

# Enhancing Phrase Retrieval for Safety Data Sheet Authoring



**Harry Baines**

Computing and Communications  
Lancaster University

This dissertation is submitted for the degree of  
*MSc Data Science*

September 2020

## Acknowledgements

Throughout the course of this project I have received a tremendous amount of support and guidance. I would like to thank my industry supervisor, Tom Hudson, who provided invaluable support to help maximise the effectiveness of the work undertaken in this project.

I want to thank all those at Yordas for giving me the opportunity to undertake a complex project with you this summer. I would also like to acknowledge all members of the IT team at Yordas who gave me motivation, assistance and guidance when necessary during the course of this project.

I would like to thank my academic supervisor, Keivan Navaie, who provided valuable academic advice to assist in writing this dissertation.

I would also like to thank Simon Tomlinson, Emma Eastoe, Chris Edwards and Clement Lee for their impeccable management of the Data Science programme at Lancaster. I would also like to acknowledge the help and support offered by Louise Innes from the Faculty of Science and Technology for her guidance to help write this dissertation.

Finally I would like to thank my family and friends who supported and motivated me throughout and helped to ensure this work was of the highest quality I could have achieved.

## **Abstract**

The adoption of artificial intelligence in information retrieval systems has revolutionised the way users retrieve relevant information from document collections. In this paper we seek to explore a wide range of these techniques to facilitate the development of an intelligent search engine. This engine will enhance the predictive capabilities of the phrase retrieval procedure for a safety data sheet (SDS) authoring software application. We iteratively implement a series of prototype models to experiment with a range of features found during research to create a refined prototype model to provide the foundation for a proposed search engine architecture. We conclude this paper by evaluating the performance of the refined prototype and quantify to what extent the SDS authoring experts agree on the relevance of the generated search results. The evaluation indicates the refined prototype produces very relevant results across all test queries and addresses many of the shortcomings found with the existing search procedure.

# Table of contents

List of figures	vii
List of tables	ix
<b>1 Introduction</b>	<b>1</b>
1.1 Safety Data Sheet Authoring at Yordas . . . . .	1
1.2 The Current System . . . . .	2
1.3 General Project Aim . . . . .	3
1.4 Aims and Objectives . . . . .	4
1.4.1 Research appropriate techniques used to implement smart search models . . . . .	4
1.4.2 Implement a series of prototype models . . . . .	4
1.5 Report Outline . . . . .	6
<b>2 Background</b>	<b>8</b>
2.1 Information Retrieval . . . . .	8
2.2 Retrieval Models . . . . .	10
2.2.1 The Boolean Model . . . . .	10
2.2.2 The Vector Space Model . . . . .	11
2.2.3 The Probabilistic Model . . . . .	13
2.3 The Inverted Index . . . . .	15
2.4 Preprocessing . . . . .	15
2.5 Spelling Correction . . . . .	16
2.6 Term Proximity . . . . .	17
2.7 Learning to Rank . . . . .	17
2.8 Chapter Summary . . . . .	20
<b>3 The Dataset</b>	<b>21</b>
3.1 Data Description . . . . .	21
3.1.1 Phrases . . . . .	21
3.1.1.1 Phrase Character Counts . . . . .	24
3.1.1.2 Phrase Word Counts . . . . .	24
3.1.2 Query Logs . . . . .	25
3.1.2.1 Phrase Searches . . . . .	25

3.1.2.2	Phrase Selections . . . . .	27
3.2	Limitations of the Dataset . . . . .	29
3.3	Chapter Summary . . . . .	29
<b>4</b>	<b>Methodology</b>	<b>31</b>
4.1	Search Engine Prototypes . . . . .	31
4.1.1	Prototype 1: Fuzzy Search Engine . . . . .	32
4.1.1.1	Implementation . . . . .	32
4.1.1.2	Issues Encountered . . . . .	34
4.1.2	Prototype 2: TF-IDF Cosine Search with Spelling Correction . . .	34
4.1.2.1	Implementation . . . . .	36
4.1.2.2	Spelling Correction . . . . .	37
4.1.2.3	Issues Encountered . . . . .	38
4.1.3	Prototype 3: TF-IDF Cosine Search with Spelling Suggestions . .	39
4.1.3.1	Implementation . . . . .	39
4.1.3.2	Issues Encountered . . . . .	40
4.2	Proposed Architecture . . . . .	40
4.3	Chapter Summary . . . . .	42
<b>5</b>	<b>Results and Discussion</b>	<b>43</b>
5.1	Evaluation . . . . .	43
5.1.1	Relevance Judgements . . . . .	44
5.1.2	Relevance of Search Results . . . . .	45
5.1.3	Ranking of Search Results . . . . .	46
5.1.4	Investigating Agreements of Relevance Scores . . . . .	47
5.2	Prototype Feedback . . . . .	49
5.3	Search Engine Features Questionnaire . . . . .	50
5.3.1	Feature 1: Spelling Correction . . . . .	50
5.3.2	Feature 2: Multi-Word Order-Independent Search . . . . .	50
5.3.3	Feature 3: Phrase Autocompletion . . . . .	51
5.3.4	Feature 4: Multi-Word Prefix Search . . . . .	51
5.3.5	Feature 5: Online Learning . . . . .	51
5.3.6	Feature 6: Frequently Selected Phrases . . . . .	52
<b>6</b>	<b>Conclusions</b>	<b>53</b>
6.1	Review of Aims and Objectives . . . . .	53
6.2	Future Work . . . . .	55
6.3	Deviations from the Proposal . . . . .	56
6.4	Lessons Learned . . . . .	56
6.5	Final Remarks . . . . .	57
	<b>References</b>	<b>58</b>
	<b>Appendix A HAZEL UI</b>	<b>61</b>

Appendix B	Phrase Selections	64
Appendix C	Questionnaire Responses	65
Appendix D	Project Specification	71

# List of figures

1.1	HAZEL system architecture. . . . .	2
2.1	General information retrieval system architecture [16]. . . . .	9
2.2	An example term-document incidence matrix for words used by Shakespeare in some of his plays. A value of 1 indicates the presence of a term with 0 indicating the absence of a term [20]. . . . .	11
2.3	Cosine similarity illustration [7]. . . . .	13
2.4	Term frequency behaviour illustration [5]. . . . .	14
2.5	Basic inverted index structure [1]. . . . .	15
2.6	Common spelling error types considered in the Damerau-Levenshtein distance metric [21]. . . . .	17
2.7	Learning to rank system [17]. . . . .	18
2.8	Learning to rank approaches [32]. . . . .	19
2.9	Percentage of queries where a user viewed the search result presented at a particular rank [11]. . . . .	19
3.1	Example phrase record from the phrases table. . . . .	22
3.2	Phrase word counts (top 100 counts). . . . .	25
3.3	Example phrase search record from the phrase_searches table. . . . .	26
3.4	Phrase searches time series. . . . .	26
3.5	Example phrase selection record from the phrase_selections table. . . . .	27
3.6	Phrase selections time series. . . . .	28
4.1	Search engine architecture (prototype 1). . . . .	33
4.2	TF-IDF cosine similarity search engine architecture (prototype 2). . . . .	36
4.3	TF-IDF cosine similarity search engine architecture (prototype 3). . . . .	39
5.1	Total agreed binary and graded relevance scores for groups A and B. . . . .	48
5.2	Cohen's $\kappa$ values for both agreed binary and graded relevance scores for the allocated search results to groups A and B. . . . .	49
A.1	Safety Data Sheet blank template (left) with HAZEL add-in (right). . . . .	61
A.2	HAZEL phrase search. . . . .	62
A.3	HAZEL phrase translations. . . . .	63
A.4	Regular and prototype search procedures toggle component. . . . .	63

C.1	Questionnaire responses for feature 1. . . . .	65
C.2	Questionnaire responses for feature 2. . . . .	66
C.3	Questionnaire responses for feature 3. . . . .	67
C.4	Questionnaire responses for feature 4. . . . .	68
C.5	Questionnaire responses for feature 5. . . . .	69
C.6	Questionnaire responses for feature 6. . . . .	70



# List of tables

1.1	Example user queries with intended phrases. . . . .	3
3.1	Phrase examples. . . . .	23
3.2	Phrase character statistics for active English phrases in HAZEL’s produc- tion database (as of 11th August 2020). . . . .	24
3.3	Phrase word counts for active English phrases in HAZEL’s production database (as of 11th August 2020). . . . .	24
3.4	Search text character counts for phrase selections (as of 11th August 2020). . . . .	28
B.1	Phrase selection counts from phrase_selections (as of 11th August 2020). . . . .	64

# Chapter 1

## Introduction

In this chapter we give an overview of the current system including details of the prototype authoring application and the search procedure it employs to retrieve phrases. A set of intended phrases will be provided in addition to example user queries to demonstrate the drawbacks of the search procedure. This will motivate the need for a more complex search algorithm to be implemented into the application. We then provide the general aim of the project, alongside further aims and their associated sub-objectives.

### 1.1 Safety Data Sheet Authoring at Yordas

Yordas Group is a leading international provider of scientific, environmental, human health, global regulatory and sustainability services. Yordas' Hazard Communications team (HZC) work to author and audit safety data sheets (SDS) for a range of clients with an array of substances for which this documentation is required. An SDS is a structured document containing occupational safety and health data, including all legally required information to accompany a product such as the chemicals it contains, any hazards the chemical presents and information on handling, storage and emergency measures in case of an accident.

Yordas have implemented and deployed a prototype authoring software application (HAZEL), accessible via a Microsoft Word Add-in, to 5 Yordas authoring experts in HZC to facilitate the entry of both complex and standardized phrases into an SDS template, with the added ability to easily translate between any of the 43 currently supported languages. This functionality is achieved via API communication between the authoring software and the list of phrases, represented in different languages, which are stored in databases managed on Yordas' servers. Following deployment of HAZEL on 13th of January 2020, the system has accumulated a corpus of time series data relating to phrase searches, selections and translations for a given SDS, along with a timestamp when each interaction was carried out.

## 1.2 The Current System

Phrases can be searched through the Add-in UI in Microsoft Word using a search-as-you-type feature which, on each keystroke, populates a list of selectable search results. The UI for this functionality is shown in Appendix A.1 for a template SDS.

A phrase is inserted at the current cursor location in the SDS upon selection from the search results. The current phrase search algorithm implemented in HAZEL is reliant on substring matching and leverages case-insensitive search, that is, a given phrase is only returned if the lowercased phrase contains a substring equal to the lowercased user query at any point in the phrase. This is used in conjunction with the filter method provided by the SQLAlchemy<sup>1</sup> Python library in the Flask application to obtain the list of English phrases for a given user query. The complete architecture of the HAZEL system is illustrated in Figure 1.1.

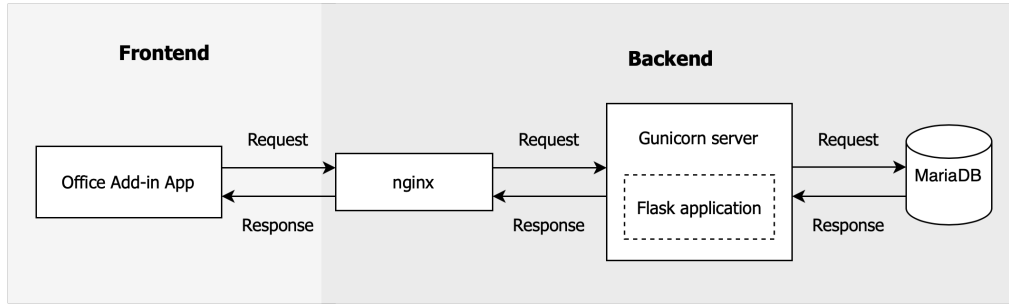


Fig. 1.1 HAZEL system architecture.

The rudimentary nature of this search procedure means only those phrases containing the user's complete, ordered input substring without grammatical errors will be returned. The authoring experts at Yordas spend a significant amount of time searching for phrases to insert into an SDS. Hence this can become particularly frustrating and time consuming for them as the search mechanism cannot tolerate minor grammatical mistakes when searching for phrases containing words which are difficult to spell, or yield phrases containing multiple words in the user's search query which may not appear consecutively in a phrase. We provide example user inputs with intended phrases in Table 1.1.

<sup>1</sup>SQLAlchemy: <https://www.sqlalchemy.org>

Table 1.1 Example user queries with intended phrases.

Query	Intended Phrase	Returned?	Reason
acute toxicity	acute toxicity	✓	Exact match
acute	acute toxicity	✓	Phrase contains the query string
toxicity	low acute toxicity	✓	Phrase contains the query string
acute toxicity	low acute toxicity	✓	Phrase contains the query string
acte	acute	✗	Spelling error
low acte toxicity	low acute toxicity	✗	Spelling error in a query word
toxicity low	low acute toxicity	✗	Words out of order in query string
low toxicity	low acute toxicity	✗	Query not consecutive in phrase
toxic acute	acute toxicity	✗	User query not in intended phrase

Therefore, Yordas envisage an intelligent predictive search model utilising both existing and contemporary data science techniques would enable users of HAZEL to more easily find the phrases they require. Accounting for minor spelling mistakes and multiple words in the user's search query (either in order or out of order) would provide an improvement on the current search procedure. In addition to ensuring intended phrases are present in the search results, we also seek to maximise the rank of an intended phrase in the results for a given query. For example, searching for 'burns' should ensure the 'burns' phrase appears at the top of the results, with other phrases containing the word 'burns' to be ranked higher than those phrases not containing it. Furthermore, we aim to explore how different ranking algorithms alter the rank of phrases containing a subset of words from a given user query. The inclusion of smarter search results based on a smarter search algorithm and logged user behaviours would elevate HAZEL's predictive capabilities, with the potential for both relevant and frequent phrases to be presented to users to assist their search experiences.

## 1.3 General Project Aim

The primary aim of this project is to explore an array of strategies available to aid in the development of a smart predictive search model prototype Yordas could utilise in the production version of HAZEL. Rather than creating a production-ready search model, a series of prototype systems will be implemented and provided to the Yordas authoring experts in HZC as a proof of concept, who will provide feedback to iteratively improve the system, resulting in a tailored and tested prototype upon completion of the project. The development of a bespoke, in-house solution to this problem was advised over utilising existing systems built to facilitate in the creation of search engines, notably Elasticsearch <sup>2</sup> and Apache Solr <sup>3</sup> (both built on the Apache Lucene engine <sup>4</sup>). Although

<sup>2</sup>Elasticsearch: <https://www.elastic.co>

<sup>3</sup>Apache Solr: <https://lucene.apache.org/solr/features.html>

<sup>4</sup>Apache Lucene engine: <https://lucene.apache.org>

these third-party software libraries provide a plethora of pre-built features to create search systems, they generally require a steep learning curve and would be costly to maintain and scale when productionising HAZEL to external clients. This motivates the need for a system which integrates into HAZEL's codebase without the requirement for any major external dependencies.

