

OPTIMIZATION OF WORD MEANING SEARCH

A PROJECT REPORT

Submitted by

Pratik Sharma 141080036

Pratiksha Dehade 141081042

Akash Kadel 141080041

Under the guidance of

Dr. V. K. SAMBHE

(Professor, Department of Computer Engineering & Information Technology)

to our satisfaction for the award of the degree of

BACHELOR OF TECHNOLOGY

in

COMPUTER ENGINEERING AND INFORMATION

TECHNOLOGY



Veermata Jijabai Technological Institute, Mumbai

Certificate

This is to certify that the following students of Bachelor of Technology in Information Technology, have completed the report entitled "Optimization of word meaning search using machine learning" to our satisfaction.

PRATIK SHARMA

141080036

PRATIKSHA DEHADE

141081042

AKASH KADEL

141080041



(Name & Signature)

Guide / Supervisor

(Name & Signature)

Head, Department of
Computer Engineering and
Information Technology

(Name & Signature)

Co-Guide / Co-Supervisor

(Name & Signature)

Director, VJTI

Certificate

This is to certify that the following students of Bachelor of Technology in Information Technology, have completed the report entitled "Optimization of word meaning search using machine learning" to our satisfaction.

PRATIK SHARMA 141080036

PRATIKSHA DEHADE 141081042

AKASH KADEL 141080041



(Name & Signature)

Supervisor / Guide

Date:

(Name & Signature)

Examiner

Place:

DECLARATION

I declare that this written submission represents my ideas in my own words and where others' ideas have been included. I have cited and referenced the original source

I also declare that I have adhered to all the principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea / data / fact / source in my submission.

I understand that any violation of the above will be cause for disciplinary action by the institute and can also evoke penal action from sources which have thus been properly cited or from whom proper permission has not been taken when needed.

Pratik Sharma
141080036

Date: May 2018

DECLARATION

I declare that this written submission represents my ideas in my own words and where others' ideas have been included. I have cited and referenced the original source

I also declare that I have adhered to all the principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea / data / fact / source in my submission.

I understand that any violation of the above will be cause for disciplinary action by the institute and can also evoke penal action from sources which have thus been properly cited or from whom proper permission has not been taken when needed.

Pratiksha Dehade
141081042

Date: May 2018

DECLARATION

I declare that this written submission represents my ideas in my own words and where others' ideas have been included. I have cited and referenced the original source

I also declare that I have adhered to all the principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea / data / fact / source in my submission.

I understand that any violation of the above will be cause for disciplinary action by the institute and can also evoke penal action from sources which have thus been properly cited or from whom proper permission has not been taken when needed.

Akash Kadel
141080041

Date: May 2018

ABSTRACT

One of the major problems faced by avid book readers is the process of searching for the meanings of difficult words they come across in a book. This may sound like an easy task, but when encountering such words very frequently, it not only irritates the reader to stop their reading and search for this word, but also takes away from the overall reading experience.

There have been a lot of studies and researches which help determine the difficulties readers face in anticipating the meanings of different words in the English language. It would certainly be groundbreaking if we were able to not only solve the problem of determining which word is difficult for that particular individual, but also provide the meanings of those words at the exact time when the reader has reached that word in the book.

In this report, we present a tool developed on the basis of various research papers, machine learning and different algorithms which can first predict the book which the user is reading based on the words the user has searched, then provide the user with the meanings of all the difficult words he/she will encounter as he/she is reading the book, and customize that experience based on their vocabulary level and reading speed for an optimal result and reading experience. Also, the user will be able to pause the application for an in-depth look at the meaning of the word and take as long as he/she wants to understand it, and then go back to the book. The user would even be able to move back and forth through the words and use the application at their ease.

The tool will have several applications, in search engines, in pdf readers, as stand-alone mobile or desktop applications and as an online service. This will change the way people read books and encourage them to read more, as the time consuming and interrupting task of going to the dictionary from the book and back will be taken care of.

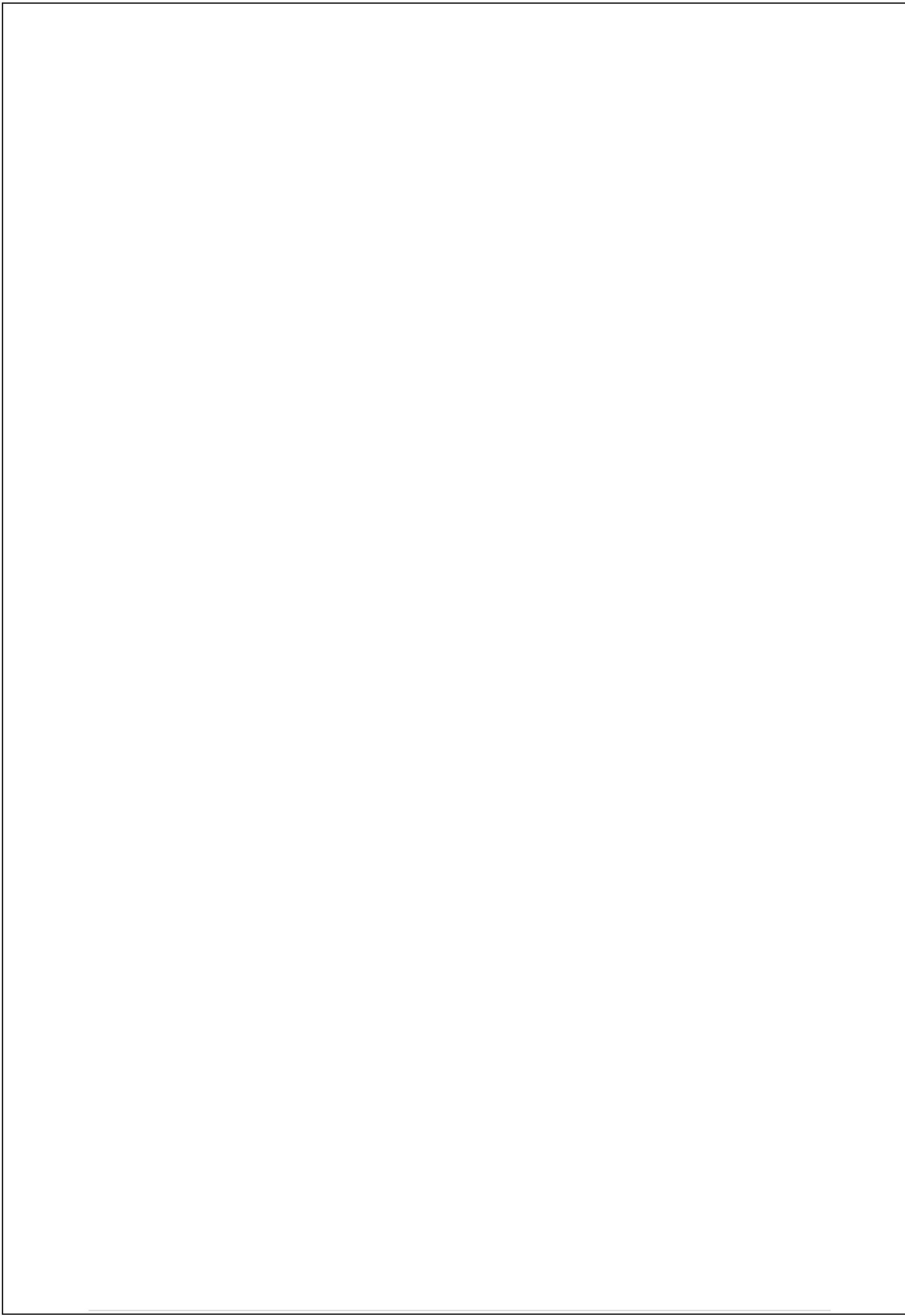


TABLE OF CONTENTS

Abstract	07
Chapter 1: Introduction	
1.1 Existing Methodology	10
1.2 Motivation	12
1.3 Problem Definition	13
1.4 Objectives	13
1.4 Methodology	14
1.5 Summary	15
Chapter 2: Literature review	
2.1 Existing methods to aid readers	17
2.2 Full Text Search	17
2.3 Determining the difficulty of words	20
2.4 Determining the speed of reading	23
Chapter 3: Implementation	
3.1 Structure of system	24
3.2 Use case of the proposed system	28
3.2.1 Conversion of files	28
3.2.2 Training the system	29
3.2.3 Performing a search	32
3.2.4 Predicting the Book	33
3.2.5 Displaying words and their meanings	35
Chapter 4: Results and discussion	
4.1 A standalone application to aid book readers	38
4.2 Other Implementations	38
4.3 Future Scope	39
Chapter 5: Conclusion and Future enhancement	
5.1 Conclusion	41
5.2 Failure Condition	42
5.2 Future Enhancement	43
5.3 Contribution	43
References	45

Chapter 1

INTRODUCTION

Reading has always been a top list hobby worldwide. Be it conventional way of reading hard copies or advanced way of reading books online. Although the number of readers reading books online have substantially increased over the enhancement of technology, it has been found out that 80 percent of the total population still prefer reading hard copies of a matter rather than its content surfing over the net. So even if we have to come up with a idea of revolutionizing the reading experience we need to pave our way towards hard copy reading targeting most of the world's population. Once the we achieve an active foothold on the physical affair then we can success-fully embed the same into online reading encounters.

