TÉCNICO LISBOA

A Virtual Assistant for Web Dashboards

Support Bot

Ricardo Serralheiro Mendes Lopes

Thesis to obtain the Master of Science Degree in

Information Systems and Computer Engineering

Supervisors: Prof. Maria Luísa Torres Ribeiro Marques da Silva Coheur
Eng. José Miguel Castanheira Cavalheiro

Examination Committee

Chairperson: Prof. Luís Manuel Antunes Veiga
Supervisor: Prof. Maria Luísa Torres Ribeiro Marques da Silva Coheur
Member of the Committee: Dr. João Freitas

October 2018

Acknowledgments

Em primeiro lugar gostaria de agradecer ao BNP Paribas a oportunidade que me deram e por me terem dado sempre as condições necessárias para desenvolver o meu trabalho.

Gostaria também de agradecer aos meus orientadores, a Professora Luísa Coheur e o Engenheiro José Cavalheiro, por toda a disponibilidade demonstrada, pelas sugestões e pelas críticas construtivas. Um agradecimento especial à Professora Luísa pela sua visão optimista, por saber sempre o que dizer e por me ouvir queixar dos meus problemas, estivessem eles relacionados com esta tese ou não.

A toda a minha família, e em especial aos meus pais, ao meu irmão e aos meus avós, por sempre me apoiarem ao longo de todo o meu percurso académico e por todo o interesse que demonstraram no meu trabalho.

Aos meus amigos, em especial à Catarina Campos, ao Tiago Vicente, ao Miguel Amaral, ao Daniel Sousa, ao Daniel Simões e ao João Duarte, por sempre me incentivarem, por me apoiarem, pelo interesse mostrado e por me ajudarem a desanuviar quando era necessário.

A todos,

Obrigado.

Abstract

The aim of this work is to implement a system capable of answering questions posed by the employees of BNP Paribas, regarding the web dashboards they use, in Natural Language, through text – a Frequently Asked Questions problem. Frequently Asked Questions are a collection of question-answer pairs, usually manually-created, which answer the most asked questions by the users of the system. Generally, questions in the Frequently Asked Questions collection are independent from each other, that is, they ask different things. As such, there is only 1 question of each "type" in the collection. However, this presents a problem in Natural Language Processing. When a user asks a new question, it is possible to have a mismatch between the terms of the new question and the terms of the question in the Frequently Asked Questions Collection. We explore how useful paraphrases can be to overcome this problem by developing a Paraphrase Generation module based on paraphrasing rules and based on Neural Machine Translation, and applying it to our manually-created questions dataset, thus enhancing our dataset. We find that using paraphrases to populate a Knowledge Base improves the accuracy of our system. Moreover, we propose to enhance our solution by asking users for their feedback, which enables the system to match new questions with the ones already in the Knowledge Base. The final solution is a system that generates paraphrases of the user question, searches for similar questions in the Knowledge Base, retrieves the final answer from a Question/Answer Database and learns from the feedback given by the users.

Keywords

Natural Language Processing, Question Answering, Paraphrase Generation

Resumo

O objectivo deste trabalho é implementar um sistema capaz de responder às perguntas dos empregados do BNP Paribas, relativas aos *web dashboards* que usam, em Língua Natural – um problema de *Frequently Asked Questions*. *Frequently Asked Questions* é uma coleção de pares pergunta-resposta, geralmente criada manualmente, que responde às perguntas mais comuns, feitas pelos utilizadores do sistema. De modo geral, as perguntas na coleção são independentes umas das outras, isto é, só há uma pergunta de cada "tipo". No entanto, isto causa um problema visto que quando um utilizador faz uma nova pergunta é possível que os termos utilizados não sejam os mesmos que os termos presentes na coleção de *Frequently Asked Questions*. Neste trabalho exploramos quão úteis são as paráfrases para ultrapassar este problema. Para tal, desenvolvemos um módulo de Geração de Paráfrases baseado em regras e baseado em *Neural Machine Translation*, e aplicamo-lo ao nosso *dataset* de perguntas, gerado manualmente a partir da documentação de um dashboard, de modo a expandi-lo. Descobrimos que usar paráfrases para popular uma Base de Conhecimentos melhora a *accuracy* do nosso sistema. Para além disso, propomos melhorar a nossa solução através do *feedback* dos utilizadores, que permite ao sistema aprender uma correspondência entre as novas perguntas e as perguntas na Base de Conhecimentos. A solução final é um sistema que gera paráfrases da pergunta do utilizador, procura por perguntas semelhantes na Base de Conhecimentos, recupera a resposta final de uma Base de Dados de Pergunta/Resposta e aprende ao longo do tempo com o *feedback* que lhe é dado pelos utilizadores.

Palavras Chave

Processamento de Língua Natural, Pergunta/Resposta, Geração de Paráfrases

Contents

List of Figures

List of Tables

Acronyms

BNPP	BNP Paribas
QA	Question Answering
QAS	Question Answering System
NLP	Natural Language Processing
IR	Information Retrieval
IE	Information Extraction
NLIDB	Natural Language Interface for Databases
ODP	Open Directory Project
AQE	Automatic Query Expansion
SVM	Support Vector Machine
CFG	Context Free Grammar
QG	Quasi-synchronous Grammar
QCFG	Quasi-synchronous Context Free Grammar
CRF	Conditional Random Field
kNN	k-Nearest Neighbors
MSRPC	Microsoft Research Paraphrase Corpus
PPDB	Paraphrase Database
MT	Machine Translation
NMT	Neural Machine Translation

NN	Neural Network
SQL	Structured Query Language
RS	Recommender System
CBF	Content-based Filtering
CF	Collaborative Filtering
HF	Hybrid Filtering
LD	Large Document
SD	Small Document
CBFC	Content-based Filtering and Classification
CFC	Collaborative Filtering and Classification
HFC	Hybrid Filtering and Classification
T2T	Tensor2Tensor
KB	Knowledge Base
PG	Paraphrase Generation
NLTK	Natural Language Toolkit

1

Introduction

Contents

BNP Paribas (BNPP) is an international banking group with a presence in 74 countries and more than 190,000 employees[1]. As a result of their operations large amounts of data are recorded for analysis. Tools such as web dashboards are used by BNPP employees to access the information they need to perform business activities. An example of a dashboard used by BNPP is the "Rollover[2] Opportunities" dashboard, which displays information regarding trades about to expire and the history of trades of a given client. For instance, users can see how many times a particular client has performed a rollover and which clients might be more interesting to talk to, in order to perform a rollover (a rollover is the process of extending the settlement date of an open position). Currently, if dashboard's users have any questions, other than asking their coworkers, they can only send an email to the support team asking for help. However, replies are not immediate, and users are forced to wait, causing productivity to drop. Examples of questions users of the "Rollover Opportunities" dashboard might ask are: "How is the rollover ratio calculated?" or "What are the filtering options of the dashboard?".

To solve this problem, we propose to implement a Question Answering System (QAS), a type of system designed to automatically answer questions made by users in Natural Language to solve the dashboard users' problem of having to wait for a reply from the support team. Instead, users can ask the same question to the QAS – through text – and receive an answer much faster.

This is a Frequently Asked Questions (FAQ) problem. As such, we will build a questions dataset (FAQ collection) based on the documentation of the dashboard. To deal with the problem of lack of data, more specifically the mismatch between the terms in the FAQ collection and the terms in the questions asked by the users, we will create a Paraphrase Generation (PG) module, to automatically generate paraphrases of the questions in our dataset, in order to expand it. We then evaluate our system to see if the use of paraphrases leads to more accurate answers.

1.1 Objectives

The objectives of this work are as follows:

- Implement a system capable of answering the dashboards users' inquiries;
- Implement a Paraphrase Generation module capable of expanding our collection of questions.

1.2 Organization of the Document

This thesis is organized as follows: Chapter 2 examines related work in the area of Question Answering (QA), with focus on each stage the QA pipeline and the generation of paraphrases. While it is not the

[1] http://www.bnpparibas.pt/en/bnp-paribas/bnp-paribas-group/ (accessed December 20, 2017)
[2] https://www.investopedia.com/ask/answers/06/rollover.asp (accessed August 18, 2018)

focus of our work, we briefly examine work related to Natural Language Interfaces for Databases and Recommender Systems, as it may be useful in a future work. In Chapter 3, we present and evaluate our Paraphrase Generation module, used in several of our proposed solutions. In Chapter 4, we propose and evaluate several solutions to build a Question Answering System, based on the amount of work required to build a QAS for a different dashboard. Finally, Chapter 5 concludes this work by discussing the contributions, current limitations and future work.

2

Related Work

Contents

Question Answering (QA) is a research area that combines the research of Information Retrieval (IR), Information Extraction (IE) and Natural Language Processing (NLP), to give an answer to a user query. Moreover, Question Answering Systems (QASs) are typically comprised of three distinct modules: Question Processing, Information Processing and Answer Processing [1, 2]. Additionally, QASs can be classified as either open-domain or closed-domain depending on the topics they can answer. While closed-domain systems focus only on selected topics, open domain systems try to answer questions from nearly any topic. In the following sections we review previous works in the area of QA, focusing on each stage of the QA pipeline and on Paraphrase Generation (PG). Moreover, we also briefly analyze existing work in the area of Natural Language Interface for Databases (NLIDB) and Recommender Systems (RSs).

2.1 Question Processing Module

Given a question in Natural Language as input the Question Processing Module analyzes and transforms the question into a representation more suitable for the Information Processing Module.

2.1.1 Question Classification

Question Classification refers to classifying the given sentence, usually based on the semantic category of the expected answer. This is a very important step in the QA pipeline as the classification can be used to filter possible answers but also to suggest different search strategies.

All the works discussed here follow Li and Roth's two-layered question taxonomy which contain six coarse-grained categories and fifty fine-grained categories [3]. An example of a coarse-grained category is "Location", which contains 5 fine-grained categories: city, country, mountain, state, other. For instance, the question "Where is Lisbon?" would be classified as LOCATION:country.

Zhang and Lee [4] compared different learning algorithms using only surface-level features (bag-of-words and bag-of-ngrams) and found that Support Vector Machines (SVMs) outperform the other algorithms for both fine-grained and coarse-grained classification. Additionally, Zhang and Lee also suggested including syntactic information. First, the sentence is parsed into a syntactic tree. The tree is then represented as a vector defined by its fragments, where the fragments of a syntactic tree are all its sub-trees with at least a terminal symbol or a production rule. A production rule is a rule from the grammar. For instance, for Context Free Grammars (CFGs) production rules are of the form

$$X \rightarrow \beta \tag{2.1}$$

where X is a non-terminal symbol and β can be any sequence of terminal and non-terminal symbols. However, a restriction was made such that production rules could not be broken into incomplete parts.

Next, they proposed a tree kernel based on the syntactic fragments and trained a SVM using that kernel. The tree kernel was significantly better for coarse-grained classification (accuracy 90% vs. 87.4%) but only slightly better for fine-grained classification (results omitted in the original work).

Huang et al. [5] trained a SVM and a Maximum Entropy model for question classification. In their approach each question is represented as a bag-of-features. The five features they used where: wh-word, head-word, hypernyms, n-grams and word shape. Wh-word corresponds to the starting of the question and could be "what", "which", "when", "where", "what", "how", "why", or "rest". The head-word is extracted from the syntactic tree of the question. Hypernyms of the head-word are used directly from WordNet [6] or indirectly by computing the similarity between the head-word and the words in the description of each category, using the WordNet Similarity package [7], an extension to WordNet. WordNet is a manually constructed network of lexical relationships and its similarity packages extends the network with several measures of similarity between words. The question category with the highest similarity is marked as a feature. For n-grams they used uni, bi and tri-grams. Word shape represents the shape of the question and can be "all upper", "all lower", "mix case", "all digits", "other". They found that using bigrams and trigrams instead of unigrams decreased accuracy and that using all three did not improve accuracy. Their best result came from using all features (accuracy of 93.6 for Coarse classification and 89.2 for Fine classification).

Silva et al. [8] used a rule-based classifier and WordNet to train a SVM. If one of the sixty hand-built patterns matches the question, then a category is returned. Otherwise, the head-word of the question is extracted and its hypernyms are followed until one matches a possible category. Their best result was achieved using the head-words and categories discovered by the rule-based classifier, together with unigrams, as features to train the SVM (accuracy of 95% for coarse-grained and 90.8% for fine-grained).

Instead of classifying questions based on the expected semantic category of the answer, it is also possible to classify them based on the topic of the question. Beitzel et al. [9] compared and combined different classifier, trained on a set of 20,000 questions manually classified into 18 topics such as "Entertainment", "Business" or "Computing" [10]. The models trained were: pre-retrieval; bridged; explicit (snippets); explicit (snippets + docs) and pre-retrieval + explicit (snippets). The first model, pre-retrieval, uses only the query string for classification and is an ensemble classifier composed of a perceptron, selectional preference and exact match, as described in [10]. The bridged model, following the work of the winning entry for the KDD cup 2005 [11], is a model that builds a "bridge" between two taxonomies. However, as opposed to the KDD entry where an intermediate taxonomy is used as a bridge, Beitzel et al. manually bridged the Open Directory Project (ODP)[1] pages into one of the existing 18 categories and trained a SVM on the ODP webpages. For the explicit models, SVMs were trained on the top 10 snippets and snippets + documents retrieved from Google for each query. Their final model was a com-

[1]ODP is no longer available since March 2017, but a static mirror can be found at http://dmoztools.net/

bination of models 1 and 3. From their conclusions, training models explicitly with the classified queries outperforms training by building a taxonomy bridge by as much as 48% in F1 score.

2.1.2 Paraphrase detection

Paraphrase detection is an important step of QA pipeline. By detecting questions with the same meaning, but worded differently, we can give the same answer to both questions instead of going through the whole pipeline or compare the previous answer with the new answer and decide which one is correct.

In order to detect paraphrases, Kozareva and Montoyo [12] used a meta-classifier composed of SVM, k-Nearest Neighbors (kNN) and Maximum Entropy classifiers. Two feature sets were used for training: word overlap and word similarity. The first set uses as features:

- Skip-grams;
- The ratio between the common consecutive n-grams in both sentences and the total number of words in each sentence;
- Longest common subsequence between the two sentences, normalized by the length of each sentence.