Through the use of automation, a smart search algorithm would ultimately reduce the amount of time spent completing an SDS, thus enhancing business efficiency. Yordas aspire to be the industry leader in SDS authoring and upon commercialisation of HAZEL to external clients, it is imperative that searching for phrases is as effortless as possible. In this paper we will explore a range of potential techniques Yordas could adopt to realise this aim, which would also enable them to gain a competitive advantage over competing SDS authoring companies.

## 1.4 Aims and Objectives

### 1.4.1 Research appropriate techniques used to implement smart search models

We place emphasis on the creation of a comprehensive overview of viable search model solutions utilising relevant data science methods which Yordas can use and implement into the production version of HAZEL. This will involve researching and exploring common data science methods employed both in research and in practice. All relevant findings will be documented in this report. This will provide Yordas with a comprehensive overview of different strategies they could implement into the production version of HAZEL and allow them to make an effective business decision of which method to pursue before productionising HAZEL.

During an initial discussion with Yordas it was proposed a predictive search model which could continuously learn user behaviours (i.e. phrase searches and selections) in real-time would further enhance predictive performance and dynamically adapt to their preferences with minimal human intervention. The identification of appropriate methods to solve this problem, notably online machine learning, is an area we aim to explore in this paper. The online aspect is paramount here - an offline model would not be able to incorporate new user behaviours and changes made to phrases unless periodic re-training is considered, as existing phrases can be edited or removed and new phrases can be added. Creating a script to re-train the search model offline on a daily or weekly basis would solve this problem, however users ideally want access to any immediate changes.

### 1.4.2 Implement a series of prototype models

Following research, a series of prototype predictive search models will be implemented. Given the short time frame within which the project will be undertaken, each prototype will be deployed into the development version of HAZEL, which will be iteratively improved following feedback from the SDS authoring experts. This will be done not only

to experiment with a range of techniques found from the research, but to also solicit constructive feedback and obtain feature recommendations to facilitate the development of a more bespoke system. Each of the new smart search prototype systems will be accessible in addition to the legacy search procedure as a fallback - in the cases where the smart search does not yield expected phrases for a given query, the authoring experts can easily toggle back to the legacy search.

A key consideration of this project is to ensure a high quality user experience (UX) when using the smart predictive model. Hence once a prototype model has been implemented and deployed into HAZEL for user testing, a qualitative evaluation will be undertaken in addition to a quantitative evaluation. However, emphasis is placed on an evaluation of quality of the search results as it is often difficult to quantify how well the predictive model is performing relative to the user's expectations. A high quality UX can be achieved by ensuring the following:

1. **Relevant search results for a given query:** This is naturally a very difficult problem to solve as determining relevance of a phrase given a query depends on the specific use case for employing the search procedure and requires a careful selection of appropriate algorithms to compute relevance scores. Ultimately, we aim to present an ordered list of the top- $k$  phrases by relevance for a given query. However relevance is inherently subjective in nature - users may not necessarily agree on whether a phrase is relevant to a given query. Hence we seek to quantify to what extent users agree on the relevance search results for a given query. In addition, a quantitative evaluation will be conducted to analyse how often relevant phrases are selected towards the top of the search results. A brief qualitative evaluation here will involve the SDS authoring experts giving verbal and textual feedback to provide an alternative point of view when evaluating how relevant phrases are for given queries. Another aspect to consider here is dealing with ambiguous prefixes. For example, typing 'ac' could yield potentially thousands of results as entire phrase strings would be considered in a smarter model. Hence the choice of search ranking algorithm will influence the order of the returned results, with those more relevant being placed towards the top.
2. **Spelling correction:** It is widely known many users enter queries into search engines containing misspellings in one or more of the search terms. The current system only supports substring matching within words in a phrase, and the spelling and order of the characters spelled in the user's query must be spelled correctly. Given the inherent spelling complexity of the phrases under consideration in this project, we seek to research and implement methods to tolerate minor grammatical mistakes.
3. **Identification of suitable factors and choice of algorithms for use in the new search engine:** Unlike in marketplace search where you have access to additional user data such as their geographical location, factors like this may not be suitable or available. A machine-learned ranking model could incorporate many of these factors with varying weights. This would enable the system to continuously

adapt to a set of users to further improve predictive performance in accordance to the user's search behaviour. Hence a challenge here is to identify suitable factors to utilise in the prototype model based on the available data. It is expected search results will take slightly longer to be generated in comparison to the regular search given the larger amount of computation required to generate them. Hence the choice of data structures and algorithms necessary to generate these suggestions should be chosen with computational speed and efficiency in mind as they will ultimately dictate how quickly search results can be generated on the server. The ideal system should provide search results nearly instantaneously as the user types into the search input.

4. **Easy to access and use the new search model:** Ensuring users can easily toggle between the regular and prototype search procedures with minimal effort will prevent any bias towards a particular search procedure. To ensure the search model is easy to use, we aim to utilise existing UI components in the front-end application as users are familiar with them.
5. **Development of testing suite for prototype model testing.** In addition to the development of the series of prototype models, we seek to develop a testing suite to enable unit tests to be performed. This will ensure the prototype model works as expected and will help identify any potential issues to be resolved during development of each prototype. This will involve developing code using Python, similar to that already developed for the HAZEL system, to rapidly and efficiently test various aspects of the prototypes.
6. **Ensure privacy and security-related matters relating to user phrase searches and selections are considered in each prototype.** An important consideration here is once HAZEL is commercialised, it is of paramount importance the company's phrases are stored and managed securely. Each company will have access to a subset of different phrases. Therefore it is imperative any further data structures implemented (such as memory caches) do not violate the integrity and access rights for the stored phrases.

## 1.5 Report Outline

The remainder of the report is structured as follows. In Chapter 2 we conduct thorough research into the range of search techniques employed in practice, which will include descriptions of the mathematical foundations underpinning relevant algorithms. Chapter 3 outlines the available data to use in this project which will aid in the selection of suitable search algorithms and choice of factors to potentially include in a machine-learned ranking model. Chapter 4 outlines the series of prototype search models developed in accordance with the techniques discussed in Chapter 2. An overview of the issues encountered with each prototype will motivate the choice of techniques employed in a further iteration. Chapter 5 presents an overview and interpretation of both the

---

quantitative and qualitative evaluation results for the refined prototype search model, as well as a general discussion regarding future directions. Chapter 6 concludes with a discussion of the aims and objectives outlined in this chapter, which includes an assessment of which were successfully accomplished, as well as presenting overall conclusions, lessons learned during the project and pointers for further work.



# Chapter 2

## Background

This chapter begins by introducing the area of information retrieval (IR) which concerns the filtering of specific information from a collection of data. Modern retrieval systems such as online web search engines can adopt a variety of different retrieval models depending on the specific application. We explore the mathematical principles underlying common types of retrieval models including the boolean, vector space and probabilistic models, as well as highlighting how each model quantifies relevance in relation to the query submitted by the user. The relative advantages and disadvantages of these models will be discussed, including an assessment of the practicability each model has to enhance HAZEL's retrieval capabilities. In addition we explore how spelling correction and proximity of terms in a document can assist the user in retrieving more relevant documents as well as improving the user experience. We conclude this chapter by exploring the application of contemporary machine learning approaches to IR systems, and how learning to rank models can improve ranking of the search results.

### 2.1 Information Retrieval

In an academic context, information retrieval can be defined as finding material (usually documents) of an unstructured nature (usually text) that satisfies an information need from within large collections (usually stored on computers) [20]. It is the area which underpins search engines, which are designed to retrieve information from a large collection of documents. A user's information need can be defined as their desire to find some information from a system to satisfy a particular need. This will heavily influence the choice of search engine design, as information needs can vary significantly, for example "how to cook pasta" and "buses from Lancaster to Manchester" represent different queries which should yield recipes to cook pasta and a bus timetable for Lancaster to Manchester for the respective queries.

Many different types of document collections exist, such as online chat conversations, books, articles and a user's personal file store. This type of data is inherently unstructured, which means there is a missing formal, semantically overt and a structure easy for a computer to understand [7]. Unstructured data has an internal structure but is not

structured via a predefined schema or data models. This is in contrast to structured data, in which we observe data which has a rigid and predefined structure (usually managed in relational databases) and is searchable using human generated queries using Structured Query Language (SQL) using field names, for example maintaining product inventories and airline reservation systems.

The birth of the World Wide Web in 1989 sparked a dramatic change in the way people can access and search through large document collections online [3]. At web scale we encounter very large collections of web pages, so the choice of data structures required to build a scalable solution for large datasets must account for this. For a small dataset, one could simply use regular expressions to extract required information, although this becomes very slow when dealing with larger document collections. In online web search the document collection consists of a list of web pages to search against and the IR system represents the search engine which yields a set of relevant documents in relation to the user's search query. The IR system must be capable of interpreting natural language queries to understand the user's information need to obtain a set of results relevant to the query. The architecture for a general IR system is depicted in Figure 2.1.

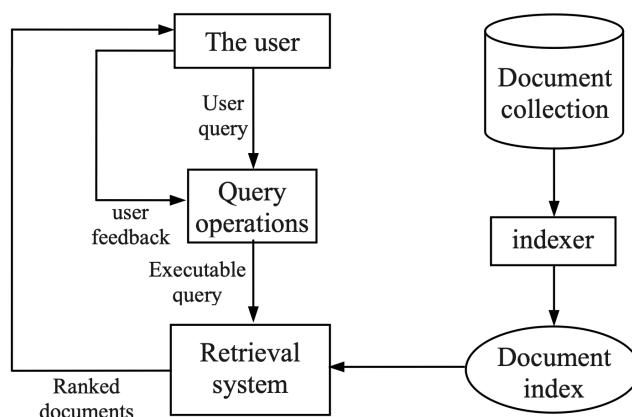


Fig. 2.1 General information retrieval system architecture [16].

In general, a user issues a query to a query operations module which performs query preprocessing and converting the query into a numerical representation. An indexer is responsible for indexing all documents in a collection to create a document index, which will facilitate efficient document retrieval. The inverted index is a well known, efficient indexing model which will be discussed further in section 2.3. Given the user's preprocessed and transformed query, the retrieval system can then compute relevance scores for each document in the index. The documents can then easily be ranked by a numerical relevance score to produce a set of ranked documents. To enhance efficiency further, a small set of documents containing at least one query term are retrieved from the document index and relevance scores are computed on this subset given a user query, as computing scores for all documents can be inefficient.

A fundamental concept in information retrieval is the notion of relevance. A document is deemed relevant if it is one that the user perceives as containing information of value

with respect to their personal information need, and not because it happens to contain all query words [20]. For example, a user searching for "python" in a web search engine may want information relating to the Python programming language and not the animal. Hence it can be difficult for such systems to decipher the user's information need with short or ambiguous queries.

The choice of an appropriate IR system architecture is paramount to realise the effectiveness of the system and ensure it works efficiently in practice. In typical web search the system must be capable of searching through billions of documents distributed across millions of computers. In this paper however we focus on enterprise search, in which a corporation's database of internal documents is stored on centralized file systems to provide search over the collection [20].

## 2.2 Retrieval Models

Here we present a formalized mathematical overview of the key retrieval models used in information retrieval, notably the boolean, vector space and probabilistic models. The way in which these models define relevance and represent both the user's query and the document collection will be explored.

### 2.2.1 The Boolean Model

The Boolean model of information retrieval (BIR) represents the document collection and the user's query as a bag-of-words/terms<sup>1</sup>, in which the order and positions of terms are ignored. Given a set of distinct vocabulary terms  $V = \{t_1, t_2, \dots, t_{|V|}\}$  from a document collection  $D$ , the user's query  $q$  and each document  $d_j \in D$  can be represented as weighted vectors:

$$q = (w_1, w_2, \dots, w_{|V|}), \quad (2.1)$$

$$d_j = (w_{1j}, w_{2j}, \dots, w_{|V|j}), \quad (2.2)$$

where each weight  $w_{ij}$  quantifies the importance of term  $t_i \in V$  in document  $d_j$ , and  $|V|$  represents the number of distinct vocabulary terms in the collection. A document collection can thus be represented as a matrix of these vectors. This representation is known as a term-document incidence matrix.

Figure 2.2 illustrates a binary term-document incidence matrix. If a term is present in a document the weight value is 1, otherwise we assign a value of 0. Given a user query containing common Boolean operators such as AND, OR and NOT as a Boolean expression, we can perform bitwise operations for each query term in binary format to

---

<sup>1</sup>A term denotes a distinct value in the vocabulary, and a document denotes a distinct unit containing a subset of these terms. The words term and word will be used synonymously, although a term could consist of more than one word.

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth	...
Antony	1	1	0	0	0	1	
Brutus	1	1	0	1	0	0	
Caesar	1	1	0	1	1	1	
Calpurnia	0	1	0	0	0	0	
Cleopatra	1	0	0	0	0	0	
mercy	1	0	1	1	1	1	
worser	1	0	1	1	1	0	
...							

Fig. 2.2 An example term-document incidence matrix for words used by Shakespeare in some of his plays. A value of 1 indicates the presence of a term with 0 indicating the absence of a term [20].

produce a binary result. A value of 1 in the result vector indicates a given document is retrieved.

Given the inherent precise structure of Boolean queries, many professionals such as those who deal with legal materials prefer Boolean models over alternative models. The Boolean model enables greater control and transparency over what is retrieved as the user's query either matches a document or does not [20]. However, this model poses many disadvantages. Notable shortcomings are the lack of support for more complex query operations such as proximity search (i.e. retrieving documents in which terms are close to each other) and partial term matching for autocomplete search suggestions, requiring exact query term matching against the document collection, lack of spelling error tolerance and lack of ordering of the retrieved documents.

A further consideration here is the term-document matrix will be very sparse given that not every document will contain every vocabulary word. For a large vocabulary and a short query, many of the weights in the matrix will be zero. For large document collections with each document containing several terms, the matrix consists of 99% of cells containing 0 values, and thus is incapable of being stored in memory. A solution to this problem is only storing the non-zero values and their positions. This idea is core to a common index structure known as the inverted index which will be discussed in section 2.3.

### 2.2.2 The Vector Space Model

The vector space model is a widely used model in information retrieval. Here we consider a document as a numerical vector of weights, in which each weight component represents a value according to the TF or TF-IDF schemes [16]. The TF (term frequency) scheme simply considers the number of occurrences of a term in a document. A shortcoming of

this approach is all other documents containing this term and their frequencies are not considered.

