so on, these are the so called 'versatile' QA systems. Compared to the traditional QA systems, these systems either provide enhanced answers (with multimedia) or enable the system to answer questions that were previously impossible to answer.

Based on which of the issues a query answering system primarily focuses on dealing with we can divide them into following categories:

[2] Taken from: http://www.nltk.org/book/ch06.html

- Multimedia QAS: Multimedia QAS refers to QA systems that provide textual and multimedia contents such as images, sounds, video and so on as answers to queries, while the query itself is in textual form only.
- Multimodal QAS: Multimodal QAS employ multiple modalities at both question and answer level. This is a superset of multimedia QAS, the difference being that these systems accept multimedia inputs as opposed to only textual.
- Multi-strategy QAS: These systems utilize multiple strategies to extract answer to any question.
- Multi-source QAS: Multi-source question answering systems utilize multiple kinds of databases as corpus for finding the answers, these systems provide uniform access to diverse knowledge resources.
- Multi-stream (or agent) QAS: A stream (or agent) is a question answering system on its own, thus multi-stream QA systems employ multiple question answering streams for finding the answer to the questions. Each of the streams can possibly use different answering strategy or knowledge sources and produces a ranked list of answer candidates.
- Multi-lingual QAS and cross-lingual QAS: Systems that can work on and interact with users in multiple languages.
- Multiple choice QAS: Where the system has to answer questions from a set of provided answers.
- Multi-sentence QAS: Multiple sentence questions as the name suggests consist of multiple sentences, multi-sentence QAS answer these kinds of questions.
- Multi-layer QAS: Questions are iteratively decomposed into a set of simpler questions that can be directly answered using the document corpus.
- Multi-perspective QAS targets opinion based questions.
- Multi-focus QAS: Question focus is the information expectations expressed by the question. Multi-focus QAS answers questions containing multiple foci.
- Multi-parser QAS: Different types of parsers are employed on different types of query data.
- Multi-document comparison QAS: Answer are constructed from parts taken from multiple documents.
- Multi-search engine QAS: Utilizes existing search engines for answer extraction.
- Multi-channel QAS: Provides multiple channels of communication or presentation of answer to the user.

Each of these systems has their own advantages and addresses and resolves a particular bottleneck of traditional systems.

Based on the scope from where the QA system is able to answer a question we can divide QAS into open and closed domain question answering systems, where closed domain QAS answer questions from a closed domain such as literature, while in open domain question answering the domain is not restricted and the questions can be about nearly anything.

2.3 QANTA[3]

This section is adopted from [**8**].

A Neural Network for factoid question answering over paragraphs by Mohit Iyyer, Jordan Boyd-Graber, Leonardo Claudino, Richard Socher, and Hal Daumé III.

The model was designed and tested using Quiz bowl questions. Quiz bowl is a trivia competition where questions consist of four to six sentences. Sentences are revealed to the players one by one, each sentence containing clues to the answer, that help uniquely identify the answer even without the context of previous sentences. Competitors may answer at any time, the more sentences are revealed the lower the score is awarded.

Like many other NLP neural networks QANTA uses the vector space model to learn representations of individual questions. These questions are then mapped to their labels, which in this case are the answers for the questions. Given that there is a limited number of possible answers this task can be viewed as multi-class classification task.

To compute distributed representations for individual questions, QANTA uses a dependency-tree recursive neural network (DT-RNN). These representations are then aggregated and fed into a multinomial logistic regression classifier, where the class labels are the answers associated with each question instance. As in other RNN, the work begins by associating each word w in the vocabulary with a vector representation. These vectors are stored as the columns of a $d\,x\,V$ dimensional word embedding matrix, where V is the size of the vocabulary. The model takes dependency parse trees of question sentences and their corresponding answers as input.

Intuitively, the goal is to encourage the vectors of question sentences to be near their correct answers and far away from incorrect answers. Given a sentence paired with the correct answer, a random set of answers is selected from the set of all incorrect answers, this set is then used with the hidden vector representation to calculate the rank of the correct answer in respect to the set of incorrect questions, because the calculation of the rank is expensive, an approximation is made by sampling random incorrect answers.

The exact calculation used in QANTA is outside the scope of this thesis and can be found in the corresponding paper. Important to note though is that the system considers each sentence independently but in case of multi-sentence questions it is able to learn from multiple sentences, this is done by averaging the representation of sentences seen in a question. This averaged DT-RNN model is called QANTA: a question answering neural network with trans-sentential averaging.

[3] Available at: http://cs.umd.edu/~miyyer/qblearn/

2.4 Existing Systems

In this section existing systems will be looked at how they work and what issues they solve, while looking into the underlying system and real life applications.

2.4.1 Watson

IBM's supercomputer capable of answering questions posed in natural language was initially developed by answering questions of Jeopardy. Later the application of this computer was far extended beyond this scope into medical, financial, educational and other areas to provide ranked answers with sources to assist humans in decision making. [**9**]

The computer boosts 15 terabytes of memory, 2880 processing cores to handle the deep QA architecture, which uses NLP, information retrieval (IR), machine learning, knowledge representation and reasoning to advance the science of automatic question answering. [**9**]

Watson successfully and in a dominating fashion beat the best human players in Jeopardy [**10**] using a self-contained knowledge base (no Internet access) just like humans have during a competition [**11**].

Before entering this battle of wits Watson had to learn from a wide variety of documents to be able to answer open-domain like questions that are present in Jeopardy. The first step in answering a question is then to analyze a question and parse it into parts of speech format and identifying the different roles the words and phrases in the sentence are playing. Using this process Watson tries to reason what type of question is asked and what type is the expected answer. For each of the interpretations of a question Watson search through millions of documents to find possible answers, weighing quantity over quality. Afterwards obviously wrong answers are eliminated, and Watson searches for further evidence to support the right answer. To do this it searches through passages to find evidence for or against possible answers to further eliminate wrong answers from the equation. In the end it returns the weighed prediction to the answer with proof. [**11**]

The real life applications of Watson in medicine show the potential that deep learning and query answering techniques have in assisting humans, Watson is able to study millions of papers from the field of medicine while examining and comparing the medical history of patients to provide possible cures and countermeasures. Watson is a prime example of what the future might hold.

2.4.2 START

START was the world's first Web-based QA system. [**12**] Start uses natural language annotations, which are machine parsable sentences and phrases that describe the content. This metadata is used to identify what type of questions a particular piece of knowledge is useable to answer. Various NLP tools are used to compare a user's query with these annotations stored in a knowledge-base. Omnibase a virtual database is used to provide start with a uniform abstraction layer web knowledge sources. [**7**]

Freely available and accessible from the Web, START represents the past, when machine learning and recursive neural networks, were less widespread and too computationally expensive to be applicable. The obvious negatives of this system are apparent on the first try as it takes on the task of answering open-domain queries without the use of RNN techniques it is fairly limited and cannot answer difficult questions that are present in Jeopardy. Nonetheless it provides an interesting contrast to other implementations on this list.

2.4.3 Google Now

Google Now is Google's "personal assistant" implementation for the Android platform, very similar to the well-known Siri, which is Apple's personal assistant for iOS devices. With the available resources Google has at its disposable in terms of resources we can expect a lot from this application. Queries can be entered by saying "Ok, Google" followed by the question or simply clicking the microphone icon on the home screen search widget.

Google Now also uses neural network(s) to answer queries, but it also leverages an enormous data structure called the Knowledge Graph. It is used to learn about entities and how they connect to each other, how relevant they are in a given context. Google does not simply aim to answer queries posed by the user, the aim is to answer queries before they are even made. [**13**]

Google Now is able to answer simpler queries such as "who painted Guernica" or "who coined the phrase 15 minutes of fame", though it is more designed to answer questions the like of "where is the nearest restaurant". It has no doubt a wide a variety of usages and aims to tackle the problem of query answering from a unique perspective.

2.5 Speech recognition and its accuracy

Speech recognition is the translation of spoken words into text. In the recent years we have seen many products developed that use voice control such as music players, television, cars and many others. Google has been on the forefront of the efforts to make speech recognition more accurate. It is another one of the fields of natural language processing that has been using machine learning to improve the accuracy.