Hence, the focus of our proposed application would be modifying, styling, and alleviating the user experience of hard copy reading using the technological tools. The same would be developed with the help subsidiary tools already available to make e reading activity less resource consuming. We would consider different aspects of machine learning tools and algorithms that would be integrated in our system. Suitable supported language and database would be used. Also, exploration on the existing learning tools would take place for better understanding of already availability of software in order to achieve maximum optimization. This way a care would be taken to give full justice to the field of Artificial intelligence such that when the system is build it should be easier to support the current applications and still could stand out.

1.1 Existing Methodology

As already discussed, not much tools have been available for elevating hard copy reading experience besides dictionaries in the form of book/electronic resource in searching for meanings of difficult words that he/she come across while reading. Finding meanings to the strenuous words, being able to make notes of certain content from the book, keeping a track of one's reading habits, taking into considerations of external interruptions etc. are some of the tasks which should be kept in mind while building a reading supportive application.

Also, these tasks are encountered several times by a reader which makes the reading activity tedious and apathetic. The following applications are the supposedly existing methodologies that support hard copy reading in way or other.

Table 1.1. Existing Solutions for Hard copy/ Soft copy reading problems

Application Name	Features	Drawbacks
Other online tools (online dictionaries, note making applications, speed tracers etc; dev. across the technological era)	<ul style="list-style-type: none"> <input type="checkbox"/> Meanings of difficult words becomes easy to search <input type="checkbox"/> Reading speeds could be tracked <input type="checkbox"/> Particular content could be marked and saved for later use 	<ul style="list-style-type: none"> <input type="checkbox"/> Not a structured way defined to make navigating through a particular book less tiresome <input type="checkbox"/> Hence no enhancement towards the idea of ease of conventional reading
Google Play Books (dev. 2010)	<ul style="list-style-type: none"> <input type="checkbox"/> The first most convenient way introduced to download a book online and read it offline <input type="checkbox"/> Several editing options, book libraries, in-built online dictionaries, easy scaling of books etc. available 	<ul style="list-style-type: none"> <input type="checkbox"/> Fail to provide features so as to provide users with the similar satisfaction of conventional reading

Amazon Kindle (dev. 2012)	<input type="checkbox"/> A light weighted, portable and extensible electronic device or reading books <input type="checkbox"/> Downloading of PDFs of any sort of document is available <input type="checkbox"/> Many text editing screen adjustments and other such enhancements available <input type="checkbox"/> Multi featured as compared to Google books	<input type="checkbox"/> Although best has been tried to integrate the online reading experience with conventional reading yet amazon kindle do not completely provide a solution for actual hard copy reading but can help us in understanding different aspects of making reading user friendly.
------------------------------	--	--

1.2 Motivation

The advancement of several technological tools to aid the online reading experience has always been appreciable. Also, we have made sufficient arguments which supports the fact of not much has been done to actually come up with tools to provide a wonderful experience of reading while physically reading a book. All those efforts of converting a virtual interface to give a real feeling of reading a book in the physical world is of course substantial. **But how great would be an idea of a user being able to actually go through a hard copy and at the same time able to use all the online tools features in an integrated manner to enrich its reading experience**

The above statement surely startled our brain cells on which we decided to on our idea of building such an application. The basic notion was to come up with such an application that would map the readers book into it, provide with meaning of all difficult words in a sequential manner and traverse through the book in such a way that is proved to be an amazing experience to the user despite of the physical interruptions. The driving force acting behind this motivation was the already available online reading tools with the help of which we would integrate several features into one application.

1.3 Problem Statement

The idea is to build an application which would elevate user's conventional book reading experience including

Application should be able to determine the reader's book with the help of words searched by the user based upon the time interval between the searches

It should be able to keep track of the speed of the reader which would later help it determine what word the user might be reading at the moment and thus provide its meaning automatically

As the reader is proceeding through the book page by page, application should be able to show the meaning of all the words that are classified as **difficult** or that could probably be searched by a user. All this should be done by traversing through all the words from the particular book which are difficult as the reader progresses.

User should be able to continue from where they left by the use of **pause** functionality provided in the application. Once the application continues from where it left the rest of the traversing and meaning suggestion should be done by the application.

User should be able to move back and forth across the words in the book thus allowing the user freedom to use the application at their ease

Considering an ideal situation of a reader completing a book at one go, if such a user uses the application, he would just be needed to input the application with first few difficult words. The rest of the time, the user would just be enjoying his book reading with no tedious searching and everything would be one glance away as per his/her reading progress.

1.4 Objectives

In order to build the application, we need to accomplish the following objectives:

1. The pdf files corresponding to the books must be converted to text files, for the system to be able to extract the words from the books.

2. The words from these files should then be used to train the model. To do so, the system builds a data structure similar to Inverted Index which will be discussed in the Literature section.
3. The next task is to develop an algorithm to search through the Inverted Index and determine the book that the user is reading based upon the following available data
 - The words that the user searches for (**Word1, Word2,**)
 - The time interval between the searches that were made (**T1, T2, ...**)
 - The reading speed of the user

Using this information, the model should first calculate the distance between the two words and should determine the book based upon that

4. The last objective, once we determine the book is to build a Graphical Interface to aid the user. This interface should provide the user with the **meanings** of the upcoming “**difficult**” words from the identified book and also at the same time allow the user to **pause** at any word and **move back and forth** through the words.

1.4 Methodology

The system is built entirely using python and its accompanying libraries and APIs.

The whole system is divided into several modules which include:

1. The module to convert the book pdfs to text files:

In order to do so, we use the PDFminer library available as a third-party download. In order to convert multiple files, we store them all in a single directory and convert each one to a **.txt** file while iterating through them and then saves it in a different directory.

2. The module to extract individual words and train the model:

Once the text files are ready, we use them to extract the words from each individual book file and then add these words to the Inverted Index which is implemented using MongoDB

To extract the words, we simply use the regular expression '**\w+**' and store that in an array. This array is then used to create the inverted index

The inverted index is simply MongoDB database table that stores the data about what word is present in what book and at what positions in a single document

3. The Search Module to determine the book:

We use basic set theory and statistical learning to determine the book. We start by creating a set **S1** of all the books that consist the two words searched by the user (**W1 and W2**) at an appropriate distance (determined based upon the interval between the two searches **T1** and the users reading speed). If this set is a singleton set, we stop the search and proceed further if not, we wait for the user to search another word and then using the second and the third word **W3** and the now obtained time interval **T2** we create another set **S2** and find the intersection of the two sets thus narrowing down the result until we are left with a singleton set.

4. The GUI module to display the meanings of the words:

We use the Tkinter library that is available in python to build the GUI for displaying the words and their meanings. Once a book has been recognized, the system gets all the words of that book from the MongoDB database and starts iterating through them while also showing the meanings of the words.

In order to get the meanings of the words, we use a third-party API called “**wordnik**” which provides definitions to the words by the use of API calls over internet.

1.5 Summary

- **Chapter 1** includes introduction which further includes description about existing technologies to support our idea, motivation for our idea, problem definition and methodologies that could be used.
- **Chapter 2** includes Literature Review which is a critical analysis of published sources or literature on topic. It is an assessment of the literature and provides a summary, classification, comparison and evaluation.
- **Chapter 3** discuss in depth about the implementation details and various technologies used.
- **Chapter 4** explore about the various possibilities of other implementation of the application which may or may not be in an integrated form i.e. discuss the diversity of features.
- **Chapter 5** discuss about the conclusions derived and future scope of enhancement for the application.

Chapter 2

LITERATURE REVIEW

When we have a large number of text files, say word, PDFs or other text data, searching could be an algorithmically time-consuming task. A sequential search for a word would take at worst a time proportional to the number of words which would lead to a very slow solution. Knowing which words, we have in a document, in the order they appear takes a long time, as we have to scan the entire document. Also, the process is further hindered by the fact that these documents will be books, which tend to comprise a lot of pages. When we have a sample space of a lot of books, most of which will be novels, it is quintessential to have an algorithm which helps us scan these documents and store the results in a structured way.

Also, we need to keep in mind that not all of the words scanned will be hard and require a meaning search. We need to classify the scanned data into two fields. To determine which word will be placed in which category, looking at each word individually will not only consume a lot of time, but also will be biased, as the words which a particular individual finds tough cannot be generalized. Hence, we need a method to classify all of these words in a non-biased way, which should be effective and correct as well.