A well-known extension to this scheme is the TF-IDF model (term frequency-inverse document frequency), which represents documents in a common vector space. The motivation for TF-IDF is to scale down terms that appear frequently in a corpus as they tend to be less informative than those terms which appear less frequently. The raw term frequency count is normalised to prevent bias towards short or long documents and is given by:

$$tf_{ij} = \frac{f_{ij}}{\max\{f_{1j}, f_{2j}, \dots, f_{|V|j}\}}, \quad (2.3)$$

where  $f_{ij}$  represents the raw frequency count of term  $t_i$  in document  $d_j$ . The denominator normalises the raw frequency count by taking the maximum frequency count of all terms in document  $d_j$ . The inverse document frequency is given by:

$$idf_i = \log \frac{N}{df_i}, \quad (2.4)$$

where  $df_i$  represents the number of documents in which term  $t_i$  occurs (document frequency). This factor diminishes the weight of terms that occur frequently in a given document and increases the weight of terms that occur rarely across documents. In essence, a term which appears more often across documents is unlikely to be important and is not discriminative. Note here that if the user's query contains a single query term, the IDF component of TF-IDF has no effect.

The TF-IDF values can now be computed by multiplying these components together to give a term weight:

$$w_{ij} = tf_{ij} \times idf_i \quad (2.5)$$

Hence each document can now be represented as a vector of term weights in a common vector space. For a given user query  $q$ , we can apply the TF-IDF scheme to each term  $t_i$  in  $q$  to obtain a weight  $w_{iq}$  for each term  $t_i$ .

In the vector space model we rank documents according to their degrees of relevance to the user's query. This is achieved by computing a similarity score between the user's query  $q$  and each document  $d_j$  in the document collection  $D$  in vector representations. Many similarity methods exist, however a common approach is to utilise the cosine similarity function which is defined as follows:

$$\text{cosine}(d_j, q) = \frac{\langle d_j, q \rangle}{\|d_j\| \times \|q\|} = \frac{\sum_{i=1}^{|V|} w_{ij} \times w_{iq}}{\sqrt{\sum_{i=1}^{|V|} w_{ij}^2} \times \sqrt{\sum_{i=1}^{|V|} w_{iq}^2}} \quad (2.6)$$

The denominator of equation 2.6 length-normalises the query and document vectors to unit vectors. If the TF-IDF vectors are normalised, the cosine similarity is equal to

the dot product (i.e. the numerator) and we can ignore the denominator to prevent redundant computations.

The cosine similarity compensates for the effect of documents having different lengths, as the orientation of vectors is considered as opposed to their magnitude. The cosine similarity between the document  $d_j$  and the query  $q$  equals 1 when the two vectors have the same orientation, 0 when they are orthogonal and -1 when they have the opposite orientation. If the TF-IDF vectors are length-normalised by dividing each component by their lengths (e.g. using the  $L_2$  norm), the vectors are mapped onto a unit sphere. This results in both short and long documents having weights of the same order of magnitude<sup>2</sup>. This is illustrated in Figure 2.3.

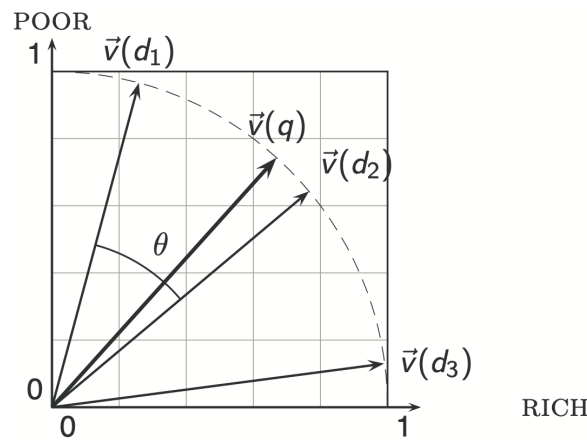


Fig. 2.3 Cosine similarity illustration [7].

Computing the cosine scores for all documents enables a list of documents to be ranked by decreasing cosine score in relation to the user's query, resulting in an ordered set of results in which the top results are considered more relevant.

The vector space model lacks the control of a Boolean model. In the latter model, terms are required to be present in a given document. This behaviour may be beneficial over a ranked set of results in which documents containing more frequent terms are recommended.

### 2.2.3 The Probabilistic Model

The probabilistic model has gathered more attention and achieved success in recent times. A famous ranking model based on this framework is BM25 (or Okapi BM25) which is known to perform well across a range of search tasks [12]. Similarly to TF-IDF, documents and queries are represented in a common vector space. Using the TF-IDF model, documents containing a term more frequently and rarely relative to all other

<sup>2</sup>Libraries such as scikit-learn [22] support length-normalised TF-IDF vector computations, so a simple dot product of the normalised query and document vectors is sufficient as opposed to a full cosine score calculation as in Equation 2.6. This will save unnecessary computations.

documents will produce a high TF-IDF weight and are thus considered relevant. Hence it rewards term frequency and penalizes document frequency. BM25 extends this model to account for document length and term frequency saturation. The BM25 score for a given query  $q$  containing terms  $t_1, \dots, t_M$  is given by:

$$\text{BM25}(d, q) = \sum_{i=1}^M \frac{\text{IDF}(t_i) \cdot \text{TF}(t_i, d) \cdot (k_1 + 1)}{\text{TF}(t_i, d) + k_1 \cdot \left(1 - b + b \cdot \frac{\text{LEN}(d)}{\text{avdl}}\right)}, \quad (2.7)$$

where  $\text{TF}(t_i, d)$  is the term frequency of  $t_i$  in document  $d$ ,  $\text{IDF}(t_i)$  is the IDF weight of term  $t_i$  (see equation 2.4),  $\text{LEN}(d)$  is the length of document  $d$ ,  $\text{avdl}$  is the average document length, and  $k_1$  and  $b_1$  are hyperparameters [17].  $k_1$  calibrates the term frequency for a given document, where 0 represents no term frequency will be used and 1 meaning raw term frequency will be used.  $b$  is a tuning parameter which scales the document length, with a value of 1 meaning the term weight is fully scaled by the document length with a value of 0 indicating no length normalisation [20].

Once a document is saturated with occurrences of a term, more occurrences should not have a significant impact on the score. A further benefit of term frequency saturation is complete matches are given a higher score compared to partial matches. This means documents which match more of the terms in a query are given a higher score compared to documents containing many matches for just one of the query terms. If a very long document mentions a query term only once, it is less likely to be relevant to the query compared to a short document mentioning the term once [5].

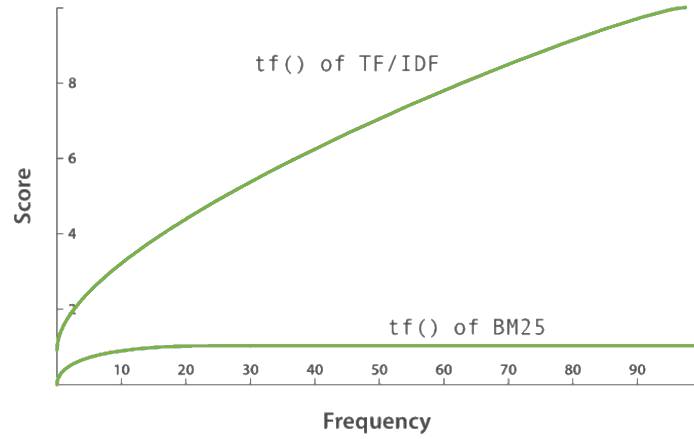


Fig. 2.4 Term frequency behaviour illustration [5].

The default parameter settings for  $k$  and  $b$  are 1.2 and 0.75 respectively, however these can be tuned using machine learning. Research has been conducted to integrate machine learning approaches, notably LambdaRank neural networks, to incorporate BM25 attributes to avoid explicit parameter tuning in a data driven approach. In order to train models like this, a large training dataset is generally required to ensure an effective and robust model is constructed which avoids overfitting. The training dataset

used in this research consisted of roughly 70,000 queries where each query was associated with on average 150-200 URLs along with feature vectors for term frequencies, document frequencies and field lengths. Each query-URL pair was assigned a human generated relevance label on a 5-level relevance scale from 0-4 in which 4 indicated the most relevant document [30].

## 2.3 The Inverted Index

The inverted index is a word-oriented data structure which facilitates fast lookups for search tasks. For each unique word in the vocabulary obtained from a list of documents, the list of documents which contain that word are stored. This overcomes the sparsity issue observed in the term-document matrix, as only the presence of terms in documents are considered.

Vocabulary	$n_i$	Occurrences as inverted lists
to	2	[1,4],[2,2]
do	3	[1,2],[3,3],[4,3]
is	1	[1,2]
be	4	[1,2],[2,2],[3,2],[4,2]
or	1	[2,1]
not	1	[2,1]
I	2	[2,2],[3,2]
am	2	[2,2],[3,1]
what	1	[2,1]
think	1	[3,1]
therefore	1	[3,1]
da	1	[4,3]
let	1	[4,2]
it	1	[4,2]

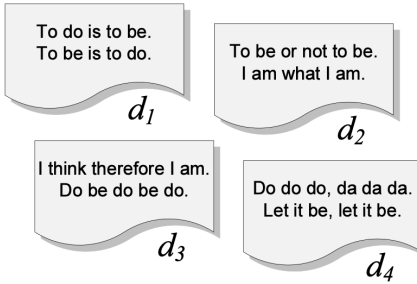


Fig. 2.5 Basic inverted index structure [1].

Once constructed, the inverted index can provide random access lookups for query words. This model can be extended to include positional information and frequencies of terms in each document.

## 2.4 Preprocessing

Before indexing, text must be preprocessed [19]. In general, the following steps are used:

1. **Tokenization:** separate the text into distinct tokens. Tokens are instances of a sequence of characters in a given document that are grouped together as a useful semantic unit [20]. One procedure is to split text on whitespace characters, however joint words separated by a punctuation mark (e.g. '-' or '/') would then be treated as a single token. Hence the tokenisation procedure employed should be chosen with care.



2. **Removal of punctuation:** following tokenisation, punctuation characters can be removed to isolate the distinct words.
3. **Removal of stopwords:** words which are common and generally uninformative (e.g. and/it/the) can be removed.
4. **Case folding:** converting all characters in a document to the same case (i.e. lowercase).
5. **Stemming:** reduce morphological variations of words to a common stem.

This procedure should be applied to both the documents and the queries.

## 2.5 Spelling Correction

It is estimated approximately 10-12% of all query terms entered in Web search engines are misspelled [8]. Information retrieval systems should therefore be able to tolerate minor grammatical mistakes in queries. Many techniques have been proposed to correct spelling errors in text [14]. Interactive spell checking approaches exist, in which the user makes a final decision on the spelling of words in their query. Another approach is automatic correction, which aims to automatically detect and correct spelling errors, however is notoriously more difficult to achieve in practice [34].

An important distinction exists between error detection and error correction. Efficient algorithms and data structures have been devised to tackle error detection, although error correction is considerably more difficult to achieve. Many existing spelling correction procedures rely on correcting each term of a multi-term query independently, without incorporating the context of the string within which each query term appears. This is known as isolated-term correction. Such techniques fail to detect typographic and grammatical errors that result in alternative valid words. Thus more complex approaches have been devised to incorporate contextual information to address common spelling errors as well as correcting words with more than one possible correction. This is known as context-sensitive correction [20].

The edit distance (also known as the Levenshtein distance) between two strings  $s_1$  and  $s_2$  is the minimum number of required edit operations to transform  $s_1$  into  $s_2$ . It is a commonly employed distance metric in spelling correction algorithms. Insertion of a character, deletion of a character and replacement of a character with another character are the commonly used operations. This algorithm has complexity  $O(|s_1|, |s_2|)$ , where  $|s_i|$  denotes the length of the string  $s_i$ . Further modifications can be made to this algorithm which can allow for weighted edit operations, such as assigning a higher weight for replacing a certain character over another. An extension to this metric is the Damerau-Levenshtein distance, which also accounts for transposition errors. This is depicted in Figure 2.6. Using the traditional edit distance, transposition would require 2 distinct changes. A further extension is the Restricted Damerau-Levenshtein distance, which ensures each substring may only be edited once.

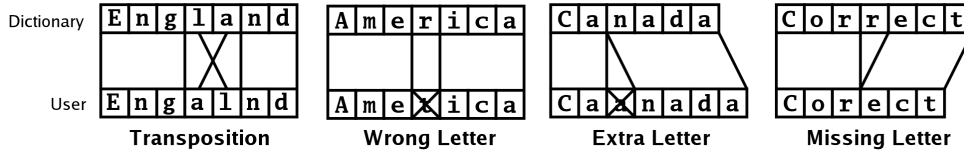


Fig. 2.6 Common spelling error types considered in the Damerau-Levenshtein distance metric [21].

The SymSpell algorithm, based on the Symmetric Delete spelling correction, utilises the Restricted Damerau-Levenshtein distance metric and is known to be 6 orders of magnitude faster than the standard Damerau-Levenshtein distance. The algorithm obtains all strings within a pre-specified maximum edit distance using only delete operations and prefix indexing [9]. However higher memory consumption and pre-calculation times are now observed.

## 2.6 Term Proximity

Many information retrieval models assume terms occur independently of other terms in a document. Traditional bag-of-words models do not account for the positional information of terms. Term proximity is a form of term dependence and uses the distance between terms in a document as a proximity measure. If terms appear within a certain number of tokens they are said to be in a lexical relation to each other [23]. This helps to promote documents containing matched query terms which are close to each other.

One can construct a positional index which is similar to the inverted index which stores positional information about each term alongside its frequency [20]. Research has shown incorporating term proximity calculations can significantly improve retrieval performance for models such as the BM25 model [31]. It has also been shown that utilising high-order  $n$ -gram language models using bigrams and trigrams can indirectly capture proximity [28]. Further research has been conducted to explore the effects of integrating term proximity into BM25 using pseudo term frequencies instead of raw term frequencies, which account for the distance between two query term occurrences as well as their order and the term weights [29]. Term proximity information has also been integrated into the BM25 model to only the top documents returned, and was shown to improve retrieval effectiveness particularly for the top retrieved documents [24].

It was found extending the TF-IDF model with term proximity information can increase effectiveness of retrieving relevant documents. The combination of span-based and pair-based methods can produce a term proximity score for a given query-document pair and added to the cosine score as outlined in Equation 2.6 [13].

