SIMPLIFYING Q&A SYSTEMS WITH TOPIC MODELING

A Thesis

Presented to

The Graduate Faculty of The University of Akron

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

Troy Kozee

May, 2018

SIMPLIFYING Q&A SYSTEMS WITH TOPIC MODELING

Troy Kozee

Thesis

Approved:

Advisor
Dr. Chien-Chung Chan

Faculty Reader
Dr. En Cheng

Faculty Reader
Dr. Yingcai Xiao

Department Chair
Dr. David Steer

Accepted:

Dean of the College
Dr. John Green

Dean of the Graduate School
Dr. Chand Midha

Date

CONTENTS

Page

CHAPTER

I. INTRODUCTION 3

1.1 About the source texts and question collection 5

1.1.1 A Connecticut Yankee in King Arthur's Court 5

1.1.2 Call of the Wild 6

1.1.3 A Tale of Two Cities 7

II. IR AND Q&A SYSTEMS 9

2.1 Information Retrieval 9

2.2 Apache Lucene 10

2.3 Q&A Systems 12

2.4 Common NLP Tasks 14

2.5 Stanford CoreNLP, JWI, and BabelNet 17

III. TOPIC MODELING 20

3.1 MALLET 22

IV. EXPERIMENT OVERVIEW 23

4.1 Preparing and Analyzing the source texts 23

4.2 Measures used 24

4.2.1 Mean Reciprocal Rank 25

V. SYSTEMS OVERVIEW 26

5.1 Baseline System 27

5.2 Control Q&A System 30

5.2.1 Passage Extraction System 32

5.3 Topic Modeling Q&A System 34

5.3.1 Negative Results 36

VI. RESULTS AND DISCUSSION 38

6.1 Future Work 41

6.1.1 Better Shingling 41

6.1.2 Better WSD and synonym retrieval 41

6.1.3 Better queries 42

6.1.4 Topic modeling for questions 42

VII. CONCLUSION 43

BIBLIOGRAPHY 44

ABSTRACT

Question answering (Q&A) systems have made great progress in recent years, but that progress has also required enormous resources in terms of computation and storage. We attempt to reduce the computational and storage footprint of Q&A systems by using topic modeling algorithms to associate questions with relevant segments of text. We compare the performance of three different Q&A systems constructed using different strategies, including topic modeling, to answer fact-based questions about three books. We compare these systems by mean total indexing time (n=10), mean index size (n=10), mean total query time over question collection (n=10), proportion of correct answers and mean reciprocal rank (MRR). We found that an NLP-based system performed best, but at the expense of much longer run times for indexing and querying. A system based around topic modeling performed worst.

ACKNOWLEDGEMENTS

Thanks to Dr. Chan for his advice and assistance throughout this project. Also thanks to Drs. Cheng and Xiao for taking the time to read my thesis and attend my oral defense.

Thanks to Francesco Cecconi and Roberto Navigli of BabelNet, for kindly increasing my daily quota of BabelNet lookups for the duration of this project.

Thanks to my wife, Hannah, for her love and support throughout the duration of this degree.

CHAPTER I

INTRODUCTION

A "question answering" (Q&A) system attempts to answer questions that are posed in an unstructured, naturalistic language familiar to any native speaker. To contrast with the "English-like" queries of SQL, users of a Q&A system do not need extensive technical training to use the system; they need only be aware of how grammatically complex a question the system is capable of answering. While voice-driven products, such as Apple's Siri and Amazon's Alexa, have recently made the public aware of Q&A systems, the first example of a Q&A System, BASEBALL, was developed in 1961 [20]. BASEBALL read questions from punched cards and was unable to parse multi-clause sentences, but 50 years of progress resulted in IBM's Watson winning the Jeopardy in 2011 against two of the best human players of the game [17].

It can be argued that improvements in Q&A systems have been enabled (if not encouraged by) the "QA" track of the annual Text Retrieval Conference (TREC), a jointly sponsored project of the U.S. Department of Defense and NIST [2]. TREC organizers hold annual competitions on a variety of information retrieval tasks, and have run a question answering (QA) track between 1999 and 2007, starting a new LiveQA track in 2015[3]. The team that developed Watson explicitly acknowledge using TREC data to test and improve their system, and members of the Watson team

were active with TREC in other capacities [18]. The TREC organization takes credit for a number of advances in information retrieval (including Q&A), and a number of Q&A systems exist for a variety of fields, such as medicine and materials science [9, 43].

This progress comes as a computational cost. IBM's Watson uses 90 IBM POWER7 servers and 16 terabytes of ram [37]. It found Jeopardy answers in a matter of seconds by applying a large number of information retrieval techniques in parallel on a very large corpus of source data [17, 45]. Any Q&A system with these computational requirements would be unable to service more than a few requests at once, but Watson's success demonstrates the power of Q&A systems. This raises the question: how can one both improve the quality of answers from a Q&A system while reducing its computational requirements?

While improving Watson itself is far beyond the scope of this paper, we investigate and compare accuracy and computation requirements of different Q&A strategies. In particular, we want to see if topic modeling approaches can improve the accuracy of a Q&A system while reducing its computational requirements. We build and compare three Q&A systems built using Apache Lucene, an open-source information retrieval platform. Each system processes and answers questions from three books. Our baseline system uses mostly-unmodified Lucene components to implement a Q&A system. Our control system supplements "stock" Lucene with a number of natural language processing (NLP) techniques, such as part-of-speech identification, named entity recognition, and coreference (pronoun) resolution, to improve upon the

baseline system. The experimental system primarily uses topic modeling approaches to associate questions with source text passages that contain words statistically related to the question. The three systems are compared using a number of measures that we'll discuss in section 4.2.

1.1 About the source texts and question collection

Copyright concerns restrict our options to those texts in the public domain. We chose three dissimilar texts from Project Gutenburg.

1.1.1 A Connecticut Yankee in King Arthur's Court

Mark Twain self-published this book in 1889. We used this book as our primary text during development of each system. It recounts the tale of a contemporary engineer who is mysteriously transported to 6th century Camelot. Using his scientific knowledge to pose as a magician, the lead character attains political power and attempts to create a just, modern American society in a medieval Great Britain beset by suffering and injustice. Twain depicts medieval people as superstitious and gullible, the culture as coarse and cruel, and the Roman Catholic Church as an oppressive political power [46, 6].

This text contains a number of interesting features for analysis. The story is primarily narrated in the first person perspective of the main character, who is variously referred to as "the Stranger," "the Boss," and "Hank." The first person character changes at the beginning and end of the book, with the author or another in-

story character narrating in the first person. The book and its language are relatively modern, though the narrator uses a number of colloquialisms, and portions of the dialogue are intentionally written to imitate medieval idioms and speech patterns. Both of these features present an interesting challenge regarding NLP algorithms, though topic modeling algorithms function correctly regardless of language or use.