Now comes one of the tougher tasks. We have to predict when the user will actually reach the word when reading the book so that we can display the word at the right time. We need an algorithm or a testing method to determine the reading speed of the user so as to customize the experience to his/her need. Once this testing method determines the reading speed of the user, it will help us determine the timing of the next difficult word, and also help us to predict the book the user is reading.

2.1 Existing methods to aid readers

There are a few methods available to aid readers in the process of finding meanings. The most common perhaps are traditional dictionary applications which provide a search functionality and some other features such as a word of the day to try improve the vocabulary of the user. Although dictionary apps are available very easily, a lot of users still prefer a quick google search to look at the meaning of a word.

Another revolution that came into the reading world was the Kindle by Amazon. In the Kindle device and its app, a simple yet effective solution is provided. As soon as the user clicks on any word he/she finds difficult, the app provides the meaning of the word, in a small pop-up window. Although easy to use and efficient in what it does, the kindle app will not do any good if you're reading a hard copy of a book. Also, predicting when the user will reach the word that is not provided here.

2.2 Full Text Search

Introduction

RDBMS support services that have a schema. Unstructured or semi- structured data may need to be indexed. It has become important to create a new platform to fulfil the demand of organization due to the challenges faced by traditional data. Big data helps to analyze growing volumes of structured transaction data, plus other forms of data that are often left untapped by conventional business intelligence (BI) and analytics programs. Elasticsearch is a big data technology which is schema less. It is a tool used to search big data. Elasticsearch uses the concept of denormalization for search. Elasticsearch uses the indexing concept. It is a document-oriented tool. Once the document is added, it can be searched within a next second as Elasticsearch is real time.

Elasticsearch is used for many use cases like analytics store, auto-complete, spell checker, alerting engine, and as a general-purpose document store; Full text search is one of it. It is a robust search engine that provides a quick full text search over

various documents. It searches within full text fields to find the document and return the most relevant result first. The relevancy of documents is good as Elasticsearch uses Boolean model to find document. As soon as a document matches a query, Lucene calculates its

score for that query, combining the scores of each matching term. The relevance of the document can be calculated using practical scoring function.

Elasticsearch uses the concept of inverted index for searching. When the query is fired, Elasticsearch looks into inverted index table to find the required data. It will show the relevant document in which the term is contained. An inverted index consists of a list of all the unique words that appear in any document, and for each word, a list of the documents in which it appears. Inverted index is the most commonly used method for FTS (Full Text Search). The inverted index searches the relevant document by mapping the term to its containing document. In the dictionary the terms are sorted which results in quick search.

2. Explanation of Elasticsearch working through examples

- (i) Creation of inverted index using Elasticsearch
- (ii) Working of Elasticsearch with stopwords list concept

(i) Creation of inverted index using Elasticsearch

If the user wants to find articles that talk about aircraft, you may search using the word “aircraft”. If you don’t have a special indexing technique, then all the records will be scanned to find a match, which is inefficient technique. “inverted index” offers the solution for this.

The following example elaborates the searching using Elasticsearch. Consider these documents to be indexed with the contents in it.

Document 1: “Elasticsearch is fast”

Document 2: “Elasticsearch is efficient”

Document 3: “Elasticsearch offers high speed”

Now to create the inverted index for the above documents, the contents of each document is split into separate words. After then, the lists of unique words, the document ids in which they are contained and the word frequency list are generated. So the inverted index generated for the above documents would be below:

Terms	Document	Position	Term Frequency
Elasticsearch	Document1,Document2 , Document 3	1,1,1	3
is	Document1,Document2	2,2	2
fast	Document1	3	1
efficient	Document2	3	1
offers	Document3	3	1
High	Document3	4	1
speed	Document3	5	1

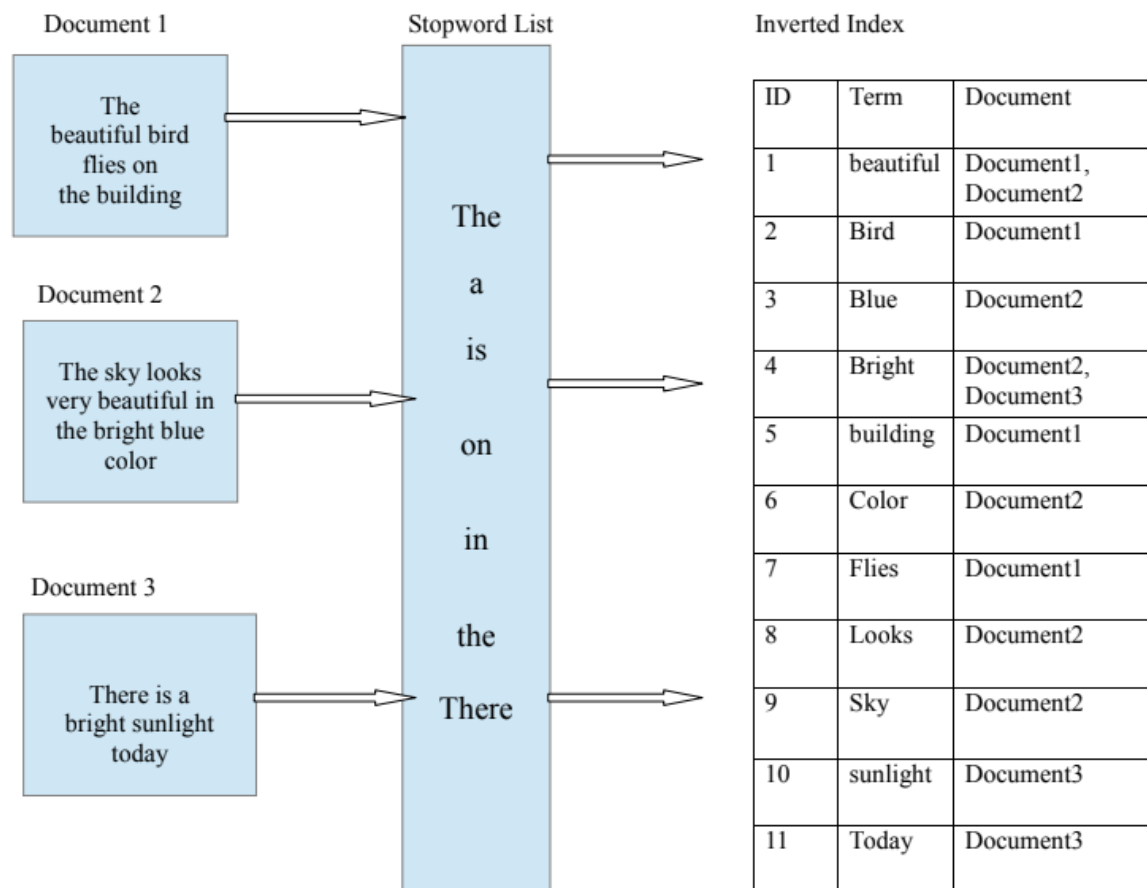
Figure 3: Creation of Inverted Index using Elasticsearch

If the query is fired to search the word “offers”, then Elasticsearch will look into its inverted index table. It will find that the word occurs in document 3 and will show that document. So due to the inverted index creation, the search becomes faster.

(ii) Working of Elasticsearch with Stopword list concept

The following diagram elaborates the working of Elasticsearch. It creates an inverted index internally.

Some words Seldom add value to a search. As per the above figure, the Elasticsearch uses the concept of Stopword list to fasten the search process. Elasticsearch has its own list of predefined stopwords. Stopwords can usually be filtered out before indexing. The inverted index is then created over the terms of document to make search faster.



Why we use it?

In order to determine the book that the user is reading, we need a structure that we could use to determine all the books that consist the word that the user has searched for. In order to do so in reasonable time, we use a data structure similar to an Inverted Index.

2.3 Determining the difficulty of words

1. The Frequency Effect

Reading is a coordinated execution of a series of processes which involve word encoding, lexical access assigning semantic roles, and relating the information contained in a sentence to earlier sentences in the same text and the reader's prior knowledge (Just and Carpenter, 1980). Successful comprehension of texts depends on the readers' semantic and syntactic encoding abilities (Marks et al., 1974), as well as their vocabulary knowledge in the language (Laufer and Ravenhorst-Kalovski, 2010; Nation, 2006). A general consensus of reading research is that lexical coverage/vocabulary knowledge are good predictors of reading comprehension (Bernhardt and Kamil, 1995; Laufer, 1992; Nation, 2001; Nation,

2006; Qian, 1999; Qian, 2002; Ulijn and Strother, 1990). A reader's vocabulary knowledge is largely related to the amount of exposure they have received to

words—often referred to as frequency effect. It is argued to be predictive of word difficulty (Ryder and Slater, 1988) and Leroy and Kauchak (2014) found that word frequency is strongly associated with both actual difficulty (how well people can choose the correct definition of the word) and perceived difficulty (how difficult a word looks). High-frequency words are usually perceived and produced more quickly and more efficiently than low-frequency ones (Balota and Chumbley, 1984; Howes and Solomon, 1951; Jescheniak and Levelt, 1994; Monsell et al., 1989; Rayner and Duffy, 1986). Consequently, a text with many high-frequency words is generally easier to understand than one with a number of rare words. Frequency of word occurrence affects not only the ease of reading, but also its acceptability (Klare, 1968). The frequency effect is based on a cognitive model assuming a higher base-level of activation for frequently-used words, so they require relatively less additional activation when they are being retrieved from the reader's mental lexicon (Just and Carpenter, 1980). This idea is supported by the findings that high-frequency words are more easily perceived (Bricker and Chapanis, 1953) and readily retrieved by the reader (Haseley, 1957). Going beyond this basic effect, in frequency-based accounts of Second Language Acquisition (Ellis, 2012), the frequency distribution of the input is a key determinant of acquisition, with regularities emerging through the learner's exposure to the distributional characteristics of the language input.