## 2.7 Learning to Rank

Learning to rank in information retrieval aims to employ machine learning techniques to automatically construct the ranking model  $f(q, d)$  where  $q$  and  $d$  denotes a query and a

document respectively [15]. A score can be assigned to a given query-document pair, with both represented as a numerical vector of features, from a locally trained ranking model. Incorporating various factors such as search log data can help to represent relevance of a document and hence machine learning can help to construct the ranking model with a range of features. Learning to rank is inherently a supervised learning problem, that is, training and testing phrases are required.

Training data consists of a set of queries, with each mapped to a set of documents with an assigned relevance score, where the higher the score the more relevant the document is to the query.

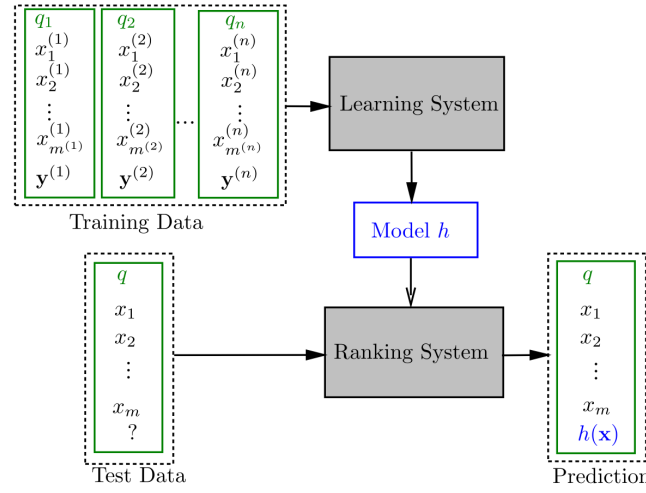


Fig. 2.7 Learning to rank system [17].

In general, learning to rank approaches can fall into one of three categories [17]:

1. **Pointwise approach:** uses regression, classification or ordinal regression on each document when ranking.
2. **Pairwise approach:** formulates the ranking problem using pairwise classification.
3. **Listwise approach:** attempts to optimise information retrieval evaluation measures or to minimise listwise ranking losses.

Some of the common algorithms used for each approach is illustrated in Figure 2.8.

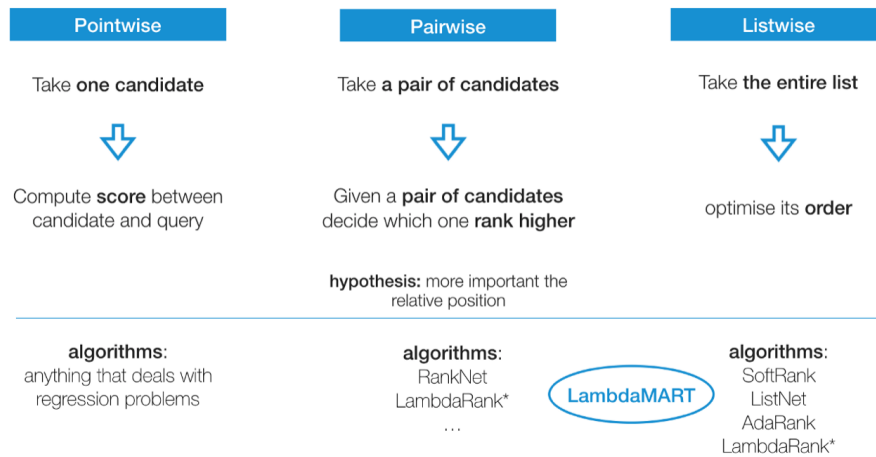


Fig. 2.8 Learning to rank approaches [32].

We elide specific mathematical details regarding these models due to their complexity. Extensive research has been undertaken into analysing the effectiveness of each approach, with more studies focusing on the listwise approach [17].

The main shortcoming of offline learning to rank is having to create significantly large labelled datasets of explicit relevance judgements. This is an expensive and time consuming process. Online learning to rank is an alternative solution and in many cases is preferable, as the system can learn online in real time from user interactions and can continuously improve ranking scores using this implicit feedback. Online learning to rank remains an active area of research. However using interaction data poses several challenges. Interactions such as clicks of results are strongly influenced by how they are presented (i.e. their rank) which introduces bias. Hence clicks cannot be represented as simple relevance judgements [10].

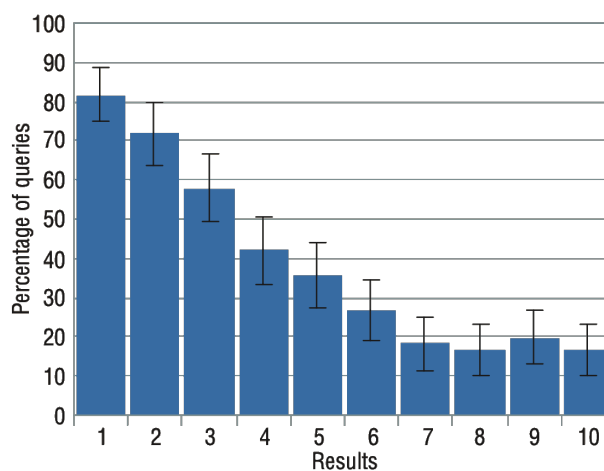


Fig. 2.9 Percentage of queries where a user viewed the search result presented at a particular rank [11].

A study was undertaken using an eye-tracking device in a controlled user study to analyse which search results were being read by users (see Figure 2.9). In general, only the top few results are viewed with a general decreasing trend for search results at lower ranks.

## 2.8 Chapter Summary

In this chapter we explored a range of retrieval models including an in-depth overview of the mathematical underpinnings of commonly used ranking algorithms. We covered preprocessing methods, spelling correction techniques, term proximity information and a brief overview of learning to rank methods using machine learning. In the next chapter we seek to conduct an analysis of the dataset at hand which will aid in the selection of suitable methods studied in this chapter to experiment with in Chapter 4.

# Chapter 3

## The Dataset

In this chapter we will describe the available data in terms of its representation, as well as conducting an exploratory data analysis to summarise the data in the form of a concise set of descriptive and quantifiable statistics. An overview of the data limitations found will also be presented. This will provide a foundation for a suitable selection of methods to be explored in the prototype model implementations and the proposed model architecture outlined in Chapter 4.

### 3.1 Data Description

A production version of the HAZEL system has been deployed and made available to HZC with an additional testing environment to facilitate testing of new features before deployment into production. All available data is maintained in a collection of MariaDB<sup>1</sup> databases for both the production and testing versions of HAZEL which are hosted on Yordas’ servers. For this project we utilise the `hazel_phrases` and `hazel_sds_requests` production databases for retrieving phrase and query log data respectively.

#### 3.1.1 Phrases

Data relating to phrases is maintained in the `hazel_phrases` database. Although this database consists of several relational tables, we focus on the key tables relevant to this project, notably the `phrases` and `languages` tables. An example phrase record is illustrated in Figure 3.1 and the meanings of each field are given below:

- **phrase\_counter**: an increasing integer value representing the total number of phrases added in total to that point (with a starting value of 10000001).
- **phrase\_id**: a unique phrase identifier. Each phrase has a unique identifier regardless of its language.

---

<sup>1</sup>MariaDB is an open source RDBMS (relational database management system): <https://mariadb.org>.

```

1  {
2      "phrase_counter": "10000063",
3      "phrase_id": "YGPID0000367120",
4      "phrase_code": "YGPC00000367120",
5      "phrase": "If the stated operational conditions are used, the
6      ↪ estimated exposures to workers of the mixture component with
7      ↪ greatest risk are shown below:",
8      "language_id": "YGS0000794204",
9      "phrase_score": "2",
10     "phrase_created": "2020-02-04 14:48:34",
11     "phrase_updated": "2020-02-04 14:48:34",
12     "phrase_active": "1"
13 }

```

Fig. 3.1 Example phrase record from the phrases table.

- **phrase\_code**: a unique phrase code. A phrase represented in different languages will share the same phrase code.
- **phrase**: a textual description of the phrase.
- **language\_id**: a unique language identifier which references the languages table in `hazel_phrases`, where each `language_id` represents a unique language.
- **phrase\_score**: a value of 1 indicates the phrase has restricted SDS uses, a value of 2 indicates the phrase is suitable for all SDS uses and a value of 3 indicates the phrase is a Yordas owned packaged phrase.
- **phrase\_created**: the date when the phrase was created.
- **phrase\_updated**: the date when the phrase was updated.
- **phrase\_active**: a value of 1 indicates the phrase is available for use, with a value of 0 indicating the phrase has been deleted.

This project concerns only the active English phrases. This will ultimately comprise the phrase indexes in the prototype models we will develop in Chapter 4. Phrases referenced in this report for data description purposes will involve those with a phrase score of 2 and 3, although developed models will operate on phrases having any of the previously described phrase scores.

The textual description of each phrase is the field which enables phrases to be retrieved for a given user query. In Table 3.1 we demonstrate the complexity and diversity of the

phrases through a series of examples, describing the different types of phrases a user can retrieve.

Table 3.1 Phrase examples.

Phrase Types	Phrase Examples
Single word phrases	Acute Toxicity Ecotoxicity Risk If
Multiple word phrases	Danger of explosion through impact. Oxides of nitrogen Acute Toxicity Low Acute Toxicity Colourless to brown liquid with perceptible odour. Clear liquid with perceptible odour. Dust may have irritant effect on eyes. The substance has irritant effect on eyes.
Phrases containing punctuation	(n-Octanol/water) Common name(s), synonym(s) of the substance [Closed cup] Solidifies 11-20°C. Ethanol (e.g. 1 drinking glass of a 40% alcoholic beverage). Store at temperatures not exceeding <... C >°C/<... F >°F. Use explosion-proof [electrical/ventilating/lighting/<... seg >] equipment. SELF-HEATING SOLID
Unique codes	LD50 P261: Avoid breathing gas. P261: Avoid breathing vapours. P403+P235: Store in a well-ventilated place. Keep cool. P371+P380+P375: In case of major fire and large quantities: Evacuate area. Fight fire remotely due to the risk of explosion.

In general we can organise the types of phrases into 4 categories which are outlined in Table 3.1. Phrases can contain only a single word. Multiple-word phrases can contain an arbitrary number of words. Many phrases consist of a varying amount of punctuation characters. Finally, phrases can contain unique codes either as a single entity or being part of a longer phrase in conjunction with several additional codes.



### 3.1.1.1 Phrase Character Counts

Table 3.2 Phrase character statistics for active English phrases in HAZEL’s production database (as of 11th August 2020).

Count	Mean	Std	Min	25%	50%	75%	Max
22077.00	38.73	43.95	1.00	16.00	29.00	49.00	1932.00

In Table 3.2 we observe a total phrase count of 22,077 phrases with the length of an English phrase varying anywhere between 1 and 1932 characters, with a median length of 29 characters. The mean length is 38.73 characters which is slightly larger than the median, indicating a small proportion of phrases are longer than the majority of phrases. Thus the median is a more appropriate approximation of the average phrase length. The database of English phrases is constantly updated with new phrases being added and existing ones being edited or deleted. Note here phrase character counts exclude whitespace and include punctuation characters. The reason for phrases having a length of 1 are for translation purposes. For example, the single character word ‘I’ translates to the two character word ‘Je’ in French.

### 3.1.1.2 Phrase Word Counts

We now proceed to compute the distribution of word counts amongst all phrases.

Table 3.3 Phrase word counts for active English phrases in HAZEL’s production database (as of 11th August 2020).

Count	Mean	Std	Min	25%	50%	75%	Max
22077.00	6.59	8.01	1.00	2.00	5.00	8.00	348.00

Table 3.3 outlines the number of words in each phrase. The number of words can vary anywhere between 1 and 348, with an average word count of 6.59 and a median of 5 words across the 22,077 phrases. These statistics indicate the word count distribution has a strong positive skew.

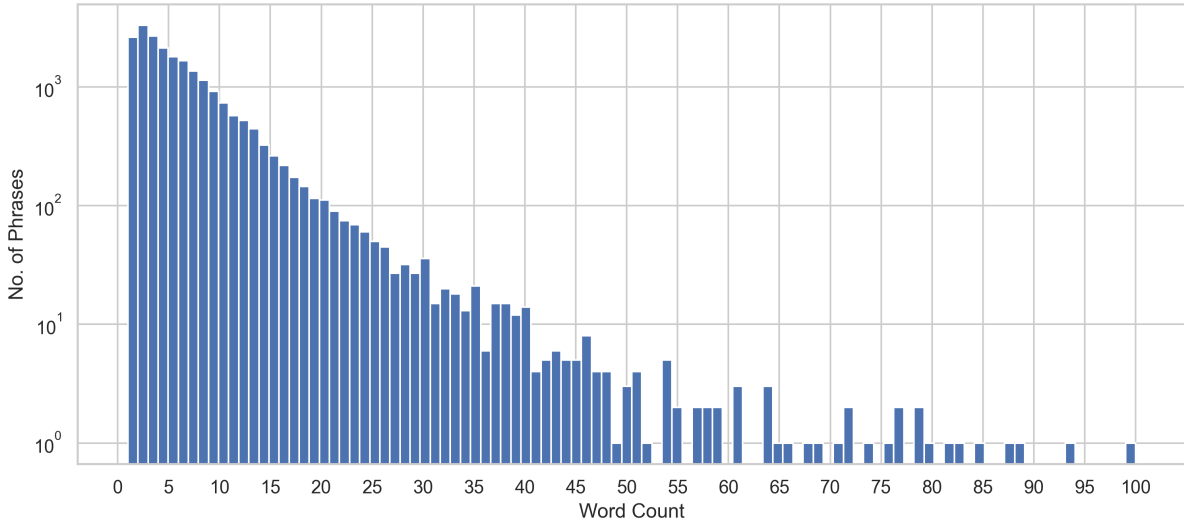


Fig. 3.2 Phrase word counts (top 100 counts).

The distribution of the top 100 word counts is illustrated in Figure 3.2 for the total words counts observed across all phrases. This represents 99.9% of the total distribution of word counts across the list of phrases. We generally notice a high proportion of phrases contain between 1 and 30 words. A large proportion of phrases contain between 1 and 10 words, with an observed modal value of 2 words. It was found 57% of all phrases contain between 1 and 5 words. The most common word count for the phrases was 2, with 15.1% of phrases containing only 2 words, 12.2% of phrases contain 3 words and 11.9% of phrases contain only a single word.

### 3.1.2 Query Logs

Query logs have been accumulated from the authoring experts since the deployment of HAZEL in the `hazel_sds_requests` database. The relational tables of interest in `hazel_sds_requests` are the `phrase_searches` and `phrase_selections` tables.