We created a collection of 39 fact-based questions, taken from the first 10 chapters of the book. Each question's answer can always be found within a relatively small passage (usually 1-3 paragraphs) of a single chapter.

1.1.2 Call of the Wild

This Jack London novella was published in 1903 and tells the story of a dog stolen from his masters and sold into servitude during the Yukon Gold Rush. Told from the third-person perspective, the main character, Buck, is a beloved family dog who is stolen by a gardener's assistant to cover a gambling debt. Buck is brutally mistreated and eventually sold to a courier, who takes him to Alaska and trains him to be a sled dog. The story depicts Buck gradually shedding his "civilized" habits and learning to survive by brutality and cunning. Buck is passed from owner to owner, and the story ends with Buck joining a pack a wolves after avenging the death of an owner who treated him well [27, 5].

This text anthropomorphizes the main character, placing human-like emotions, intellect and motives onto a dog. It's also written in London's flowing style, which involves complicated and complex sentences. London also implies more than

he states, so the meaning of a passage may be clear to a human reader even though the meaning isn't directly stated.

We created a collection of 29 fact-based questions based off the first two chapters of this eight-chapter novella. These questions were designed to be slightly more challenging than the questions for the previous text.

1.1.3 A Tale of Two Cities

Charles Dickens wrote "A Tale of Two Cities" in 1859. It tells the story of British and French emigres living in London around the times of the American and French Revolutions. A young couple, with complicated pasts, meet and marry before becoming embroiled in the French Revolution and the later Reign of Terror. The story ends with one character sacrificing himself to save the husband of his unrequited love from the guillotine.[15, 8].

This is both the earliest, longest, and most grammatically complex of the three books in this study. Despite this complexity, we found a "directness" to Dickens's writing that made it an easier read. We chose this book with the expectation that word matching and grammatical parsing would struggle to extract answers from this text, but a topic-based approach may have more success.

We created a collection of 20 fact-based questions from the first 11 chapters of the book. Instead of taking the questions directly from the text, we took them from the chapter summaries at *www.sparknotes.com*. Again, our expectation is that

a topic-based approach would not be confused by this additional degree of separation from the text.

CHAPTER II

IR AND Q&A SYSTEMS

2.1 Information Retrieval

A basic information retrieval (IR) system consists of a components for tokenizing document text, indexing those tokens, and efficiently retrieving documents containing some arbitrary set of tokens. IR systems are document-token centric; documents contain a large number of individual tokens, roughly grouped into collections called "fields". The purpose of a query is to quickly retrieve those documents that contain [most of] some set of query tokens in certain fields. This design implies that the query system anticipates incomplete matches (unlike a traditional DBMS), and retrieves the documents that best match the query.

IR systems can apply a number of simple transformations during indexing to make the text easier to search for relevant phrases [16]. Examples of these transformations are

- Replacing uppercase letters with lowercase
- Removing common "stop-words" which add very little unique information to a text, such as "a", "and", or "the."

- Replacing word variants with the root form of a word (ie. replace "bringing" with "bring"). This can be done algorithmically (stemming) or by dictionary lookup (lemmatization).

These transformations are also applied at query time so that the query tokens' text matches the indexed tokens. This can result in less relevant "false positive" documents being retrieved, but also ensures that relevant documents aren't missed because (for example) the query used a different verb tense from the source text.

2.2 Apache Lucene

Lucene is an actively developed, open source information retrieval platform initially written by Doug Cutting, and now managed by the Apache Foundation. It is over 20 years old and has had several dozen major and minor releases over that time [10].

Indexed information is stored as a number of discrete documents, each with an arbitrary number of character fields. Tasks such as tokenization, lemmatization, and stop-word removal are handled by an object of the Analyzer superclass, and can be set on a per-field basis. Analyzers can be thought of as pipelines, consisting of a Tokenizer to generate an initial stream of tokens and zero or more Filters that modify the token stream in some way. Lucene provides a handful of complete Analyzers, as well as a number of Tokenizers and Filters that can be assembled into a custom Analyzer. Tokenizer and Filter subclasses can also be easily written to address a specific analysis task (such as named entity recognition).

When a document's fields are indexed, the analyzer generates a steam of tokens which are usually stored in an "inverted document index." This "inverted index" relates each unique token with the documents containing that token. In the most basic sense, a query can be thought of as retrieving from the index the sets of documents containing each query token, and intersecting those sets to get a list of documents containing all (or most) of the tokens in the query. However, merely having all (or most) of the tokens in a query does not imply that a document is pertinent to the query. Matching documents need to be scored.

Documents are scored by generating a sparse, N-dimensional term vector for each document, where N is the number of unique tokens in the entire index. A non-zero value for any entry in the vector represents the presence in the document of the token associated with that entry. At query time, a vector using the query's tokens is similarly generated and compared with the document vectors. Documents whose vectors are most similar to the query vector are returned as a search result.

Precisely how document vectorization and comparisons are done depends on the scoring algorithm, and Lucene allows for scoring algorithms to be easily changed. Prior to Lucene 6, Lucene used term frequency-inverse document frequently (tf-idf) weighting and cosine similarity scoring [39]. Term vector entries were calculated using tf-idf, where tokens that both

1. Occur frequently in a given document (tf)

2. Do not occur in many documents in the collection (idf)

receive the highest values, as those terms are understood to be important to the document in question. Similarity is computed using a traditional dot product formula to compute the cosine angle between the two vectors in N-dimensional space. The idea is that similar vectors will have a small angle between them [16].

Lucene 6 has seen the introduction of the Okapi BM25 scoring model. BMxx stands for "Best Match," and refers to a family of scoring functions with different parameter values that impact the weighing of different parts of the scoring function (particularly tf). In addition to different weighting, BMxx also accounts for document lengths in individual cases and across the collection, improving the ranks of relatively short documents that also have high tf and idf [7, 1, 42]. Lucene maintainers judged the system sufficiently superior to make it the default scoring algorithm, and experiments with our baseline system demonstrate BM25's superiority over the previous scoring method.

2.3 Q&A Systems

A question-and-answer (Q&A) system adds the following components to an IR system[43, 40], with the goal of being able to correctly answer questions posed in a natural language

1. A natural-language processing (NLP) component for detecting and recording semantic features in the document text during indexing. These are discussed in section 2.4.