2. Experimental Setup

Before turning to the three experiments carried out, let us introduce the resources and the general procedure used. As source of the frequency and CD1 information, we used the SUBTLEXus (Brysbaert and New, 2009) and the SUBTLEXuk (van Heuven et al., 2014) resources. We ran all experiments with two distinct frequency resources to be able to study the impact of the choice of resource. As corpus for exploring the approach and 10-fold cross validation testing we used the leveled text corpus WeeBit (Vajjala and Meurers, 2012). For independent crosscorpus testing, we trained on WeeBit and tested on the exemplar texts from Appendix B of the Common Core State Standards (CommonCore, 2010). For machine learning, we used the basic k-nearest neighbor algorithm implemented in the R package class given that in our initial exploration

it turned out to perform on a par or better than other commonly used algorithms such as Support Vector Machine or Decision Trees.

2.1 The SUBTLEX Lists

The SUBTLEXus (Brysbaert and New, 2009) contains 74,286 word forms with frequency values calculated from a 51-million-word corpus of subtitles from 8,388 American films and television series broadcast between 1900 and 2007. The SUB-

TLEXuk (van Heuven et al., 2014) is the British counterpart, consisting of 160,022 word forms with frequency values calculated from a 201.7-million-word corpus of subtitles from nine British TV channels broadcast between January 2010 and December 2012. The SUBTLEX resources provide frequency information in several forms motivated in van Heuven et al. (2014); we made use of the frequencies given on the Zipf scale (\log_{10} of the frequency per billion words), as well as the CD values, for which each film or TV program counted as a context.

2.2 The WeeBit and Common Core Corpora

The WeeBit corpus used in a number of readability and text simplification studies (Vajjala and Meurers, 2012; Vajjala and Meurers, 2013; Vajjala and Meurers, 2014) was collected from the educational magazine *Weekly Reader* used in earlier readability research (Petersen and Ostendorf, 2009; Feng et al., 2010) and the BBC-Bitesize website. As summarized in Table 1, it is a 789,926-word corpus of texts labeled with five grade reading levels. The Common Core corpus consists of exemplar texts from Appendix B of the English Language Arts Standards of the Common Core State Standards. The corpus we use for testing in our experiments

is exactly the same as the one used by Nelson et al. (2012). They eliminated the lowest (K–1) level of the original six levels and removed repetition, dramas, and texts intended for teacher to read aloud, resulting in 168 remaining passages at five levels.

Grade Level	Age Group	# Articles	# Words / Article
WR Level 2	7–8	616	152.63
WR Level 3	8–9	616	190.74
WR Level 4	9–10	616	294.91
BiteSize KS 3	11–14	616	243.56
BiteSize GCSE	14–16	616	400.51

Table 1: Details of the WeeBit corpus

Why we use it?

While displaying the meanings of the words, we cannot go on and display the meanings to all the words. That would both be overwhelming and inefficient. Hence we need to come up with a factor that we could use to determine the difficulty of the words. This paper thus helps us determine such a criterion i.e. “frequency”. Which accounts for 60% of word difficulty.

2.3 Determining the speed of reading

Determining the speed of reading of the user is an essential part of the entire application. To determine this speed, a test has to be given by the user. Here, we have to focus on the difficulty of the extract we will give the user to read. Various tests are available which span across different levels of difficulties in a single extract and measure the time user has taken to read the whole text.

To make this process robust, a questionnaire is added at the end of the text to make sure the user has actually read the test in the most natural way possible. Once the speed of reading for the user is calculate, we can begin working on the calculations required to predict when the next word should pop up on the screen.

Chapter 3

IMPLEMENTATION

3.1 Structure of the System

The whole system is divided into several modules which include the conversion module, training module, searching module and the user interface. In this chapter we will study how each individual module works and is integrated together to form the entire system.

The system is built entirely using python and its accompanying libraries and APIs. This gives us a lot of flexibility and speed in building the application and the availability of external libraries certainly reduces the number of lines of code and the time it takes to write it.

The complete system has been built as a framework and hence can be used beneath any application to provide the various functionalities. For example if an application needs to search for books based on words and the reading intervals between them, the framework provides a simple function to do so.

Through training we gather a huge amount of data, account for the large number of books it trains across. To get a perspective, we store all the words that there are in all the books with their positions. Then there is a lot more. This data has to be stored somewhere. Hence, for the role of database, we choose MongoDB which provides us with a highly scalable architecture to support the amount of data that we generate.

Why MongoDB for Deep Learning?

Developers and data scientists can harness MongoDB as a flexible, scalable, and performant distributed database to meet the rigors of AI application development. MongoDB is the only database that harnesses the innovations of

NoSQL – data model flexibility and distributed, scale-out design – while maintaining the foundations of rich query capabilities and strong consistency that have made relational databases the platform for enterprise applications over the past three decades.

Flexible Data Model

MongoDB's document data model makes it easy for developers and data scientists

to store and combine data of any structure within the database, without giving up sophisticated validation rules to govern data quality. The schema can be dynamically modified without application or database downtime that results from costly schema modifications or redesign incurred by relational database Systems. This data model flexibility is especially valuable to deep learning, which involves constant experimentation to uncover new insights and predictions:

- Input datasets can comprise rapidly changing structured and unstructured data ingested from clickstreams, log files, social media and IoT sensor streams, CSV, text, images, video, and more. Many of these datasets do not map well into the rigid row and column formats of relational databases
- The training process often involves adding new hidden layers, feature labels, hyperparameters, and input data, requiring frequent modifications to the underlying data model.

A database supporting a wide variety of input datasets, with the ability to seamlessly modify parameters for model training, is therefore essential.

Rich Programming and Query Model

MongoDB offers both native drivers and certified connectors for developers and data scientists building deep learning models with data from MongoDB. The PyMongo driver is the recommended way to work with MongoDB from Python, implementing an idiomatic API that makes development natural for Python programmers. The community developed MongoDB Client for R is also available for R programmers. The MongoDB query language and rich secondary indexes enable developers to build applications that can query and analyze the data in multiple ways. Data can be accessed by single keys, ranges, text search, graph, and geospatial queries through to complex aggregations and MapReduce jobs, returning responses in milliseconds. To parallelize data processing across a distributed database cluster, MongoDB provides the aggregation pipeline and MapReduce. The MongoDB aggregation pipeline is modeled on the concept of data processing pipelines. Documents enter a multi-stage pipeline that transforms the documents into an aggregated result using native operations executed within MongoDB. The most basic pipeline stages

provide filters that operate like queries, and document transformations that modify the form of the output document. Other pipeline operations provide tools for grouping and sorting documents by specific fields as well as tools for aggregating the contents of arrays, including arrays of documents. In addition, pipeline stages can use operators for tasks such as calculating the average or standard deviations across collections of documents and manipulating strings. MongoDB also provides native MapReduce operations within the database, using custom JavaScript functions to perform the map and reduce stages. In addition to its native query framework, MongoDB also offers a high performance connector for Apache Spark. The connector exposes all of Spark's libraries, including Python, R, Scala and Java. MongoDB data is materialized as DataFrames and Datasets for analysis with machine learning, graph, streaming, and SQL APIs. Performance, Scalability & Redundancy Model training time can be reduced by building the deep learning platform on top of a performant and scalable database layer. MongoDB offers a number of innovations to maximize throughput and minimize latency of deep learning workloads:

- WiredTiger is the default storage engine for MongoDB, developed by the architects of Berkeley DB, the most widely deployed embedded data management software in the world. WiredTiger scales on modern, multi-core architectures. Using a variety of programming techniques such as hazard pointers, lock-free algorithms, fast latching and message passing, wiredTiger maximizes computational work per CPU core and clock cycle. To minimize on-disk overhead and I/O, WiredTiger uses compact file formats and storage compression.
- For the most latency-sensitive deep learning applications, MongoDB can be configured with the In-Memory storage engine. Based on WiredTiger, this storage engine gives users the benefits of in-memory computing, without trading away the rich query flexibility, real-time analytics, and scalable capacity offered by conventional disk-based databases.
- To parallelize model training and scale input data sets beyond a single node, MongoDB uses a technique called sharding, which distributes processing and data across clusters of commodity hardware. MongoDB sharding is fully elastic, automatically rebalancing data across the cluster as the input dataset grows, or as nodes are added and removed.
- Within a MongoDB cluster, data from each shard is automatically distributed to multiple replicas hosted on separate nodes. MongoDB replica sets provide