Before designing the application with speech-to-text as its primary method of input, it was important to assess how accurate the translation process is and whether or not it is something that comes handy in many situations or is a useless frustrating marketing gimmick. In order to test the accuracy a corpus had to be collected first, a set of sentences which would be spoken into the microphone of an Android device by me. A set of Jeopardy quiz questions were used as the input sentences that are available online.

Each sentence was input 5 times to eliminate variations (such as unstable internet, outside noise etc.) and to see if the engine is capable of learning based on previous inputs. It is important to keep in mind that I am not a native English speaker and while I cannot hear it when talking I no doubt have a Middle/Eastern-European accent when talking.

The input was done using the Speech Engine API using a dummy application that simply output the result of the translation. The main inaccuracies in converting speech to text arose from either similar sounding words, names or long input sentences. Words such as called and cold, word and verb were confused multiple times.

Well known named distinct sounding named entities such as Vermont were recognized with a very high or complete accuracy, while other named entities such as Lake Vostok were never recognized and I was unable to input them using speech to text functionality. Some longer input sentences were converted with no problem, while with others only part of the sentence was output or the output sentence was completely different from the input sentence. Punctuation of course is not output by the speech engine.

Examples:

- Input: Lake Vostok is the largest lake here
- Outputs:
  - linguascope is the largest lake here
  - linguascope is the largest lake here
  - linguascope is the largest lake here
  - little star is the largest lake here
  - like what is the largest lake here
- Input: Cross country skiing is sometimes referred to by these two letters the same ones used to denote 90 in Roman numerals
- Outputs:
  - cross country skiing is sometimes referred to by these two letter the same ones used to clean out 90 in roman numerals
  - cross country skiing is sometimes referred to by these two letters the same ones used to denote 90 in roman numerals
  - cross country skiing is sometimes referred to by these two letter the same ones you studying all day 19 in roman numerals
  - cross country skiing is sometimes referred to by these two letters the same ones used to denote 90 in roman numerals

  - cross country skiing is sometimes referred to by these two letters the same as you study now to 19 in roman numerals

As seen in the examples there is no direct correlation between the length of the inputs and the accuracy of outputs and in some cases longer inputs are handled easily and output accurately.

In conclusion I found using the speech input ranging from painfully frustrating to incredibly useful and easy, the main advantage being the speed and the elimination of need to pay attention to the screen. Still the speech engine does not work as accurately as it would be needed or at least wanted from an application using only speech input, but it is certainly useful as the preferred option.

3 Employed Technologies

In this chapter the reader can find the description and functionality of the technologies used in implementing the system that are of outside source and not the work of the author of this thesis, but are necessary for full functionality.

3.1 ElasticSearch[4]

For storing and indexing of the parsed Wikipedia database dump ElasticSearch is used which is built on Apache Lucene. It is a flexible and distributed open-source full-text information retrieval engine. It allows both full-text and structured search, analytics and all three in combination. For document representation it uses the JSON format and it aims to make full-text search easy by hiding the complexities of Lucene behind a simple, coherent, RESTful API. Thanks to its real-time response time, reliability and availability of a Python API, ElasticSearch is a well-equipped solution for backend use in this project.

3.2 Android OS

Android is a mobile operating system based on the Linux kernel and currently developed by Google. Android was unveiled in 2007, along with the founding of the Open Handset Alliance - a consortium of hardware, software, and telecommunication companies devoted to advancing open standards for mobile devices. [**14**]

Nowadays you would have to look very hard to find anyone who has not used or at least seen a device running Android OS, it has been deployed across many platforms and markets with various degree of success. Electronic products using the “Smart” prefix are most likely built on Android, Smart TVs, Smart Watches, Smart Cars and of course Smartphones just to name the most well-known areas where Android has significant market share. It has found its way onto gaming consoles, video cameras and even fridges [**15**]. These areas might all deploy different ways of interacting with the user, for Smartphones and Tablets the interface is based on direct manipulation and touch input [**16**].

Android apps are written in the Java programming language. The Android SDK tools compile code – along with any data and resource files into an APK: an Android package, which is an archive file with the .apk extension. One APK file contains all the contents of an Android app and is the file that Android-powered devices use to install the app. [**17**]

[4] Available at: https://www.elastic.co/products/elasticsearch

3.3 Android Studio[5]

As part of Googles continuous efforts to make development for Android easier and more accessible the Android Studio IDE was released.

Android Studio is now the official IDE for Android application development, it is built on IntelliJ IDEA the popular IDE for java by JetBrains and is available across all three mainstream platforms (Windows, Linux, Mac OS X). Android studio boosts built in support for Android Virtual Devices as well as an Android Device Monitor, SDK manager and other useful tools such as built-in integration, intelligent code editor and code templates, all free of charge.

3.4 Word2vec[6]

This tool provides an efficient implementation of the continuous bag-of-words and skip-gram architectures for computing vector representations of words. The word2vec tool takes a text corpus as input and produces the word vectors as output. It first constructs a vocabulary from the training text data and then learns vector representation of words.

Word2vec is used to initialize the data fed into QANTA, this should significantly help with both accuracy and training time.

3.5 Stanford Core-NLP[7]

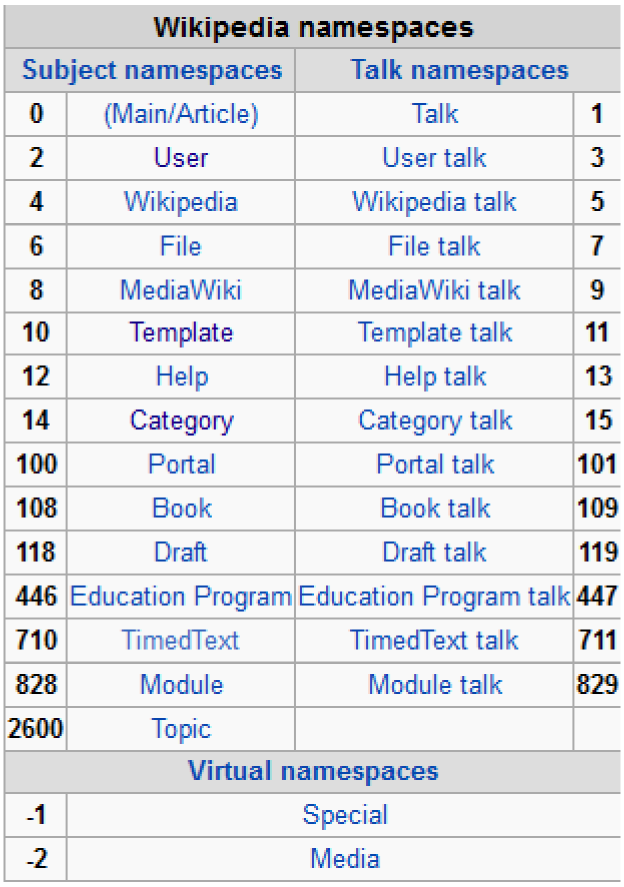
“Stanford CoreNLP provides a set of natural language analysis tools which can take raw text input and give the base forms of words, their parts of speech, normalize dates, times, and numeric quantities, and mark up the structure of sentences in terms of phrases and word dependencies, indicate which noun phrases refer to the same entities, indicate sentiment, etc. Stanford CoreNLP is an integrated framework. Its goal is to make it very easy to apply a bunch of linguistic analysis tools to a piece of text.” [**18**]

Neural Network Dependency Parser (NNDP) will be employed, to replace the PCFG Parser originally used by QANTA. This NNDP provides substantial performance boost as opposed to the PCFG parser. Upon usage of these parsers a speed increase from 6-7 sentences/second to 300-400 sentences/second was observed.

[5] Available at: https://developer.android.com/sdk/index.html
[6] Available at: https://code.google.com/p/word2vec/
[7] Available at: http://nlp.stanford.edu/software/corenlp.shtml

3.6 WikiExtractor[8]

For extracting information and data into plain text format with html markup for headers and URLs from the Wikipedia database dump WikiExtractor is used. It is a parser for parsing the Wikipedia database dump into clean text format, it is a script written in python that requires no additional library to function and aims at high accuracy in the extraction task. It provides the functionality to only parse selected namespaces, namespace in Wikipedia is a set of Wikipedia pages whose names begin with a particular reserved word. For this task the only relevant namespace is the 0th (Main) namespace which contains the articles and pages.