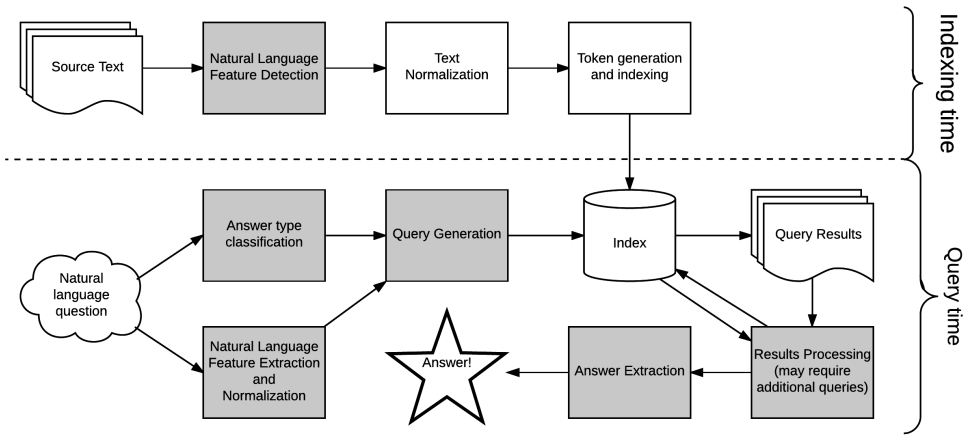

Figure 2.1: Complete Q&A System. Components in gray are added to an underlying IR system (in white).

2. A component for converting a natural language query into a query that the IR system can perform. This component ought to use the same NLP techniques used at indexing. It may also employ

   (a) Synonym substitution to account for differences that naturally occur between a posed question and the queried documents.

   (b) Answer type classification, which identifies the kind of thing that would answer a question. For example, questions that start with "who" can usually be answered with by referencing a person or organization. This can be used to refine query results.

3. A component for analyzing the document results returned from a query. The aim of this component is to locate the document passage that can best answer the posed question. The simplest version of this matches tokens from the query

with tokens in the source text, however a more advanced component may also use NLP data that the underlying IR system isn't designed to use. It may also submit additional queries in an attempt to better locate an answer.

When assembled into a complete Q&A system (see Figure 2.1), natural language questions are parsed, and the most important features are extracted and transformed into a query for the underlying IR system. The query results are then analyzed in an attempt to extract the passage that most likely answers the question.

2.4 Common NLP Tasks

Natural Language Processing (NLP) involves identifying and extracting the semantic features of text in a "human-spoken" language. This can be a very difficult problem for a machine, despite the fact that school-age children learn how to solve this problem in the course of their education. While a machine might not be able to "understand" the meaning of a passage of text in the same way a human can, NLP strategies allow a machine to identify relationships between words within a passage. These relationships can be exploited in a number of ways, such as in finding coinciding relationships between two different passages (e.g. a passage of text and a question about the passage).

Any analysis of text starts with identifying words (tokens) within a passage, and organizing those words into sentences. Each sentence can then be parsed in isolation, labeling words and phrases for their parts-of-speech (POS). Named Entities

(NEs), such as people, places, dates, etc. can be recognized and identified in the text. Pronouns can be properly associated with the noun they reference. The sentiment (positive or negative) of a passage can even be identified.

NLP tasks rely to some degree on machine learning and heuristics. Using an annotated corpus of source text, such as Penn Treebank-3 [30], a machine learner can be trained to identify the desired features in another passage. The accuracy of the identification relies on the similarity between training text and the passage being identified. For example, a classifier trained on text from the 1960s may be difficulty correctly identifying features in the Canterbury Tales (14th century English). As mentioned in section 1.1, these shortcomings pose real problems for this project.

Because word form and use can vary based on context, it's also common for words to be reduced to a root form and appropriate synonyms referenced. Lemmatization, replacing a word with its root form, can be done referencing a dictionary such as Princeton's WordNet [32]. Stemming, algorithmically reducing a word to its root form, is less common in NLP because generated stems are sometimes not valid words.

Synonym substitution can help show relationships between passages that a machine can't find due to differences in word choice. WordNet organizes words into collections of synonyms called "synsets," which can be used to find appropriate synonyms for a word. However, the meaning (or sense) of a word depends heavily on context. The word "bank" can be either a noun or a verb, referring to a financial institution, a sloped patch of earth, a formation of clouds, or the act tilting to change

Algorithm 1 Lesk's Algorithm (Simplified)

1: **input** word W^*, context words $W_1, ..., W_m$, dictionary D

2: **let** $S_1, ..., S_n$ be the senses of W^* in D

3: **initialize** $weight(S_k) = 0$ for k=1 to n

4: **for each** W_i

5: **for each** S_k

6: **if** (definition or examples in S_k contain W_i),

7: increment $weight(S_k)$ by $weight(W_i)$

8: **output** the S_k with the greatest weight

$weight(W_i)$ isn't strictly defined, but [22] uses idf

direction. Each sense of this word has its own synonyms, so it's vital that the appropriate sense be identified.

Word Sense Disambiguation (WSD) refers to the difficult problem of programmatically deducing the correct sense of a word from its context. Lesk's algorithm [24, 22] was one of the first to address this problem and illustrates one way that WSD may be accomplished. The core insight of Lesk's algorithm is that lexical dictionaries (like WordNet) often have definitions and sample sentences that illustrate the correct use a particular sense of a particular word. The proper sense of a word can be deduced by recording the number of surrounding "context" words that also appear in the definition or sample sentences for each sense. The most likely sense will have the

most context words that appear in the definition or sample sentences. A version of the algorithm can be found in Algorithm 1.

Though more a machine learning task, Answer Type Classification (ATC) is a helpful component in any Q&A system, especially whencombined with NE recognition. Using the kind of answer a question requires, queries to the IR system can be refined to focus on an appropriate NE (eg. focus on people for questions requiring a person). Li and Roth [25, 26] propose a detailed system for ATC that begins with course classifications like "human" or "location," each of which refines into more specific ideas like "individual," "group," "city," or "country." Morton [33] uses a simpler typology which specifies 11 different answer types and a precise method for training a classifier for this typology (which we use in our control system).

2.5 Stanford CoreNLP, JWI, and BabelNet

Stanford CoreNLP [29] is open-source language processing toolkit written in Java. In addition to basic tasks such as tokenization, CoreNLP is capable of performing most of the NLP tasks mentioned above, such as tokenization, sentence boundary detection, named entity recognition, POS annotation, and coreference (pronoun) resolution. Each task is handled by an Annotator, which provides appropriate information for the token in question. Annotators can have prerequisites for use: NER requires POS markings, and everything requires tokenization. Despite these prerequisites, the Annotators can generally be accessed independently of each other, once the text is analyzed. CoreNLP also implements a number of machine learning algorithms, such

as Naive Bayes, Maximum Expectation and SVM. These components can be (and were) used to train an Answer Type Classifier.

A number of Java libraries exist for using WordNet and most of them use a similar interface that mirror's WordNet's design. The best-maintained of these is Java WordNet Interface (JWI) [19]. The system for ATC described in [33] identifies important nouns in a question being processed, and uses WordNet to provide synonyms (specifically, hypernyms) for these words. JWI is used during training and use of the ATC component.