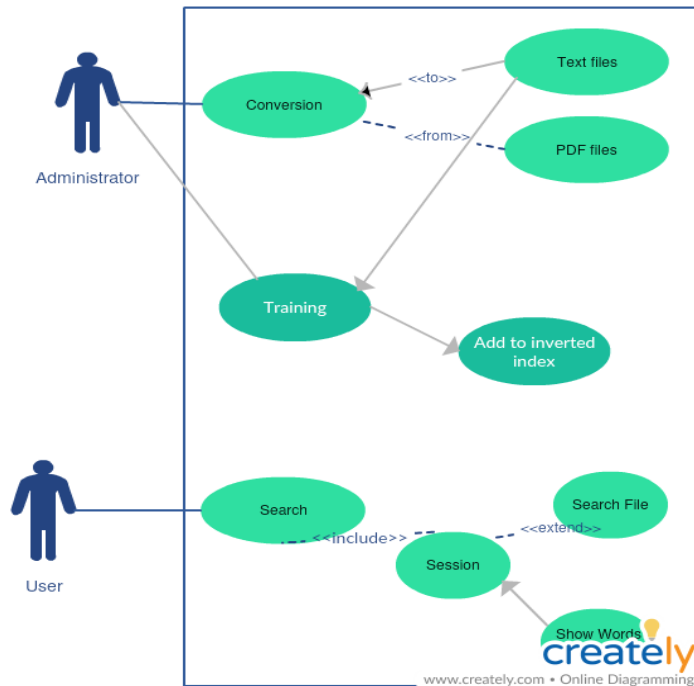
redundancy to recover training data in the event of a failure, reducing the overhead of checkpointing.

Proposed System goes through the following phases

1. We have a number of PDFs that need to be added to the inverted index data structure. We first convert these into text files with the same name.
2. Each of this text file is then fed to the training module which extracts each word that is present in the text file and adds it to the reverse index database.
3. Once this information is added to the underlying MongoDB database we can now perform the search operation.
4. When the user searches for the words, we record the times between the two consecutive searches and by estimating the number of words between the occurrence of two words and then searches for the file containing such a pattern
5. Once the system recognizes the book that the user might be reading they can now just take a glance at the screen and find the next difficult word that they might come across.

3.2 Use case of the proposed system

The system consists of three use cases conversion to text, training, search and recommendation



3.2.1 Conversion of files

Our system primarily aims at aiding the readers to better understand and enhance the reading experience. To train the model for it to search and then discover the book that the user might possibly be reading, we make use the PDF files to extract the text and ultimately the words from the files. We convert the PDF file to text files first as it is easier to extract the individual words from the text files.

PDFminer:

PDFMiner is a tool for extracting information from PDF documents. Unlike other PDF-related tools, it focuses entirely on getting and analyzing text data. PDFMiner allows one to obtain the exact location of text in a page, as well as other information such as fonts or lines. It includes a PDF converter that can transform PDF files into other text formats (such as HTML). It has an extensible PDF parser that can be used for other purposes than text analysis.

We use PDFMiner to convert multiple PDF files in a directory to text files. The module named 'ConvertToText.py' consists of various methods to choose the directory that contains the PDF files and then uses another function named **convertmultiple()** to convert these files to text and store it in another directory.

3.2.2 Training the System

Once we have the text files ready, we use them to extract the words and add them to the **InvertedIndex**. Let us create a context to understand the whole process.

1. We extract each word from the text files and store it in a Python Dictionary. While we do so, we also keep on recording the position of each word. We will later know how would that help.
2. These words are then stored in the MongoDB database in such a way that all the occurrences of the word in particular book are stored as a separate document

A single document would be of the following form:

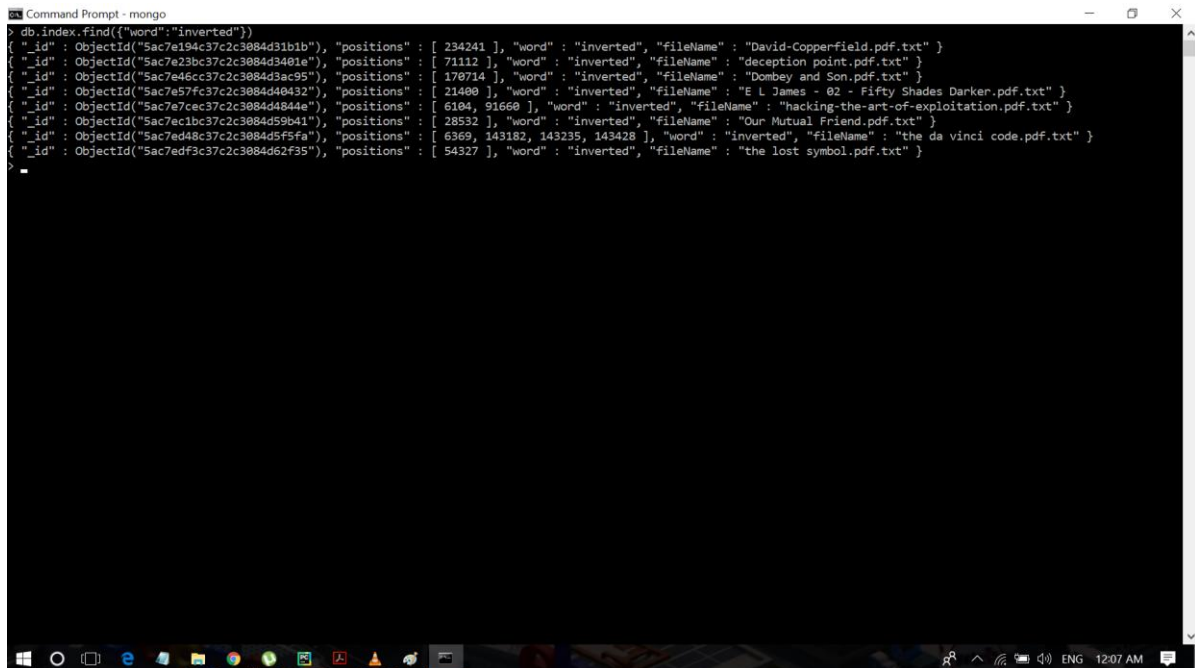
```
{ "_id" : ObjectId("5ac7d655c37c2c06f443a682"), "positions" : [ 2128, 2596, 4187, 10016, 10795, 13894, 18624, 24666, 35002, 37568, 39870 ], "word" : "eye", "fileName" : "1- Stormbreaker.pdf.txt" }
```

3. Along with this data we also record the frequency of all the words that have ever occurred in any of the books that we have recorded in the **"freq"** table. This helps us to understand the difficulty of the words (0.6 correlation)
4. We keep track of which files have we recorded by the system in the **"AllFileDirectory"** table.
5. In order to serve the user with all the word meanings once the particular book has been identified, the system also has a list of all the words belonging to a particular book in order in the **"WordsInFileList"**

We have seen how the inverted index works in the Literature section. The data structure that we use is a variation of the same. Instead of having a single entry for a particular **"word"** in which case it would become overwhelming to store the position data for each book in that single entry, we add multiple entries for each word. This allows us to store both the file

names that consists of the word, but also their exact positions. Hence, for each word that we scan for a particular book, we make separate entries with the exact positions of the word stored in a list. This helps in searching, as we will see the upcoming sections.