Wikipedia namespaces			
Subject namespaces		Talk namespaces	
0	(Main/Article)	Talk	1
2	User	User talk	3
4	Wikipedia	Wikipedia talk	5
6	File	File talk	7
8	MediaWiki	MediaWiki talk	9
10	Template	Template talk	11
12	Help	Help talk	13
14	Category	Category talk	15
100	Portal	Portal talk	101
108	Book	Book talk	109
118	Draft	Draft talk	119
446	Education Program	Education Program talk	447
710	TimedText	TimedText talk	711
828	Module	Module talk	829
2600	Topic		
Virtual namespaces			
-1	Special		
-2	Media		

Figure 3 Wikipedia Namespaces[9]

3.7 Python 2.7

Python is one of the most popular programming languages of the recent years, with strong community backing and extensive libraries. It is an interpreted, object-oriented, dynamically typed language. Python is freely available for all three major platforms and can be freely distributed free of charge. [**19**]

[8] Available at: http://medialab.di.unipi.it/wiki/Wikipedia_Extractor
[9] Taken from: http://en.wikipedia.org/wiki/Wikipedia:Namespace

Being a general purpose language it can be used to various tasks as it offers high level data structures. QANTA is also written in Python 2.7 what made this language the go-to language for this project.

One of the biggest advantage of python over other languages is the vast amount of packages available. These libraries include the Natural Language Toolkit for NLP tasks, which also comes with a book written by the authors of the package. The NumPy package for scientific computing and Scikit-learn for machine learning.

4 Design

In this chapter a concept for the system is drafted, starting with the server side and then moving on to the application itself.

4.1 Server

Taking into consideration that Wikipedia database dumps are created once a month [**20**] the server must be able to refresh itself when a new dump is created. This task must not interfere with the user's ability to use the application, it must not shut the server down nor cause any significant delays in serving the incoming requests. Since there is no way to guarantee a hundred percent uptime for the server re-launching it should be as easy and effortless as possible while giving options to configure some of the more important aspects without having to look at or modify the source code.

The option is provided to define a separate folder for the Wikipedia dump (hereinafter *wikifolder*), the name of the data folders (hereinafter *name*) and the name of the index for ElasticSearch (hereinafter *elastic)*.

The first step after starting the server is checking if a model is already trained, this is useful for recovering from shutdown as it significantly decreases startup time by skipping the training. If the model has not yet been trained, first the latest Wikipedia dump is downloaded if it is not available in the *wikifolder*. The following step is parsing the dump, the target directory is a new directory inside *wikifolder* named *name*. Subsequently the data extraction process is initiated.

The extraction process starts with extracting the articles which are then indexed using ElasticSearch to the index *elastic*. The development set is extracted from these articles and the testing data from a ~200 000 question dataset form the quiz competition called Jeopardy. Following this the datasets are parsed using the NNDP parser and converted to dependency tree format. After this the training process begins, upon completion of which a concurrent TCP server is started that serves the incoming requests from clients and periodically checks if a new a version of the Wikipedia dump has been released. Detecting a new version of the dump triggers an update cycle which runs parallel to the server.

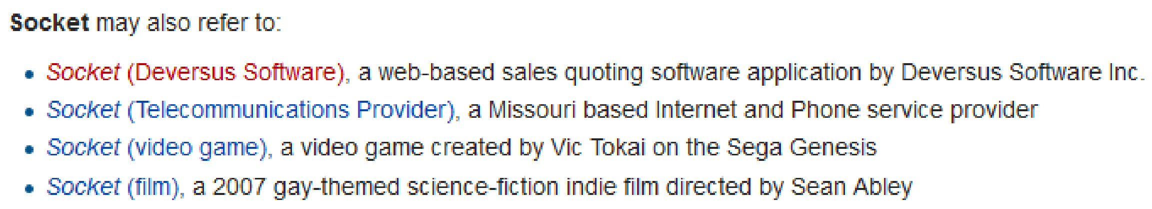
For the Flow Chart see the 1. Appendix.

4.2 Data Preparation

The contents of Wikipedia articles are mapped to their titles by treating each sentence in the article as a question sentence and the title as the answer. Based on the recommendation of Mohit Iyyer author of QANTA, we treat answers as a single entity, to do this the spaces in the page titles of Wikipedia are replaced with underscore (e.g. Pablo Picasso becomes Pablo_Picasso). To further increase accuracy vocabulary is reduced by converting the corpus to lower case.

WikiExtractor is used with section and link switches to keep sections and hyperlink annotations in the text. This output is used for answering queries and is indexed using ElasticSearch before being processed. The code of WikiExtractor was modified to use HTML header markup for section titles instead of plain text, this helps with term frequency as the section titles such as „Early Life“ or „History“ are often used across many Wikipedia pages and also makes it easier to discard this data while keeping the result delivered to the application formatted. Answer occurrences in an article that are not the title of the current article are replaced with the underscored markup. Hyperlinks in the articles always refer to another wiki article, as the *External Links* have a separate, dedicated section. A hyperlink example from a parsed wiki page is as follows *<a href="pop art">pop art>*, which is a link that refers to the page Pop Art, using regular expressions links are matched and replaced with the underscored URL part of the link (*pop_art*).

Due to the nature of Wikipedia's naming scheme we are unable to follow the same methodology for answer occurrences, as can be seen on Figure 4 page titles are often suffixed with parentheses that contain additional information. Given the example below if we were to find the word "socket" in an article we would not be able to tell which of the four possible answers should be inserted.


Socket may also refer to:

- *Socket* (Deversus Software), a web-based sales quoting software application by Deversus Software Inc.
- *Socket* (Telecommunications Provider), a Missouri based Internet and Phone service provider
- *Socket* (video game), a video game created by Vic Tokai on the Sega Genesis
- *Socket* (film), a 2007 gay-themed science-fiction indie film directed by Sean Abley

Figure 4 Socket Multiple Entries[10]

The corpus is then sentence tokenized using NLTK library's sentence tokenizer. Each sentence is checked and skipped if consists of only stop words as stop words are not used for the training process of QANTA and the empty sentence would cause errors. At the end of each sentence that passes the check the HTML markup for line break *<br>* is appended as it is used as the sentence delimiter in the parsing process. These sentences are then wrapped as question-answer pairs in a container.

Next a test set is created using the Jeopardy question dataset mentioned in the previous section. First the questions are filtered based on some characteristics such as inclusion of visual clues, which can be identified by the *„seen here“* phrase, inclusion of hyperlinks in the question etc. For questions that passed the filter the answer is matched with the titles indexed in ElasticSearch and every match is added to the test set as question-answer pairs.

[10] Screenshot of page: http://en.wikipedia.org/wiki/Socket

4.3 Graphical User Interface

The main philosophy behind the interface is to keep it intuitive, clear and minimalistic. The goal is to provide the user with the best user-experience possible, for me this means keeping the layout simple and well-spaced, with larger UI elements.

As previously multiple times mentioned speech recognition should be the primary method of input of questions, though not everyone knows about this functionality of Android so based on the simple drawing in Figure 5 a GUI is implemented that should guide the user to take this course of action. On the other hand some people might prefer not to use this function at all and would rather stick with keyboard input, for them a text field is added.

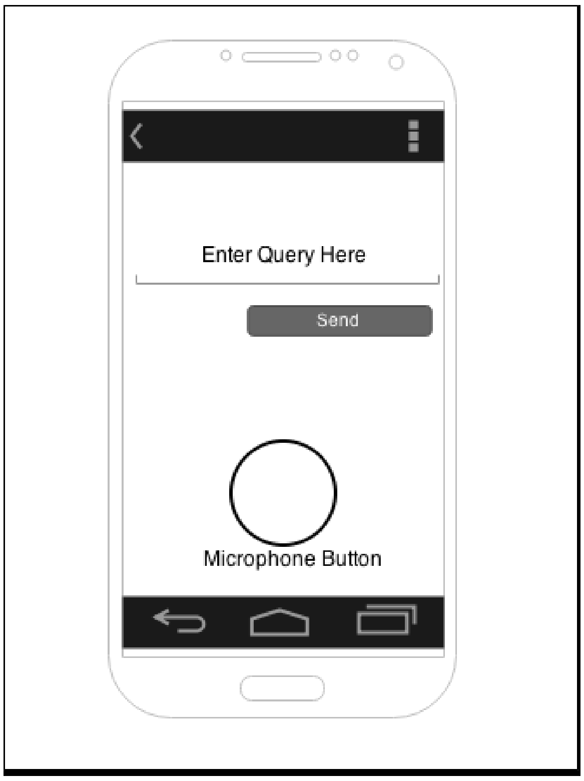


Figure 5 Basic UI design for Application

Toast

“A toast provides simple feedback about an operation in a small popup. It only fills the amount of space required for the message and the current activity remains visible and interactive and the message itself automatically disappears after a timeout.” [**21**]

There will not be permanent a connection between the server and the client, for this reason there is important to present some info about the connectivity to the user. For this purpose the application is going to use toast messages to communicate if the query was sent or if it failed because of no internet connectivity or the server not running.