The same procedure used to extract question features for training the ATC can also be used to highlight important question features when transforming a question into an IR system query, particularly when it comes to identifying words whose synonyms should be included. WordNet's resource of synonyms can be of great help in providing those synonyms, but only if the correct meaning (or sense) of the word in question is used. The ATC component avoids this problem by always using the first WordNet sense for a word, which is sufficient for classification, but not for actual queries. Word Sense Disambiguation (WSD) must be used to insert the correct list of alternative synonyms for a query, but a good Java WSD library is difficult to find. The ones that exist are either abandoned, focused on contests like SenseEval, or require dozens of lines of boilerplate code.

We eventually settled on BabelNet [34], an online, multi-lingual, WordNet-like lexical dictionary that brings together lexical information for a number of sources, including WordNet and Wikipedia. Access is provided through a RESTful interface.

Aside from expanded access to synonyms, BabelNet provides WSD functionality for a few lines of code [35]. As we only needed to use WSD in the query component of our system (and only for single-sentence queries), BabelNet had the perfect "footprint" for our needs. BabelNet is unfortunately a metered service, and fixed number of lookups can be performed in a 24-hour period. Finding the sense of a word, retrieving its synset, then identifying and accessing related synsets could require a dozen BabelNet requests per term. The additional network latency makes BabelNet an imperfect solution to our problem. Local caching can correct both of these issues, but also distorts measurements taken to compare the different systems.

CHAPTER III

TOPIC MODELING

Topic modeling is machine learning technique that allows for quickly detecting co-occurring groups of words within a collection of documents. Topic modeling algorithms are context free, language independent, and require very little in the way of manually processing or labeling the text being analyzed. Probably the most popular topic modeling algorithm, Latent Dirichlet Allocation (LDA), was first published in 2003 by Blei, et al. [12] and incrementally improved over the intervening years.

The algorithm uses a "generative model" that assumes the text being analyzed was randomly generated [11]. A topic is a "bag of words" with probability distribution over those words. Any semantic relationships between topic words are beneficial for human understanding, but entirely disregarded by the algorithm. A document collection usually consists of a distribution over multiple topics. According to the generative model, every document is the result of the following random process described in Algorithm 2.

It must be stressed that LDA assumes the every document is randomly generated, and human comprehension of the document is just a convenient coincidence. Given a collection of documents and predetermined number of topics, LDA finds the allocation of words to topics, and mixture of topics themselves, that maximizes the

Algorithm 2 LDA's generative model for documents

1: **input** the number of words to be generated, n,

topic distribution over collection T, where...

each topic is a distribution over a vocabulary of words.

2: **for each** word, w_i, to be generated

3: **randomly select** topic $T_j \in$ T

4: **randomly select** word $\hat{w} \in T_j$

5: **assign** $\hat{w} \rightarrow w_i$

6: **output** document as$\{w_i\}_{i=1}^{n}$

likelihood of the collection occurring at random. Humans find the resulting topics understandable because they came from a collection of documents that humans can understand.

The LDA algorithm does not address the matter of choosing an appropriate number of topics, but a number of approaches have been proposed. The most simple approach is the blanket statement "between 200 and 400 topics," from the makers of the Java library: MALLET [4]. More programmatic approaches combine LDA with a non-parametric, probabilistic process (like Hierarchical Dirichlet Process) to pick an optimal topic number [44]. [48] avoids using another probabilistic process by utilizing n-fold cross validation to find a number of topics that minimizes perplexity.

3.1 MALLET

MALLET is a language-oriented machine learning toolkit from the University of Massachusetts [31]. It is written in Java, and contains a number of machine learning algorithms, including LDA. Use is similar to CoreNLP's machine learning algorithms, in that one creates a pipeline to emit a stream of tokens, which are concatenated into a feature vector, called an Instance. A collection of these Instances can be given to MALLET's LDA implementation, which generates a collection of topics. MALLET also provides an Inferencer, which uses an existing topic model to classify the topic mixtures of new documents.

CHAPTER IV

EXPERIMENT OVERVIEW

4.1 Preparing and Analyzing the source texts

We acquired plain text copies of each novel from Project Gutenberg and processed them to ease in chapter and paragraph detection. Each text was divided at indexing time into non-overlapping, five-paragraph passages (ie. shingles), with each shingle indexed as a Lucene document along with chapter number and chapter title. Passage/shingle texts were analyzed during indexing by each Q&A system as described in Chapter 5. Lucene queries retrieve the five-paragraph passage(s) that best match the query, meaning that returned passages need to be re-analyzed and scored to extract the 512-character excerpt that contains the most appropriate answer.

Our choice of five-paragraph shingles was the result of several trade-offs, and shingle size is one area where all three systems could be tuned for better performance, as will be discussed in Section 6.1. Smaller shingles would simplify the final step of answer extraction (see scoring discussion in 4.2.1), but would impede finding relevant documents for answers that span more paragraphs than the shingle size. This would be especially problematic for answers that occur in dialogue, as authors typically use new paragraphs to signal alternating speakers in a conversation. This literary device

also means that shingles vary in length from a dozen words to several thousand characters, posing problems for operaitons that rely on sufficient context, such as coreference (pronoun) resolution.

Indexing by chapter solves many of these problems, but also means that entire chapters are returned from queries. This shifts the work of finding relevant passages from an efficient IR system like Lucene to a less-efficient answer extraction system that does a linear scan through the text.

4.2 Measures used

The three systems are compared using the following metrics

- **MIT10** - Mean indexing time over 10 trials. This includes the time needed to tokenize and analyze the source text, as well as storage time.
- **IS** - Index Size
- **MTQT10** - Mean total query time over 10 trials for the entire question collection. This includes the time to analyze the question, query the index, process the result, and check the answer for correctness.
- **PAC** - Proportion of questions answered correctly. This is counted whenever the system returns 512-character excerpt that contains the correct answer to a question, regardless of how system ranks the fragment. Detection of correct answers is done semi-automatically.

- **MRR** - Mean reciprocal rank. This is similar to the previous measure, but accounts for how highly the system ranked the returned excerpt. See 4.2.1. As above, this is calculated on a semi-automatic basis.

These tests are conducted on a 3.6 Ghz Intel i7-4790 CPU with 16 GB of ram running Windows 10. During testing, the system is monitored to prevent scheduled, resource-hungry tasks (such as updating or search indexing) from distorting the timed results. To account for run-time improvements due to caching, some initial trials were run and discarded.

4.2.1 Mean Reciprocal Rank

For each question q_i in collection Q, let r_i be the rank of the correct answer returned by the system (ie. $r_i = 1$ if the system ranks the returned correct answer most highly). If the correct answer is not found for q_i, let $r_i = \infty$ and define $\frac{1}{\infty} = 0$.