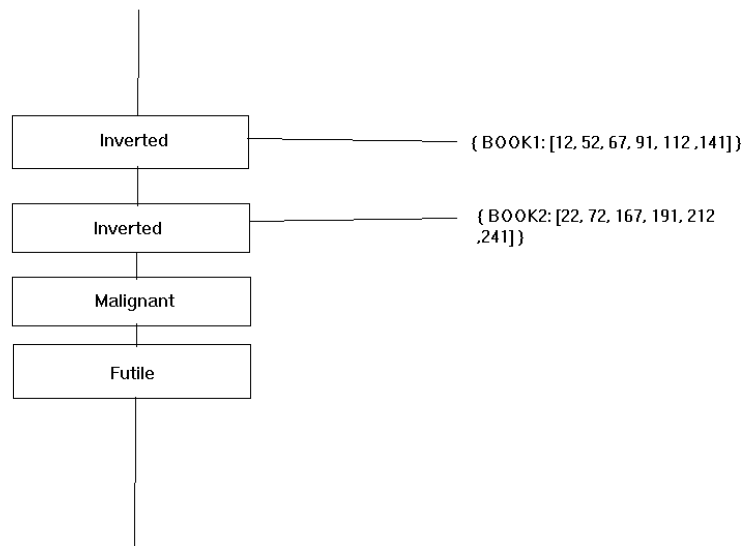
The following image shows an example of all the entries for the word “inverted” in the database.



```
Command Prompt - mongo
> db.index.find({"word":"inverted"})
{ "_id" : ObjectId("5ac7e194c37c2c3084d31b1b"), "positions" : [ 234241 ], "word" : "inverted", "fileName" : "David-Copperfield.pdf.txt" }
{ "_id" : ObjectId("5ac7e23bc37c2c3084d3401e"), "positions" : [ 71112 ], "word" : "inverted", "fileName" : "deception point.pdf.txt" }
{ "_id" : ObjectId("5ac7e46cc37c2c3084d3ac95"), "positions" : [ 170714 ], "word" : "inverted", "fileName" : "Dombey and Son.pdf.txt" }
{ "_id" : ObjectId("5ac7e57fc37c2c3084d40432"), "positions" : [ 21400 ], "word" : "inverted", "fileName" : "E L James - 02 - Fifty Shades Darker.pdf.txt" }
{ "_id" : ObjectId("5ac7e7c37c2c3084d4844e"), "positions" : [ 6104, 91660 ], "word" : "inverted", "fileName" : "hacking-the-art-of-exploitation.pdf.txt" }
{ "_id" : ObjectId("5ac7ec1bc37c2c3084d59b41"), "positions" : [ 28532 ], "word" : "inverted", "fileName" : "Our Mutual Friend.pdf.txt" }
{ "_id" : ObjectId("5ac7ed48c37c2c3084d5f5fa"), "positions" : [ 6369, 143182, 143235, 143428 ], "word" : "inverted", "fileName" : "the da vinci code.pdf.txt" }
{ "_id" : ObjectId("5ac7edf3c37c2c3084d62f35"), "positions" : [ 54327 ], "word" : "inverted", "fileName" : "the lost symbol.pdf.txt" }
```

All the entries for word “inverted”

Another thing to notice when working with the framework is that all the words that are there are stored in lower case form. This helps save space and also allows us to keep any anomalies away.



Data structure similar to Inverted Index

3.2.3 Performing a search

The system will allow the user to search the meanings for the words that they find difficult. While doing so, it will also keep track of what word the user has searched for and accordingly depending upon the time interval between the searches, the system will try and make a prediction regarding what book the user might be reading. Hence this part will explain how the user performs searches and how does the system go on to predict the book that the user might be reading.

The user is provided with a small window GUI to perform search of the words. When the user searches for a word, they expect to get the meaning of the word. We have utilized a third party API named **Wordnik** to get the meanings or definitions of the words. Wordnik offers both examples and definitions of words through API calls. To use the API we first need to register with it on their websites and get the API key for our account. Using the key and the URL we create a wordnik client which allows us to communicate with the API.

```
apiUrl = 'http://api.wordnik.com/v4'
apiKey = 'cbf25abcf44677fc4a12f0b721e0c2f4c5d9c5e13f14a9079'
client = swagger.ApiClient(apiKey, apiUrl)
wordApi = WordApi.WordApi(client)
```

Image: Client creation for Wordnik API

As this will require an internet connection, it introduces a dependency in our system, that the system must always be connected to the internet. Yet, if the underlying framework was to be used along with a full fledged Search engine it would surpass this problem, as the search engine would anyway require an internet connection. The function call to get the definition of the word entered by the user would look like the following snippet.

```
definition = []
try:
    definition = self.wordApi.getDefinitions(word)
except():
    self.meaningoutput.configure(text="Some error occurred while trying to connect to API")
    self.labels.configure(text="")

if definition is not None:
    text = definition[0].text + " "*(200 - (len(definition[0].text) % 200))
    self.meaningoutput.configure(text=text, font='Helvetica 9 italic')
    time.sleep(4)
```

Image: Using Wordnik to get the definition

As can be seen, the call to the API has to be enclosed in a try-except block. This is because

if a connection error occurs, the API will return an error which if not caught will hinder the working of the system. If the API encounters a word that doesn't have a meaning, it will return an empty object. Hence, we first check whether the object is empty before we try and print the result to avoid any errors.

If the word is a valid one, the API returns a list of all the definitions that are available. The system then goes on to display those in the GUI. While this is happening, the system in the background is also doing something else that we will see in the next section.

3.2.4 Predicting the book

While the user searches for the words that he/she finds difficult, while providing no context about what book might they be reading, the system on its own does the job of prediction in the following way.

When the user opens up the application for the first time, they are presented with a paragraph in the English language and are made to read it. The time(**T**) in minutes that they took to read the text is recorded and is used to determine the reading speed of the user. If there are say **W** words in the text, the reading speed of the user(**R**) is given as:

$$R = \frac{W}{T} \quad \text{eq. 3.2.3.1}$$

This reading speed is then used to make predictions regarding the books and also while displaying the words.

Next it comes to searching. When the user searches for the first word (**W1**) a timer starts recording the time in seconds, the timer stops when the user searches for the second word **W2**. Let's say that the time interval between the search of two words is **t1**.

$$D1 = t1 \times R \quad \text{eq. 3.2.3.2}$$

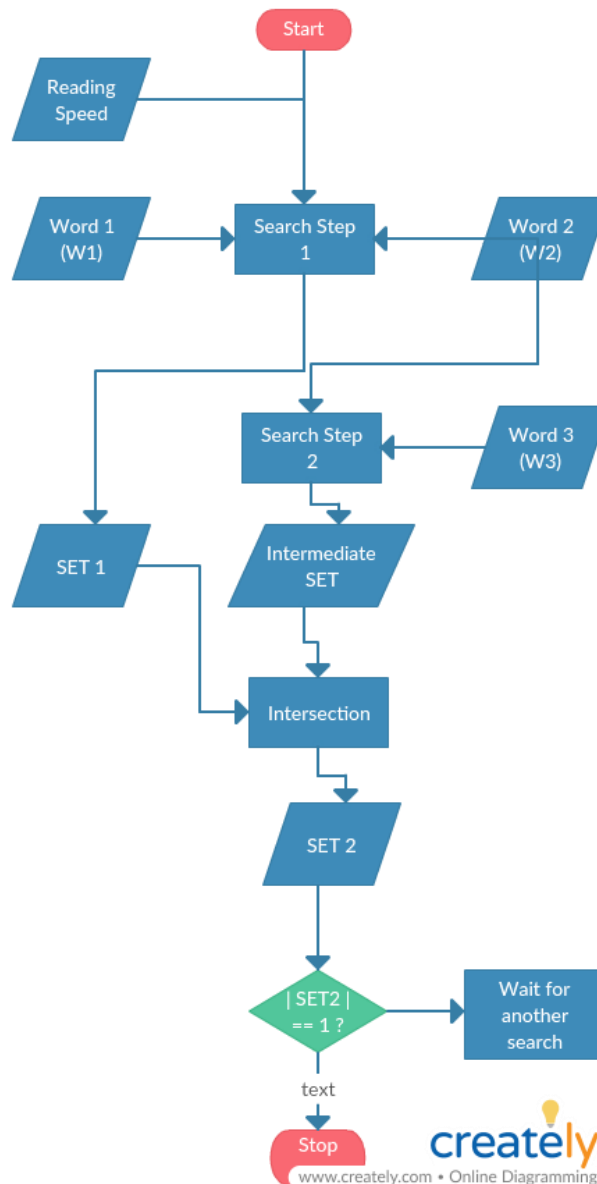
Using the time, we can estimate the number of words that might have occurred between the two words that the user searched for. Say it's **D1**. Using this information,

namely **W1**, **W2** and **D1** we first create a set of all the files that contain the two words W1 and W2 at a distance of D1 lets call it **SET1**.

$$SET1 = \{ x \mid distance(W1, W2) = D1 \text{ and } W1 \& W2 \in x \} \quad \text{eq. 3.1.2.3}$$

If $|SET1| = 1$ our search ends as we have found the book that contains the two words at the desired distance. If such is not the case, the system then resets the timer and waits for the user to search another word W3. When that happens, a similar set is created and an intersection is performed between the two sets thus yielding a set of books containing the words at appropriate distances. This search thus goes on until a set containing only one book remains.