Skip-grams is a technique similar to n-grams, but instead of only allowing adjacent tokens it allows words to be "skipped". The word similarity feature set is based on computing the similarity for each verb/noun pair in both sentences using the WordNet Similarity package. The three classifiers were trained on the Microsoft Research Paraphrase Corpus (MSRPC) [13]. Their best result was achieved using the votes of the three classifiers and a combination of both feature sets (accuracy of 76.64%).

Fernando and Stevenson [14] also used the WordNet Similarity package but, instead, build similarity matrices for the pairs of sentences being compared. Each matrix element w_{ij} represents the similarity between word p_i and word p_j, according to some similarity measure. They experimented on the MSRPC with six already existing similarity measures from the similarity package with the best result being given by the "jcn" similarity measure by Jiang and Conrath [15] (accuracy of 74.1%). Given two concept nodes C_1 and C_2 in WordNet, the jcn similarity measure is computed as:

$$sim_{jcn} = \frac{1}{(IC(C_1) + IC(C_2) + 2 \times IC(LCS(C_1, C_2))} \tag{2.2}$$

where LCS is the Least Common Subsumer of the two concept nodes, defined as the most specific node that is ancestor of both concept nodes and IC is the Information Content of a concept c, defined as:

$$IC(c) = -logP(c) \tag{2.3}$$

where $P(c)$ is the probability of seeing concept c in a corpus.

More recently, Madnani and Tetreault [16] used eight machine translation metrics as features to train classifiers, also on the MSRPC. Their best result is achieved with a meta-classifier composed of a logistic regression, SVM and kNN, using all metrics as features (accuracy of 77.4%).

An application of paraphrase detection in QA systems can be seen in [17], where the authors detect whether the new query is a paraphrase of one of the cached queries and, if so, simply return the answer for that query instead of going through the usual process of retrieval.

2.1.3 Automatic Query Expansion

Automatic Query Expansion (AQE) is a technique used to deal with the mismatch occurring between the words used in the documents and the words used in the input query. The query is expanded by adding words or phrases with similar meaning to those in the query. The key factors for AQE are the source of the expansion terms and the method for ranking the expansion terms [18]. According to Carpineto et al. [19] the several approaches to AQE can be divided into five groups, based on the method for selecting the expansion terms: linguistic analysis, corpus specific, query specific, search log analysis and web data.

2.1.3.A Linguistic analysis

These approaches are usually based on knowledge representation sources such as dictionaries, thesauri or WordNet. However, using a general purpose thesaurus for technical domains can result on a big lack of coverage, seeing that those often have their own sublanguage [20]. There is also the problem of word ambiguity as these knowledge sources show all the possible meanings of a word even when only a few are valid for the specific query.

2.1.3.B Corpus specific

Also known as global analysis, corpus based techniques analyze the contents of a full-text database to identify features used in similar ways.

Qiu and Frei [21] built a thesaurus based on term-term similarity. Terms are represented as vectors indexed by the documents and the thesaurus is built by determining the similarity between all term-term pairs using the scalar vector product.

Another approach is that of Crouch and Yang [22], where they built a thesaurus by clustering the document collection using the complete-linkage clustering algorithm. This algorithm starts with as many clusters as documents and at each step the similarity between all clusters is computed. The most similar clusters are merged until only 1 cluster remains, which results in a hierarchy of clusters. Then,

for building the thesaurus, the cluster hierarchy is traversed, and thesaurus classes are generated. The queries and documents are then augmented by the thesaurus classes generated.

2.1.3.C Query specific

Also known as local analysis, these techniques typically extract expansion terms from the top N retrieved documents. First, the original query is issued and the documents are ranked. Then all terms are extracted from the top N retrieved documents and ranked based on their potential contribution to the query. The top terms are appended to the query and the final, expanded query is reissued yielding the final documents ranking.

One example is the work of Xu and Croft [23] who proposed extracting concepts (groups of nouns) from passages (300 words in their experiments) instead of documents. The reason for using passages instead of documents is that documents can be very long and cover multiple topics, which can lead to augmenting the query with irrelevant information. Another drawback of these techniques is efficiency, because of the need to retrieve documents during query evaluation time. Billerback and Zobel [24] discussed ways to improve the efficiency of the query evaluation process.

2.1.3.D Search log analysis

The idea behind search log analysis is to mine query associations and extract expansion terms from there. Billerbeck et al. [25] associated a query with the top N documents returned for that query. The set of associated queries was then used to represent the document. Associations can be used to rank the original query, as a source for expansion terms, or both. Cui et al. [18] used query logs to try and extract relationships between query terms and document terms. For each query, the documents the users clicked on were stored and used to establish probabilistic correlations between the query terms and the document terms. At expansion time the probabilistic correlations were used to select terms from documents. Alternatively, Jones et al. [26] used past queries directly as a source for expansion terms.

2.1.3.E Web data

These approaches make use of data available on the web. The most common approach is to use anchor texts (the clickable text of a hyperlink) as most of the time they are a brief description of the webpage it links to. Kraft and Zien [27] proposed a method for AQE using anchor texts and compute the rank of anchor texts as the median rank aggregation of:

- Weighted number of occurrences of an anchor text. Weights were based on the destination page (same directory, same website, different website);
- Number of elements in the anchor text;

- Number of characters in the anchor text;

Riezler et al. [28] took a different approach. They used 10 million question-answer pairs from Frequently Asked Questions (FAQ) pages to learn lexical correlations between words in questions and answers. They expanded queries with content words not present in the original query.

2.2 Information Processing Module

The Information Processing Module retrieves documents or passages and ranks them based on their relevance to the query.

2.2.1 Passage Ranking

The output of the Passage Ranking step is a list of the k most relevant passages for the input query.

According to Callan [29] the types of passages can be grouped into 3 classes:

- Discourse - based on paragraphs, sentences, sections, etc.
- Semantic - based on the content of the text
- Window - based on the number of words

Tellex et al. [30] compared 8 different passage retrieval algorithms. From their experiments, differences between passage retrieval algorithms are only significant depending on the document retrieval system used. Their results using Lucene[2] show that differences in performance between algorithms are not statistically significant, using this retrieval engine.

Robert and Gaizauskas [31] proposed 4 different methods for passage retrieval and compare it to Okapi [32]. The first approach preprocesses all documents and creates a new collection made of all passages. For comparison with Okapi, paragraphs are used as passages. It returns the best passages, possibly more than 1 from the same original document. The second approach retrieves the top N documents for the original query and post-processes them into passages. A document collection is then created from its passages. The same retrieval engine used to retrieve the documents is used to determine the best passage from each document collection. For the ith ranked document, its best passage is returned as the ith best passage in the final ranking. Approach number 3 also retrieves the top N documents and post-processes them into passages. The best passages are returned, allowing multiple passages from the same original document to be returned. Approach number 4 post-processes the top N documents retrieved, and the best N passages are returned but at most 1 per original document.

[2] https://lucene.apache.org/

This approach is similar to the second approach, but the ranking is based on the passage score, instead of the document ranking.

However, while there is some research work done in passage ranking for QA, most systems rely on already existing information retrieval systems such as Elasticsearch[3]

2.3 Answer Processing Module

Given a set of passages from the Information Processing Module, the Answer Processing Module filters the relevant passages and extracts the final answer from the passages.

2.3.1 Answer Sentence Selection

The goal of Answer Sentence Selection is to decide whether or not a given sentence contains the correct answer to the question and re-rank the sentences such that the ones more likely to contain an answer are higher ranked [33].

Wang et al. [34] made use of a Quasi-synchronous Grammar (QG), originally by Smith and Eisner [35]. A QG is a monolingual grammar that translates sentences from the source-language to the target-language. To do this, every node of the target grammar is annotated with a set of zero or more nodes from the source tree. For instance, the rules of a Quasi-synchronous Context Free Grammar (QCFG) would be of the form:

$$(X, \alpha) \rightarrow (Y, \beta)(Z, \gamma) \tag{2.4}$$

$$(X, \alpha) \rightarrow w \tag{2.5}$$

where X, Y and Z are non-terminal nodes of the target grammar, α, β and γ are nodes of the source tree and w is target terminal node. Based on these grammars Wang et al. proposed a generative model that estimates the probability of the question tree, given the answer tree. For training they used questions from TREC 8-12 (100 manually-judged and 2,293 automatically-judged) and questions from TREC 13 for development and testing (84 and 100 questions, respectively). This experimental setup is now the most used for the task of Answer Sentence Selection and all works discussed here are evaluated with this dataset. They achieved a **MAP!** (**MAP!**) of 0.6029%.

In 2010, Wang and Manning [36] proposed another generative model, but based on Conditional Random Fields (CRFs), which model the alignment between question and answer as a set of tree-edits

[3] https://www.elastic.co/ (accessed December 2, 2017)

between the two parse trees. Using the same experiment setup as the previous work, they achieved similar performance (MAP: 0.5961%).

In the same year, Heilman and Smith [37] proposed a tree-edit model to solve this problem. They modeled question/answer pairs as a sequence of tree edits that transforms one tree into the other. In order to find the minimum edit sequence they used a greedy best-first search with a tree kernel heuristic as its evaluation function and trained a logistic regression classifier on labels and features of the edit sequences. Their performance was also similar to the previous two works (MAP: 0.6091%).

In 2013, Yao et al. [38] expanded upon the work of Heilman and Smith by using 15 additional syntactic features, as well as semantic information from WordNet. While there was an increase in Precision and Recall over the previous works, since the evaluation set they used was smaller the results were not statistically significant and should be considered on par with the previous works (MAP: 0.6371%).

In the same year, Severyn and Moschitti [33] trained a binary SVM with tree kernels using shallow syntactic information to classify question/answer pairs as correct or not. The answers were then re-ranked such that the higher ranked sentences were the ones more likely to contain a correct answer (MAP: 0.6781%).

Also in 2013, Yih et al. [39] took a different approach and instead of using syntactic information proposed a model based on lexical semantic resources. The lexical models used were:

- Degree of Synonymy/Antonymy;
- Hypernymy and Hyponymy;
- Semantic word similarity based on distributed representations.

They used six features to train their models: Identical Word Matching, Lemma Matching, WordNet, Lexical Semantics, Name Entity Matching and Answer Type Checking. Their system greatly outperformed previous works (MAP: 0.7648%).

2.3.2 Answer Extraction

Answer Extraction corresponds to extracting the final answer to the query from the selected passages [1]. Early approaches to Answer Extraction were based on exploiting surface-level features for building or learning patterns. While having a good precision, these mods usually have a low recall. Ravichandran and Hovy [40] proposed an algorithm for learning patterns from examples. For a given question type (e.g. Birthyear) they would store the top 1000 retrieved documents from a web query and select the sentences containing both the question and answer term (e.g. “Mozart” and “1756”). From then a suffix tree was built for each sentence and computed the matching substring between both sentences. The resulting patterns were matching substrings that have both the question and the answer term. Problems

with this approach were the fact that it only supports one question term, low recall and the fact that it cannot model long-distance relationships.

In [41], Ravichandran et al. proposed an extension to this algorithm allowing more than one question term. They derived a set of 22,353 patterns from 16,228 QA pairs and used a maximum-entropy classifier to learn the weights of the patterns.

Peng et al. [42] proposed a hybrid model combining linguistic features and pattern learning. The linguistic features they extracted were:

- Copula - a linking verb (e.g. “is”);
- Apposition - pairs of noun phrases in which one modifies the other (e.g. in the sentence “Lisbon, the capital of Portugal, is a beautiful city”, “Lisbon” is in apposition to “the capital of Portugal”);
- Proposition - represents predicate-argument structures (e.g. eat(John,apple));
- Relations - relationships between two objects (e.g. “spouse-of”, “parent-of”)

They experimented using only the linguistic features and only the learned patterns and concluded that they have a comparable performance. However, using a combination of manual patterns, learned patters and linguistic information increased the performance. Another conclusion they drew was that “what” type questions can be very diverse and some of them do not contain any of the linguistic features they used.

In 2005, Sun and Tan [43] used shallow semantic parsing to solve this problem. For each question and answer sentence they would store a parse tree and at classification time compute the similarity between verb-argument structures in the question and in the answer sentence. From their early experiments they concluded that it was better to take in consideration all arguments together rather than separately and thus use the Jaccard coefficient to measure similarity between the set of arguments from the question and the set of arguments from the answer sentence.

More recently, in 2013, Yao et al. [38] approached the problem of answer extraction as a problem of sequence tagging using CRFs and Tree Edit Distance. The goal is to take all possible answer sentences and label each token as being either in the Beginning, Inside or Outside (BIO) the answer. They define an edit script between trees as the sequence of edits needed to transform one tree into the other. The tree edit distance is the sum of all those costs. As features for training they used the edit distance between the question and the answer as well as features extracted from the question and the edit script.

Also in 2013, Severyn and Moschitti [33] trained a SVM with a tree kernel using shallow syntactic information as features. At classification time, given a question and a candidate answer sentence all noun phrase (NP) nodes are marked as answers (ANS), one at a time. The resulting trees are then classified using the tree kernel and the highest scored one is selected.

2.4 Paraphrase Generation

Paraphrase Generation corresponds to the automatic generation of paraphrases.

In 2017, Iyer et al. [44] use automatically generated paraphrases to expand their training set. They generate paraphrases using Paraphrase Database (PPDB) [45]. PPDB contains millions of paraphrase pairs, in 22 different languages, divided in 6 sets (S–XXXL). Iyer et al. use one-to-one (lexical) and one-to-many (phrasal) paraphrases from the Large paraphrase pack. To expand their training examples they select a random non stop-word and replace it with a random paraphrase. For each training example they generate ate most 3 paraphrases. They evaluate their system on 2 datasets and show that using paraphrases to add examples to the training set increases performance (84.8 vs. 81.8 with and without paraphrases, for a dataset, and 86.2 vs. 84.3, for the other).