5 Implementation

In this chapter the implementation process of the server system and the paired Android application is described. Starting off with the directory structure of the server then moving on to describe the modules used and what they implement. As the final step the implementation of a simple Android application is described that will serve the Wikipedia page as the answer to the query entered by the user.

5.1 Server

The implementation follows the imperative programming paradigm. The code is divided into modules in a way so the name of each modules should give a clear idea what it implements and what kind of code can be found in it.

5.1.1 Modules

This section is subdivided into subsections each named by the corresponding module they describe.

Main

The server can be started using main.py. The available command line arguments are as follow:

- --settings [file] XML file containing the settings of the server
- --force Forces the system to reparse the dump and retrain the model before launching the client-server

The *settings* module is used to read in and parse the XML file with the settings in it, the choice of storing these options rather than having a list of command-line arguments is twofold, first of all the list would have been very long and impractical to use, secondly a server is usually started once then left alone, rather than started over and over (excluding the phase of development), so having a static file to startup the server with is more desirable than the alternative. Given that python has no implicit data structure like language C, a Class is used instead to store all relevant data. This class named Folder is defined in this module also and it stores several settings for the code and some of the data that is being used by the script. This implementation allows to only the reference to this object to be passed around the functions. All data held by the *Folder* that becomes unnecessary is dumped either to files or to the trash.

The flow of work is decided by checking the availability of the trained model (classifier). If the classifier or the indexed data is missing the training path is taken. First the *janitor* script is called to cleanup folders for the output, it cleans and creates folders inside the *data* folder, the name of this folder is given by the *name* option. The training starts with downloading and parsing.

Downloader

This module implements the downloading and parsing of wikidump. It is done using *wget* command called through python's *subprocess* module. *Wget* is a Linux package with command line interface used to download files. An option of this program is *timestamping* which timestamps the downloaded file and the ability to *continue* downloading. Combining these two options a functionality is achieved where a file is downloaded if a newer version of it becomes available at the source. The program is called and the standard error output caught and searched for the "200.*OK" confirmation code which means the file is being downloaded. Parsing is done in a similar fashion without checking the output of WikiExtractor.

Extract Traindata

After the dump has been parsed the extraction process begins. The parsed output is split into 100 MB files by WikiExtractor and these files are processed sequentially, while each file itself is processed in parallel. The indexing is done on a file by file basis meaning after a file was processed the articles in the file are indexed in bulk to avoid overloading ElasticSearch. The extracted dataset is stored in the *folder* object as question-answer pairs while all answers are also stored in a list of unique answers. Questions from the Jeopardy dataset that have matching answers in the *list of answers* are stored as the *test set* as question-answer pairs. This module also creates the initial word embedding, although this is done after QANTAs preprocessing script finishes as the word embeddings are created using the vocabulary extracted by it. Mirroring previous methods the *word2vec* program is called through *subprocess*. The output of *wod2vec* is a text file where each line is a word and its corresponding vector representation with *d* dimensionality. This means the format of the output is *V*ocabulary size *x d*, this is not the desired input form of QANTA (2.3), so it is transposed into a *d x V* output.

Dparse to dtree

This module is inside the QANTA folder in the preprocess subdirectory. This module had to been heavily refactored to be useable in the server. While the core functionality has not been changed it has been significantly improved from a performance standpoint by parallelizing tasks and providing a new parsing script that calls NNDEP parser instead of PFCG parser. This module also dumps the *train and test sets* into the corresponding directory in the *data* folder, dumping the question file and then calling a *subprocess* to parse the questions from a file was faster than feeding the data into the parser through a pipe. This also has the added benefit of being able to look at the training and testing data and control for parsing errors and the like.

QANTA

An interface has been added to QANTA to avoid having to load preprocessed files from pickled dumps and for better integration. The folder instance passed to this interface contains all the necessary data for the interface to function.

Getters

This module contains the functions used for answer prediction and the function that trains and returns a classifier using the trained model. This classifier is then passed to the other getter functions to generate a class label for given input (question-sentence). This module employs the *dyn_parse.sh* script which is used to parse text on the fly as opposed to *parser.sh* from preprocessing which is parsing from text files.

Server

The *server* module first loads and trains a classifier using the *getter* module and then starts a threaded TCP server on port 13338. A handler for interrupt signals has been written into this module to shutdown gracefully on demand (e.g. killing the process). After this the server is started to serve forever and handling incoming requests. While the requests are served the main thread is free to check for updates on the wikidump, this is done by calling the downloader module. If there is no new file available the main thread is put to sleep for 12 hours.

5.2 Persistence and performance

The persistence of the server files is achieved using the shelve module, which allows "shelf files" to be used to store data using key-value pairs. This allows the server script to store all data needed to train a model and restart a failed server in one neatly packed file.

Due to the memory intensive nature of the calculations for QANTA (2.3), shelving is used to store temporary data that is not necessary at given moment to save operating memory. In the process of optimizing memory usage a serious design flaw was discovered.

As previously written in (2.3), the rank computation for each dependency tree for a sentence chooses 100 random wrong answers from a set of all wrong answers. The original code stored an array of indexes pointing to the location of all of the wrong answers in the vocabulary for each dependency tree (dtree) introducing a huge amount of redundant data. For each question there is one right answer and *n – 1* wrong answers where *n* is the total number of answers available. Each dtree object also had to (2.1.1) store the right answer (label), meaning each dtree object stored *n* indexes to answers in the vocabulary.

For solving this issue the code had to be refactored. Given that the order in which words are added to the vocabulary is inconsequential (but the order must remain stable after indexing of data is done), using the *list of answers* recorded during the preprocessing process all answers are added to the vocabulary before the dependency trees are created, this way all answers in the vocabulary are in index range {*0,n}*. As previously established the dtree stores the index of right answer, after adding *n* the number of all answers to the dtree as well, we are able to generate 100 random indexes in range *{0,n}*. Upon exclusion of the the index of the right answer from this range the indexes to all wrong answers to given dtree are available. This way the memory overhead is reduced from $Q\ x\ (A-1)\ x\ 4\ (bytes)$

to $Q \ x \ 1 \ x \ 24 \ (bytes)$ where Q is the number of all sentences in the dataset, A is the number of all answers. Despite eliminating this massive memory hungry part, the model remains memory expensive to the point where it is more limited by memory usage than processing power. This is further solved by splitting the dataset into batches of 10000-15000 questions which are then split into 25 mini-batches. This has a negative impact on training time, but makes the training on larger datasets feasible.

5.3 Serving the requests

When a query arrives from the user a new thread is created that is tasked with handling it. The first step is heuristics, checking if the query is an exact match to an article, in this case the answer is sent immediately without any question answering taking place.

If the received query is not a title the query is first parsed using the NNDP parser, then passed to the classifier to generate predictions. These predictions are then sorted, inflated to a probability score based on the test results in (**Error! Reference source not found.**). For each prediction the corresponding article is retrieved from ElasticSearch, then the query is tokenized using the NLTK word tokenizer. In the next step stop words are removed from the query and the remaining words are matched with the texts from the articles. The score *(matched words) / (all words in query)* is used to adjust the outcome score of the answer predictions of the classifiers. At this point the server sends out a JSON package with the predictions and corresponding matched keywords and a confidence score.