$$SET2 = SET1 \cap IntermediateSet \quad \text{eq. 3.1.2.4}$$



Flow chart of the determination process

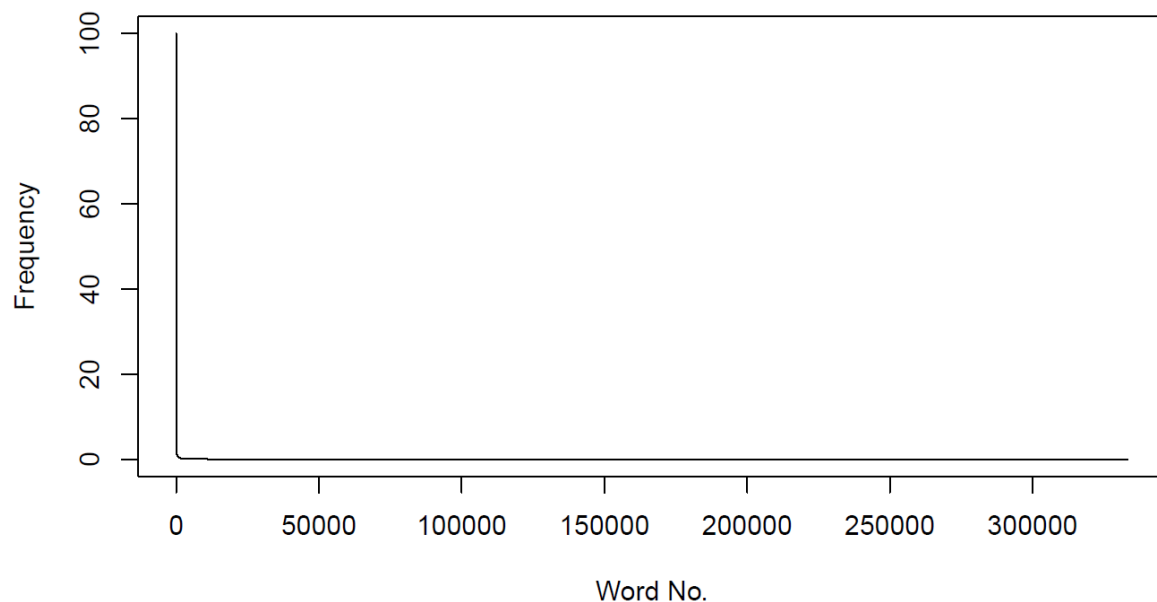
We thus now know what book the user might be reading. This then brings us to our next step i.e helping the user discover meanings of words without having to type out any of the words.

One of the things that need to be taken care of is, that if the user hasn't searched for a word for some threshold time, the search session should expire. In order to stop the application for producing vague results.

3.2.5 Displaying all the words and their meanings

At this point the system knows the speed with which the user reads, the book that the user

might be reading, it even has all the words from that book in order. Now the system displays on the GUI the upcoming words from the book. To do so the **MeaningSearchGUI** module comes into play. It provides the user with a GUI consisting of words from the book scrolling across the screen while also providing the feature to pause at a word to read out meanings. The user can move forward and backwards across the words at their convenience. Now one of the major issues is, that we cannot display the meanings to all the words there are because that would be inefficient and overwhelming at the same time. It would thus be better to decide upon a metric to determine the difficulty of a word. As was discussed in the literature review section in the word difficulty paper, the difficulty of a word depends upon various factors that include the frequency, context and number of syllables and a few more. While it would be an overwhelming task to determine the context for the words in different books, the frequency of words could provide us with an appropriate metric to deal with this problem. The frequency of words tends to show a 0.6 correlation with word difficulty. Hence, to decide upon which words are to be displayed, we take into account, the frequency of the words. We decide upon a threshold frequency β which is decided by normalizing the global word frequencies between 0-2000. That seems like a huge number. We had to scale it that way because the words with very high frequencies shunted the scores of the lower frequency words.



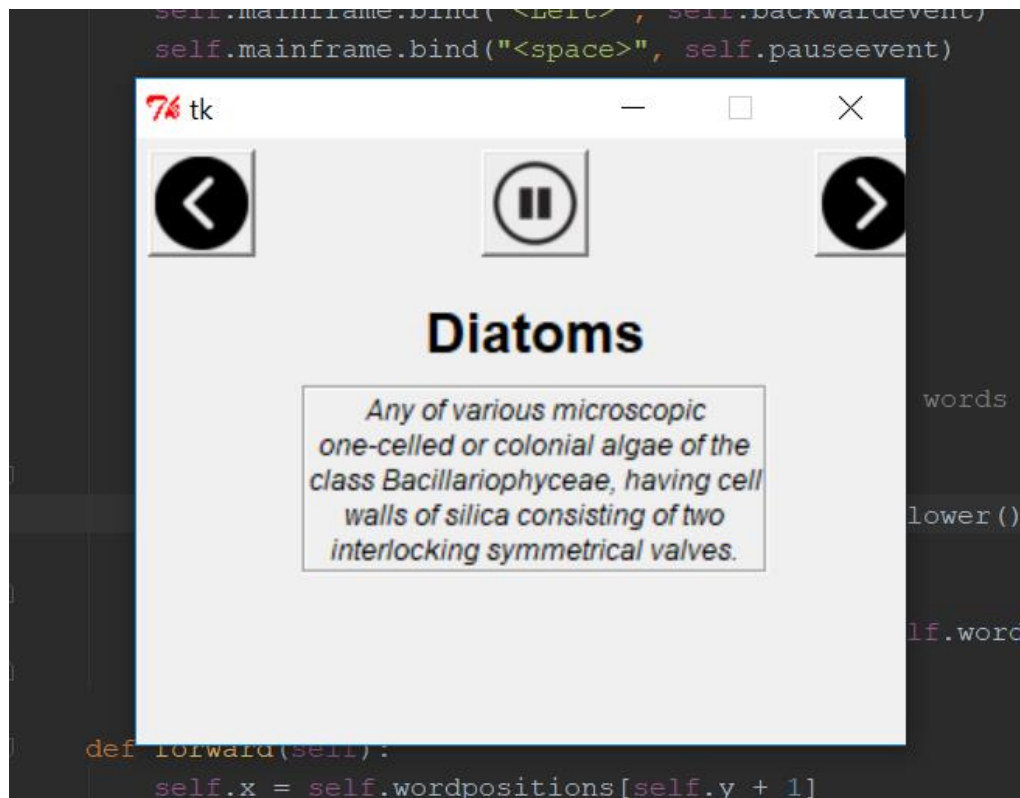
Normalized Frequency Vs Word plot

The above plot shows us the staggering differences in frequencies, with frequencies normalized between 0-100. Most frequencies tend to reach near zero values. Which is why the scale had to be expanded.

Now, for deciding what the value of β would be, we take a safe path. From the graph it could be observed that at one point the frequencies drop suddenly. We choose this point as the threshold value, terming all the values below this threshold as difficult.

The words occurring in between the harder words are also iterated through, they are just not displayed, this allows to have an appropriate time interval between the difficult words. Each difficult word and its meaning are displayed for 4 seconds to allow the user to read through. If the user wishes to, they can pause the flow and return back to reading whenever they wish.

The feature to move backward and forward through the words is also provided. This can be accessed using the on-screen buttons, or the arrow keys. Likewise, the SPACE button can be used to pause the flow.



GUI to continuously display the words and their meanings

Chapter 4

RESULTS AND DISCUSSIONS

4.1 A standalone application to aid book readers

After integrating all the data, we had with the developed algorithms as mentioned in the implementation section, we now have a standalone application which will aid users reading hard-copy of books.

The users had to go through a test to determine their reading speed, which then the application uses to optimize the results. The book detection capacity of the application worked accurately, providing the user a generous gap between two-word searches, thus including a kind of mitigation for human errors.

Once the book was determined by the application, the pop-up window, after loading in a matter of seconds, started displaying the difficult words for the user. The time gap between the words was determined with the help of the reading speed of the user determined earlier. A considerable amount of time was provided for the user to read and understand the meaning of the word. Also, the pause button worked just fine and made the user feel free to take his/her time to look at the word meanings or to stop reading for a while, but not lose the synchronization developed with the application. The application was able to match the reading speed of the user and show word meanings in time, thanks to the tolerance limits added.

Thus, overall, the app performed admirably and the solution to the problem statement was found.

4.2 Other implementations

Along with the classic stand-alone application, other implementations of the project are possible. These other implementations explore possibilities of the developed project being used in different scenarios with respective prerequisites.

One of the implementations of the project is to integrate it inside a pdf or ebook reader, so that reading soft-copies of books becomes hassle-free. When implemented, the pdf or ebook reader (like the Kindle) will have a small section

somewhere on the screen where the pop-ups will appear. Essentially, when the user is reading the text(book) on the screen, as soon as the user reaches a particular hard word, the pop-up section will show the meaning of that word. It will be there for a considerable amount of time, in which the user will be able to read the meaning. Unlike the Kindle, where the user has to select and highlight the word to see its meaning, here, he/she will just have to shift eyes to the pop-up section for the meaning of the word he/she is looking for.

Another implementation of this module will be if it is integrated inside a search engine. Many people still use the quickest method, just an internet search of the word for the meaning they're looking. Also, due to current rises in the AI implementations of voice assistants, most people just talk to their smartphone for a quick reply. If implemented correctly within a search engine like Google, it will automatically recommend users books it thinks they're reading. The user will not even have to know that such an application exists and have to download it separately. Whenever a person searches for a few word meanings in a row, the voice assistant (in smartphones) and the search engine itself (in browsers) will show the user recommendations of the book he/she is reading. Also, as a search engine has massive reach over all the data present, storage of soft copies of books and training of the tool with those will not be a huge load anymore.