In the same year, Dong et al. [46] generate paraphrases using 3 different methods: PPDB, a Neural Machine Translation (NMT) approach and using rules mined from the WikiAnswers corpus [47].

Regarding the PPDB method, Dong et al. use the S pack of paraphrases and focus on Lexical and Phrasal paraphrases. They generate at most 10 paraphrases per sentence.

For their NMT-based approach, they follow the implementation details of Mallinson et al. [48]. They train a bi-directional Recurrent Neural Network on data released as part of the WMT15 shared translation task (4.2 million sentence pairs), and used English/German as the language pair. They used the top 15 decoding results as candidate paraphrases.

Regarding the approach using rules mined from the WikiAnswers corpus, they follow the approach of Fader et al. [49]. The corpus contains millions of clusters labeled as paraphrases by users. They begin by extracting question templates, i.e. questions with at most one wild-card, that appear in at least 10 clusters. They consider as paraphrases templates that co-occur in the same cluster (both appear more than 5 times), and that have the same arguments.

Their results show that using the paraphrases improved their performance. Furthermore, the method that improved performance the most was the NMT approach.

2.5 Natural Language Interface for Databases

A NLIDB is a system where the interaction between the user and the database is performed through Natural Language rather than through a formal language specific for database queries such as the Structured Query Language (SQL). These systems allow non-expert users to query databases without specific knowledge of the relationships and attributes of the database. Furthermore, these interfaces are used with the QAS in problems where the knowledge source is a relational database rather than a full-text database.

According to Nihalani et al. [50] NLIDBs are usually composed of two sub-components:

- Linguistic component - transforms the input into a formal language query (Analysis) and generates an answer based on the results (Answer Generation);
- Database component - performs traditional Database Management functions.

Additionally, there are 4 main approaches to building these systems [50]:

- Based on Pattern Matching;
- Based on Syntax;
- Based on Semantic Grammars;
- Based on Intermediate Representation Languages.

Approaches based on pattern matching were the earliest approaches and relied on patterns to map the input directly into the database terms.

Syntax-based approaches first parse the input and build a tree. The mapping is then done based on the tree nods. However, not all nodes are usually mapped, and it is not always clear which nodes should or not be mapped. Moreover, creating the rules on which the mapping is based on is not always trivial.

Approaches based on semantic grammars are similar to syntax-based approaches as both build a parse tree and map the tree nodes to a query. The idea behind approaches based on semantic grammars is to use some prior knowledge to build the grammar rules in order to simplify the resulting parse tree as much as possible. The main problems with this methodology are requiring prior knowledge and being domain-dependent.

Lastly, approaches based on intermediate representation languages first translate the input into a logical form and then map the logical form into a database query language.

Li and Jagadish [51] proposed a method for building a NLIDB with user feedback throughout the process in order to guarantee the correctness of the query. First, the input is parsed, and a tree is built. Then the system finds the nodes that can be directly mapped into SQL. If any node cannot be mapped, a message is sent to the user. Moreover, if there are multiple mappings for one node then one is picked, and the choice reported to the user. After this step is concluded it is assumed that all tree nodes are understood. However, it is possible that the tree is invalid from the database perspective (e.g. missing the names of tables or attributes). Adjustments are made to the structure of the tree, under user supervision, after which the resulting tree is considered valid is translated to SQL.

2.6 Recommender System

RSs predict or recommend items based on the preferences of the users. In order to make these predictions RSs need to gather user information either explicitly (e.g. through movie ratings) or implicitly

(e.g. through websites visited, links clicked). They are used in many different areas such as movie recommendations, music, documents or e-commerce [52]. Typical approaches to RSs can be classified as [53]:

- Content-based filtering;
- Collaborative filtering;
- Hybrid Filtering.

Content-based Filtering (CBF) makes recommendations based on the profile of the user and the content of the items. For instance, if a user has watched several comedy movies the system is more likely to recommend another comedy instead of an action.

Collaborative Filtering (CF) makes recommendations by predicting the rating of new items based on the rating of users with the same interests. For example, if two users A and B gave a good rating to the same movies and user B also gave a good rating to movie X then the system is likely to recommend movie X to user A.

Hybrid Filtering (HF) combines CF methods and CBF methods. Hybrid methods can be classified as:

- implement collaborative and content-based methods separately and combine their predictions
- incorporate some content-based characteristics into a collaborative approach
- incorporate some collaborative characteristics into a content-based approach
- construct a unifying model that incorporates both content-based and collaborative approaches

Arguello et al. [54] studied different document representations for recommending blogs based on an input query. This differs from regular document recommendation because a blog can be seen as a collection of documents (posts) instead of a single document. Additionally, blogs can cover several different topics whereas documents are usually restricted to a few topics. They propose 2 different representations: Large Document (LD) and Small Document (SD). The LD representation treats the blog as a single document while the SD representation views each post as a document and aggregates their rankings in the end. The LD representation gave much better results than the SD representation.

Liu et al. [55] built a recommender system for sharing documents in personal folder environments. They propose 3 different architectures:

- Content-based Filtering and Classification (CBFC);
- Collaborative Filtering and Classification (CFC);

- Hybrid Filtering and Classification (HFC);

CBFC generates a document profile from its contents, a category profile from the documents in a given category and a user profile based on the content of the documents the user has in their personal folder. Then, similarity between documents and categories is computed and a list of document-category pairs is generated for recommendation.

CFC first generates a category and a user profile. Then, it finds similar users based on the similarity of the category profiles, predicts the ranking for other document-category pairs and generates a list of document-category for recommendation.

HFC is a combination of the previous models. It predicts the ranking of document-category pairs by merging the score given by the CBFC and the CFC models. Their experiments showed that the HFC outperforms both CBFC and CFC.

Chandrasekaran et al. [56] proposed a concept-based document recommendation system for recommending documents based on user publications. Their architecture has 3 components:

- Classifier module - Classifies each document into a set of concepts;
- Profiler module - Creates a user profile by extracting the concepts from the user's published works;
- Recommender module - Searches for documents that have concepts similar to the profile to create a candidate list. Converts the vector representation of the user profile and document into weighted concept trees and computes the tree edit distance between both trees. The 10 closest documents are then recommended.

From their results they concluded that the concept tree algorithm outperforms both keyword similarity and vector similarity methods.

3

Paraphrase Generation

Contents

This chapter discusses the methodology followed in order to build a Paraphrase Generation (PG) module for our Question Answering System. The main objective of this module is to generate paraphrases of the questions inputted by the users, which can be seen as a method of Automatic Query Expansion. We performed an experiment using a Paraphrase Generation framework and publicly available corpora. Since results were poor, we used 2 other methods for paraphrase generation: based on a database of paraphrases (PPDB-based) and based on Neural Machine Translation (NMT). In order to evaluate these approaches we first had to create a question dataset. We conclude this chapter with the evaluation of those 2 methods and the discussion of the results.

3.1 Preliminary experiment

A first experiment to generate paraphrases was made using a Question Generation framework developed by Rodrigues et al. [57]. The framework takes as input sentence/question pairs to create semantic patterns. Questions are then generated by applying the patterns created to new sentences. When a match is found, a question is generated. In our experiment we wanted to see if it was possible to utilize the framework developed by Rodrigues et al. to paraphrase questions, instead of only generating questions from sentences. That is, to input question/question pairs, create semantic patterns and generate questions by matching the patterns with new questions. To do this we used the crowdsourced paraphrase corpora made available by Wang et al. [58] as input seeds to the framework. There are 8 different corpora: basketball; blocks; calendar; geo880; housing; publications; recipes; regex; restaurants and socialnetwork. The corpora were created using Amazon Mechanical Turk (AMT)[1] by having crowd workers (turkers) paraphrase canonical utterances, such as "meeting whose date is at least date of weekly standup" or "player whose number of steals (over a season) is smaller than 3".

We defined 2 experimental cases: canonical/paraphrase and paraphrase/paraphrase. In the first case we used the canonical utterance given to the turkers and one paraphrase as seeds. We used the first 26 pairs of canonical utterance and paraphrase as input. In contrast, in the second case we used two paraphrases generated by the turkers, from the canonical utterances. We used one paraphrase/-paraphrase pair for each canonical utterance. Additionally, to generate new questions we used canonical utterances in the first case and human-generated paraphrases in the second case.

Table 3.1 shows the number of seed pairs we used for each corpus and for each of the experiments.

Table 3.1: Number of seed pairs for each corpus and experiment type

	bball	**blocks**	**cal**	**geo880**	**housing**	**pubs**	**recipes**	**regex**	**rest**	**socialnet**
canon/para	26	26	26	26	26	26	26	26	26	26
para/para	252	463	190	475	228	144	123	1359	339	613

[1] https://www.mturk.com/ (accessed May 30, 2018)

An example of a pair used in the first case is “player whose number of steals (over a season) is smaller than 3” and “name a player with fewer than three steals?”. For the second case, an example of a pair we used is “name a player with fewer than three steals?” and “how many players have less than 3 steals on them in a season?”

Unfortunately, results from preliminary experiments were extremely poor. Countless questions where generated but close to none had any syntactic or semantic correctness.

Examples of generated questions can be seen in Figure 3.1

```
point guard how many assists played a season ?
point guard had of by ?
how many assists in a season played point guard have ?
how many assists in point guard have ?
kobe bryant how many assists had during the a season ?
how many assists has played the point guard ?
how many assists played point guard have in a season ?
how many assists have in a season ?
point guard at least a season ?
how many assists have over the a season ?
how many assists have had the least point guard of in a season ?
how many assists many were played by a season ?
how many assists has the most point guard ?
how many assists season on or ?
season how many assists have over ?
how many assists had of by ?
```

Figure 3.1: Examples of questions generated by the framework

3.1.1 Main problems with this approach

Here we will briefly discuss which were the problems behind this approach and the reasons why it failed.

3.1.1.A Dataset problems

As previously mentioned, we used the corpora made available by Wang et al. as input to this experiment. Furthermore, we had assumed that paraphrased sentences would have the same meaning, but found that it was not always the case.

Figure 3.2 shows that some workers write questions with a different meaning. For the original canonical utterance “season of player kobe bryant whose number of blocks (over a season) is 3” one worker wrote the question “what player had more blocks than kobe bryant whose number of blocks is 3”, which has a completely different meaning from the original. Another worker wrote “in which season did kobe bryant average 3 blocks”. By adding “average” to the question, it also changes its meaning from “total blocks” to “average blocks”.

However, the fault does not always lie with the workers. Some canonical utterances are vague and open to interpretation. This can be seen in Figure 3.3, with the canonical utterance “number of position”.

```
original - season of player kobe bryant whose number of blocks (over a season) is 3
  para - this is considered the season of player kobe bryant with 3 blocks, worker - A19LTHK3TA57SF
  para - what year did kobe bryant have three blocks, worker - A2RRYVYUDNK2GB
  para - in which seasons kob bryant made 3 blocks, worker - A2TUNPP2GOQ1N3
  para - what season did kobe bryant have 3 blocks, worker - A32CIL5S21Y4MR
  para - during which seasons did kobe bryant make 3 blocks, worker - A2ZD8NY75G9JSX
  para - what player had more blocks than kobe bryant whose number of blocks is 3, worker - A2CPBDUGX4QF2C
  para - what seasons did kobe bryant have only three blocks, worker - AUQTHU5AOZVQC
  para - in which season did kobe bryant average 3 blocks, worker - AKY7ZFPCHAXMJ
```

Figure 3.2: Example of original canonical utterance and paraphrases made by workers

While most questions mean “how many different positions are there”, one worker wrote “what is the number of the player in this position”, which has a different meaning.

```
original - number of position
  para - how many different positions are there, worker - A11GZNNVLEWRGF
  para - what is the number of positions, worker - A1L9XASV7HRLBT
  para - how many positions, worker - A2TEWO3WJ5YKGG
  para - what is the number of the position, worker - A32CIL5S21Y4MR
  para - what is the number of the player in this position, worker - A2XRC7P3AORIPH
  para - how many positions are there, worker - A1L09AVOUS47FK
```

Figure 3.3: Example of vague canonical utterance and respective paraphrases made by workers

3.1.1.B Framework limitations

We also found some limitations with the framework. It requires that every predicate argument in the sentence (or first question, in our case) to exist in the question. Taking as an example the questions “player who had less than 3 steals in a season” and “who has had less than 3 steals?”, since “player” does not exist in the second question, no pattern is created. Furthermore, using the question pair “In which seasons did kobe bryant make 3 blocks?” and “Season in which 3 blocks were made by kobe bryant?” as example, in the first sentence, “In which seasons” is labeled as *AM-TMP* denoting a temporal relation, but on the second sentence “Season in which” is labeled as *Loc* relating to a location. As such, no pattern is created between those questions either. Additionally, since the framework is designed to receive as input a sentence/question pair, it requires the first input to be bigger than the second input. While this is not a limitation per se, as it can be overcome by always placing as first input the question with bigger length, it was inconvenient.

3.2 Creating a Dataset for evaluation

Since we wanted to evaluate our paraphrasing approaches we first had to create a questions dataset, to which we would apply our paraphrasing methods, to generate paraphrases.

The development process began by manually writing questions based on the documentation of the dashboard “FXLM Rollover Opportunities”. The documentation of this dashboard consisted of approxi-

mately 6 pages of information, which resulted in 606 different questions.

The next step was to manually group the 606 questions by expected answer, that is, to group the questions such that each group contains only questions that have the same answer – we did this in order to be able to test the quality of our paraphrases as we assume that paraphrases of a question have the same answer. The outcome of this grouping were 130 different groups. Examples of questions from the group with highest frequency are “How can I see what clients are more likely to rollover?”, “How can we find potential clients to speak with to rollover with BNPP?” and “How can I know what clients will rollover?”. The following table and figure present more details regarding the dataset created. In Table 3.2 we can see the number of questions, number of groups, maximum, minimum and mean questions per group, as well as the standard deviation. Figure 3.4 shows the distribution of questions per group, through a bar chart.