Mean reciprocal rank is calculated

$$MRR = \frac{1}{|Q|} \sum_{q_i \in Q} \frac{1}{r_i}$$

It should be clear that $0 \leq MRR \leq 1$ where 1 represents the system returning the correct answer to every question and scoring that answer highest each time. 0 represents the opposite – the system failing to retrieval the correct answer at all for each question. What is less clear is that the system severly penalizes 2nd place answers (from 1 to .5), but is progressively less harsh in its treatment of answers ranked 3rd and lower (.5 to .333 to .25 to .2 ...).

CHAPTER V

SYSTEMS OVERVIEW

Each system maes use of Lucene for the core task of storage and retrieval. Analysis of the text to be stored (tokenization, lemmatization, etc) or queried happens within the framework provided by Lucene.

NLP tasks are primarily performed using CoreNLP, though this is somewhat of a trade-off as we had several examples of systems/components built on Apache OpenNLP. We found CoreNLP to be a "better" NLP system than OpenNLP, and only regretted that decision once or twice. An ATC system was built using CoreNLP. Both WordNet and BableNet serve as our lexical resources, with WordNet access from the JWI library, and BabelNet access from a pure-Java wrapper to BabelNet's HTTP interface.

Topic modeling was handled by MALLET from the University of Massachusetts, which came highly recommended from several sources.

These libraries and the three Q&A systems are written in (or compatible with) 32-bit Java 1.8.

5.1 Baseline System

The baseline system primarily consists of "stock" classes that are distributed with Lucene. Only modifications allowed by class interface were used. No Lucene classes were subclassed.

Analysis at indexing and query time is performed by the EnglishAnalyzer, which consists of these components [7]

1. A StandardTokenizer, which implements the Unicode Text Segmentation algorithm
2. A StandardFilter, which simply passes on tokens without modifications
3. An EnglishPossessiveFilter, which strips Unicode variations of "'s" from token text.
4. A LowercaseFilter, which converts all token characters to lowercase
5. A StopwordFilter, which removes a predefined set of common words (e.g. "a", "an", "the", "is", etc.) from the token stream. Using the EnglishAnalyzer API, the baseline system also uses this filter to remove common question words ("who", "what", "where", "do", "about", etc) that could interfere with retrieval.
6. A SetKeywordMarkerFilter, which marks a predefined set of words as "keywords," which are unchanged by the next filter. The baseline system does not make use of this filter.

7. A PorterStemFilter, which uses Porter's Stemming Algorithm to reduce words to a more simple form by removing a variety of word endings. For example, a Porter stemmer would render the underlined portion of the previous sentence as "to reduc word to a more simpl form by remov a varieti of word end". This is not a faithful rendering of the text, but it does ensure that word variations (e.g. remove, removes, removed, removing) are all mapped to the same pseudo-root word to improve retrieval.

The entirety of each novel was indexed using this filtering pipeline. Query text is similarly analyzed as described above, but queries themselves undergo some extra processing (especially for Connecticut Yankee).

One of the novels is written in the first person, though questions are written in the third person using one of the main character's names. For example, the question "Where was the Stranger born?" is intended to retrieve the second sentence of the book "I was born and reared in Hartford,..." A series of simple substitutions are applied to these questions, replacing phrases like "the Stranger" and "the Boss" with the in-book reference "I". The question "Where was the Stranger born?" becomes "Where was I born?" After the EnglishAnalyzer has operated on this query text, only the tokens "i" and "born" remain.

Generally speaking, tokens that are emitted by the analyzer are put into individual TermQuerys, using a BooleanQuery to apply a logical "AND". For the book written in the first person, the token "i" needs to be to mapped to a number of first-person pronouns that may appear in a relevant passage. This is accomplished by

expanding "i" into a "first person query", consisting number of TermQuerys for first-person pronouns in a larger BooleanQuery using the logical "OR". This "first person" query is then included in the "AND" BooleanQuery mentioned above.

To complete the above example, the tokens "i", "born" would generate a query ("born" AND ("i" OR "we" OR "me" OR "us" OR ...)).

Queries can retrieve up to 15 documents, though if a query returns no documents, terms are randomly dropped until at least 15 documents have been retrieved [13]. Each document represents a five-paragraph passage from the text, which needs to be reexamined to extract the most relevant 512-character fragments. Lucene's Highlighter class does precisely this. Using term vectors stored at indexing time, the Highlighter class

1. Splits document text into non-overlapping fragments of a [mostly] uniform length. The baseline system uses a maximum length of 512 characters.

2. Scores each fragment by counting the number of fragment tokens that match query tokens

3. Returns the 10 highest scoring fragments for the query.

Each returned fragment now has two scores: the fragment score, s_f, assigned by the highlighter and the document score, s_d, assigned at query time. To create an aggregate score, the baseline system uses the formula $s_d \cdot ln\,(s_f + b_f)$, where b_f is a fragment boost factor to more strongly favor high fragment scores over high document scores. The baseline system uses $b_f = 25$.

The returned fragments are semi-automatically tested to see if they correctly answer the question.

5.2 Control Q&A System

Our control system is built around the natural language processing capabilities provided by CoreNLP, and roughly follows the design set forth by [16]. CoreNLP is used for tokenizing the source text, lemmatization (instead of stemming), NER, sentence parsing (POS labeling) and coreference (pronoun) resolution. All Lucene tokens have an attribute called "Position Increment Gap," which specifies the space between the token at hand and the previous token in the stream. The default value for this is [sensibly] 1, but it can be increased to show, for example, that stop-words were removed from the stream. A gap of 0 implies that different tokens are synonymous. Both NER and pronoun resolution use this feature to place NE labels or pronoun references "on top" of the tokens they are describing. This allows queries to find the alternative terms without distorting the length of the stream of tokens (which some queries use for scoring).

POS labels are recorded as a token "payload." Token payloads are not searchable, but can be retrieved from query results for additional processing. The "answer extraction" feature of the control system uses POS labels to make stronger matches between query terms and returned results.

Like the Baseline system, the control system uses simple transformations, like stop-word removal and case removal, to make the text more easily searchable. These

transformations are done after CoreNLP processing, which needs the unfiltered token stream.

Queries are built and executed in a manner similar to the baseline system, but with several modifications

1. An ATC system based on [16, 33] was adapted to CoreNLP and trained on [33]'s corpus of questions. Several controlled trials showed this system performs comparible to the original system. The original text of each question was analyzed, and the result used to add searches for specific NE labels along with important question terms in the query. Several answer types have associated NE labels, which are "ORed" into a Lucene BooleanQuery and treated as wildcards by the passage extraction system (see below)

2. A synonym substitution component that expands remaining noun, verb and adjective question terms (lemmas) with their synonyms. BabelNet provides both WSD and synonym lookups, though caching is used to limit online activity. Synonyms are packaged together with the original lemma and "ORed" in a BooleanQuery.