4.3 Future scope

The tool developed will be useful not only in the current scenario, but also in the future. As AI and machine learning make progress and find a wider way into our lives, there won't be many aspects it will not affect. This tool, combined with the future optimization of neural networks, ML algorithms, sensors and their capacity to record real time data, can change the way the next generation will read books.

People will have a choice to take a book, a hard or a soft copy, sit down, relax and read the book as they want, taking as many breaks in between. Their habits, the time of the day and how it changes their reading speed will be taken into consideration with advanced sensory networks and this data will be channeled into improving the tool even more.

This tool and the way the algorithms are used in the project may also affect upcoming projects and optimizations in other applications dealing within the same domain.

Chapter 5

CONCLUSION AND FUTURE ENHANCEMENT

5.1 Conclusion

After integrating the data, we collected, the algorithms built and implemented, and the results from various research papers referenced, we now have a tool which can efficiently:

- Recognize the book the user is reading based on the words he/she has searched.
- Successfully record the reading speed of the user.
- Start displaying the meanings of the tough words the user encounters at the exact time he/she reaches that word in the book.
- Pause/resume the process as per the user's commands.

The recognition process of the book the user is reading was quite a comprehensive process, as it took into consideration all of the factors which affect the determination of the book, viz. the reading speed of the user, the words (two or more) which have been searched and the gap between those words based on the average reading speed of the user. This process made up the first part of the tool and took 2-3-word searches at the maximum to guess the book correctly from the database. Even as the number of books in the database will increase to tens of thousands, it will not take more than 5-6 searches, as the exact combination of those particular words, in that order and at that exact word distance is almost a mutually exclusive property for any particular book, or any pair of pages.

The tool developed was equipped with large enough time windows to give the user the time to read the meaning, as well as provide some level of fault tolerance. Also, the tool showed great resilience to erroneous matching of speed with that of the user's by re-calculating the reading speed of the user. At this stage, a small test can be added based on the excerpt to test whether the user has read the given text properly or not.

The threshold for the words to be considered as difficult was set by taking into consideration a huge database of all the words used in the English language, considering the frequency of their occurrence as a parameter. This enabled us to determine which words the average

reader may consider as a difficult one. This, of course, does give us a lot of accuracy, but not quite complete. This is because difficulty of words for a person is a subjective matter and cannot be determined 100% accurately based on some mathematical factors alone. In spite of this, the accuracy of the tool was above satisfactory, providing with meanings the user considered difficult in almost all of the cases. To further optimize this, a small vocabulary test could be included into the first-time installation process of the application.

The tool thus provided an excellent and smart solution to the problem statement, and the efficiency of the solution provided was quite high, as the results were satisfactory and not a lot of erroneous cases were seen.

Now, as for what the future holds, the tool will only increase its efficiency as it is trained more and more on a lot of books. Advances in AI and other technologies which will be able to record a whole lot of factors which affect the reading capacity and the vocabulary level of a user will help the tool to be even more accurate by interacting with the user to a whole new level. The tool will not only change the reading scenario for novel readers, but also for students studying in the classroom. New, interactive ways to teach students and make them read and learn can be fabricated. A huge number of ways to use the tool come to mind.

As this newly developed technology grows, further optimizations made to it will make it compatible with other future projects which will be able to use this as a module to complete a specific function. This could be used as a reference to build something new, something larger.

The tool will find its way into future implementations as a standalone app, in pdf/ebook readers and also in search engine enhancement. The future scope of the project is wide, as optimizations in sensor networks and AI not only facilitates, but also encourages further implementations of the tool.

5.2 Failure Conditions

Failure conditions for the system to make it inoperable

- The PDF files that were used to gather the training data should be in the right format in order to provide correct data
- An internet connection is needed at all times in order for the system to work correctly

- The system would face difficulty in cases where the book that the user is reading requires extra cognitive thinking to read and understand.

5.3 Future Enhancement

By finding the alternative way to overcome the above problems is the area of research.

- Better approach: In future more, sensors integrated with the mobile devices can be used to track the users cognitive and physical state to better adjust the speed
- There are many variables that affect the reading of a user. In future if there are technologies that could keep track of that information, this system could be enhanced much further

5.4 Contribution

Given the advancement of technology over the years, it is safe to say that technology has almost overpowered every corner of our lives. Gone are the days when having practical experience of an activity required an actual execution in the real world. Nowadays, no matter whatever we desire to try our hands at, it's just one tap away. And all the credit goes to the digital devices such as our mobile phones, tablets, personal computers etc. All these things are fancied by every age group in any sector of his/her professional or personal life. People have become glued to devices to such an extent that, in today's scenario, no matter what experience they want to have it is sure going to be a virtual one.

Like we know everything has its own merits and flaws. So what good comes out of this? Gaining a learning experience by just sitting in a corner of your bedroom and being able to explore the world by just scrolling through your mobile phones? This entire idea of embedding a real-world experience into a virtual platform by the use of technology surely is lucrative. Also, this would keep doing good to the mankind until and unless it's not killing our natural instincts. And this would only be possible if we always try to maintain a thin line between these two platforms

The basic objective of our framework lies in the idea of how we could be able to use technology for the betterment of society. We focused on how we would support in the growth of user's natural instincts with the help of our application. This can only be maintained if we let the users follow their conventional methods of learning and use

technology for maximum optimization of the learning process. Since we were motivated towards contributing something for the avid readers we took the most rational way i.e. hard copy reading as our experimentation base and try to build it upon with our technical knowledge.

Almost 80 percent of the readers in the world are hard copy readers. So rather than shifting the entire population towards e-reading why not just provide them a platform where they could still follow their own way and integrate technology into it only in the required form possible. That's exactly where our application could contribute in the best way possible. This would save the efforts of trying to convert the majority of readers into e-reading which they might not be fascinated with. This would not only save a lot of resources in the technological front but also contribute towards adding magnitude to social development in an optimized manner. So that you still can laze in the corner of your bedroom, sip your cup of coffee, have a physical interaction with your book in the real world and yet use technology in way which would boost your learning process.

REFERENCES

Developing the Reverse Index Data structure and the search algorithm

Pearl, J. (1984). Heuristics: intelligent search strategies for computer problem solving.

Subramaniam, P., Zoss, J., Ying, J. J., Caltabiano, M., & Malden, M. S. (2004). *U.S. Patent No. 6,728,702*. Washington, DC: U.S. Patent and Trademark Office.

Zobel, J., & Moffat, A. (2006). Inverted files for text search engines. *ACM computing surveys (CSUR)*, 38(2), 6.

Determining the difficulty of the words

Breland, H. M. (1996). Word frequency and word difficulty: A comparison of counts in four corpora. *Psychological Science*, 7(2), 96-99.

Tamayo, J. M. (1987). Frequency of use as a measure of word difficulty in bilingual vocabulary test construction and translation. *Educational and Psychological Measurement*, 47(4), 893-902.

Rudell, A. P., & Hua, J. (1997). The recognition potential, word difficulty, and individual reading ability: On using event-related potentials to study perception. *Journal of Experimental Psychology: Human Perception and Performance*, 23(4), 1170.

Kibby, M. W. (1977). Note on relationship of word difficulty and word frequency. *Psychological Reports*.

Zobel, J., & Moffat, A. (2006). Inverted files for text search engines. *ACM computing surveys (CSUR)*, 38(2), 6.

Elish-Piper L 2010. Understanding reading comprehension: Information and ideas for parents about reading comprehension. *Illinois Reading Council Journal*, 38(3):49-52.

Farr R & Carey RF 1986. *Reading: What can be measured?* (2nd ed). Newark: International Reading Association.

Feasibility of Word Difficulty Prediction

Ricardo Baeza-Yates^{1,2(B)}, Mart´ı Mayo-Casademont², and Luz Rello³

"Dolch word list." Wikipedia, n.d. Web. 15 Nov. 2014.

Fry Edward. Dr. Fry's Spelling Book Levels 1-6. Teacher Created Resources, 2000.
Print

"Sight Word Lists." Student Resources. Tarpey Elementary School, n.d. Web. 15 Nov. 2014.

"Sight Words: Grades K, 1, 2 & 3." Pawnee CUSD 11. Simplified Online Communication System, n.d.
Web. 15 Nov. 2014.

U.S. Department of Education, Office of Career, Technical, and Adult Education (OCTAE). "Dolch Basic

Sight Word List." Literacy Information and Communication System (LINCS), n.d.
Web. 15 Nov. 2014.

"WISD High Frequency Word Lists by Grade Level." Waxahachie Independent School District, n.d.
Web. 15 Nov. 2014.

"Assessing the Lexile Framework: Results of a Panel Meeting." National Center for Education Statistics, Aug 2002.

"How Lexiles Harm Students." Mike Mullen, October 22, 2012
<http://mikemullin.blogspot.com/2012/10/how-lexiles-harm-students.html>