Table 3.2: Original Dataset Details

Original Dataset Details	
Nr. of questions	606
Nr. of question groups	130
Max questions per group	72
Min questions per group	1
Average questions per group	4.662
Standard Deviation	7.967

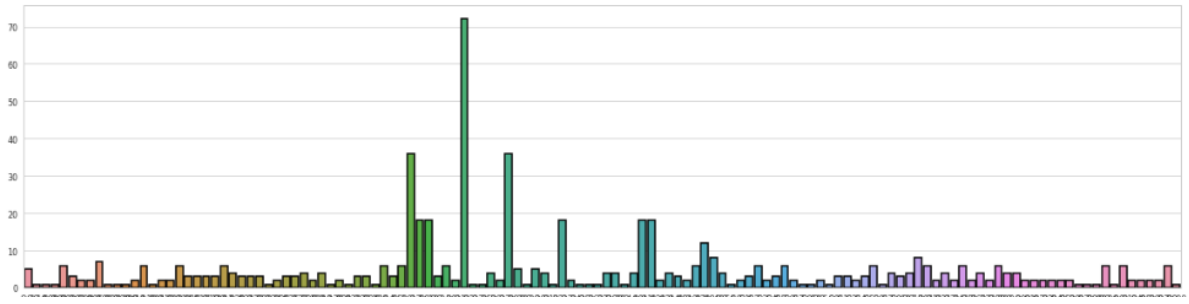


Figure 3.4: Questions per group

3.3 Generation based on the ParaPhrase DataBase

Paraphrase Database (PPDB) [45] is, as the name suggests, a Database of paraphrases. It offers paraphrase packs in 22 different languages in sizes ranging from S (small) to XXXL (extra-extra-extra-large). The paraphrase packs are plain text files with a paraphrase rule per line. The rules for the English pack are formatted as follows:

$$LHS|||PHRASE|||PARAPHRASE|||(FEATURE = VALUE)^*|||ALIGNMENT|||ENTAILMENT$$

Where LHS is the constituent label for the paraphrase pair, PHRASE is an expression, PARAPHRASE is its paraphrase, (FEATURE=VALUE)* is a list of features and respective values, ALIGNMENT is the alignment between the PHRASE and PARAPHRASE, and ENTAILMENT denotes an entailment relation between PHRASE and PARAPHRASE (e.g. Equivalence for the pair restriction/limitation or ForwardEntailment for the pair dog/animal). An example of a PPDB rule can be seen in Figure 3.5

Figure 3.5: Example of a PPDB2.0 rule for the English language

```
[NN] ||| transplant ||| transplantation ||| PPDB2.0Score=5.24981
PPDB1.0Score=3.295900 -logp(LHS|e1)=0.18597 -logp(LHS|e2)=0.14031
-logp(e1|LHS)=11.83583 -logp(e1|e2)=1.80507 -logp(e1|e2,LHS)=1.46728
-logp(e2|LHS)=11.47593 -logp(e2|e1)=1.49083 -logp(e2|e1,LHS)=1.10738
AGigaSim=0.63439 Abstract=0 Adjacent=0 CharCountDiff=5 CharLogCR=0.40547
ContainsX=0 Equivalence=0.371472 Exclusion=0.000344 GlueRule=0
GoogleNgramSim=0.03067 Identity=0 Independent=0.078161 Lex(e1|e2)=9.64663
Lex(e2|e1)=59.48919 Lexical=1 LogCount=4.67283 MVLSASim=NA Monotonic=1
OtherRelated=0.372735 PhrasePenalty=1 RarityPenalty=0
ForwardEntailment=0.177287 SourceTerminalsButNoTarget=0 SourceWords=1
TargetComplexity=0.98821 TargetFormality=0.98464 TargetTerminalsButNoSource=0
TargetWords=1 UnalignedSource=0 UnalignedTarget=0 WordCountDiff=0
WordLenDiff=5.00000 WordLogCR=0 ||| 0-0 ||| OtherRelated
```

Following the work of Dong et al. [46] we focused on **LEXICAL** (single word) and **PHRASAL** (multiword to single/multiword) paraphrases and used the S sized pack for precision.

Instead of using all the rules in the pack, we selected only those with an entailment relation we thought appropriate for paraphrases, namely: Equivalence, ForwardEntailment (hyponym) and ReverseEntailment (hypernmym). A hyponym is "A word of more specific meaning than a general or superordinate term applicable to it" [2] as exemplified above, whereas is the opposite.

Additionally, we also filtered the number of rules based on the features "PPDB1.0Score" and "PPDB2.0Score", and a threshold for those features. We selected the thresholds for filtering after a small empirical experiment.

We chose 4 different thresholds, based on quartiles:

- PPDB1.0Score above second quartile and PPDB2.0Score above second quartile (50-50);
- PPDB1.0Score above third quartile and PPDB2.0Score above second quartile (75-50);
- PPDB1.0Score above second quartile and PPDB2.0Score above third quartile (50-75);
- PPDB1.0Score above third quartile and PPDB2.0Score above third quartile (75-75).

We then filter the rules in two ways:

- By selecting those with PPDB1.0Score and PPDB2.0Score above the threshold (Case A);

[2] https://en.oxforddictionaries.com/definition/hyponym (accessed September 22, 2018)

- By grouping the rules according to their Entailment Relation and then selecting those with PPDB1.0Score and PPDB2.0Score above the threshold, for each group (Case B).

The following tables show some statistics regarding the values of the features we used, as well as the number of rules before and after filtering. Table 3.3 shows the number of rules as well as the max, min, mean and standard deviation of the rules' score relative to lexical rules and Table 3.4 relative to phrasal rules. Table 3.5 contains the number of rules after filtering, for lexical and phrasal rules, and for each filtering case.

Table 3.3: Statistics of lexical rules, grouped by relation

	PPDB1.0Score					PPDB2.0Score				
Relation	Count	Max	Min	Mean	Std	Count	Max	Min	Mean	Std
Equivalence	7,090	27.33	2.15	12.31	4.82	7,090	4.70	3.47	3.74	0.21
ForwardEntailment	6,588	27.33	2.15	12.61	4.87	6,588	4.63	3.47	3.73	0.19
ReverseEntailment	23,011	26.6	1.49	7.89	3.67	23,011	5.22	3.48	3.78	0.25
All Relations	36,689	27.33	1.49	9.59	4.70	36,689	5.22	3.47	3.77	0.23

Table 3.4: Statistics of phrasal rules, grouped by relation

	PPDB1.0Score					PPDB2.0Score				
Relation	Count	Max	Min	Mean	Std	Count	Max	Min	Mean	Std
Equivalence	354,208	27.94	1.39	8.80	5.40	354,208	6.13	4.26	4.54	0.23
ForwardEntailment	251,408	27.99	1.39	12.80	6.14	251,408	6.01	4.26	4.50	0.20
ReverseEntailment	427,632	27.99	1.39	14.45	6.14	427,632	6.03	4.26	4.50	0.20
All Relations	1,033,248	27.99	1.39	12.11	6.45	1,033,248	6.13	4.26	4.51	0.21

Table 3.5: Number of rules after filtering

	Case A				Case B			
	50-50	50-75	75-50	75-75	50-50	50-75	75-50	75-75
Lexical	8304	3656	4429	2038	8414	3820	4130	1867
Phrasal	239145	112052	111554	49230	243758	116770	117819	54559

We then applied each set of rules to our manually created 606 questions and removed duplicates. The number of questions after this step is presented in Table 3.6.

Table 3.6: Number of questions after applying rules

	Case A				**Case B**			
With duplicates	**50-50**	**50-75**	**75-50**	**75-75**	**50-50**	**50-75**	**75-50**	**75-75**
Lexical	1,507	1,040	1,199	920	1,719	1,190	1,116	879
Phrasal	3,564	1,692	1,905	988	3,218	1,347	1,761	1,058
Without duplicates	**50-50**	**50-75**	**75-50**	**75-75**	**50-50**	**50-75**	**75-50**	**75-75**
Lexical	1,046	755	859	723	1,153	844	821	704
Phrasal	2,970	1,412	1,573	837	2,830	1,097	1,483	863

3.4 Generation based on Neural Machine Translation

NMT is the an approach to Machine Translation (MT) that uses Neural Networks (NNs) to translate text or speech from one language to another. We follow the work of Dong et al. [46] to generate paraphrases by translating sentences from English to German and then translating back to English. At first we attempted to use the same library as Dong et al., however we faced many problems due to a conflict of versions between several libraries. We then searched for other NMT libraries and found tensorflow's Tensor2Tensor (T2T) library[3] [59], which we eventually used. More specifically, T2T's implementation of the Transformer model from the paper by Vaswani et al. [60].

We trained 2 models with Hparams "transformer_base" on data released as part of the WMT15 task (4.2 million English and German aligned sentences) for 21k steps each, on 8 K80 GPU's, using Google Cloud[4].

To decode (i.e. translate), we set the parameters "Beam_size" to 10 and "Alpha" to 0.6, after a brief empirical experiment.

An example of a paraphrase generated using this method is "What is a billing date?". The original question was "What is a settlement date?" and the German-translated question was "Was ist ein Abrechnungsdatum?".

Using this method, each original question may, or not, generate a paraphrase since the final translated question may be the same as the original question. Additionally, it is possible that different questions generate the same paraphrase. From the 606 generated paraphrases only 562 were left after removing duplicates. Additionally, from those 562, 232 already existed in the original questions.

3.5 Evaluating our approaches

In order to evaluate our PPDB-based and NMT-based approaches we designed 3 experiments: Big Knowledge Base, Small Knowledge Base and Real Case Simulation. In all experiments the goal was to classify the "test" questions using the questions in our "Knowledge Base (KB)". For example, if we had in our KB a question manually classified as being in group 1 "What is the main goal of the dashboard?" and a question manually classified as being in group 2 "What does the dashboard show?", given a new question "What can I see in the dashboard?" the goal would be to classify it as belonging to group 1 or 2.

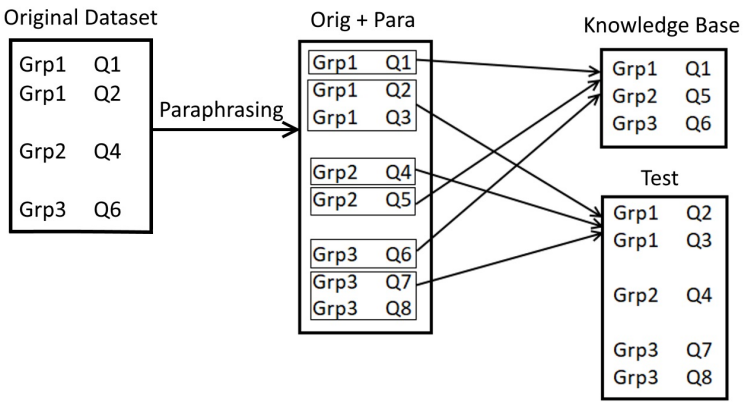
In the first experiment, Big KB, we simulate the setting of having a big knowledge base. We start by generating paraphrases of our original questions. Then, we choose 1 question from each group for testing and use the remaining as our KB. After that, we compute the distance between the test

[3] https://github.com/tensorflow/tensor2tensor (accessed June 22, 2018)
[4] https://cloud.google.com/ (accessed June 24, 2018)

questions and the KB questions and classify the group of each test question as the group of the KB question closest to it.

In the Small KB experiment we begin as in the previous experiment, by generating paraphrases of the original questions, and then do the opposite: choose 1 question from each group for the KB and use the remaining questions for testing. This is better shown in Figure 3.6. After selecting questions for the KB and testing, we compute the distance between the test questions and the KB questions, and select as the group for each of the test questions the group of the KB closest to it, as in the Big KB experiment. This setting can also be seen as a "lazy" setting and more likely to occur in the "real world" as it simulates a Frequently Asked Questions (FAQ) setting, where we have 1 answer for each FAQ.

Figure 3.6: Small KB Setting



The experiment "Real Case Simulation" is similar to the Small KB experiment since we also choose 1 question from each group for the KB and use the remaining for testing. However, in contrast to the Small KB experiment, we first choose the questions for the KB and testing and only then generate paraphrases of the questions in the KB. The process of selecting the questions can be seen in Figure 3.7

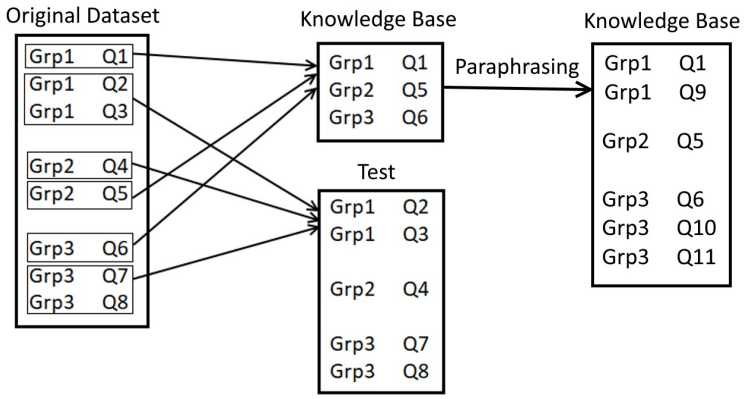
For each experiment we tested using:

- only the manually generated questions;
- only PPDB-based generated paraphrases (each paraphrase set individually);
- only NMT-based paraphrases;
- original questions + best lexical and phrasal PPDB-based paraphrases + NMT-based paraphrases

We classified the questions using 7 different distance metrics based on the Levenshtein Distance [61], Jaccard Distance [62] and Word2Vec [63]:

- Character-wise Levenshtein Distance

Figure 3.7: Real Case Simulation Setting



- Word-wise Levenshtein Distance
- Character-wise Jaccard Distance
- Unigram Jaccard Distance
- Bigram Jaccard Distance
- Trigram Jaccard Distance
- Distance between the sums of Word2Vec Embeddings

3.5.1 Distance Metrics

As previously mentioned, we computed 7 different distance metrics. Here, we will explain how to compute each one of them.