3. A passage extraction system based on [41, 16] that finds the passage of a returned document that best matches the query. This system is described in more depth in 5.2.1.

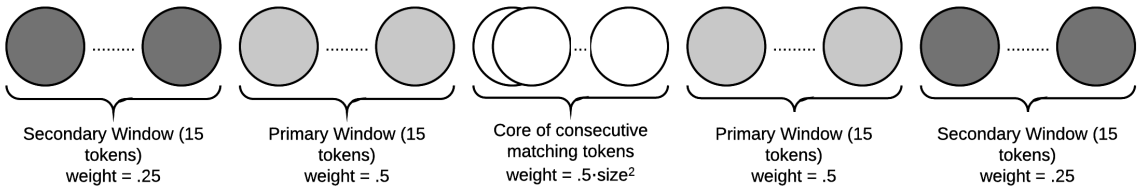

Figure 5.1: Windows around a core of matching tokens.

A more complicated query generation system using CoreNLP's grammar parser with [47] was considered, but the existing word filters did a sufficient job of extracting important words from a question and discarding the rest.

The top 10 scoring fragments generated by the answer extraction system are semi-automatically tested to see if they correctly answer the question, and results are recorded below.

5.2.1 Passage Extraction System

As mentioned above, the passage extraction system is based on a system proposed by AT&T at TREC-8. It works by building two different windows around a "core match" between a source token and a query token, scoring additional matches based on distance from the "core." Passages with several matching tokens in close proximity will score higher, and are more likely to contain the answer to the question.

Lucene allows for document term vectors (generated at indexing) to be stored with the document and retrieved with the document at query time. This term vector can be used to easily recreate a document's token stream without having to re-analyze

the document's text. The stream undergoes some additional preprocessing to make NE and POS labels easier to match.

The extractor scans through the stream of source tokens, looking for matches against query tokens. If a match is found, it is labeled a "core match", and the core is expanded if a token adjacent to the core also matches. Around this core of consecutive matching tokens, primary and secondary windows of fixed length (see Figure 5.1) demarcate the text fragment in question. The windows are searched looking for additional matches. The passage is scored by summing the weighted scores of each match (token text, POS and NER labels), where weights depend on which window the match appears in. Matches in the secondary window (16-30 tokens away from the core) are weighted at .25, while primary window matches (2-15 tokens away) are scored at .5. Core tokens are quadratically weighted by the number of tokens in the core: $.5 \cdot (length)^2$. If all of a query's tokens are matched within the passage, a doubling bonus is applied to emphasize that passage's similarity to the query.

Searching for a specific type of NE (due to ATC) is done using a wildcard query token that specifically matches against the NE attribute of a source token. Matching this wildcard token occurs after the other query tokens are matched, but before the passage it scored, and (if present in the queries) must also match for the passage's score to receive a doubling bonus.

Because matching is done within a 60 token window, it isn't guaranteed that a returned passage will fall within the 512 character limit. However, matching passages that fall under 512 characters are expanded to meet the 512 character limit.

5.3 Topic Modeling Q&A System

The topic modeling system generally follows the architecture of [21], which was submitted to the LiveQA track of TREC 2016. It uses the less computationally-intensive elements of CoreNLP for Lucene indexing, search and retrieval, with a topic-based answer extraction system. Specifically, CoreNLP is used for tokenization and lemmatization at index time, along with several of the "normalizing" filters that were used in the other systems. Questions are also tokenized and lemmatized using CoreNLP, with the same BabelNet-based synonym substitution that the Control system uses. If a question returns no results, a random term is dropped and the query re-run. As in the other systems, queries can return up to 15 five-paragraph passages.

Answer extraction is done by way of topic modeling, instead of matching query tokens with passage tokens. MALLET provides for both training a topic model and using it to classify the topic mixtures of new documents. An LDA topic model (500 topics) was trained using chapters from the source text. This model was employed by the answer extraction system to classify the topic mixtures of the returned passages and the original question text. To compensate for short question texts (which pose a problem for topic model inferencing), synonyms are used to "expand" nouns,

verbs and adjectives, insert synonymous words alongside the original question word [38].

Returned passages were split into non-overlapping, 512-character fragments, and Jensen-Shannon (JS) distance was used to calculate the similarity between each passage and the question. This matches the strategy used in [21]. A low score implies high topic similarity, so the 10 fragments with the lowest JS distance to the question were returned by the system.

JS distance is built off of Jensen-Shannon divergence, which is also called "information radius" [14], and itself built off of Kullback-Liebler divergence. Figure 5.2 shows the relationships between these. Also known as "relative entropy", KL divergence measures how much one probability distribution differs from another distribution [28, 23]. KL divergence is problematic as a general measure for similarity, because it isn't symmetric and requires creative calculations if one distribution has an outcome with zero probability. Jensen-Shannon divergence overcomes both of these constraints by averaging the two distributions together [14], and taking a square root of JS divergence results in a proper metric [36].

As topic mixtures can rightly be thought of as probability distributions, JS distance is a natural means differentiating passages that are "similar" to the question from passages that are very different.

5.3.1 Negative Results

Searches based on topic mixture were tried, but discarded for performing significantly worse than the baseline system. JS distance is a good measure of similarity, but searching by JS distance would require comparing the question to every indexed passage, entirely negating the benefit of searching with Lucene. Because topics are numerically labeled, it is possible to compute an average topic for a passage and perform a search using the average topic of a question. This is system did far worse than the baseline system.

Another approach selected a "baseline" passage and calculated the JS distance of every other passage (and question) to that baseline. If a question and passage have small JS distance to each other, they will have a similar JS distance to that baseline, so it was possible to search by distance to baselines. This system also performed worse than the baseline system.

Given two discrete probability distributions over sample space, S, with respective pdfs $p(x)$ and $q(x)$

$$KL(P,Q) = \sum_{x_i \in S} p(x_i)\, log \frac{p(x_i)}{q(x_i)} \tag{5.1}$$

where $p(x_i) = 0 \leftrightarrow q(x_i) = 0 \rightarrow log\frac{p(x_i)}{q(x_i)} = 0$

$$JS(P,Q) = KL(P, \frac{P+Q}{2}) + KL(\frac{P+Q}{2}, Q) \tag{5.2}$$

$$JSD(P,Q) = \sqrt{JS(P,Q)} \tag{5.3}$$

Figure 5.2: KL divergence (1), JS divergence (2) and JS distance (3)

CHAPTER VI

RESULTS AND DISCUSSION

As expected, the baseline system had the fastest mean run-time in indexing and querying, and the smallest memory footprint (Table 6.1). What is surprising is that it also performed relatively well in answering questions (Table 6.2), despite its minimal processing of the source text. In question answering, the control system outperformed the baseline system, but also took several orders of magnitude more time to index the data and process the questions. This was largely due to two factors

1. Coreference resolution of shingled paragraphs. Coreference resolution was, by far, the most computationally intensive part of processing the source text, and paragraph shingling effectively quintuples the amount of text to be processed.