#### 3.1.2.1 Phrase Searches

When searching for phrases, each keystroke is logged in `phrase_searches` for a specific user along with a timestamp and is represented in Figure 3.3.

The fields in Figure 3.3 have the following meanings:

- **search\_id**: an increasing integer value identifying a particular search record.
- **search\_string**: the search query entered by the user.
- **search\_timestamp**: the time the user submitted the search query.
- **search\_user**: the user who issued the search query.

```

1 {
2   "search_id": "7681",
3   "search_string": "according to ec regula",
4   "search_timestamp": "2020-02-13 14:51:43",
5   "search_user": "user@email.com"
6 }

```

Fig. 3.3 Example phrase search record from the phrase\_searches table.

Each phrase search record represents a single search string entered by the user after a given keystroke. Hence for a user input of length  $N$ ,  $N$  individual phrase search records will be stored with each record representing all characters typed up to character  $N$  in the search string.

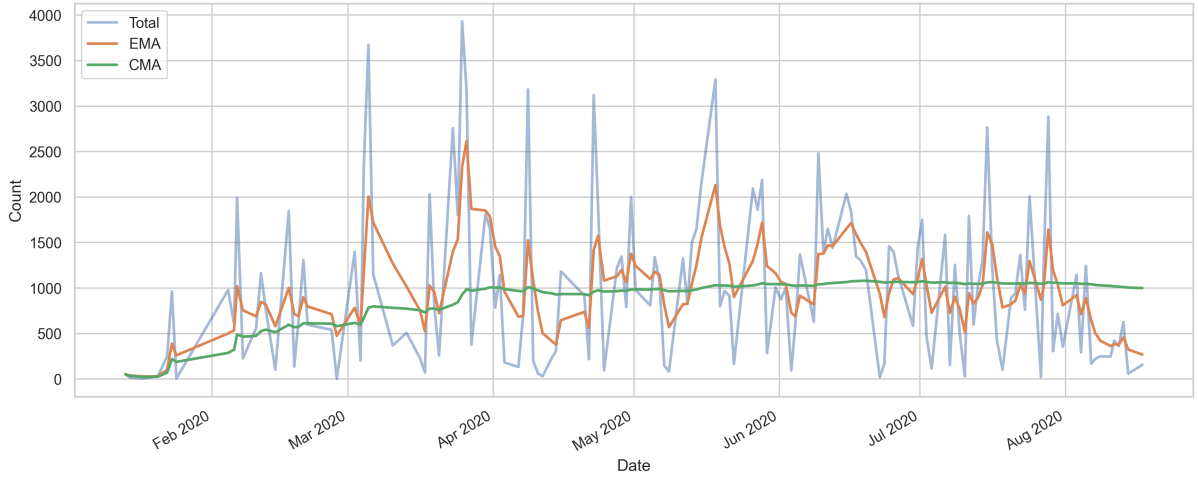


Fig. 3.4 Phrase searches time series.

Figure 3.4 illustrates the behaviour of phrase searches from 13th January 2020 to 17th August 2020. For this timeframe, a total of 143,917 phrase selections have been recorded. The blue line indicates the total number of phrase searches over this timeframe. The total number of phrase searches per day can vary significantly anywhere up to 4000 total searches. Note here an individual search represents a single typed character. The orange line represents a 5-day exponential weighted moving average where the exponential smoothing factor  $\alpha = \frac{1}{3}$ . The observed trend is similar to that of the total number of phrase searches, with the number of searches ranging anywhere up to around 2500 searches for a given day. The green line represents the cumulative moving average and helps to observe the average over time up to time  $t$ . We generally notice here from

```
1 {  
2     "selection_id": "16106",  
3     "selection_phrase_code": "YGPC00010896325",  
4     "selection_phrase_text": "Acute dermal toxicity",  
5     "selection_search_string": "Acute Dermal Toxicity",  
6     "selection_timestamp": "2020-08-11 15:18:06",  
7     "selection_user": "user@email.com"  
8 }
```

Fig. 3.5 Example phrase selection record from the phrase\_selections table.

HAZEL's deployment the average number of searches increased up to a relatively constant value of around 1000 searches per day.

### 3.1.2.2 Phrase Selections

When phrases are selected from the search results, a record representing which phrase was selected along with the search string at the time of selection is stored. An overview of all fields stored is given in Figure 3.5.

The fields in Figure 3.5 have the following meanings:

- **selection\_id**: an increasing integer value identifying a particular phrase selection record.
- **selection\_phrase\_code**: the phrase code of the selected phrase.
- **selection\_phrase\_text**: the textual description associated with the phrase at the time of selection.
- **selection\_search\_string**: the search query entered by the user at the time of phrase selection.
- **selection\_timestamp**: the time the user selected the phrase.
- **selection\_user**: the user who issued the selection.

The top 25 most frequently selected phrases are outlined in Appendix B.1. The top 5 phrases have a high frequency of use compared to the other selected phrases. Many selected phrases consist of only 1-2 common words. A variety of stopwords (e.g. and/not/to) and rarer words (e.g. harmonised/acetate) are present across the selected phrases. This demonstrates the heterogeneity of the words within the phrases among those commonly used.

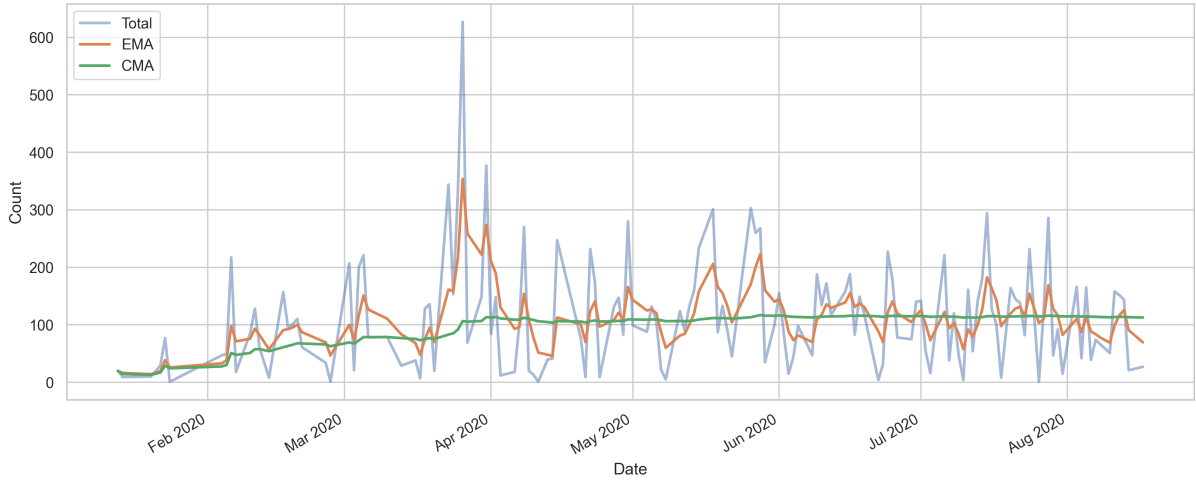


Fig. 3.6 Phrase selections time series.

Figure 3.6 illustrates the behaviour of phrase selections from 13th January 2020 to 17th August 2020. For this timeframe, a total of 16,114 phrase selections have been recorded. The blue line indicates the total number of phrase selections over this timeframe. The total number of phrase selections per day can vary significantly anywhere up to around 380 total selections, although a larger value of around 625 searches was observed, however this was generally an extreme value compared to the rest. The orange line represents a 5-day exponential weighted moving average where the exponential smoothing factor  $\alpha = \frac{1}{3}$ . This aids in filtering out noise and to identify the overall trend. The observed trend is similar to that of the total number of phrase selections, with the number of selections ranging anywhere up to around 350 selections for a given day. The green line represents the cumulative moving average and helps to observe the average over time up to time  $t$ . We generally notice here from HAZEL's deployment the average number of selections increased up to a relatively constant value of around 100 selections per day.

Table 3.4 Search text character counts for phrase selections (as of 11th August 2020).

Count	Mean	Std	Min	25%	50%	75%	Max
3282.00	15.28	20.50	1.00	5.00	8.00	16.00	166.00

Table 3.4 outlines summary statistics regarding the character lengths of search texts for a total of 3282 individual phrase selections. On average we expect the user to type around 15 characters, although in some cases users tend to type a larger amount of characters (i.e. up to 166). The standard deviation value of 20 characters indicates the total number of typed characters before a phrase is selected can vary significantly. In addition to these statistics it was found 9.3% of the phrase selection search strings were equal. Although users tend to search for complete phrases, this is expected behaviour as their intuition of how the regular search works requires them to enter an ordered sequence of characters

with no spelling mistakes. It is anticipated in a future iteration of the search algorithm users will be able to type fewer characters to yield the phrases they require with less effort.

## 3.2 Limitations of the Dataset

The available data was found to have the following limitations:

1. Document locations for selected phrases were not stored. This will prevent the search engine from incorporating a document location factor to predict the most likely phrases to be selected at a given position in an SDS.
2. The ranks of the selected phrases were not stored. This information would become particularly useful when concerning learning to rank models as discussed in Chapter 2. As we seek to evaluate the performance of the prototype search engines, a `selection_phrase_rank` column representing the rank of the selected phrase was integrated into `phrase_selections`. The phrase selected at the  $n$ -th position in the search results will have a rank of  $n$ , with a value of 1 indicating the top search result was selected.
3. A list of relevant and non-relevant phrases at the time of phrase selection were not stored. As discussed in Chapter 2, many learning to rank models involve classifying relevant and non-relevant items from a set of data. Hence with the current set of data, many learning to rank techniques cannot be applied until such data is collected.
4. Query log data not sufficiently large enough to create a machine-learned ranking model. As discussed in Chapter 2, these methods generally do not perform well with a small amount of data. Given we do not have a sufficiently large enough set of query log records, we delegate the integration of machine-learned ranking functions to the proposed implementation and the application of such techniques will be discussed further in section 4.2.

The limitations identified here could provide potential improvements to HAZEL in a further iteration of the system. Generating a sufficiently larger dataset of user behaviours will enable a more diverse range of methods to be explored such as learning to rank models as discussed in section 2.7. Hence in Chapter 4 we prioritise the application of the more foundational techniques required to create search engines as well as the proposed architecture, with an increasing level of complexity in further prototype models in terms of the features they contain.

## 3.3 Chapter Summary

In this chapter we provided an overview of the available dataset. A series of phrase examples was given to demonstrate the diversity of the phrases at hand. The representa-

tion of phrase records and statistics such as phrase character and word counts was also given. We then gave an overview of the available query logs, notably the phrase search and selection data, with an overview of the representation and the behaviour over time. Finally we presented some important data limitations. In the following chapter we seek to explore a range of suitable techniques based on this available data and to facilitate experimentation with methods explored in Chapter 2.

# Chapter 4

## Methodology

Due to the plethora of features found from the research undertaken in Chapter 2 and the available data outlined in Chapter 3, we seek to explore the practicability of many of these features through a series of prototype search engines. In this chapter we follow the agile methodology. Each prototype will be iteratively implemented into HAZEL's testing environment to facilitate rapid experimentation with a variety of different techniques and to solicit feedback from the SDS authoring experts in HZC. We conclude this chapter with an outline of the final proposed search engine architecture, including potential areas to explore further in order to maximise retrieval effectiveness in a further iteration of the system.

### 4.1 Search Engine Prototypes

To provide a fallback for the authoring experts when testing the new prototype search engines, the legacy search engine will be deployed alongside each prototype search engine in the front-end application (see Figure 1.1). Depending on the selected search engine type in the add-in application, the Flask application will allow user queries to be directed towards the relevant search procedure, that is, the regular or prototype search engine. This will allow the authoring experts to have access to a working phrase retrieval procedure should the prototype search not return the intended phrases for a given user query. Therefore we begin by implementing a toggle component in the front-end add-in application to allow users to easily switch between the search procedures (see Appendix A.4). With the current database structure, it would not be possible to differentiate between records for both searched and selected phrases. Hence an additional 2 columns, `search_type` and `selection_search_type`, were added to the `phrase_searches` and `phrase_selections` tables respectively. This will aid in identification of searched and selected phrases according to the search engine type that was used at a specific point in time. In addition to the new `selection_phrase_rank` column discussed in Chapter 3, these new columns will enable an evaluation of search performance between both the new and legacy search systems in Chapter 5.



The heterogeneity of textual representations for the phrases discussed in Chapter 3 poses an interesting dilemma. Many phrases contain a variety of punctuation characters, unique prefix codes and complex words in terms of their spelling. An annoyance with the regular search meant words entered for a given query in a different order to how they appear in a phrase means they will not be retrieved. Users would need to remember exact prefix codes such as P261 in order to retrieve these phrases. In the situations where many phrases exist with the same prefix code, a user must either scroll through the search results to find their required phrase or enter further characters to yield their required phrases. However this requires knowledge of the exact characters which follow the prefix code. It may be the case users know additional information which would further narrow down the set of possible phrases. Hence the prototypes seek to implement techniques to tolerate minor grammatical mistakes, thus yielding similar phrases and alleviating the requirement for users to type words exactly.

### 4.1.1 Prototype 1: Fuzzy Search Engine

We begin prototype development by exploring the concept of approximate/fuzzy string matching. Due to the large diversity of both standard and chemical-related phrases in terms of spelling variation and the number of characters they contain, it can become difficult for users to spell words within these phrases correctly across the range of searchable phrases. Considering the current system, if a user searches for a phrase containing several complex words in terms of their spelling, the user must ensure each word is spelled correctly for their intended phrases to be present in the search results. Therefore ensuring the new search engine can be tolerant to minor spelling mistakes would alleviate this issue, thus yielding phrases similar to the user input and consequently relaxing the constraint for users to spell words correctly and exactly.

As discussed in Chapter 3, the median number of characters in a given phrase was 29. The median number of words in a given phrase was found to be 5. The distribution in Figure 3.2 illustrated the majority of phrases consisted of anywhere between 1 and 10 words with a general decreasing trend in the total number of words across the phrases. This suggests if a user was to type in only a few words they would be able to retrieve their intended phrases subject to no spelling errors in the query. Fuzzy string matching can alleviate this issue.

#### 4.1.1.1 Implementation

As discussed in section 2.5, we can utilise the Levenshtein distance algorithm to compute the minimum number of required operations to transform a query into a phrase. Therefore a query containing only a few character adjustments to transform it into a given phrase would promote that phrase as a suitable suggestion as it is considered close to the query. This metric can also facilitate ranking of the results - phrases are ordered according to their edit distance in relation to the users input string, with the smallest distance appearing at the top of the results.