The **Levenshtein distance** is computed between two strings and is given by the minimum number of editions necessary to transform one string into the other. Possible editions are: deletion, insertion and substitution. Each edition has a cost of 1. We compute this distance based on characters and words. For the first case, using as an example the words “dogs” and “log” the Levenhstein distance is obtained as follows:

- dogs $\rightarrow$ logs (substitute ’d’ for ’l’)
- logs $\rightarrow$ log (remove ’s’)

Using the sentences “I love all dogs” and “I have cats”, the Levenshtein distance based on words is computed as follows:

- love $\rightarrow$ have (substitute “love” for “have”)

- all $\rightarrow \emptyset$ (delete "all")
- dogs $\rightarrow$ cats (substitute "dogs" for "cats")

The **Jaccard distance** is used to compute the *dissimilarity* between two sets A and B. It is computed as follows:

$$d_j(A, B) = \frac{|A \cup B| - |A \cap B|}{|A \cup B|} \tag{3.1}$$

We compute the Jaccard distance character-based, unigram-based, bigram-based and trigram-based.

Using the same example as above, the character-based Jaccard distance between "dogs" and "log" is:

$$d_j(\text{“}dogs\text{”}, \text{“}log\text{”}) = \frac{|\{d, o, g, s\} \cup \{l, o, g\}| - |\{d, o, g, s\} \cap \{l, o, g\}|}{|\{d, o, g, s\} \cup \{l, o, g\}|} = \frac{5 - 2}{5} = 0.6$$

N-grams are a contiguous sequence of *n* items. For instance, the bigrams for the sentence "I love all dogs" are ("I", "love"), ("love","all"), ("all","dogs") and trigrams are ("I", "love", "all"), ("love", "all", "dogs").

Word Embeddings are techniques to represent words as dense vectors, instead of a one-hot representation where each word corresponds to a vector of the size of the vocabulary, with all entries set to 0 except for one that is set to 1 (which corresponds to that word, in that vocabulary). For instance, using the previous sentences "I love all dogs" and "I have cats" as example, the vocabulary is $V = \{\text{“}I\text{”}, \text{“}love\text{”}, \text{“}all\text{”}, \text{“}dogs\text{”}, \text{“}have\text{”}, \text{“}cats\text{”}\}$ and the respective one-hot representation is:

- "I" = [1,0,0,0,0,0]
- "love" = [0,1,0,0,0,0]
- "all" = [0,0,1,0,0,0]
- "dogs" = [0,0,0,1,0,0]
- "have" = [0,0,0,0,1,0]
- "cats" = [0,0,0,0,0,1]

The one-hot representation of a sentence is then obtained by simply summing the one-hot vectors of each word. For instance, for the sentence "I love all dogs" the one-hot representation is [1,1,1,1,0,0]. The problem with this representation is that it creates very sparse vectors. Word embeddings address this problem by reducing the dimensionality of the vocabulary.

We used gensim's[5] [64] implementation of Word2Vec [65], using Word2Vec's[6] vectors. We compute the distance between sentences A and B in 2 ways: the cosine distance between the sum of all embed-

[5] https://github.com/RaRe-Technologies/gensim/ (accessed September 1, 2018)
[6] https://code.google.com/archive/p/word2vec/ (accessed September 1, 2018)

dings in sentence A and the sum of all embeddings in sentence B and the cosine distance between the average of all embeddings in sentence A and the average of all embeddings in sentence B.

3.5.2 Results

Since we randomly choose a sample of questions for the knowledge base and testing, for each of the 2 experiments we classify the questions and compute the accuracy of the classification 20 times and average the results. Accuracy is defined as

$$Acc = \frac{TP + TN}{TP + TN + FP + FN}$$

Where TP where TP are correctly identified instances, or True Positives, TN are correctly rejected, or True Negatives, FP are incorrectly identified, or False Positives and FN are the incorrectly rejected instances, or False Negatives.

3.5.2.A Big Knowledge Base

The results for this experiment can be seen in Tables 3.7 to 3.11.

For the original questions, whose results can be seen in Table 3.7, the best distance metric was the Jaccard Distance using bigrams (0.678). However, all except for the Word-based Levenshtein Distance (0.513) and Unigram-based Jaccard Distance (0.603) have comparable performances.

Table 3.7: Big KB Original Questions Results

	Big KB Results
Metrics	**Original Questions**
Lev-char	0.666
Lev-word	0.513
Jac-char	0.655
Jac-uni	0.603
Jac-bi	**0.678**
Jac-tri	0.659
W2V-sum	0.650

PPDB-based lexical results are shown in Table 3.8.

The best result is obtained by the question set “Case A-50-50” and using the Character-based Levenshtein Distance (0.778). Moreover, except for the question set “Case B-75-75”, where the best metric is the Jaccard Distance using bigrams, the best metric was the Levenshtein Distance for characters. Comparing these results with the ones obtained using only the original questions, we can see that except for the “Case B 75-75” paraphrase set, the best result was always better than the best result of the original questions. Furthermore, the overall best result for this experiment – ”Case A 50-50”, improved

Table 3.8: Big KB PPDB Lexical Results

	PPDB Big KB Lexical Results							
	Case A				**Case B**			
Metrics	**50-50**	**50-75**	**75-50**	**75-75**	**50-50**	**50-75**	**75-50**	**75-75**
Lev-char	***0.778***	**0.728**	**0.734**	**0.700**	**0.756**	**0.713**	**0.715**	0.671
Lev-word	0.570	0.545	0.538	0.522	*0.581*	0.540	0.536	0.503
Jac-char	0.694	0.675	0.680	0.647	*0.712*	0.661	0.658	0.648
Jac-uni	0.644	0.627	0.632	0.623	*0.661*	0.633	0.614	0.606
Jac-bi	*0.733*	0.693	0.700	0.692	0.722	0.691	0.676	**0.677**
Jac-tri	*0.692*	0.653	0.638	0.642	0.678	0.640	0.637	0.643
W2V-sum	*0.675*	0.665	0.659	0.658	0.671	0.645	0.651	0.639

the original results by 10% (0.778 vs. 0.678).

The PPDB-based phrasal results can be seen in Table 3.9.

Table 3.9: Big KB PPDB Phrasal Results

	PPDB Big KB Phrasal Results							
	Case A				**Case B**			
Metrics	**50-50**	**50-75**	**75-50**	**75-75**	**50-50**	**50-75**	**75-50**	**75-75**
Lev-char	**0.710**	**0.659**	**0.697**	**0.666**	***0.723***	0.656	**0.682**	0.654
Lev-word	0.572	0.502	0.560	0.515	*0.590*	0.519	0.553	0.499
Jac-char	0.630	0.609	0.641	0.604	*0.661*	0.623	0.637	0.618
Jac-uni	0.645	0.590	0.647	0.620	*0.666*	0.601	0.633	0.600
Jac-bi	0.692	0.654	0.683	0.681	*0.719*	**0.659**	0.679	**0.661**
Jac-tri	0.643	0.613	0.627	0.644	*0.657*	0.627	0.637	0.644
W2V-sum	*0.685*	0.659	0.672	0.656	0.681	0.646	0.657	0.638

Similarly to the lexical results, the best metric is almost always the Levenshtein Distance. When it is not the best metric it still gives a comparable performance. The best overall result is obtained by the "Case B-50-50" question set and the Levenshtein Distance based on characters. Once again, by comparing the results of using phrasal paraphrases with the results of using only the original questions, we see an increase in the best accuracy when using paraphrases (0.723 vs. 0.678).

Regarding the NMT paraphrases, results can be seen in Table 3.10.

Table 3.10: Big KB NMT Results

	Big KB Results
Metrics	**NMT**
Lev-char	0.585
Lev-word	0.473
Jac-char	0.573
Jac-uni	0.560
Jac-bi	0.558
Jac-tri	0.555
W2V-sum	**0.643**

The best metric are Word Vectors. However, results are slightly poorer in comparison with the original

questions (0.650 vs 0.643). This is perhaps due to having less questions in the knowledge base, since the NMT approach generated less paraphrases than the number of original questions.

The results of combining the Original Questions with the best PPDB questions (best lexical – "Case A-50-50" and best phrasal – "Case B-75-50") and NMT questions are shown in Table 3.11.

Table 3.11: Big KB Combination of All Methods Results

	Big KB Results
Metrics	**Original Questions + NMT + PPDB**
Lev-char	**0.819**
Lev-word	0.657
Jac-char	0.716
Jac-uni	0.704
Jac-bi	0.718
Jac-tri	0.675
W2V-sum	0.725

From the results we can see the best metric is the Character-based Levenshtein Distance (0.819). Furthermore, we can conclude that using our paraphrases to populate the Knowledge Base has a positive impact – we see an increase in the accuracy when using the paraphrases and our best overall result is achieved when we combine the original questions with both of our paraphrasing methods. This suggests that having more examples increases the accuracy, which is expected as we have more data in our KB on which to base our classification. However, the quality of the paraphrases is also important seeing that the accuracy for some of the phrasal paraphrases question sets (e.g. Case B-50-75) have accuracy comparable to the original questions and NMT paraphrases (0.659 vs. 0.678 vs. 0.643), while having more questions in the KB (1,097 vs. 606 vs. 562 questions).

3.5.2.B Small Knowledge Base

The results for this experiment can be seen in Tables 3.12 to 3.16.

Table 3.12 shows the results for the original questions. The best metric was Word2Vector (0.682).

Table 3.12: Small KB Original Questions Results

	Small KB Results
Metrics	**Original Questions**
Lev-char	0.648
Lev-word	0.624
Jac-char	0.478
Jac-uni	0.681
Jac-bi	0.667
Jac-tri	0.611
W2V-sum	**0.682**

PPDB-based lexical results are shown in Table 3.13.

Table 3.13: PPDB Small KB Lexical Results

	PPDB Small KB Lexical Results							
	Case A				**Case B**			
Metrics	**50-50**	**50-75**	**75-50**	**75-75**	**50-50**	**50-75**	**75-50**	**75-75**
Lev-char	*0.723*	0.678	0.702	0.679	0.721	0.705	0.700	0.674
Lev-word	*0.711*	0.652	0.695	0.662	0.693	0.680	0.685	0.655
Jac-char	0.587	0.526	0.543	0.540	0.570	*0.588*	0.536	0.536
Jac-uni	***0.753***	0.702	**0.745**	**0.720**	0.729	**0.732**	**0.728**	**0.712**
Jac-bi	*0.727*	0.684	0.726	0.696	0.701	0.716	0.723	0.706
Jac-tri	0.676	0.630	*0.679*	0.649	0.463	0.655	0.672	0.654
W2V-sum	*0.747*	**0.706**	0.719	0.715	**0.744**	0.730	0.723	0.708

The best result is obtained by the question set "Case A-50-50" and using the Unigram-based Jaccard Distance (0.753). We can see that by using lexical paraphrases our accuracy increased, in comparison with the accuracy of using only the original questions (0.753 vs. 0.682). Additionally, the best metric for all the question sets was either the Unigram-based Jaccard Distance or Word2Vector and both have always comparable performance to each other, except for the "Case A-75-50" question set.

The PPDB-based phrasal results can be seen in Table 3.14.

Table 3.14: PPDB Small KB Phrasal Results

	PPDB Small KB Phrasal Results							
	Case A				**Case B**			
Metrics	**50-50**	**50-75**	**75-50**	**75-75**	**50-50**	**50-75**	**75-50**	**75-75**
Lev-char	0.531	0.668	0.659	0.651	0.473	0.607	*0.678*	0.655
Lev-word	0.573	0.676	0.653	0.647	0.488	0.585	*0.691*	0.662
Jac-char	0.365	0.450	0.402	0.455	0.379	0.428	0.370	*0.464*
Jac-uni	0.681	**0.750**	**0.736**	0.697	0.641	0.698	*0.776*	0.736
Jac-bi	**0.699**	*0.747*	0.723	0.701	0.627	0.685	0.729	0.707
Jac-tri	0.610	*0.645*	0.645	0.633	0.581	0.614	0.597	0.613
W2V-sum	0.647	0.742	0.733	**0.716**	**0.642**	**0.692**	***0.788***	**0.741**

The best overall result is obtained by the "Case B-75-50" question set and Word2Vector (0.788). Once again, we see that using paraphrases improves the classification (0.788 vs. 0.682). The best metrics are the Jaccard Distance computed on unigrams and Word2Vector, except for the "Case A-50-50" question set, which is the Bigram-based Jaccard Distance.

Results of the NMT-based approach can be seen in Table 3.15.

The best results, as with the original questions, are obtained with Word Vectors. However, a worst performance than with the original questions is achieved (0.587 vs. 0.682).

The results of combining the Original Questions with the NMT questions and the best PPDB questions (best lexical – "Case A-50-50" and best phrasal – "Case B-75-50") are shown in Table 3.16.

The best metrics, in line with the individual results, are Word2Vector and Unigram Jaccard Distance, with the overall best being Word2Vector (0.703). Using all the paraphrases we managed to obtain a

Table 3.15: Small KB NMT Results

	Small KB Results
Metrics	**NMT**
Lev-char	0.547
Lev-word	0.506
Jac-char	0.458
Jac-uni	0.571
Jac-bi	0.526
Jac-tri	0.492
W2V-sum	**0.587**

Table 3.16: Small KB Combination of All Methods Results

	Small KB Results
Metrics	**Original Questions + NMT + PPDB**
Lev-char	0.600
Lev-word	0.605
Jac-char	0.403
Jac-uni	0.691
Jac-bi	0.638
Jac-tri	0.521
W2V-sum	**0.703**

better performance than by using only the original questions (0.703 vs. 0.682). However, while this result is better than the one obtained using the original questions and NMT paraphrases, it is worse than using only PPDB lexical or phrasal paraphrases. Perhaps if we had not included the NMT paraphrases, which had a worse accuracy than the original questions, we would have achieved a better result. However, we did not test this hypothesis. It is interesting to see than the best metric for the Small KB experiment, a setting where we classify many questions based on a single question per group, are Word Vectors. This means that the classification is not "bound" to the structure of the sentences.

3.5.2.C Simulating a real case

Instead of having a big knowledge base, in a real case it is far more likely to have one question of each group in the knowledge base. As previously mentioned, in this experiment we select one question per group for our knowledge base, expand those questions using our Paraphrase Generation Module (NMT + PPDB generation) and classify the remaining questions. The results can be seen in Table 3.17.