Table 6.1: Run-time and size comparison for Connecticut Yankee

System	**MIT10**	**IS**	**MTQT10**
Baseline	984 ms	3791 KB	33 ms
Control	88m 14s	4323 KB	17857 ms*
Exp	17110 ms	3831 KB	41154 ms*

*This figure represents predominantly cached BabelNet lookups. Caching speeds lookup times tenfold.

Table 6.2: PAC and MRR per system, per text

System	Yankee		Wild		Cities	
	PAC	**MRR**	**PAC**	**MRR**	**PAC**	**MRR**
Baseline	.4872	.3307	.2069	.1264	.15	.085
Control	.5897	.4637	.2759	.2253	.05	.05
Exp	.3846*	.1916	.2069*	.0883	.05*	.0125

*Topic modeling is somewhat random. Sometimes an additional question will be correctly answered.

Without these two transformations of the text, indexing time drops to several seconds.

2. Synonym substitution of question terms. Between more processing of more complex queries and the overhead of using a remote service, mean query time was also several orders of magnitude larger than the baseline system, even with caching.

The trade-off was better performance, but the improvement is not so drastic as unambiguously justify the slower run time. In fact, the baseline system actually did a better job answering the questions for one of our texts.

The poor performance of the topic-based experimental system is disappointing, as it under-performed every other system in answering questions. In fact, the success rate is poor enough to label this particular experiment a failure. Many variations of a topic-based system were tried, and all of them under-performed the baseline system.

One possible explanation for this failure is that topic modeling doesn't work at well on short fragments of text, such as a single-sentence question. This is indicated by the research being done to apply topic modeling to Twitter content, which is most certainly characterized by short fragments of text. Techniques such as concatenating multiple tweets into a larger block of text do not really apply to our case.

We were equally disappointed by how poorly all three systems performed on a "Tale of Two Cities." The method of question generation definitely played a factor here, as questions were drawn from a secondary source, covering only the first 11 chapters (or so) of the text. The entire text was indexed, so questions involving a carriage from the first 11 chapters (for example) retrieved every passage that contained a carriage (of which there were many). That potential exists in the other texts, but likely mitigated because the texts were read directly. Questions were also chosen and worded in a way that would frustrate the token-matching system that lay at the core of the baseline and control systems. We hoped that that would allow the experimental system to "shine." This was not the case.

The very high success rate of all three systems on "A Connecticut Yankee in King Arthur's Court" may be somewhat distorted. During the development of all three systems, this text was used during testing and debugging. What seems like remarkable success may simply be the fact that all three systems were "optimized" for success on this novel. The results from "Call of the Wild" may be a better indication as to the systems' capabilities.

6.1 Future Work

In the wake of a failed experiment, it is tempting to recite a litany of roads not traveled (such as Stanford OpenIE or Apache UIMA). It is unlikely that any one decision is responsible for the poor performance of the experimental system, but there are three areas of improvement that could increase the PAC and MRR of all three systems, while decreasing the run-time of the control system.

6.1.1 Better Shingling

Shingle length could be less variable, so that you don't have shingles with dozens of characters alongside shingles with thousands. In addition, shingles could be generated after the text is tokenized and analyzed. This is not a well-documented feature of the Lucene API, but it is present. Both of these changes would improve run-time while ensuring that high scoring, but "trivially short" shingles don't displace larger shingles in the search results.

6.1.2 Better WSD and synonym retrieval

BabelNet's API required minimal disruption of our existing code, but that came at the cost of increased run-time, limited retrievals in a 24-hour period (even with caching), and uncertain behavior as the BabelNet API isn't well documented beyond a set of code examples. As discussed above, other WSD libraries are available, and they would allow us to work entirely off of a local copy of WordNet with a better-documented Java library, but that approach requires more radical changes.

6.1.3 Better queries

The Lucene BooleanQuery does not account for token proximity, only presence within a document. Other Lucene queries (PhraseQuery or SpanNearQuery) do account for proximity (and were tried in our systems), but have difficulty with partial matches. With strategic synonym substitution and dropping query terms, the SpanNearQuery may do a better job at ranking relevant documents than the BooleanQuery.

6.1.4 Topic modeling for questions

Topic modeling does not work well for short fragments of text (eg. questions), and discovering a method that successfully applies topic modeling to questions could dramatically improve the experimental system. There were also several approaches that were tried and discarded in the development of the runtime system. These may become viable if a better method for modeling questions can be found.

CHAPTER VII

CONCLUSION

We built and tested three Lucene-based systems for answering natural language questions about a work of fiction. Each system represented a different strategy for processing the text, and the questions about the text. It can be argued that the best overall system was also the most simple, using only standard components that are distributed with Lucene. A more complicated system using OpenNLP (with Lucene) did have a higher success rate, but at the expense of drastically increased indexing and query times. The investigatory focus of our project, a system based around topic modeling, did not perform as well as either of the other two systems.

BIBLIOGRAPHY

[1] Information-Retrieval - TFxIDF.

[2] Text REtrieval Conference (TREC) Overview.

[3] Text REtrieval Conference (TREC) QA Track.

[4] Topic Modeling.

[5] The Call of the Wild, October 2017. Page Version ID: 805588062.

[6] A Connecticut Yankee in King Arthur's Court<, August 2017. Page Version ID: 797405709.

[7] Lucene Source Code, September 2017. original-date: 2016-01-23T08:00:06Z.

[8] A Tale of Two Cities, October 2017. Page Version ID: 806089258.

[9] Asma Ben Abacha and Pierre Zweigenbaum. MEANS: A medical question-answering system combining NLP techniques and semantic Web technologies. *Information Processing & Management*, 51(5):570–594, September 2015.

[10] Andrzej Bialecki, Robert Muir, and Grant Ingersoll. Apache Lucene 4. In *SIGIR 2012 workshop on open source information retrieval*, page 8, Portland, Oregon, 2012.

[11] David M. Blei. Probabilistic topic models. *Communications of the ACM*, 55(4):77–84, 2012.

[12] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent Dirichlet Allocation. *J. Mach. Learn. Res.*, 3:993–1022, March 2003.

[13] Wray Buntine. Topic-specific scoring of documents for relevant retrieval. 2005.