We utilise the fuzzy string matching functionality accessible via the fuzzywuzzy<sup>1</sup> Python package which is built on the Levenshtein distance algorithm. The Levenshtein distance can be represented as a similarity ratio using the following formula:

$$\text{ratio} = \frac{(|a| + |b|) - \text{lev}_{a,b}(i, j)}{|a| + |b|}, \quad (4.1)$$

where  $|a|$  and  $|b|$  are the lengths of the strings  $a$  and  $b$  respectively, and  $\text{lev}_{a,b}(i, j)$  is the Levenshtein distance between strings  $a$  and  $b$ .

The fuzzywuzzy package provides a set of similarity functions which compute a single similarity score in the range 1-100 using equation 4.1, with 100 indicating two strings are equal. We utilise the standard ratio function to compute a single similarity score in the range 1-100 with 100 indicating two strings are equal. Further similarity functions exist in the package such as *partial\_ratio*, *token\_sort\_ratio* and *token\_set\_ratio*. The *partial\_ratio* function supports substring matching and produces a similarity score of 100 when a string  $b$  exists as a substring in string  $a$ . The *token\_sort\_ratio* function first tokenizes and lowercases the strings as well as ignoring punctuation. The tokens are then alphabetically sorted and concatenated together. The function then proceeds to compute the Levenshtein ratio. This function does produce a similarity score of 100 if strings  $a$  and  $b$  contain the same characters in different orders. However this is not suitable as we require full control over the tokenization procedure as well as which punctuation characters are removed. The *token\_set\_ratio* function works similarly to *token\_sort\_ratio*, however the intersection of common tokens is now considered, thus yielding higher similarity scores for strings containing similar tokens.

After experimentation it was found the simple *ratio* function was more suitable. This was because many phrases were found to be similar in terms of their content with only slight discrepancies between them, and these differences would help to rank phrases higher or lower depending on their content.

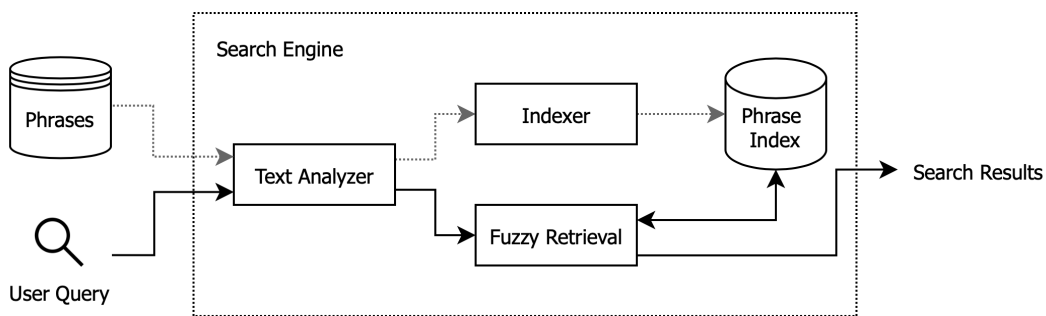


Fig. 4.1 Search engine architecture (prototype 1).

Figure 4.1 outlines the fuzzy search engine architecture. When the Flask application is loaded, the list of active English phrases are retrieved from `hazel_phrases` and passed to the text analyzer. The textual representation of each phrase is preprocessed which

<sup>1</sup>fuzzywuzzy Python package: <https://github.com/seatgeek/fuzzywuzzy>

involves case folding (converting text to lowercase) and removing preceding and trailing whitespace characters. The indexer is responsible for constructing the phrase index which, in this prototype, is simply a Python dictionary in which unique numbers are mapped to each of the preprocessed phrases. The user can then submit a query containing one or more search terms via the front-end Word add-in to the Flask application. The query is then analysed using the same procedure which was applied to the indexed phrases. The processed query is then passed to the fuzzy retrieval stage in which a similarity score is calculated against each phrase in the phrase index. Finally an ordered set of search results by similarity score is paginated and returned to the user.

#### 4.1.1.2 Issues Encountered

Following deployment of prototype 1 into the testing environment the following issues were found:

1. Although the system could tolerate minor spelling mistakes for short phrases, those phrases containing multiple words were generally not retrieved near the top of the search results for many single and multi-word queries. This was mainly due to the fact that the number of operations required to transform a single word query to a long phrase containing many other words is larger compared to shorter phrases. This behaviour was found to be particularly frustrating for the authoring experts as many search results were deemed irrelevant given a user query. In general, phrases containing only a single word were retrieved for a user query containing a single search word.
2. Longer phrases containing search words for a given user query were not retrieved towards the top of the search results. Instead, phrases which were closer to the search query in terms of the edit distance were returned, and those phrases containing a correctly spelled search word as well as other words were not retrieved towards the top of the results. This is mainly due to the required number of operations required to transform the user query to that phrase.
3. Expensive to compute similarity scores between the user query and each phrase in the phrase index. On each keystroke, a Levenshtein similarity ratio was computed. This was noticeable in the front-end application, with a 1-2 second delay in returning search results.

The issues outlined here demonstrate the need for a more sophisticated search procedure, given that the only factor contributing to the retrieval of phrases for a given user query was the Levenshtein distance.

#### 4.1.2 Prototype 2: TF-IDF Cosine Search with Spelling Correction

A noticeable shortcoming of prototype 1 was having to compute the Levenshtein distance between the users query and the list of phrases on each keystroke. In addition, the

algorithm was computationally expensive and many query-phrase distance calculations were redundant as not all phrases would be relevant to a given query. This meant computation time was wasted. This motivates the need for a data structure to store the vocabulary of words contained in the phrases to facilitate fast lookups at runtime and prevent pairwise computations for all possible query-phrase pairs in the phrase index.

Therefore we employ the TF-IDF cosine similarity search model discussed in section 2.2.2 to alleviate the issues found with prototype 1. This model will enable the following factors to be considered:

1. Frequency of terms in a phrase. Phrases which contain multiple occurrences of a given search term indicate the phrase is likely to be about that term and hence term frequency can be considered as a proxy for relevance.
2. For short phrases that contain only a few terms, it is likely those terms will have a term frequency of 1, therefore the IDF component of TF-IDF becomes a significant contributor to the relevance of a phrase given a search term. The IDF component also only has an effect on queries containing at least two terms and has no effect on ranking one term queries. Note here that in Chapter 3 we discovered in general phrases had a median length of 29 characters and hence are relatively short. Hence we do not remove stopwords here as they may contribute to the identification of short phrases and are thus considered potentially useful words.
3. Identification of unique/rare words. Phrases which contain rare search terms would increase relevance of a phrase containing those terms. The TF-IDF metric would help to identify terms which occur in certain phrases relative to all other phrases in the corpus. It may be the case that a complex phrase contains a particularly unique word, which would facilitate the phrase search for a user, which prevents them from typing multiple words or even the entire phrase for it to be returned in the search results. Hence rarity of terms can help to identify phrases which are relevant to a query as the occurrence of a rare term is significant.
4. Ranking of search results. Unlike in the boolean retrieval model where we can either match documents or not, the vector space model supports ranking of the search results via the cosine similarity function. This enables more relevant documents to be placed at the top of the search results.

In addition to these factors, we aim to implement spelling correction and partial string matching similar to that implemented in prototype 1.

#### 4.1.2.1 Implementation

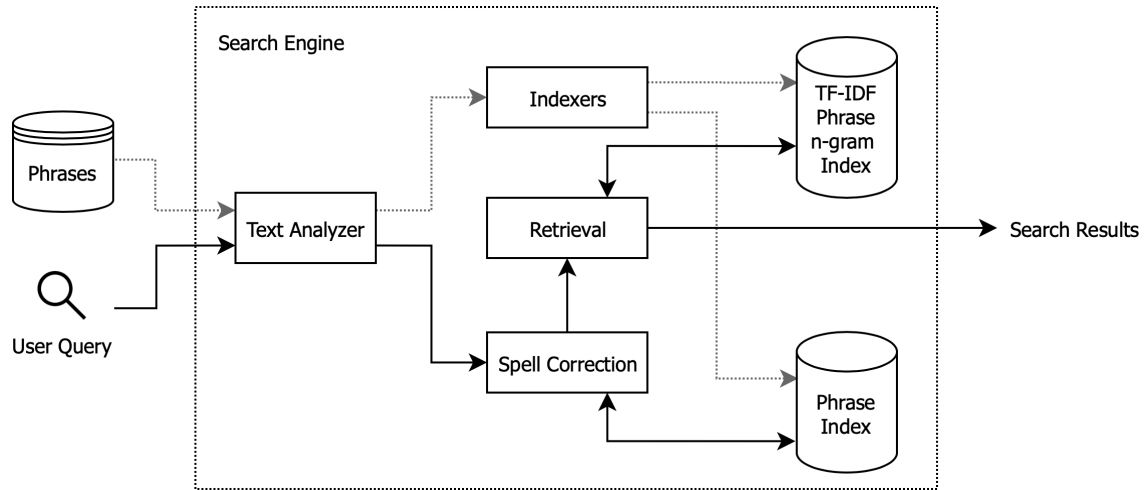


Fig. 4.2 TF-IDF cosine similarity search engine architecture (prototype 2).

Figure 4.2 outlines the prototype 2 search engine architecture. When the Flask application is loaded, the list of active English phrases are retrieved from `hazel_phrases` and passed to the text analyzer. The preprocessing steps here are akin to those conducted in prototype 1 - the textual description of each phrase is converted to lowercase, and preceding and trailing whitespace is removed. We then proceed to replace punctuation characters separating two independent terms with a whitespace (see Table 3.1 for examples). Removing these characters would combine these terms and would be treated as a single token during tokenization although they should be treated as distinct tokens. All remaining punctuation characters are simply removed as they are uninformative in determining relevance of a phrase<sup>2</sup>. The preprocessed text is then tokenized into words using the `word_tokenize` function from the NLTK<sup>3</sup> package.

Note here we apply conservative normalisation in the text analyzer. As discussed in section 2.4, common NLP preprocessing techniques such as stemming and removal of stopwords can be employed. However, if we were to stem the words obtained from the list of phrases we would lose the extra characters which were stemmed which would have helped to differentiate between complex phrases. This can result in many unwanted phrase matches for a given query containing the stemmed words. In the context of this project we do not remove stopwords as many phrases contain only a few words and are likely to contain these types of words and hence can add value to the search procedure.

The TF-IDF model in isolation would require completed words to be entered in the query to assess relevance. Ideally the search engine should be capable of producing near instantaneous search results for partial inputs and should recommend phrases similar to that input if any query words were misspelled. Therefore we utilise 2 distinct indexes:

<sup>2</sup>We utilise the Python package `string.punctuation` to remove the remaining punctuation characters: <https://docs.python.org/2/library/string.html#string.punctuation>

<sup>3</sup>NLTK: <https://www.nltk.org>

1. **Phrase index:** this index stores the tokenized phrases following preprocessing, which provides a vocabulary of the terms in the list of phrases to facilitate isolated-term spelling correction (see section 2.5).
2. **TF-IDF  $n$ -gram index:** this index stores the character  $n$ -grams of the tokenized phrases (i.e. the consecutive sequence of  $n$  characters from a given word) alongside their corresponding TF-IDF scores. An  $n$ -gram range of 1-3 was chosen as the default range to include unigrams, bigrams and trigrams. This enables us to boost relevance of a phrase containing more  $n$ -gram matches from the user input. Therefore the search engine can tolerate minor grammatical errors in the user query and support search-as-you-type behaviour for partially entered words. The user query is also decomposed into  $n$ -grams on each keystroke.

To facilitate the creation of the TF-IDF  $n$ -gram index structure we utilise the *TfidfVectorizer* class provided by the scikit-learn library [22] as an indexing mechanism, which calculates a matrix of TF-IDF features from the list of  $n$ -grams. Here we utilise the default parameter settings of the vectorizer<sup>4</sup>. The *norm* parameter is set to  $L_2$  to compute the unit norms of each vector. As discussed in section 2.2.2, a simple dot product is sufficient to compute the cosine similarity as the features are already normalised.

This greatly speeds up the development process as one is no longer required to construct an inverted index from first principles as well as implementing the previously defined functionality which the vectorizer provides. The vectorizer first computes a matrix of token counts using *CountVectorizer* followed by a *TfidfTransformer* to transform the matrix to a normalized TF-IDF representation. The final result is represented as a sparse matrix<sup>5</sup>, in which each  $n$ -gram has an assigned TF-IDF score and mapping to the phrase it was contained in. The vectorizer can then be applied to the user query (in an  $n$ -gram format) in the retrieval stage to compute a TF-IDF vector representing the user query. The cosine similarity between this query vector and the list of  $n$ -grams in the TF-IDF index are computed. If a phrase in the search results is identical to the user query, the score is changed to 1.0 thus boosting an exact match. Finally, the query-phrase pairs are ordered by the cosine similarity score and the ordered list of phrases are returned to the user. It was found in general search results were generated in the order of a few hundred milliseconds which is an improvement on prototype 1.

#### 4.1.2.2 Spelling Correction

In addition to the new core search functionality we implement an automatic spelling correction mechanism to support automatic correction of misspelled words in the users query. Using the TF-IDF cosine similarity search alone would yield no phrases for words which are not spelled correctly. This motivates the need for spelling tolerance. The

<sup>4</sup>All parameter settings can be found here: [https://scikit-learn.org/stable/modules/generated/sklearn.feature\\_extraction.text.TfidfVectorizer.html](https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html)

<sup>5</sup>This sparse representation of the term-document matrix is equivalent to the inverted index data structure.

vocabulary of words extracted from the phrases list provides a corpus we can utilise to detect misspelled words. In this way, we can tailor likely corrections for a misspelled term to the vocabulary as opposed to a generic list of all possible dictionary words. Here we implement isolated-term spelling correction which was discussed in section 2.5.

Firstly, we check for the presence of each query term against the vocabulary which is an  $O(1)$  operation. If a query term is not present in the vocabulary, we compute the Levenshtein distance between each query term and each term in the vocabulary. Vocabulary words which yield a distance less than or equal to 2 are considered candidate term corrections. This value was chosen as the maximum edit distance lookup to ensure terms with up to 2 spelling mistakes can be retrieved in the candidate correction pool. Increasing this value would therefore result in a larger set of candidate corrections. For each candidate, we replace the misspelled term with the candidate term and run the search procedure using the TF-IDF  $n$ -gram index. The results for each search are aggregated and sorted by their similarity scores. This functionality enables us to automatically yield phrases for a query which may contain up to 2 spelling mistakes in each query term without manual human correction of spelling mistakes.