If we compare these results with the ones using only the original questions (seen in Table 3.12), we see that the overall accuracy of the classification decreased but the best results for each case do not differ that much (0.681 vs. 0.657). This might mean that the quality of our generated paraphrases is not the best.

Table 3.17: Real Case Simulation Results

	Results of Simulating a Real Case
Metrics	**Original Questions + NMT + PPDB**
Lev-char	**0.657**
Lev-word	0.606
Jac-char	0.467
Jac-uni	0.618
Jac-bi	0.634
Jac-tri	0.619
W2V-sum	0.653

3.5.3 Effects of pre-processing

After evaluating our 3 experiments we decided to evaluate the effects of pre-processing on the classification as we believed it would yield better results. For instance, without using any pre-processing the occurrences "(Client Contribution)", "Client Contribution", "client contribution" and "(client contribution)" are all different due to differences in casing and characters due to the brackets. While this does not greatly affect the Character-based Levenshtein Distance, others are more affected. We experimented only with the "Small KB" method and using the original questions as well as the NMT and PPDB paraphrases. We set all letters to lower-case, separated symbols with spaces (e.g "that is," $\rightarrow$ "that is ," and "+/-" $\rightarrow$ "+ / -") and removed stop-words using the default stop-word list for the English language from Natural Language Toolkit (NLTK) [66].

The results, which can be seen in Table 3.18, are overall worse than without preprocessing but the best accuracy for both cases is comparable (0.703 without pre-processing and 0.700 with pre-processing). Nonetheless, the results with pre-processing and using the paraphrases are still better than without pre-processing and only the original questions (0.700 vs. 0.682), which, once again, shows that using the paraphrases is beneficial for the classification.

Table 3.18: Small KB Pre-Processing Vs. No Pre-Processing Results

	Pre-processing Vs. No Pre-processing Results	
	Original Questions + NMT + PPDB	
Metrics	**Pre-Processing**	**No Pre-processing**
Lev-char	0.576	0.600
Lev-word	0.591	0.605
Jac-char	0.393	0.403
Jac-uni	**0.700**	0.691
Jac-bi	0.515	0.638
Jac-tri	0.366	0.521
W2V-sum	0.323	***0.703***

It is possible that removing the stop-words decreases the accuracy. Since we used the default stop-word list, the question-words (e.g. "what", "how", etc.) are removed. Thus, some questions are reduced

to only one or two words. For instance the question “what is the maturity date ?” is transformed to “maturity date ?”. Perhaps a better accuracy could be achieved without removing the stop-words or by choosing our own stop-words instead of using NLTK’s. We did not, however, test this.

4

General Solution

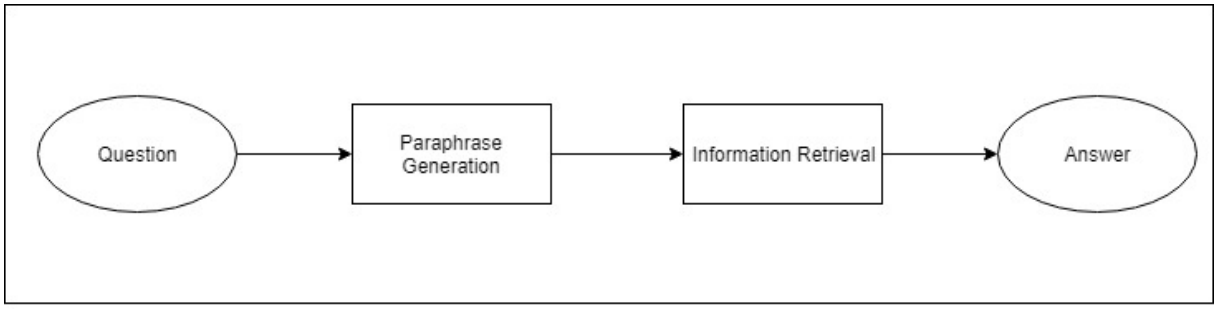
Contents

This chapter presents our solution to build a Questions Answering system capable of solving **BNP!** (**BNP!**)'s problem. We begin by explaining the general architecture of our solution. After that, we propose and evaluate different setups for our solution. Next, we discuss how to enhance our solution by taking advantage of user feedback. Finally, we discuss how to apply our solution to new dashboards.

4.1 General Architecture

Our general architecture, which can be seen in Figure 4.1, is fairly simple.

Figure 4.1: General Architecture



We begin by taking as input a user question. Then, we generate paraphrases of that question using our Paraphrase Generation module, which we presented and evaluated in the previous chapter. The original question and the paraphrases are passed on to the Information Retrieval (IR) module, which returns the final answer to the user question.

Depending on the setup, which we will discuss in more detail in the following section, our IR module may use Elasticsearch, a pre-existing IR engine or perform retrieval using question/question similarity based on the metrics mentioned in the previous chapter. Furthermore, when using ElasticSearch we use as our knowledge source the documentation of the dashboard.

Since we only index and return paragraphs as the answer to the users' questions, we manually create a new documentation, by reorganizing the original documentation, so that the information we consider to be the answer to a question is all in the same paragraph. For instance, consider the question "Into how many steps is the workflow logic split?". While in the original the answer to this question would be split into different paragraphs, one for each step, in the new documentation we put all this information into the same paragraph. Notice that if the answer was contained in different paragraphs it would be difficult to give a correct answer. We use either the original documentation or the new documentation as the knowledge source, when using ElasticSearch, depending on the setup.

When we perform retrieval based on question/question similarity we use the dataset we manually created, extended with paraphrases, as our knowledge source. We classify the user question given our

knowledge source, as we did to evaluate our paraphrase methods in the previous chapter, and then retrieve the answer from a Question Group/Answer database that we manually created.

4.2 Proposed Setups

As mentioned in Chapter 3, we manually created a dataset consisting of 606 questions based on the documentation of a web dashboard, and manually grouped them according to their expected answer. Manually creating and grouping questions from text is a tedious work and involves allocating human resources that could be being used in a different task, perhaps one more relevant to BNP Paribas (BNPP)'s business activities. As such, we present in this chapter different solutions, each requiring more human work than the one before. The solutions we present are:

- H_0: ElasticSearch + Original Documentation
- H_1: Paraphrase Generation + ElasticSearch + Original Documentation
- H_2: ElasticSearch + New Documentation
- H_3: Paraphrase Generation + ElasticSearch + New Documentation
- H_4: Paraphrase Generation + Retrieval Based on Question/Question Distance + Question/Answer Database

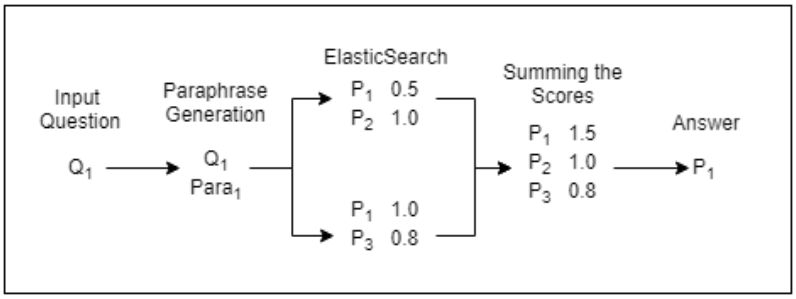
4.2.1 H_0: ElasticSearch + Original Documentation

This solution is our baseline. We index the original documentation by paragraph, using ElasticSearch and, when a question is asked, we perform a query on ElasticSearch and return the paragraph with the highest score.

4.2.2 H_1: Paraphrase Generation + ElasticSearch + Original Documentation

In this solution we index the original documentation as in the previous solution but instead of querying using only the original question we first perform a step of paraphrase generation. We query ElasticSearch with the original questions and paraphrases, with each query yielding a set of paragraphs and the corresponding scores. We sum the scores of the paragraphs individually and return the one with the highest score. An example would be a user asking a question, Q_1 and our system generating a paraphrase $Para_1$. We then query ElasticSearch with Q_1 and $Para_1$, which returns paragraphs P_1 and P_2 with scores 0.5 and 1.0, for Q_1, and paragraphs P_1 and P_3 with scores 1.0 and 0.8, for $Para_1$. Summing the scores for each paragraph returns 1.5 for P_1, 1.0 for P_2 and 0.8 for P_3. As such, we return P_1 as the answer to Q_1. This example can be seen below in Figure 4.2.

Figure 4.2: H_1 Example



4.2.3 H_2: ElasticSearch + New Documentation

This solution is the same as H_0, but using the new documentation as the knowledge source.

4.2.4 H_3: Paraphrase Generation + ElasticSearch + New Documentation

This solution is the same as H_1, but using the new documentation as the knowledge source.

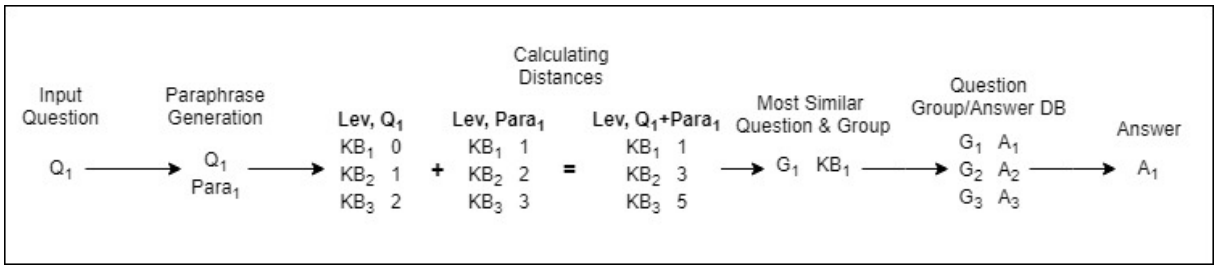
4.2.5 H_4: Paraphrase Generation + Retrieval Based on Question/Question Distance + Question Group/Answer Database

This solution corresponds to the Big Knowledge Base experimental case described in the evaluation section of Chapter 3 (Section 3.5). It uses as the Knowledge Base the question dataset we created manually (question + question group) and the paraphrases we generated using our Paraphrase Generation Module (Big Knowledge Base (KB) setting). Additionally, we manually created a Question Group/Answer Database which we use to find the answer to the user question.

We use what we learned from our evaluation in the previous chapter and only use the paraphrase sets that gave us the best results, for the Big KB case (original + best lexical – "Case A 50-50" + best phrasal – "Case B 75-50" + NMT). Furthermore, we noticed that the best accuracy for this experiment was achieved using the Character-based Levenshtein Distance. As such, we discarded all other metrics and base our classification only on the Levenshtein Distance.

To retrieve the answer to a user question we first generate paraphrases of the input question. Then, we compute the distances between the original question and its paraphrases, and the questions in our Knowledge Base. In the following step, we sum the all the distances for the original question and respective paraphrases, and select the most similar question (i.e. smallest distance). We assume that the group of the user question is the same of the most similar question and use the Question Group/Answer Database to return the answer to the question. An example can be seen below in Figure 4.3.

Figure 4.3: H_4 Example



4.3 Results and Discussion

In this section we present and discuss the results of the solutions shown in the previous section.

We manually evaluate solutions H_0 through H_3 using a sample of 20 questions from the dataset we created. For solutions where ElasticSearch is used (H_0 – H_3) we check if the top ranked paragraph is the correct and if the correct answer is in the top 5 ranked paragraphs. For H_4 we consider the accuracy of this solution as the best accuracy of the Big Knowledge Base setting in Chapter 3. Since we assume that the Question Group/Answer Database we created is correct, evaluating this solution corresponds to evaluating the classification of the questions and paraphrases, which is what was evaluated in Chapter 3.

The results can be seen in Table 4.1.

Table 4.1: Solutions Results

	Accuracy	Top-5 Accuracy
H_0	0.65	0.80
H_1	0.65	0.80
H_2	0.70	0.90
H_3	0.70	0.90
H_4	0.82	No data

In order to evaluate the impact of paraphrasing in the retrieval process we must compare the results of H_0 and H_1, and the results of H_2 and H_3. From those results, we see that the accuracy is the same with or without paraphrasing. However, upon inspecting the number of generated paraphrases for each question we find that our system was unable to generate a single paraphrase for 8 of the 20 questions. Furthermore, for the questions that our H_0 and H_1 setups wrongly classified (7 questions out of 20), our system generated 1 paraphrase for 1 of the questions and 0 for the rest. Regarding the wrongly classified questions for the H_2 and H_3 setups (6 out of 20), our system generated 2 paraphrases for 1 question and 0 for the remaining questions. As such, it is hard to evaluate the impact of paraphrasing in the retrieval process. More statistics regarding the number of paraphrases generated per question can be seen in Table 4.2

It seems our system is capable of generating multiple paraphrases for a single question, but, in

Table 4.2: Statistics regarding the number of paraphrases per question

	Number of Paraphrases per Question						
Questions	**Count**	**Max.**	**Min.**	**Mean**	**Median**	**Mode**	**Std. Dev.**
All questions	20	16	0	3.45	1.5	0	4.893
Wrongly classified H_0 & H_1	7	1	0	0.143	0	0	0.378
Wrongly classified H_2 & H_3	6	2	0	0.333	0	0	0.817

turn, it fails to generate paraphrases for many questions. This is perhaps due to the fact that we use domain-general rules, while our vocabulary is domain-specific.

Comparing the H_0 results with the H_2 results, we see that using our new documentation improved not only the accuracy but also the top-5 accuracy, which means that while the answer is not the paragraph with the highest score, it is contained in the top-5 paragraphs with the highest score.

Our best overall accuracy is given by the H_4 setup, which greatly outperforms the other setups. Notice that the accuracy achieved in chapter 3, which we also report as the accuracy of our H_4 solution in Table 4.1, is obtained considering that the test questions are not present in the Knowledge Base. In practice, in a Frequently Asked Questions (FAQ) setting, that is not always the case. Users tend to ask multiple times the same question.

We can conclude that paraphrases improve the accuracy when we use them to populate a Knowledge Base of questions. However, we can not take any conclusions from the impact of paraphrases on the accuracy of a retrieval engine since our system was unable to generate paraphrases for 8 questions from our sample of 20. Perhaps if we had used another sample of questions, or a bigger sample, we would have achieved different results.