JSON structures are used over XML as JSON is faster and has less overhead for smaller data chunks while XML has bigger overhead but scales better for many objects and more data. [**22**] The bodies of the articles are not sent with the results, only after the user selects one is the text of a Wiki page sent to the client. The heuristic implemented for exact article match is used here, as upon selecting from the results the user is sending another query to the server, which is this time served immediately with the desired data and that data only. This helps to save valuable bandwidth which is especially important on mobile devices using mobile internet.

5.4 The application

Following the goals established in (4.3) the layout of the final application remained minimalistic in nature, which should provide good performance across all devices. Two activities are used to deliver the content to the user and a dialog fragment to deliver search results.

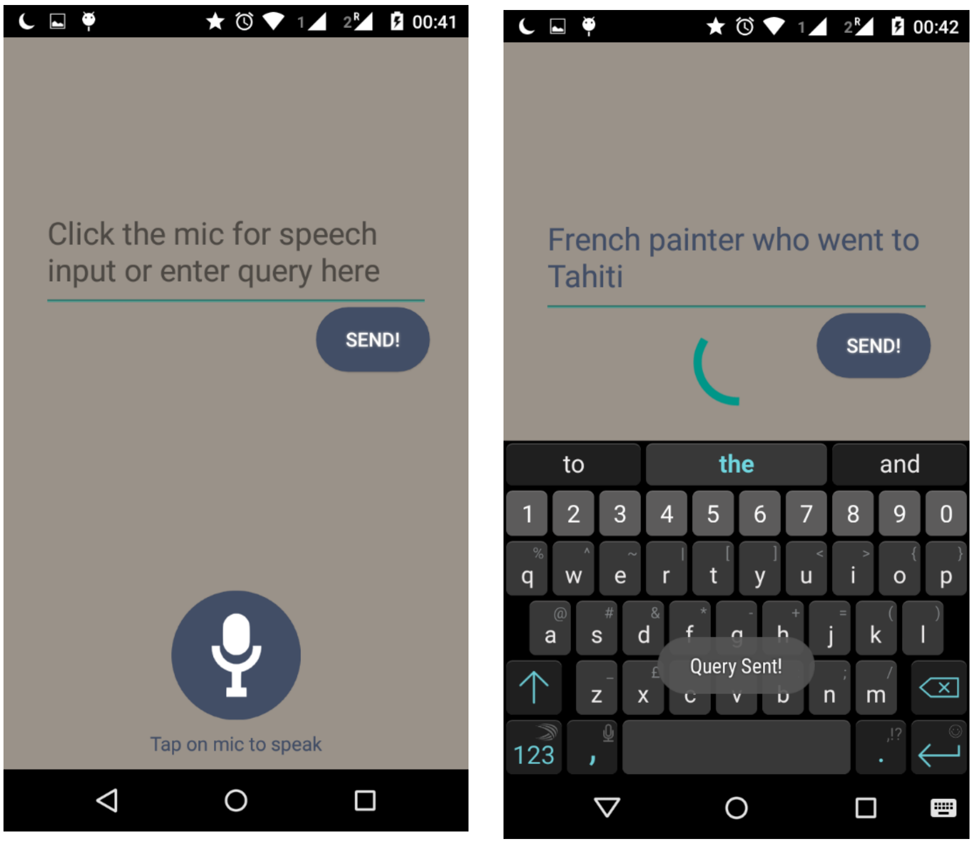
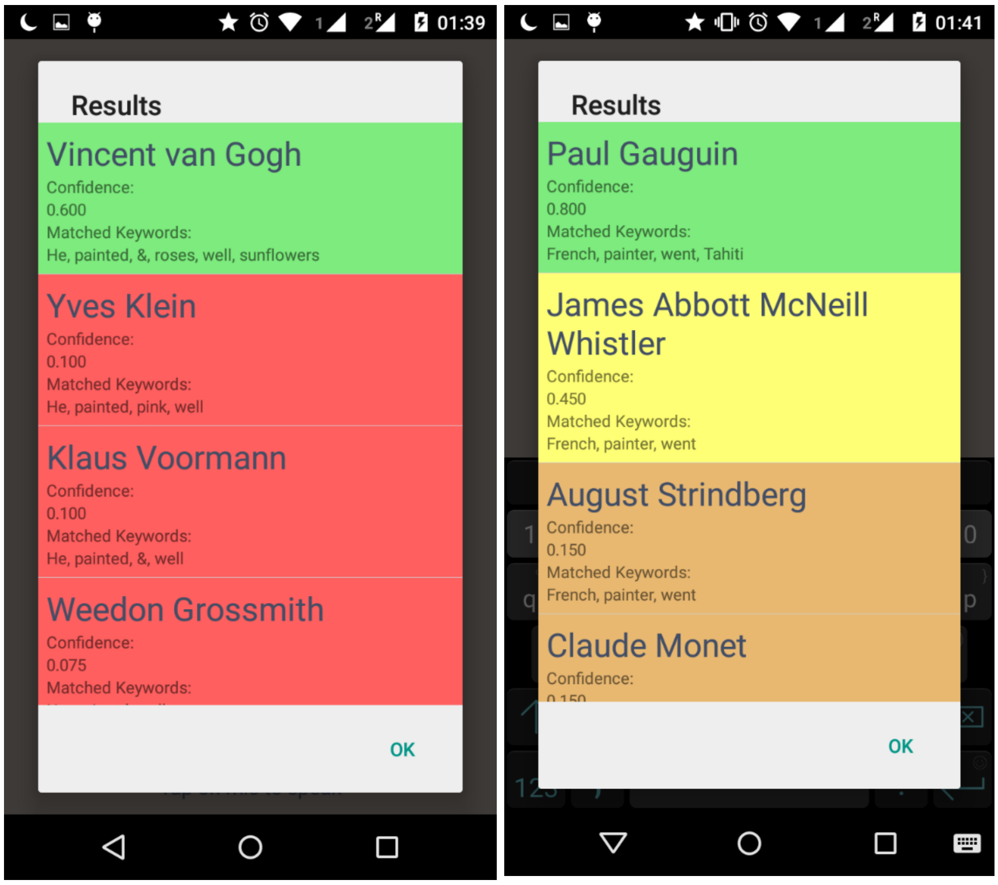

00:41
Click the mic for speech input or enter query here
SEND!
Tap on mic to speak
00:42
French painter who went to Tahiti
SEND!
to
the
and
Query Sent!
123

Figure 6 Final Application Layout

The lower part of the GUI presents the option to use speech input which is hidden by the keyboard upon “clicking” the text field. On the right picture in Figure 6 we can see the Toast notification informing us that the Query has been indeed sent, while a spinner is spinning indicating that the server is processing the query. Information about lack of internet connection or failure to connect to the server is server in the same fashion leveraging Toast messages.

Results are delivered into a dialog fragment, where the top 5 scored answer predictions are displayed starting with the page title (in this case name of painter), the confidence score and the matched keywords from the query. The results are also color coded for easier orientation. The dialog fragment can be dismissed using the OK button. Upon clicking on one of the answers a new activity is started to display the content of the Wikipedia pages. The user is given the option to enter new queries from this screen rather than having to go back to the main activity or to the results. Using the back button here as mentioned takes the user back to the previous activity or the search results based on if there search results delivered or the Wikipedia article exactly matching the query. (5.3)



Results

Vincent van Gogh
Confidence:
0.600
Matched Keywords:
He, painted, &, roses, well, sunflowers

Yves Klein
Confidence:
0.100
Matched Keywords:
He, painted, pink, well

Klaus Voormann
Confidence:
0.100
Matched Keywords:
He, painted, &, well

Weedon Grossmith
Confidence:
0.075
Matched Keywords:

OK

Results

Paul Gauguin
Confidence:
0.800
Matched Keywords:
French, painter, went, Tahiti

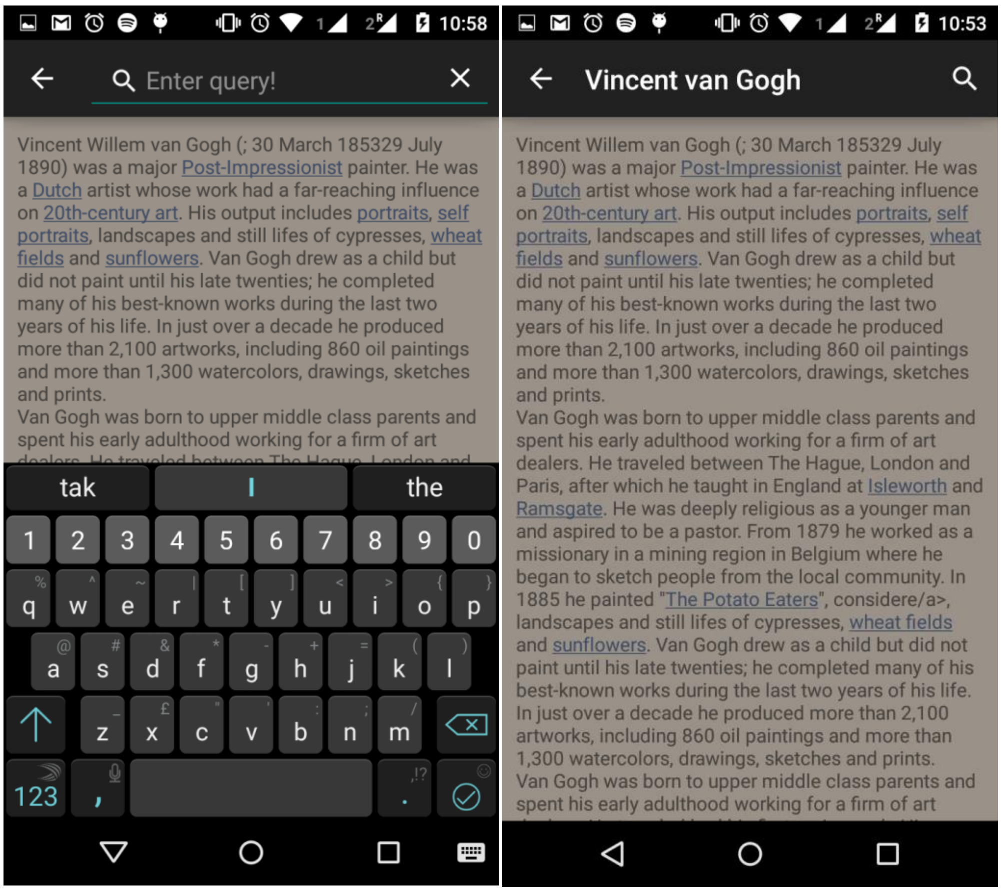
James Abbott McNeill Whistler
Confidence:
0.450
Matched Keywords:
French, painter, went

August Strindberg
Confidence:
0.150
Matched Keywords:
French, painter, went

Claude Monet
Confidence:

OK

Figure 7 Result Dialog delivered to the user



Vincent Willem van Gogh (; 30 March 185329 July 1890) was a major Post-Impressionist painter. He was a Dutch artist whose work had a far-reaching influence on 20th-century art. His output includes portraits, self portraits, landscapes and still lifes of cypresses, wheat fields and sunflowers. Van Gogh drew as a child but did not paint until his late twenties; he completed many of his best-known works during the last two years of his life. In just over a decade he produced more than 2,100 artworks, including 860 oil paintings and more than 1,300 watercolors, drawings, sketches and prints.
Van Gogh was born to upper middle class parents and spent his early adulthood working for a firm of art

Vincent van Gogh

Vincent Willem van Gogh (; 30 March 185329 July 1890) was a major Post-Impressionist painter. He was a Dutch artist whose work had a far-reaching influence on 20th-century art. His output includes portraits, self portraits, landscapes and still lifes of cypresses, wheat fields and sunflowers. Van Gogh drew as a child but did not paint until his late twenties; he completed many of his best-known works during the last two years of his life. In just over a decade he produced more than 2,100 artworks, including 860 oil paintings and more than 1,300 watercolors, drawings, sketches and prints.
Van Gogh was born to upper middle class parents and spent his early adulthood working for a firm of art dealers. He traveled between The Hague, London and Paris, after which he taught in England at Isleworth and Ramsgate. He was deeply religious as a younger man and aspired to be a pastor. From 1879 he worked as a missionary in a mining region in Belgium where he began to sketch people from the local community. In 1885 he painted "The Potato Eaters", considere/a>, landscapes and still lifes of cypresses, wheat fields and sunflowers. Van Gogh drew as a child but did not paint until his late twenties; he completed many of his best-known works during the last two years of his life. In just over a decade he produced more than 2,100 artworks, including 860 oil paintings and more than 1,300 watercolors, drawings, sketches and prints.
Van Gogh was born to upper middle class parents and spent his early adulthood working for a firm of art

Figure 8 Articles as seen by the user

6 Testing

While the memory requirements of QANTA were significantly decreased and the system was optimized to handle the task at hand as best as possible, the available hardware resources were still not enough to handle the training of the model on every Wikipedia article. Tests were done using a dataset of first (as parsed by WikiExtractor) 500 Wikipedia articles about painters. Roughly 38 000 training sentences. With 693 question answer pairs extracted from the Jeopardy question collection (hereinafter test-set).

6.1 Named entity insertion

The goal of the first set of tests was to see if replacing answer entities in the text (named entity recognition) has positive effect on the models ability to predict the answer as established in (4.2). The test was conducted using 4 datasets, one set with no answer insertion to the training corpus, the second set were the URLs where replaced with answers, the third with replacing the occurrences of the answer in the corresponding sentences with the entity representation and in the fourth where both the URL and page title replacement was done.

For each sentence from the test-set the top 5 ranked answer predictions were looked at and compared with the expected answer. As we can see on Figure 9 inserting answers into the text does not result in an increased accuracy. The model trained on the dataset with no insertion heavily outperforms the other 3 models by guessing more questions right as the first prediction and providing the correct guess in the top 5 predictions for more questions overall. The total column indicates the total number of correctly answered questions from the 693 questions.

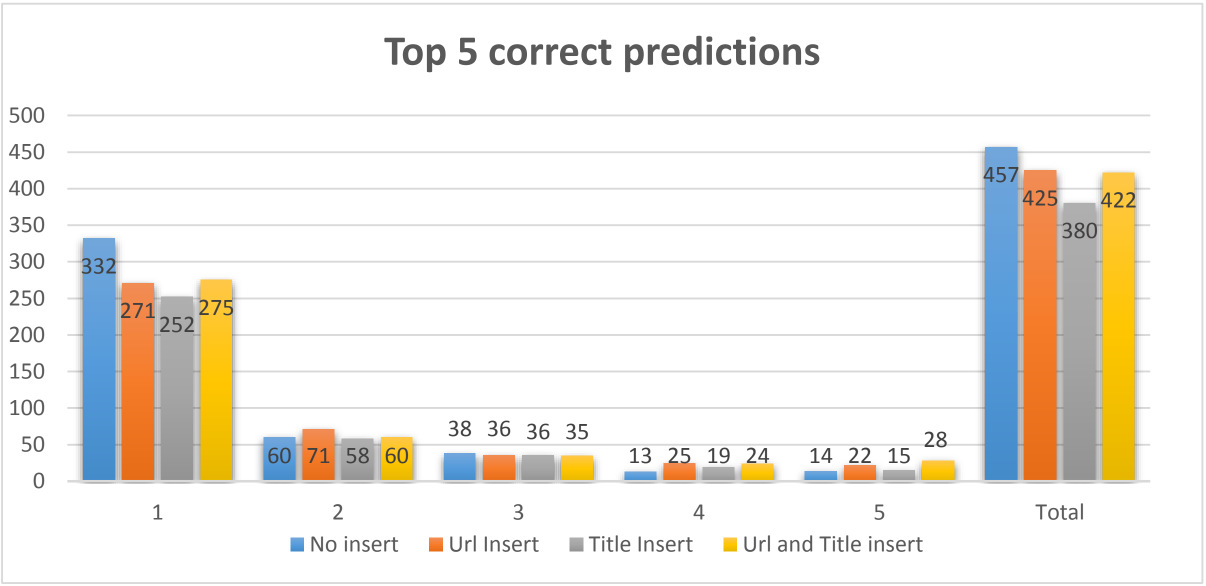


Figure 9 Correct prediction at given position in TOP 5

Based on these results default configuration is changed to avoid inserting the named entities into the dataset.