#### 4.1.2.3 Issues Encountered

During development of this prototype the following issues were found:

1. Lack of clarity regarding the generated search results with automatic spelling correction enabled. On many occasions the authoring experts in HZC found it difficult to understand why certain phrases were returned. For example, a misspelled or partially completed word could be corrected to a different word within an edit distance of 2 which changes the overall meaning of the query thus yielding completely different phrases.
2. The bag-of- $n$ -grams model that doesn't take into account positional information of terms in documents. This could be solved by implementing a positional inverted index although this would increase development time.
3. Phrase index construction using the TF-IDF representation requires periodic updates to the index when existing phrases are edited or deleted and new phrases are added to ensure users have access to an up-to-date phrase corpus. This is an intrinsic property of the TF-IDF metric - the term frequency component can be easily computed for terms in a phrase, although the inverse document frequency component is reliant on terms present in a phrase relative to the phrases in the corpus which also contain these terms. Hence in the worst case the TF-IDF values for all phrases in the corpus would have to be updated if a new phrase is added or an existing phrase is edited or deleted. Subsequently it is advised the phrase index updating phase is executed with minimal side effects to the user experience. In addition to this index, the generic phrase index should also be updated accordingly. Preventing time delays for user search requests due to an index update is strongly recommended to prevent an undesirable degradation in the quality of the user

experience. With the corpus of circa 20,000 phrases under consideration, updating the phrase index took around 10 seconds when a single phrase containing two words was added. Upon commercialisation of HAZEL to external clients the volume of phrases added and how often they are added is unknown, hence it is of paramount importance the search engine can effectively manage changes to the index to prevent unwanted time delays for user search requests as this would degrade the user experience. Ideally the system should completely mask the updating phase from the user and ensure the existing phrase index (albeit an outdated version) is still accessible until the index is updated.

### 4.1.3 Prototype 3: TF-IDF Cosine Search with Spelling Suggestions

This prototype was implemented to address the lack of clarity issue observed in prototype 2 regarding automatic spelling correction. In this prototype we implement ‘did you mean’ suggestions akin to those usually found in search engines such as Google. This will enable smart spelling corrections to be preserved in the search engine, whilst giving the user the option to either correctly spell their query words or use the smart suggestion to aid their search.

#### 4.1.3.1 Implementation

The architecture of prototype 3 is similar to that of prototype 2 (see Figure 4.3).

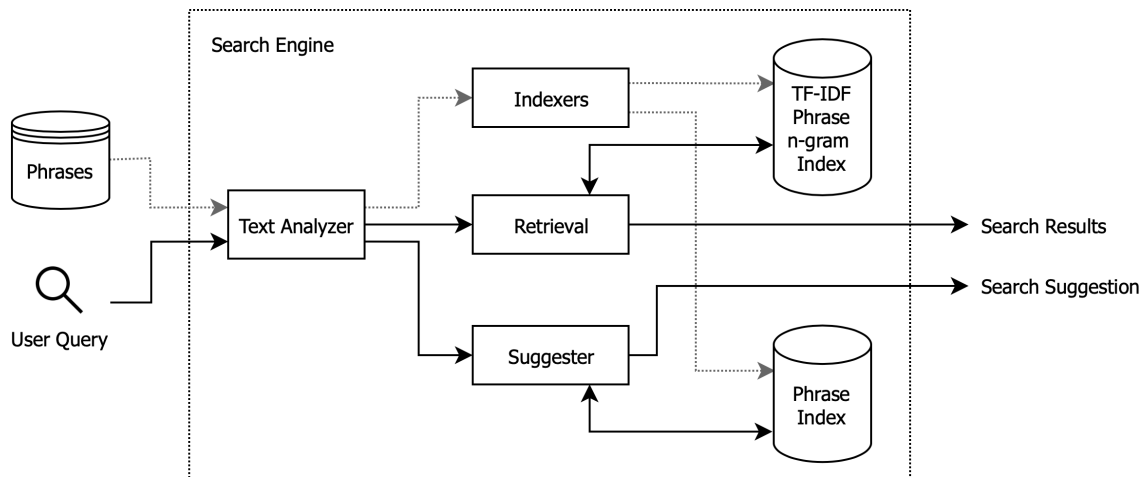


Fig. 4.3 TF-IDF cosine similarity search engine architecture (prototype 3).

The spelling correction module was replaced with a suggester module to facilitate rapid lookups for potentially misspelled query words. Hence in this implementation, a search suggestion is only returned if at least one of the query words is spelled incorrectly. Note here also that search results will be generated in addition to the search suggestion so that the user always has access to a list of results regardless of any spelling mistakes in



their query. The spelling suggestion mechanism utilises the SymSpell algorithm outlined in section 2.5. If any entered word is not a recognised term in the vocabulary, the most frequent word having an edit distance smaller than 2 will be the recommended suggestion for that word. This process is repeated for all query terms and a suggestion is constructed from these suggested words which is then presented in the front-end add-in application. It was found in general both the search results and search suggestions were generated in the order of a few hundred milliseconds which is similar to prototype 2.

In this implementation we also apply sublinear term frequency scaling by replacing term frequency  $tf$  with  $1 + \log(tf)$ . This was chosen because if a term  $a$  was to appear  $x$  times more frequently than a term  $b$ , it is not necessarily the case that term  $a$  is  $x$  times more important than term  $b$ .

#### 4.1.3.2 Issues Encountered

During development of this prototype the issues found were similar to those found in prototype 2 although the lack of clarity issue regarding spelling corrections was solved in this prototype. Issue 3 regarding phrase index updates was still an issue observed in prototype 3. We elide specific details regarding the solution to this issue here and delegate it to the considerations for the proposed architecture.

## 4.2 Proposed Architecture

Following feedback from the authoring experts it was found prototype 3 generally was a significant improvement on the current regular search procedure. Users could retrieve phrases containing minor spelling mistakes and partially entered words from a list of ranked search results. Therefore prototype 3 will provide the foundation for the proposed architecture.

Upon commercialisation of HAZEL to clients, the length of documents and the number of terms within them will vary between clients depending on their requirements. As BM25 is an extension to TF-IDF, it has the added benefit of accounting for term frequency saturation and document length as discussed in section 2.2.3. For modern full text search collections, it makes sense to pay attention to term frequency and document length, however as discussed in section 3.1.1.1 our phrases have different lengths with many being very short. Hence BM25 can help rank documents containing terms in short phrases when both terms are present in the query.

We propose an identical architecture to that outlined in section 4.1.3 for prototype 3 with the replacement of TF-IDF scores with BM25 scores. Cosine similarity computations will no longer be required and all previously implemented features such as search suggestions will be maintained.

Here we outline some advised future considerations for the proposed system:

1. **Updating the phrase indexes.** When phrases are added, edited or deleted, the phrase indexes must be updated accordingly. A procedure to re-index will therefore

be required. In addition to this, it is advised the indexes are stored on disk instead of having to re-compute the entire index every time the application loads. This would be in the form of centralized indexes and would facilitate their management.

2. **Autocompletion:** A useful feature in modern search systems is autocomplete or “search-as-you-type”, which guides the user towards the results they want by prompting them with possible completions of the phrase they are currently typing. This ultimately reduces the amount of time taken for users to find documents they require as they are not required to enter a completed phrase. This can be achieved by using edge  $n$ -grams. They differ from traditional  $n$ -grams as they enumerate all  $n$ -character subsequences from the start of a string (e.g. ‘the’ produces the edge  $n$ -grams ‘t’, ‘th’ and ‘the’ with an edge  $n$ -gram size of 3).
3. **Caching of search results for frequently entered queries:** This would prevent retrieval computations on the server for common queries and queries recently entered which contain complex words. However the cache would need to be refreshed periodically to reflect any new phrase changes on the server.
4. **Further preprocessing procedures:** For example expanding contractions into distinct tokens (e.g. won’t becomes will not) and experimenting with different tokenization methods.
5. **Incorporating click-based features:** Unlike in marketplace search in which clicked search results indicate relevance of the result relative to a query, this is not always the case. If the search results are biased and not returning intended phrases for a given query, a user may be inclined to click on a query similar to the one they intended to prevent wasting time. This is an important consideration as the overall quality of an SDS will be reduced as more relevant phrases to the user’s information need may not be included. If a click-based feature was utilised in a machine-learned ranking function, exact query-phrase matches should be prioritised. This is because more popular phrases in terms of their total number of clicks may be higher than exact query-phrase matches. Phrases should not necessarily be boosted based on their frequency of use as phrases are chosen on a per-SDS basis. An extra consideration here would be the window in which frequently selected phrases are considered. This could be implemented to realise those phrases frequently selected within the last week, month or year.
6. **Learning to rank:** Learning to rank systems would become an interesting area of research once a large corpus of query data is collected. This would enable experiments to be conducted to assess how useful machine learned ranking systems will be. An important consideration for implementing such systems into the proposed architecture is ensuring search behaviour is normalised for all users (see section 5.3.5). The hyperparameters for BM25 could also be tuned using machine learning as only the default settings were proposed.

### 4.3 Chapter Summary

In this chapter we experimented with a range of retrieval and ranking models to yield a set of phrases given a query. We began with a fuzzy text matching prototype which was found to be insufficient to obtain relevant phrases. The TF-IDF cosine similarity model was implemented in prototype 2 and alleviated the issues found with prototype 1 and incorporated automatic isolated-term spelling correction. This was found to produce a lack of clarity for the authoring experts regarding the generated search results, therefore prototype 3 explored the use of ‘did you mean’ suggestions as an alternative. This gave users more control over which results were returned and alleviated the lack of clarity issue in prototype 2. Finally we concluded with the proposed architecture which utilised prototype 3 as the foundation and BM25 scores instead of TF-IDF scores. We also provided future improvements for this prototype and considerations for implementing it into HAZEL’s production system.

# Chapter 5

## Results and Discussion

In this chapter we conduct a thorough evaluation of the refined prototype (prototype 3) using the query log data generated following deployment. The metrics computed for prototype 3 in this chapter will provide baseline measures and an indication of overall search performance. We seek to assess the relevance of the search results using a procedure to obtain a set of relevance judgements from the authoring experts to conduct a quantitative evaluation. The ranking of the search results will also provide insight into the average rank for the highest relevant search result for a given query. Using the labelled relevance scores we also quantify the extent to which the authoring experts agree on their assigned judgements. We then provide a summary of the feedback obtained for each of the prototypes outlined in Chapter 4. Finally, we conclude this chapter with the results obtained from a questionnaire presented to the authoring experts which outlined a set of potential features to implement in a further iteration of the system.

### 5.1 Evaluation

One could evaluate a search engine by analysing the total number of characters typed for both the previous and proposed systems. The autocompletion feature of a search engine can significantly reduce the search effort required by a user in terms of the number of characters they type. However one should not evaluate the search engine's performance based on the number of characters typed alone as the system would be unable to produce the best set of suggestions given only a few characters. Hence it is generally more advisable for the user to invest more effort when entering their search terms than to produce search suggestions which are not representative of the user's intention [33]. In addition, it is generally required for the user to enter further characters to reduce the search space given the overlap of common words in the set of phrases we consider in this project.

In this section we seek to assess the relevance of the generated search results by conducting a thorough evaluation of the search results at the time of phrase selection for prototype 3 following its deployment (see section 4.1.3). In general, relevance of a search result is relative to the information need and not the query [20]. This means phrases

are relevant if the information need is addressed and not because it happens to contain all query words. Due to time constraints, we were unable to experiment with different parameter settings and in a future evaluation one could evaluate system performance using a range of parameter settings for a range of query test collections to maximise performance on these collections. This would yield an unbiased estimate of performance [20].

### 5.1.1 Relevance Judgements

A notable challenge with determining relevance is the subjectivity of relevance scores assigned to a set of search results across a range of different users. Although the selected phrases and their ranks have been stored, assuming they are relevant and treating them as relevance judgements introduces a significant amount of bias. This can be due to the type of query and the rank in the results list. Hence it is not safe to assume that clicked phrases are relevant [2]. Therefore to quantify relevance of the search results, a set of ground truth relevance judgements are required - this involves the authoring experts in HZC marking search results as relevant or non-relevant for a given query. In many situations, a given phrase may contain some or all of the query words with extra words not present in the query and therefore an expert assessment is required to ascertain which of the search results are relevant.

Given the search results at the time of phrase selection were not stored, it was necessary to first retrieve all relevant phrase selections for prototype 3 and re-run the search using the search text at the time of selection. This results in a list of search results that would have been generated at the time of searching. In total, 190 phrase selections were accumulated with prototype 3, which stored the search text at the time of selection. Although this corpus of generated query logs is not significantly large, we nonetheless proceed to compute standard evaluation metrics to assess performance of this prototype in a quantifiable manner. A larger dataset would ensure a more appropriate and representative evaluation could be conducted.

Given a search text query, we seek to quantify how relevant the generated search results were for that particular query. The following procedure was used to construct an appropriately formatted dataset for evaluation:

1. Extract all phrase selection records from `phrase_selections` for prototype 3. This will return a list of phrases selected for a given search text.
2. Randomly shuffle the set of records retrieved from Step 1.
3. For each search text query, run the search procedure using prototype 3 to obtain a list of search results generated for that query. Here we consider only the top 10 search results as this equates to the total number of results present on the first page of the search results.
4. For each of the 5 authoring experts in HZC, randomly partition the experts into 2 groups (group A and group B), with each group having 2 and 3 experts respectively.

This step is done due to the subjective nature of assessing relevance. Hence this step will also enable us to quantify to what extent the users agree on the relevance of a subset of the search results for a given phrase selection, as each group will share some queries in common.

5. Allocate 20 phrase search queries from the shuffled list to both group A and group B. Hence group A has 2 users sharing 20 search queries in common for both users, with group B sharing 20 search queries in common for all 3 users.
6. Allocate 30 unique phrase search queries to each user from the remaining set. Each user now has 50 queries to evaluate, with 20 queries shared between at least another user in the same group, as well as 30 distinct search queries each.

The granularity of relevance we have chosen extends simple binary judgements to provide a more informative evaluation compared to using binary judgements. Given the inherent nature of the phrases, some will be more relevant to a given query as they may consist of all query words or similar words with slight discrepancies such as synonyms. Hence we chose to use a graded relevance scale to further differentiate between those phrases which were deemed very relevant, somewhat relevant and not relevant. This allows for more flexibility for the experts, although more subtle distinctions are required to be made to differentiate between relevance categories [26].