[14] Asli Celikyilmaz, Dilek Hakkani-Tur, and Gokhan Tur. LDA Based Similarity Modeling for Question Answering. In *Proceedings of the NAACL HLT 2010 Workshop on Semantic Search*, SS '10, pages 1–9, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics.

[15] Charles Dickens. *A Tale of Two Cities*. Chapman and Hall, London, 1859.

[16] Andrew L. Farris, Grant S. Ingersoll, and Thomas S. Morton. *Taming Text: How to Find, Organize, and Manipulate It*. Manning Publications, December 2012.

[17] D. A. Ferrucci. Introduction to "This is Watson". *IBM Journal of Research and Development*, 56(3.4):1:1–1:15, May 2012.

[18] David Ferrucci, Eric Brown, Jennifer Chu-Carroll, James Fan, David Gondek, Aditya A. Kalyanpur, Adam Lally, J. William Murdock, Eric Nyberg, John Prager, Nico Schlaefer, and Chris Welty. Building Watson: An Overview of the DeepQA Project. *AI Magazine*, 31(3):59–79, July 2010.

[19] Mark Alan Finlayson. Java libraries for accessing the princeton wordnet: Comparison and evaluation. In *Proceedings of the 7th Global Wordnet Conference, Tartu, Estonia*, volume 137, 2014.

[20] Bert F. Green, Jr., Alice K. Wolf, Carol Chomsky, and Kenneth Laughery. Baseball: An Automatic Question-answerer. In *Papers Presented at the May 9-11, 1961, Western Joint IRE-AIEE-ACM Computer Conference*, IRE-AIEE-ACM '61 (Western), pages 219–224, New York, NY, USA, 1961. ACM.

[21] Josue Balandrano Coronel. Using Topic Modeling to Answer Complex Questions. In *TREC LiveQA 2016*, page 6, Gaithersburg, MD, November 2016. National Institute of Standards and Technology.

[22] Adam Kilgarriff and Joseph Rosenzweig. English Senseval: Report and Results. In *LREC*, volume 6, page 2, 2000.

[23] S. Kullback and R. A. Leibler. On Information and Sufficiency. *The Annals of Mathematical Statistics*, 22(1):79–86, March 1951.

[24] Michael Lesk. Automatic Sense Disambiguation Using Machine Readable Dictionaries: How to Tell a Pine Cone from an Ice Cream Cone. In *Proceedings of the 5th Annual International Conference on Systems Documentation*, SIGDOC '86, pages 24–26, New York, NY, USA, 1986. ACM.

[25] Xin Li and Dan Roth. Learning Question Classifiers. In *Proceedings of the 19th International Conference on Computational Linguistics - Volume 1*, COLING '02, pages 1–7, Stroudsburg, PA, USA, 2002. Association for Computational Linguistics.

[26] Xin Li and Dan Roth. Learning question classifiers: the role of semantic information. *Natural Language Engineering*, 12(3):229–249, 2006.

[27] Jack London. *The Call of the Wild*. Project gutenburg edition, 1903.

[28] David JC MacKay. *Information theory, inference and learning algorithms*. Cambridge university press, 2003.

[29] Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky. The Stanford CoreNLP Natural Language Processing Toolkit. In *Association for Computational Linguistics (ACL) System Demonstrations*, pages 55–60, 2014.

[30] Mitchell P. Marcus, Beatrice Santorini, Mary Ann Marcinkiewicz, and Ann Taylor. Treebank-3, 1999.

[31] Andrew Kachites McCallum. MALLET: A Machine Learning for Language Toolkit. 2002.

[32] George A. Miller. WordNet: a lexical database for English. *Communications of the ACM*, 38(11):39–41, 1995.

[33] Thomas Morton. Using semantic relations to improve information retrieval. *Dissertations available from ProQuest*, pages 1–111, January 2005.

[34] Roberto Navigli and Simone Paolo Ponzetto. BabelNet: The Automatic Construction, Evaluation and Application of a Wide-Coverage Multilingual Semantic Network. *Artificial Intelligence*, 193:217–250, 2012.

[35] Roberto Navigli and Simone Paolo Ponzetto. Multilingual WSD with just a few lines of code: the BabelNet API. In *Proceedings of the ACL 2012 System Demonstrations*, pages 67–72. Association for Computational Linguistics, 2012.

[36] Frank Nielsen. A family of statistical symmetric divergences based on Jensen's inequality. *arXiv:1009.4004 [cs, math]*, September 2010. arXiv: 1009.4004.

[37] Tony Pearson. IBM Watson: How it Works, and What it means for Society beyond winnin....

[38] Zengchang Qin, Marcus Thint, and Zhiheng Huang. Ranking Answers by Hierarchical Topic Models. In *IEA/AIE*, pages 103–112. Springer, 2009.

[39] Robert Muir. [LUCENE-6789] change IndexSearcher default similarity to BM25 - ASF JIRA, September 2015.

[40] Syarilla I. Ahmad Saany, Ali Mamat, Aida Mustapha, Lilly S. Affendey, and M. Nordin A. Rahman. Semantics Question Analysis Model for Question Answering System. *Applied Mathematical Sciences*, 9(130):6491–6505, 2015.

[41] Amit Singhal, Steven P. Abney, Michiel Bacchiani, Michael Collins, Donald Hindle, and Fernando CN Pereira. AT&T at TREC-8. In *TREC*, 1999.

[42] Stephen E. Robertson, Steve Walker, Susan Jones, Micheline Hancock-Beaulieu, and Mike Gatford. Okapi at TREC-3. *Proceedings of the Third Text REtrieval Conference (TREC 1994)*, 3, November 1994.

[43] A. A. Stupina, E. A. Zhukov, S. N. Ezhemanskaya, M. V. Karaseva, and L. N. Korpacheva. Question-answering system. In *IOP Conference Series: Materials Science and Engineering*, volume 155, page 012024. IOP Publishing, 2016.

[44] Yee Whye Teh, Michael I. Jordan, Matthew J. Beal, and David M. Blei. Hierarchical Dirichlet Processes. *Journal of the American Statistical Association*, 101(476):1566–1581, 2006.

[45] Clive Thompson. I.B.M.'s Supercomputer to Challenge 'Jeopardy!' Champions. *The New York Times*, June 2010.

[46] Mark Twain. *A Connecticut Yankee in King Arthur's Court.* Charles L. Webster and Co., project gutenburg edition, 1889.

[47] Xuchen Yao and Benjamin Van Durme. Information Extraction over Structured Data: Question Answering with Freebase. In *ACL (1)*, pages 956–966, 2014.

[48] Weizhong Zhao, James J Chen, Roger Perkins, Zhichao Liu, Weigong Ge, Yijun Ding, and Wen Zou. A heuristic approach to determine an appropriate number of topics in topic modeling. *BMC Bioinformatics*, 16(Suppl 13):S8, September 2015.