4.4 Enhancing H_4 with User Feedback

In this section we propose a method of enhancing our H_4 setup by taking advantage of user feedback.

As previously mentioned, in this solution we first take a user question as input, generate paraphrases and find the closest question in our Knowledge Base to the original question and its paraphrases. Using the group of that question, the one in our KB, we retrieve the answer from our manually-created Question Group/Answer Database.

After showing the user the answer to their query we ask for feedback: “Was this answer helpful?”, a “yes” or “no” question.

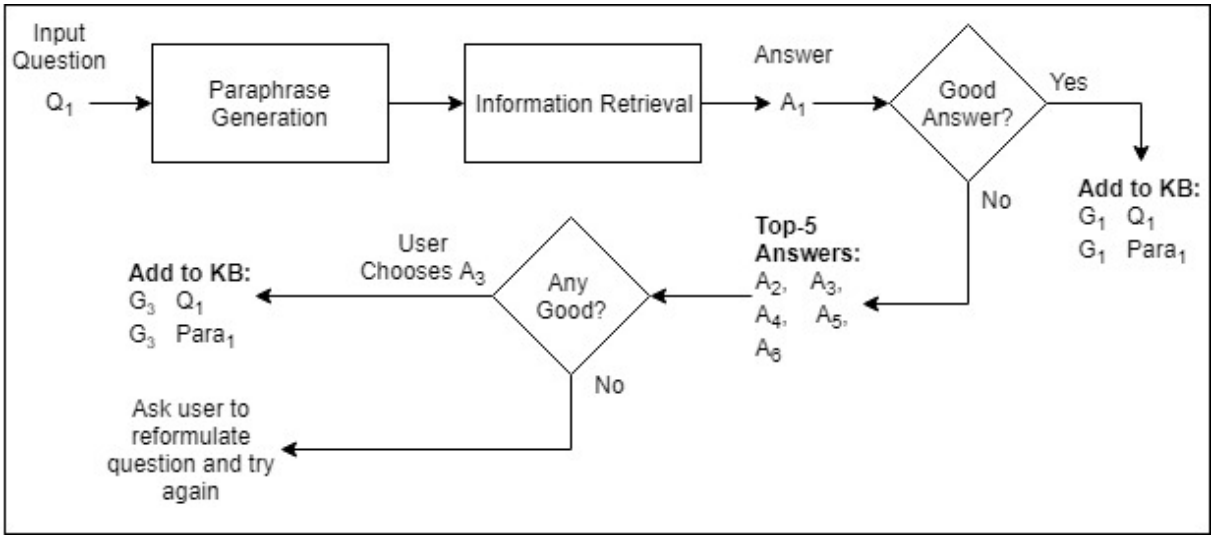
If the user answers affirmatively, the user question and its paraphrases are added to the Knowledge Base, with the group associated with that answer in the Question Group/Answer Database.

On the other hand, if the user answers negatively, we show them the top-5 answers, corresponding to the top-5 closest question groups and ask the user if any of those answers is satisfactory. If so, we add the question and paraphrases to the KB, with the group associated with the answer chosen by the

user. Otherwise, we ask the user to reformulate their question and try again. This process is illustrated in Figure 4.4.

While we expect the results to improve with the addition of user feedback, we were unable to evaluate this.

Figure 4.4: Example of User Feedback



4.5 Applying the setups to new dashboards

An important part of this work is the "plug and play" capability of the setups. Since **BNP!** has many dashboards it is important to be able to apply the setups as automatically as possible, while retaining a good accuracy.

All in all we:

- Created a question dataset and grouped the questions (around 12 hours)
- Reorganized the documentation (around 3 hours)
- Created a Question Group/Answer Database (around 2 hours – requires dataset and grouped questions)

However, not all setups require these steps. A table showing which steps are necessary for each of the proposed setups can be seen in Table 4.3

If an effort is made to create a question dataset, a question group/answer database and reorganize the documentation, then the best solution is H_4 since it had the best accuracy in our evaluation, which we also expect to increase with user feedback. If, on the other hand, only an effort to reorganize

Table 4.3: Necessary steps for each solution

	Reorganize Documentation	**Create Dataset**	**Create Question Group/Answer Database**
H_0	No	No	No
H_1	No	No	No
H_2	Yes	No	No
H_3	Yes	No	No
H_4	Yes	Yes	Yes

the documentation is made, then the best solution is H_3 or H_4, since we could not conclude anything regarding the usage of paraphrases in the process of retrieval. Finally, if no effort is made, the best solution is H_0 or H_1.

We recommend the final solution, H_4. However, it is up to whoever wants to create a Question Answering System for a new dashboard to decide which of the solutions to apply, given the results of each solution and manual work associated.

5

Conclusion

Contents

With this work we presented a Question Answering System capable of answering the questions of the dashboard's users – a Frequently Asked Questions (FAQ) problem. Since we lacked data, we created a Paraphrase Generation (PG) module to automatically generate paraphrases from our data, and use them to populate a Knowledge Base (KB). Our final solution leverages the power of the automatically generated paraphrases and user feedback to greatly improve our baseline, and learn over time.

From our evaluation we concluded that using our generated paraphrases improved the accuracy of the system when used to populate the KB, but could not conclude anything when we applied the PG module to single questions, for querying. This was due to the fact that we were unable to generate a single paraphrase for 8 of the 20 questions we used on our manual evaluation.

In order to develop and evaluate our system we created a question dataset, based on the documentation of the dashboard, where questions were grouped based on their expected answer (i.e. questions with the same answer belong to the same group). Furthermore, since we return paragraphs as the answer to users' questions, we reorganized the original documentation so that information we considered was the answer to a possible user question, and was contained in multiple paragraphs, would be contained in a single paragraph. Finally, and to support our final solution, we created a Question Group/Answer Database.

Since BNP Paribas (BNPP) has many dashboards, it was important to develop a system that was able to be easily applied to a different setting. With that in mind, we proposed different setups based on the amount of work required to implement the system. While it is up to whoever wants to create a Question Answering System (QAS) for a new dashboard to choose which solution to use, we suggest our final solution, as it is the only one to take advantage of paraphrases and user feedback.

5.1 Contributions

We developed and evaluated a Paraphrase Generation Module based on rules (Paraphrase Database (PPDB)) and based on Neural Machine Translation (NMT). We evaluated several distance metrics on a Paraphrase Detection task (part of the PG Module evaluation). We proposed and evaluated several solutions to build a Question Answering System based on the amount of work required to implement the QAS. Our final solution, despite not evaluating all the aspects of it, had the best accuracy of all the solutions we proposed.

5.2 Limitations

There are several limitations with the solutions we proposed, which can be organized into:

- Data Limitations

- Paraphrase Generation Limitations
- Retrieval Limitations

Regarding the **Data Limitations**, all the work involving the manual creation of the dataset, reorganization of the original documentation and creation of the Question Group/Answer Database, was performed by the same person. This may lead to a bias and/or human error. Furthermore, due to the way that the dataset was created, questions that belong to the same group are, almost always, already lexical paraphrases (i.e. only 1 word changes). Finally, if a question whose answer is in the documentation is asked, but that we have not considered and have not added to our dataset and to our question group/answer database, the system fails to answer correctly.

Regarding the **Paraphrase Generation Limitations**, as already mentioned, we only generate lexical and phrasal paraphrases – no changes are made to the structure of the questions. Furthermore, our system has a difficulty in generating paraphrases for some questions, as highlighted by our manual evaluation in Chapter 4. This is because we use domain-general rules in our PPDB-based approach and trained our NMT model on data from a general domain, whereas our corpus is domain-specific. Related to this, our NMT method fails to translate some questions to German and, as such, fails to generate paraphrases.

Regarding the **Retrieval Limitations**, we only consider the Levenshtein Distance in our final solution. If two questions have the same answer meaning but are written very differently, for instance the words being in a different order, our system is very likely to give a wrong answer. This also applies to acronyms (e.g. “CC” in the question and “Client Contribution” in the KB) and numerals (e.g. “What is step 1?” and “What is the first step?”). Finally, it is impossible to answer questions whose answer is spread across multiple paragraphs in the documentation.

5.3 Future Work

This system can be a target of many improvements. As future work it would be interesting to experiment with different methods of generating paraphrases. For instance, acronym expansion or by reorganizing the sentences syntactically. Additionally, we believe that creating domain-specific paraphrasing rules, either manually or automatically, may improve the system’s performance. A more thorough evaluation of the NMT-based method of paraphrasing should be conducted. Experiments with different distance metrics should be made, as well as, experiments with different ways of selecting the final answer, given a set of answers. Relating to the final solution we proposed, it must be properly evaluated with real users to see the impact of user feedback and to understand if the system was capable of learning. Coupling this system with a Natural Language Interface for Databases (NLIDB) would be interesting to answer different types of questions. It would also be interesting to have the history of questions for each user,

base the answers on that history and recommend dashboards to users based on the question they make and their past questions. To this end, we have already reviewed some literature in Chapter 2 regarding NLIDBs and Recommender Systems (RSs), which may provide useful.

Bibliography

[1] A. M. N. Allam and M. H. Haggag, "The question answering systems: A survey," *International Journal of Research and Reviews in Information Sciences (IJRRIS)*, vol. 2, no. 3, pp. 211–220, 2012. [Online]. Available: https://www.researchgate.net/profile/Ali_Allam/publication/281969283_The_Question_Answering_Systems_A_Survey/links/5600003308ae07629e51fe42.pdf%5Cnhttp://aliallam.com/QASurveyPaper(IJRRIS).pdf

[2] P. Gupta and V. Gupta, "A Survey of Text Question Answering Techniques," *International Journal of Computer Applications*, vol. 53, no. 4, pp. 1–8, 2012. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.258.7801&rep=rep1&type=pdf

[3] X. Li and D. Roth, "Learning question classifiers," in *Proceedings of the 19th international conference on Computational linguistics* -, vol. 1. Morristown, NJ, USA: Association for Computational Linguistics, 2002, pp. 1–7. [Online]. Available: http://www.aclweb.org/anthology/C02-1150http://portal.acm.org/citation.cfm?doid=1072228.1072378

[4] D. Zhang and W. S. Lee, "Question classification using support vector machines," *SIGIR '03: Proceedings of the 26th annual international ACM SIGIR conference on Research and development in information retrieval*, pp. 26–32, 2003. [Online]. Available: http://www.kernel-machines.org/papers/upload_29525_p31189-zhang.pdf

[5] Z. Huang, M. Thint, and Z. Qin, "Question classification using head words and their hypernyms," in *Proceedings of the Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 2008, pp. 927–936. [Online]. Available: https://pdfs.semanticscholar.org/94a9/af119df61f501980cf095700f35c2a7762a3.pdfhttps://dl.acm.org/citation.cfm?id=1613835

[6] G. A. Miller, "WordNet: a lexical database for English," *Communications of the ACM*, vol. 38, no. 11, pp. 39–41, 11 1995. [Online]. Available: http://portal.acm.org/citation.cfm?doid=219717.219748

[7] T. Pedersen, S. Patwardhan, and J. Michelizzi, "WordNet::Similarity - Measuring the relatedness of concepts," *Proceedings - Nineteenth National Conference on Artificial Intelligence (AAAI-2004):*

Sixteenth Innovative Applications of Artificial Intelligence Conference (IAAI-2004), pp. 1024–1025, 2004. [Online]. Available: http://www.scopus.com/inward/record.url?eid=2-s2.0-9444261848&partnerID=40&md5=4e9bffbd9bab71be4f29b2b6da51dddc

[8] J. Silva, L. Coheur, A. C. Mendes, and A. Wichert, "From symbolic to sub-symbolic information in question classification," *Artificial Intelligence Review*, vol. 35, no. 2, pp. 137–154, 2011.

[9] S. M. Beitzel, E. C. Jensen, A. Chowdhury, and O. Frieder, "Varying Approaches to Topical Web Query Classification," *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval - SIGIR '07*, pp. 783–784, 2007. [Online]. Available: http://ai2-s2-pdfs.s3.amazonaws.com/3a81/42c1d16cbb2b8f929dbb4ca10425bd112ece.pdf

[10] S. M. Beitzel, E. C. Jensen, O. Frieder, D. D. Lewis, A. Chowdhury, and A. Kołcz, "Improving automatic query classification via semi-supervised learning," *Proceedings - IEEE International Conference on Data Mining, ICDM*, pp. 42–49, 2005. [Online]. Available: https://pdfs.semanticscholar.org/ef98/07b6281ace577480e6fc31a615212c8c3108.pdf

[11] D. Shen, J.-T. Sun, Q. Yang, and Z. Chen, "Building bridges for web query classification," *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, pp. 131–138, 2006. [Online]. Available: http://doi.acm.org/10.1145/1148170.1148196

[12] Z. Kozareva and A. Montoyo, "Paraphrase Identification on the Basis of Supervised Machine Learning Techniques," *Advances in Natural Language Processing: 5th International Conference on NLP (FinTAL 2006)*, pp. 524–533, 2006. [Online]. Available: http://link.springer.com/chapter/10.1007/11816508_52

[13] B. Dolan, C. Quirk, and C. Brockett, "Unsupervised construction of large paraphrase corpora," *Proceedings of the 20th international conference on Computational Linguistics - COLING '04*, p. 350, 2004. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1220355.1220406

[14] S. Fernando and M. Stevenson, "A Semantic Similarity Approach to Paraphrase Detection," *Proceedings of the 11th Annual Research Colloquium of the UK Special Interest Group for Computational Linguistics (CLUK 2008)*, pp. 45–52, 2008. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.144.4680&rep=rep1&type=pdf

[15] J. J. Jiang and D. W. Conrath, "Semantic Similarity Based on Corpus Statistics and Lexical Taxonomy," *Proceedings of International Conference Research on Computational Linguistics*, pp. 19–33, 9 1997. [Online]. Available: http://arxiv.org/abs/cmp-lg/9709008

[16] N. Madnani, J. Tetreault, and M. Chodorow, “Re-examining Machine Translation Metrics for Paraphrase Identification,” *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL HLT ’12)*, pp. 182–190, 2012. [Online]. Available: http://dl.acm.org/citation.cfm?id=2382055

[17] S. Harabagiu, D. Moldovan, and M. Pasca, “Falcon: Boosting knowledge for answer engines,” *Proceedings of the Ninth Text REtrieval Conference (TREC 2000)*, pp. 479–488, 2000. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=EDB98637B28815625DB28B2F514B4E80?doi=10.1.1.12.1766&rep=rep1&type=pdfhttp://trec.nist.gov/pubs/trec9/papers/smu.pdf

[18] H. Cui, J. R. Wen, J. Y. Nie, and W. Y. Ma, “Query expansion by mining user logs,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 15, no. 4, pp. 829–839, 2003.