6.2 Distribution of top prediction

The distribution of correct and wrong predictions is looked in given score ranges. The distribution can be seen on Figure 13, where we can see that any prediction with score over 0.6 is correct in more than 90 % of the times, any answer prediction with a score between 0.4 and 0.6 is correct over 80 % of the times, 60 % of the predictions with score between 0.2 and 0.4 is correct and any lower score for a top prediction results in very unreliable answer prediction.

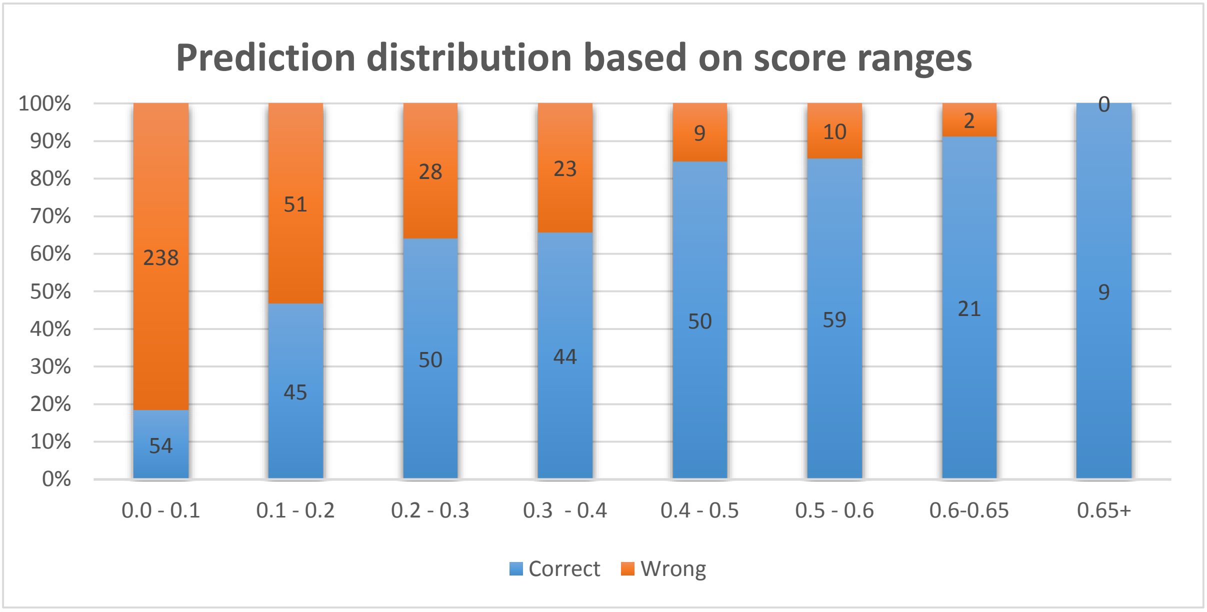


Figure 10 Correct/wrong answer prediction in given ranges

6.3 Multi-Sentence answering

In this final test the 693 questions are randomly coupled into multi sentence queries based on the expected answer. The question answer pairs are first sorted by the answer then same answer questions are randomly coupled into multi-sentence queries. After this predictions are created for the multi-sentence queries and the correct answers counted. Because the coupling is random the total number of questions is also recorded. As we can see in Figure 11, where total is the total number of new questions generated, the system is capable of consistently answering multi-sentence queries at a higher accuracy than single sentence queries.

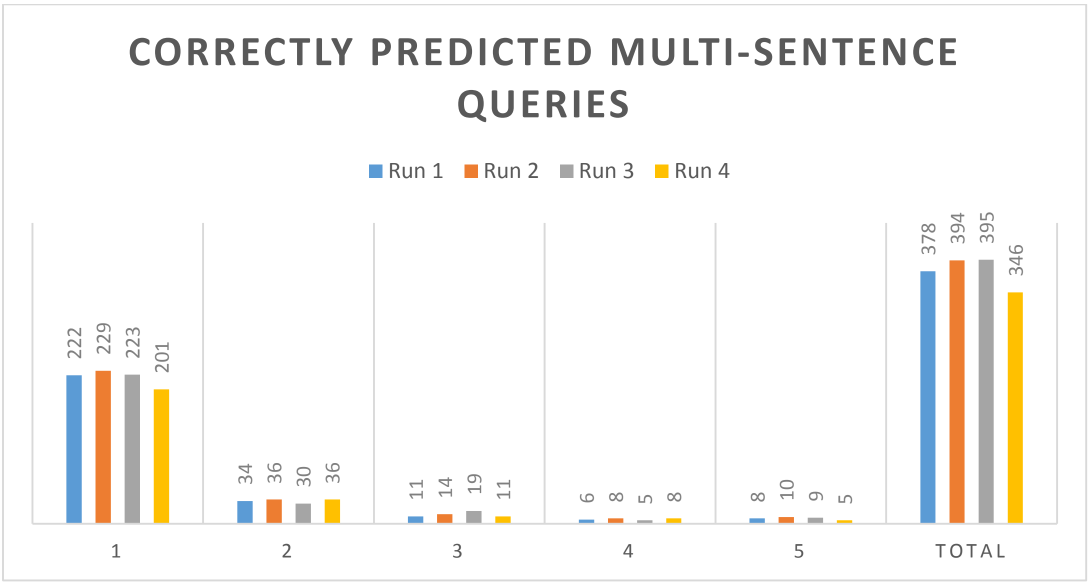


Figure 11 Multi - sentence accuracy

7 Conclusion

The goal of this work was to develop a system that allows query answering over Wikipedia on mobile devices with Android operating system. While the available system resources were a limiting factor a server was successfully developed with a paired client Android application. The user is delivered a minimalistic well performing application that offers usage for demanded use-case scenarios and the server is easy to deploy as a package and is capable of updating itself. The resulting system is a Multi-sentence closed-domain question answering system that learns titles of Wikipedia articles as answers.

In this thesis the topic of natural language processing and the challenges encapsulated by it were introduced, with a look taken at existing query answering systems, their capabilities and real life applications. After research into related fields and acquirement of the knowledge necessary to tackle the task at hand a server and a client application was designed and implemented which can be easily distributed and is comfortable to use as the end user. Various challenges arose during the realization of the project and multiple iterations were done each adding or improving on the functionality or speed of the programs. In the end both system and application were tested for functionality and accuracy which is described in the previous chapter.

Some of the implementation details of the system could be easily exchanged with a substitute, the application being an example. I decided to stick with an application as it offers better control over the content, offers built-in speech engine option and arguably better potential discover-ability on the Google Play marketplace. A webpage interface on the other hand is far easier to implement and is also available from desktop computers.

The work on this thesis brought me valuable insight into the field of natural language processing and machine learning, knowledge that is undeniable more and more important and relevant. Additionally I had the chance to learn about developing for the Android platform and acquired invaluable experience working with one of the most popular languages python.

Potential future development of the system includes complete redesign of QANTA to take advantage of distributed computing, expanding the functionality of the Android application and possibly improvement of the model by leveraging advanced named entity retrieval or natural language annotations.

References

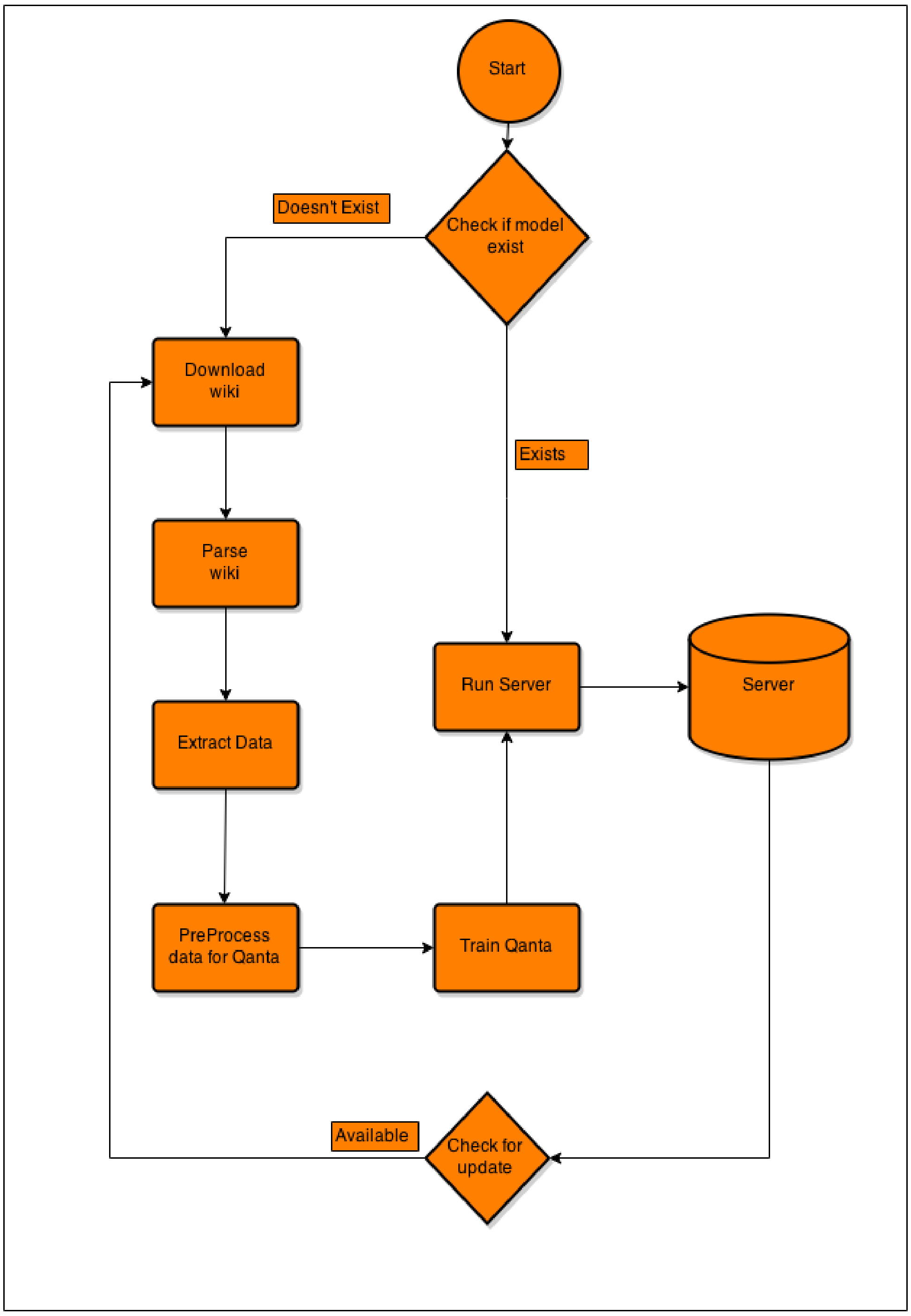
1. EBIZMBA:THE EBUSINESS GUIDE. *Top 15 Most Popular Websites | May 2015* [online]. [cit. 2015-May-14]. Available from: http://www.ebizmba.com/articles/most-popular-websites
2. IDC. *Smartphone OS Market Share, Q4 2014* [online]. [cit. 2015-May-14]. Available from: http://www.idc.com/prodserv/smartphone-os-market-share.jsp
3. ENCYCLOPEDIA BRITANNICA. *Machine Learning* [online]. [cit. 2015-May-14]. Available from: http://www.britannica.com/EBchecked/topic/1116194/machine-learning
4. HASTIE, T. R. TIBSHIRANI and J. FRIEDMAN. The Elements of Statistical Learning: Data mining, inference and Prediction. 2nd. [cit. 2015-May-14]. ISBN 978-0387848570.
5. BIRD, S. and E. A. L. E. KLEIN. Natural language processing with Python, ch. 6. In: *data mining, inference, and prediction*. Springer series in statistics. Beijing: O´Reilly, 2009 [cit. 2015-May-14]. Available from: http://www.nltk.org/book/ch06.html
6. XIAO, R. Corpus Creation. In: *Handbook of Natural Language Processing, Second Edition*. Nitin Indurkhya and Fred J. Damerau. INDURKHYA, N. and F. J. DAMERAU, eds. Boca Raton (Florida): CRC Press, Taylor and Francis Group, 2010 [cit. 2015-May-15]. ISBN 978-1420085921. Available from: http://cgi.cse.unsw.edu.au/~handbookofnlp/index.php?n=Chapter7.Chapter7
7. MITTAL, S. and A. MITTAL. Versatile question answering systems seeing in synthesis, ch. 2. In: *Int. J. Intelligent Information and Database Systems*, sv. V. 2001, s. 119-42. Also available from: https://www.academia.edu/2475776/Versatile_question_answering_systems_seeing_in_synthesis
8. IYYER, M. J. BOYD-GRABER and L. C. DAMÉ III. A Neural Network for Factoid Question Answering over Paragraphs. In: *Empirical Methods in Natural Language Processing*. Doha, Qatar: 2014 [cit. 2015-May-14]. Available from: http://cs.umd.edu/~miyyer/pubs/2014_qb_rnn.pdf
9. YOUTUBE.COM. *CHM Revolutionaries: A Computer Called Watson with IBM Research's David Ferrucci* [online]. [cit. 2015-May-15]. Available from: https://www.youtube.com/watch?v=ZvbWyREkMkw
10. PC WORLD. *IBM Watson Wins Jeopardy, Humans Rally Back* [online]. [cit. 2015-May-15]. Available from: http://www.pcworld.com/article/219900/IBM_Watson_Wins_Jeopardy_Humans_Rally_Back.html
11. IBM. *IBM Watson: The Science Behind an Answer*. [cit. 2015-May-15]. Available from: https://www.youtube.com/watch?v=DywO4zksfXw

12. START. *Natural Language Question Answering System* [online]. [cit. 2015-May-14]. Available from: http://start.csail.mit.edu/index.php
13. THE VERGE. *Google Now: behind the predictive future of search* [online]. [cit. 2015-May-15]. Available from: http://www.theverge.com/2012/10/29/3569684/google-now-android-4-2-knowledge-graph-neural-networks
14. OPEN HANDSET ALLIANCE. *Industry Leaders Announce Open Platform for Mobile Devices* [online]. [cit. 2015-May-15]. Available from: http://www.openhandsetalliance.com/press_110507.html
15. TECH NEWS: NDTV GADGETS. *Samsung's T9000 smart refrigerator runs on Android, includes apps like Evernote and Epicurious* [online]. [cit. 2015-May-15]. Available from: http://gadgets.ndtv.com/others/news/samsungs-t9000-smart-refrigerator-runs-on-android-includes-apps-like-evernote-and-epicurious-320610
16. WIKIPEDIA.ORG. *Android (operating system)* [online]. [cit. 2015-May-15]. Available from: http://en.wikipedia.org/wiki/Android_%28operating_system%29
17. ANDROID DEVELOPERS. *Application Fundamentals* [online]. [cit. 2015-May-14]. Available from: http://developer.android.com/guide/components/fundamentals.html
18. THE STANFORD NLP GROUP. *Stanford CoreNLP* [online]. [cit. 2015-May-14]. Available from: http://nlp.stanford.edu/software/corenlp.shtml
19. PYTHON.ORG. *What is Python? Executive Summary* [online]. [cit. 2015-May-15]. Available from: https://www.python.org/doc/essays/blurb/
20. META-WIKI. *Data dumps - Meta* [online]. [cit. 2015-May-15]. Available from: http://meta.wikimedia.org/wiki/Data_dumps#How_often_dumps_are_produced
21. ANDROID DEVELOPERS. *Toasts* [online]. [cit. 2015-May-15]. Available from: http://developer.android.com/guide/topics/ui/notifiers/toasts.html
22. MAJID KHOSRAVI. *XML vs JSON parsing in Android* [online]. [cit. 2015-May-15]. Available from: http://www.majidkhosravi.com/xml-vs-json-android/

Appendices

Appendix 1

Server Flowchart



Appendix 2

DVD

- /technical report/ - This report in DOCX and PDF format
- /server/ - The code that implements the server
    - /stanford-corenlp/ - The Stanford CoreNLP package
    - /word2vec/ - The word2vec program
    - /wikiextractor-master/ - WikiExtractor
    - /qanta/ - QANTA
    - /data/ - Jeopardy Questions
- /application/ - The source code and apk package of the Android application