[19] C. Carpineto and G. Romano, “A Survey of Automatic Query Expansion in Information Retrieval,” *ACM Computing Surveys*, vol. 44, no. 1, pp. 1–50, 2012. [Online]. Available: https://www.iro.umontreal.ca/~nie/IFT6255/carpineto-Survey-QE.pdfhttp://dl.acm.org/citation.cfm?doid=2071389.2071390

[20] S. Gauch and R. P. Futrelle, “Experiments in automatic word class and word sense identification for information retrieval,” *Proceedings of the 3rd Annual Symposium on Document Analysis and Information Retrieval*, pp. 425–434, 1993. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.48.8876&rep=rep1&type=pdf

[21] Y. Qiu, Y. Qiu, H.-P. Frei, and H.-P. Frei, “Concept based query expansion,” *Proceedings of the 16th annual international ACM SIGIR conference on Research and development in information retrieval - SIGIR ’93*, pp. 160–169, 1993. [Online]. Available: http://portal.acm.org/citation.cfm?doid=160688.160713

[22] C. J. Crouch and B. Yang, “Experiments in Automatic Statistical Thesaurus Construction,” *15th Annual International SIGIR*, pp. 77–88, 1992. [Online]. Available: https://pdfs.semanticscholar.org/ca56/018ed7042d8528b5a7cd8f332c5737b53b1f.pdf

[23] J. Xu and W. B. Croft, “Query expansion using local and global document analysis,” in *Proceedings of the 19th annual international ACM SIGIR conference on Research and development in information retrieval - SIGIR ’96*, vol. 19. New York, New York, USA: ACM Press, 1996, pp. 4–11. [Online]. Available: http://portal.acm.org/citation.cfm?doid=243199.243202http://www.eng.utah.edu/~cs7961/papers/XuCroft-SIGIR96.pdf

[24] B. Billerbeck and J. Zobel, “Techniques for Efficient Query Expansion,” in *String Processing and Information Retrieval*, 2004, pp. 30–42. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.63.8466&rep=rep1&type=pdf

[25] B. Billerbeck and H. E. Williams, “Query Expansion using Associated Queries,” *CIKM ’03 Proceedings of the twelfth international conference on Information and knowledge management*, pp. 2–9, 2003. [Online]. Available: http://portal.acm.org/citation.cfm?doid=956863.956866

[26] R. Jones, B. Rey, O. Madani, and W. Greiner, “Generating query substitutions,” in *Proceedings of the 15th international conference on World Wide Web - WWW ’06*, no. March. New York, New York, USA: ACM Press, 2006, pp. 387–396. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1135777.1135835

[27] R. Kraft and J. Zien, “Mining anchor text for query refinement,” in *Proceedings of the 13th conference on World Wide Web - WWW ’04*. New York, New York, USA: ACM Press, 2004, pp. 666–674. [Online]. Available: http://portal.acm.org/citation.cfm?doid=988672.988763http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.1.8820&rep=rep1&type=pdf

[28] S. Riezler, A. Vasserman, I. Tsochantaridis, V. Mittal, and Y. Liu, “Statistical Machine Translation for Query Expansion in Answer Retrieval,” *Annual Meeting- Association for Computational Linguistics*, vol. 1, pp. 464–471, 2007. [Online]. Available: http://www.stefanriezler.com/PAPERS/ACL07.pdf

[29] J. P. Callan, “Passage-Level Evidence in Document Retrieval,” in *Proceedings of the 17th annual international ACM SIGIR conference on Research and development in information retrieval*, 1994, pp. 302–310. [Online]. Available: http://link.springer.com/10.1007/978-1-4471-2099-5_31

[30] S. Tellex, B. Katz, J. Lin, A. Fernandes, and G. Marton, “Quantitative evaluation of passage retrieval algorithms for question answering,” *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval - SIGIR ’03*, pp. 41–47, 2003. [Online]. Available: http://portal.acm.org/citation.cfm?doid=860435.860445

[31] I. Robert and R. Gaizauskas, “Evaluating passage retrieval approaches for question answering,” *Advances in Information Retrieval*, vol. 2997, pp. 72–84, 2004. [Online]. Available: https://pdfs.semanticscholar.org/42ba/d33ba573445cc7b70e78be9dab6514ddf689.pdf

[32] S. Robertson, S. Walker, S. Jones, M. Hancock-Beaulieu, and M. Gatford, “Okapi at TREC-3,” p. 109–126, 1994. [Online]. Available: http://pdf.aminer.org/000/630/294/okapi_at_trec.pdf

[33] A. Severyn and A. Moschitti, “Automatic Feature Engineering for Answer Selection and Extraction,” *Empirical Methods in Natural Language Processing*, vol. 13, no. October, pp. 458–467, 2013. [Online]. Available: http://disi.unitn.it/~severyn/papers/emnlp2013.pdf

[34] M. Wang, N. A. Smith, and T. Mitamura, "What is the Jeopardy Model? A Quasi-Synchronous Grammar for QA," in *Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, 2007, pp. 22–32. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.81.5229

[35] D. A. Smith and J. Eisner, "Quasi-synchronous grammars: alignment by soft projection of syntactic dependencies," pp. 23–30, 2006. [Online]. Available: https://dl.acm.org/citation.cfm?id=1654655

[36] M. Wang and C. D. Manning, "Probabilistic Tree-Edit Models with Structured Latent Variables for Textual Entailment and Question Answering," *Proceedings of the 23rd International Conference on Computational Linguistics (Coling 2010)*, no. August, pp. 1164–1172, 2010. [Online]. Available: https://nlp.stanford.edu/pubs/wang-manning-coling10.pdf

[37] M. Heilman and N. a. Smith, "Tree edit models for recognizing textual entailments, paraphrases, and answers to questions," *The 2010 Annual Conference of the North American Chapter of the ACL*, pp. 1011–1019, 2010. [Online]. Available: http://dl.acm.org/citation.cfm?id=1858143

[38] X. Yao, B. Van Durme, C. Callison-Burch, and P. Clark, "Answer Extraction as Sequence Tagging with Tree Edit Distance," *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, no. June, pp. 858–867, 2013. [Online]. Available: http://www.aclweb.org/anthology/N13-1106http://www.aclweb.org/old_anthology/N/N13/N13-1106.pdf

[39] W.-t. Yih, M.-W. Chang, C. Meek, and A. Pastusiak, "Question Answering Using Enhanced Lexical Semantic Models," *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1744–1753, 2013. [Online]. Available: http://www.aclweb.org/anthology/P13-1171

[40] D. Ravichandran and E. Hovy, "Learning surface text patterns for a Question Answering system," *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics - ACL '02*, p. 41, 2001. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1073083.1073092

[41] D. Ravichandran, A. Ittycheriah, and S. Roukos, "Automatic derivation of surface text patterns for a maximum entropy based question answering system," *Proceedings of the HLT-NAACL Conference*, no. May 2003, pp. 85 – 87, 2003. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.6.2418

[42] F. Peng, R. Weischedel, A. Licuanan, and J. Xu, "Combining Deep Linguistics Analysis and Surface Pattern Learning: A Hybrid Approach to Chinese Definitional Question Answering," *Proceedings of Human Language Technology Conference and Conference on Empirical*

Methods in Natural Language Processing, no. October, pp. 307–314, 2005. [Online]. Available: http://www.aclweb.org/anthology/H/H05/H05-1039

[43] R. Sun, J. Jiang, Y. F. Tan, H. Cui, T.-S. Chua, and M.-Y. Kan, "Using Syntactic and Semantic Relation Analysis in Question Answering," in *Proceedings of the 14th Text REtrieval Conference (TREC 2005)*, 2005. [Online]. Available: http://www.cuihang.com/publications/nus.qa.2005.pdf

[44] S. Iyer, I. Konstas, A. Cheung, J. Krishnamurthy, and L. Zettlemoyer, "Learning a Neural Semantic Parser from User Feedback," 2017. [Online]. Available: http://www.upwork.comhttp://arxiv.org/abs/1704.08760

[45] J. Ganitkevitch, B. Van Durme, and C. Callison-Burch, "PPDB : The Paraphrase Database," *Proceedings of NAACL-HLT*, no. June, pp. 758–764, 2013. [Online]. Available: http://paraphrase.org.http://cs.jhu.edu/~ccb/publications/ppdb.pdf

[46] L. Dong, J. Mallinson, S. Reddy, and M. Lapata, "Learning to Paraphrase for Question Answering," *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pp. 875–886, 2017. [Online]. Available: http://aclweb.org/anthology/D17-1091http://arxiv.org/abs/1708.06022

[47] A. Fader, L. Zettlemoyer, and O. Etzioni, "Open question answering over curated and extracted knowledge bases," in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '14*, 2014, pp. 1156–1165. [Online]. Available: http://dx.doi.org/10.1145/2623330.2623677http://dl.acm.org/citation.cfm?doid=2623330.2623677

[48] J. Mallinson, R. Sennrich, and M. Lapata, "Paraphrasing Revisited with Neural Machine Translation," in *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, vol. 1, 2017, pp. 881–893. [Online]. Available: http://aclweb.org/anthology/E17-1083http://www.aclweb.org/anthology/E17-1083

[49] A. Fader, L. Zettlemoyer, and O. Etzioni, "Paraphrase-Driven Learning for Open Question Answering," pp. 1608–1618, 2013.

[50] M. N. Nihalani, S. Silakari, and M. Motwani, "Natural language Interface for Database: A Brief review," *IJCSI International Journal of Computer Science Issues ISSN*, vol. 8, no. 2, 2011. [Online]. Available: www.IJCSI.org

[51] F. Li and H. V. Jagadish, "Constructing an interactive natural language interface for relational databases," *Proceedings of the VLDB Endowment*, vol. 8, no. 1, pp. 73–84, 2014. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2735461.2735468

[52] J. Bobadilla, F. Ortega, A. Hernando, and A. Gutiérrez, "Recommender systems survey," *Knowledge-Based Systems*, vol. 46, pp. 109–132, 2013. [Online]. Available: http://irntez.ir/wp-content/uploads/2016/12/sciencedirec.pdf

[53] G. Adomavicius and A. Tuzhilin, "Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions," *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, no. 6, pp. 734–749, 2005. [Online]. Available: http://pages.stern.nyu.edu/~atuzhili/pdf/TKDE-Paper-as-Printed.pdf

[54] J. Arguello, J. L. Elsas, J. Callan, and J. G. Carbonell, "Document Representation and Query Expansion Models for Blog Recommendation," *Proceedings of the 2nd International Conference on Weblogs and Social Media ICWSM*, pp. 10–18, 2008. [Online]. Available: http://www.aaai.org/Papers/ICWSM/2008/ICWSM08-009.pdf

[55] D. R. Liu, C. H. Lai, and C. W. Huang, "Document recommendation for knowledge sharing in personal folder environments," *Journal of Systems and Software*, vol. 81, no. 8, pp. 1377–1388, 2008. [Online]. Available: https://ir.nctu.edu.tw/bitstream/11536/8483/1/000258800900008.pdf

[56] K. Chandrasekaran, S. Gauch, P. Lakkaraju, and H. P. Luong, "Concept-based document recommendations for CiteSeer authors," in *Adaptive Hypermedia and Adaptive Web-Based Systems*. Springer, 2008, pp. 83–92.

[57] H. Rodrigues, L. Coheur, and E. Nyberg, "Improving question generation with the teacher's implicit feedback," pp. 301–306, 2018.

[58] Y. Wang, J. Berant, and P. Liang, "Building a Semantic Parser Overnight," *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 1332–1342, 2015. [Online]. Available: http://aclweb.org/anthology/P15-1129

[59] A. Vaswani, S. Bengio, E. Brevdo, F. Chollet, A. N. Gomez, S. Gouws, L. Jones, L. Kaiser, N. Kalchbrenner, N. Parmar, R. Sepassi, N. Shazeer, and J. Uszkoreit, "Tensor2Tensor for Neural Machine Translation," *CoRR*, vol. abs1803.07, 2018. [Online]. Available: httparxiv.orgabs1803.07416

[60] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention Is All You Need," 6 2017. [Online]. Available: http://arxiv.org/abs/1706.03762

[61] V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions and reversals." *Soviet Physics Doklady*, vol. 10, no. 8, pp. 707–710, 1966.

[62] P. Jaccard, “Nouvelles recherches sur la distribution florale,” *Bull. Soc. Vaud. Sci. Nat.*, vol. 44, pp. 223–270, 1908.

[63] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean, “Distributed Representations of Words and Phrases and their Compositionality,” *CoRR*, vol. abs/1310.4, 2013. [Online]. Available: http://arxiv.org/abs/1310.4546

[64] R. Rehurek and P. Sojka, “Software Framework for Topic Modelling with Large Corpora,” in *LREC 2010 Workshop on New Challenges for NLP Frameworks*. Valletta, Malta: ELRA, 5 2010, pp. 46–50.

[65] T. Mikolov, G. Corrado, K. Chen, and J. Dean, “Efficient Estimation of Word Representations in Vector Space,” *Proceedings of the International Conference on Learning Representations (ICLR 2013)*, pp. 1–12, 2013. [Online]. Available: http://ronan.collobert.com/senna/http://arxiv.org/pdf/1301.3781v3.pdf

[66] E. Klein and E. Loper, *Natural Language Processing with Python.* O’Reilly Media, 2009, no. January 2009.