A series of Microsoft Excel files were then constructed for each user using Python. For a given search result for a given search query, we asked the users to assess relevance of each of their 50 allocated search results. A value of 2 would indicate a result is very relevant, a value of 1 would indicate the result is somewhat relevant and a value of 0 would indicate the result is not relevant. This scale was chosen such that phrases which are more relevant than phrases which are somewhat relevant could be differentiated. Here a relevant search result for a given query would suggest that result is a possible intended phrase to search given the query and not based on if the search result contained some or all of the query words.

Following assessment from the authoring experts a total of 170 queries were evaluated from a possible 250.

### 5.1.2 Relevance of Search Results

In this section we explore evaluation metrics for ranked retrieval results for prototype 3. A metric commonly used in information retrieval is precision, which measures the accuracy of the system by computing the fraction of documents which are relevant for the retrieved list. We can extend this by computing the proportion of top- $k$  results which are relevant, as typically we are only concerned with the top few documents. This is known as Precision at  $k$  ( $P@k$ ). The value of  $k$  can be chosen based on how many documents the user will view, and typically a value of 10 is an appropriate choice. To calculate this measure, we must obtain relevance judgements for the top- $k$  documents. This has the advantage over recall, a further metric, which can only be computed if all relevant documents have been identified [6].

An extension to precision is average precision (AP) which is a quantity combining both recall and precision for ranked retrieval results. For a given information need, AP is the mean of the precision scores after each relevant document is retrieved:

$$AP = \frac{\sum_r P@k}{R}, \quad (5.1)$$

where  $k$  is the rank of each relevant result,  $R$  is the total number of relevant results and  $P@k$  is the precision of the top- $k$  retrieved results [35].

A further metric commonly used in information retrieval is Mean Average Precision (MAP) which computes the average of the average precision values for the set of top  $k$  documents [20]. If the set of ranked retrieval results for an information need  $q_j \in Q$  is  $\{d_1, \dots, d_{m_j}\}$  and  $R_{jk}$  is the set of ranked results from the top result until document  $d_k$  (i.e. equation 5.1), then MAP can be computed as follows:

$$MAP(Q) = \frac{1}{|Q|} \sum_{j=1}^{|Q|} \frac{1}{m_j} \sum_{k=1}^{m_j} \text{Precision}(R_{jk}) \quad (5.2)$$

For the total of 170 marked queries a MAP value of 0.882 (3.d.p) was obtained. This value indicates on average a given search query will produce relevant phrases towards the top of the results list. As previously mentioned this analysis was conducted on a small number of queries, therefore to yield a more representative result a larger dataset should be used if possible.

### 5.1.3 Ranking of Search Results

The first metric we utilise is the Mean Reciprocal Rank (MRR). This measure first computes the reciprocal of the rank of the first relevant document in the search results for a given query. The MRR is then calculated by averaging the reciprocal ranks across all queries [6]. Therefore the MRR can be computed with the following equation:

$$MRR = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \frac{1}{\text{rank}_i}, \quad (5.3)$$

where  $\text{rank}_i$  represents the rank of the first relevant document for the  $i$ -th query.

Computing the MRR on the previously discussed phrase searches evaluation dataset, a value of 0.968 was obtained for all queries. This metric however only accounts for the first relevant search result, and in many cases multiple search results may be deemed relevant. MRR is applicable in the situations where users intend to find a known item (i.e. the user searches for a document that they have either seen before or they know exists), or there is only one relevant result. MRR is equivalent to MAP where each query has exactly one relevant document [6]. However in this project many phrases may be relevant for a given search query and when many variants of the required document exist with minor discrepancies and many results are considered relevant, MAP is generally more useful as it considers the position of all relevant items in the search results.

Finally we explore another common metric known as DCG (discounted cumulative gain) which is known to work well for evaluating document ranking using graded relevance judgements [25]. Highly relevant results appearing lower in the search results are penalized. At rank position  $p$ , DCG is calculated as follows:

$$\text{DCG}_p = \sum_{i=1}^p \frac{rel_i}{\log_2(i+1)} \quad (5.4)$$

A normalized version of DCG (nDCG) which uses an ideal DCG (IDCG) is used to produce a metric comparable to different queries in varying result set sizes. IDCG first sorts the search results by their relevance scores to produce an ideal DCG at rank  $p$  to normalize DCG:

$$\text{nDCG}_p = \frac{\text{DCG}_p}{\text{IDCG}_p} \quad (5.5)$$

The nDCG values can be averaged across all queries and an ideal value is 1.0. A value of 0.988 was returned for nDCG, a strong value close to 1 which indicates the search results generally ranked relevant results towards the top as marked by the authoring experts.

#### 5.1.4 Investigating Agreements of Relevance Scores

Here we provide a quantitative assessment of the extent to which the authoring experts agree on relevance scores for the set of relevance judgements collected using the procedure discussed in section 5.1.1. We can evaluate the subset of allocated queries for groups A and B, with each group sharing 20 randomly allocated queries between each member of the group.

Collecting relevance assessments was found to be a time-consuming process. Due to the timeframe within which prototype 3 was deployed and relevance judgments were collected, we were only able to collect a small dataset for evaluation. The relevance judgement collection procedure enabled the formation of a dataset to facilitate the evaluation of inter-expert agreement for the assigned relevance scores. Group A, consisting of 2 experts, returned relevance judgements for their 20 shared queries. Group B, consisting of 3 experts, returned relevance judgements for their 20 shared queries for only 2 experts. Hence we can evaluate the agreed relevance scores for a total of 20 queries per group, with each query having 10 search results.



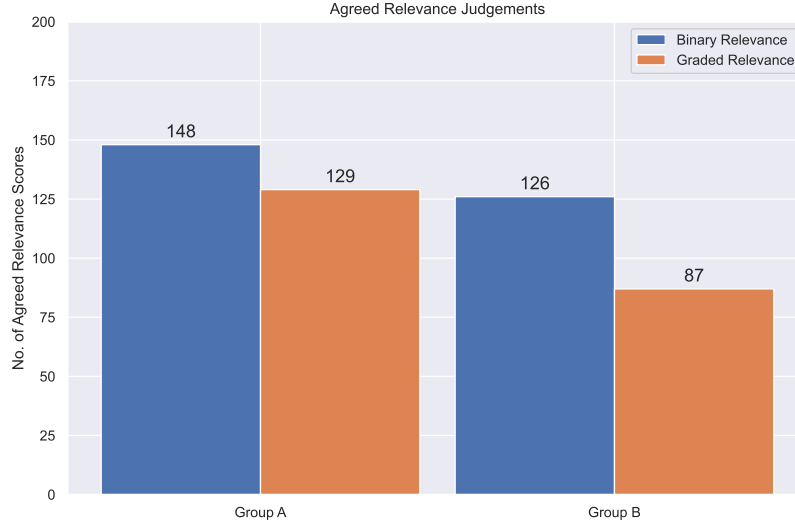


Fig. 5.1 Total agreed binary and graded relevance scores for groups A and B.

The agreed relevance scores are shown in Figure 5.1. Here we consider the complete set of relevance judgements for each group when binary relevance and graded relevance are considered respectively. For groups A and B, 148 and 126 binary relevance scores were agreed respectively. The number of agreed graded relevance scores in both groups were generally lower than the agreed binary relevance scores, with values of 129 and 87 being observed from both groups respectively. This was mainly due to having a very relevant category and given relevance is subjective, the chance of multiple assessors agreeing on relevance over a larger number of categories is expected to be less.

A commonly used statistic known to be more robust than a simple percentage agreement is Cohen's  $\kappa$  which accounts for the possibility of the agreement occurring by chance [4]. We utilise this measure to quantify agreement between the experts in the separate groups A and B. This statistic can be calculated as follows:

$$\kappa = \frac{P(A) - P(E)}{1 - P(E)}, \quad (5.6)$$

where  $P(A)$  is the proportion of time the judges agree (inter-annotator agreement) and  $P(E)$  represents the proportion of the times they would be expected to agree by chance [18, 20]. This produces a value between -1 and 1, with scores above 0.8 generally considered good agreement, with 0 or lower indicating no agreement [22].



Fig. 5.2 Cohen's  $\kappa$  values for both agreed binary and graded relevance scores for the allocated search results to groups A and B.

Figure 5.2 shows the Cohen's  $\kappa$  values for both groups when binary and graded relevance scores are considered. We computed these values using scikit-learn's *cohen\_kappa\_score* function. For both groups A and B, values of 0.497 and 0.423 were obtained respectively for the  $\kappa$  values for binary relevance scores. These values indicate a moderate level of agreement between the experts in both groups when agreeing on relevant/not relevant assignments to each of their 200 allocated search results. When graded relevance was considered the  $\kappa$  values were generally lower. A value of 0.285 for group A indicates a minimal level of agreement and 0.109 for group B indicates almost no agreement between the graded relevance scores. It is expected a larger relevance scale with a finer level of granularity would increase disagreements of the relevance scores between the experts.

## 5.2 Prototype Feedback

As outlined in Chapter 4 we devised a series of prototype search engines, which enabled us to solicit feedback from the authoring experts and experiment with different features found during the research conducted in Chapter 2.

Prototype 1 explored the use of approximate string matching which generally only worked for short phrases consisting of only 1 or 2 short words. Searching for a correctly spelled word should return phrases containing that word and not phrases containing words which are close in terms of their edit distance. Therefore the authoring experts concluded a more sophisticated approach was required which could incorporate multiple factors to search for both rare and complex words occurring in a phrase as well as producing a ranked set of search results.

This was manifested in prototype 2 which utilised the TF-IDF cosine similarity search model and overcame all the limitations of prototype 1. Automatic spelling correction

was also implemented to relax the constraint for users to spell complex words correctly. Despite receiving more positive feedback compared to prototype 1, automatic spelling correction was found to generally produce a lack of transparency over the ranking of the search results.

Prototype 3 addressed the complexities found with automatic spelling correction by utilising a search suggestion module to recommend words which were misspelled in the user's query. This was found to be a more appropriate strategy for dealing with spelling mistakes, as it was discussed the authoring experts often know the phrases they require in advance and therefore they should ensure words are spelled correctly. Having search suggestions overcame the lack of transparency issue found with automatic spelling correction, and gave the user the option to search for phrases containing minor spelling mistakes without the search engine performing too much processing on the user's behalf. Following this positive feedback, the proposed implementation outlined in section 4.2 used prototype 3 as its foundation.

### 5.3 Search Engine Features Questionnaire

Following implementation and deployment of the series of prototype models, we devised a questionnaire consisting of both open-ended and closed-ended questions to solicit further feedback from the authoring experts. This was undertaken to better understand and prioritise further development of search engine features which would be better suited to the users requirements. In the following sections we summarise the qualitative and quantitative feedback received for a total of 4 out of 5 received questionnaire responses.

#### 5.3.1 Feature 1: Spelling Correction

Overall, spelling correction was identified as a relatively important feature (see Appendix C.1). Should a given user incorrectly spell a word contained in a phrase by 1 or 2 letters, the phrase should still be retrieved. Incorporating spelling tolerance overcomes the difficulty of having to spell entire phrases correctly which may contain several complex words and misspelling any would not retrieve results.

Users were also asked of the importance of the search engine having the ability to identify minor language variation between words in phrases (e.g. sulfur and sulphur should return phrases containing words having either of these spellings). Although this feature was identified as useful, it was not deemed a necessary feature to have as phrases containing words with potential spelling variations are minimal. The authoring experts mainly author in UK English so this feature was found to be not essential.

#### 5.3.2 Feature 2: Multi-Word Order-Independent Search

This feature was recognised as an essential feature for the search engine to have (see Appendix C.2). Users generally said only a few words are entered and not the whole phrase as it becomes difficult to remember entire phrases by heart. This was identified as

a limitation of the regular search algorithm, which was unable to search multiple words regardless of their order in the phrase. Hence a search engine which can yield phrases containing several words at any position within a phrase should be returned in the search results.

### 5.3.3 Feature 3: Phrase Autocompletion

This feature was recognised as a useful feature, as many users described they were unable to remember a complete phrase or long words in a phrase (see Appendix C.3). In many cases, it was described the users would prefer smart search completions in the search results to prevent typing complete words. However, this feature was not as important as searching for multiple words regardless of their order, and employing both these features in conjunction poses further implementation difficulties as partial search words can now appear in any order and should populate appropriate search results.

### 5.3.4 Feature 4: Multi-Word Prefix Search

A multi-prefix matching algorithm was analysed to suggest terms whose words' prefixes contain all words in the string entered by the user from a large medical ontology [27]. An idea similar to this was proposed in the questionnaire - users would enter short prefixes for multiple words and have the algorithm automatically complete each word based on each prefix. This was identified as a generally useful feature, as many long words would no longer be required to be fully typed (see Appendix C.4). Although it was found this feature is of a low priority and this style of searching would require users to take time to become accustomed to it. It was also mentioned users could have access to a prefix list for quick lookups.

### 5.3.5 Feature 5: Online Learning

As discussed in section 2.7 many search engines can incorporate machine learning to rank search results in accordance with the logged user behaviours. This feature generally yielded a mixed response (see Appendix C.5). An interesting point raised here was the negative implications for the search engine to promote automatic authoring and real-time learning of user behaviours. For SDS authoring, it is of paramount importance for authoring experts to be in control over which phrases are selected. A user is more inclined to select a phrase similar to the one they intended if it is present at the top of the the search results as you will not need to question your own judgement, and thus the quality of the SDS is reduced. This could ultimately lead to an inaccurate SDS.

A further point noted here was the potential for many relevant phrases to be 'lost' further down the search results which would devalue the database of phrases, meaning most standard phrases could be always present at the top and thus chosen more often. This would ultimately make the SDS document sub-par and many of the desired phrases to make an SDS stand out would require further effort to find in the results.

The inclusion of machine learning has the potential to provide a lack of transparency and control over which results are returned. The search results should be relatively deterministic and as such the search results should not necessarily promote phrases in the results if they were selected recently or more frequently. If this feature was to be implemented, having the ability to disable self-learning for a given user was highly recommended.

Despite these shortcomings, it was noted many phrases at given positions in an SDS are likely to be chosen more often and hence should be ranked higher in the search results for a similar search query.

### **5.3.6 Feature 6: Frequently Selected Phrases**

The final proposed feature was the separation of commonly used phrases in a given document into a separate tab in the front-end application. This would alleviate the issues with those discussed for Feature 5, as deterministic search behaviour for the search engine is preserved and the phrases in the results are not dependent on their selection frequency. It was noted that many phrases were present at multiple locations within a given SDS document and hence having a log of these phrases would be a useful reference for the users. Although it was discussed to incorporate phrase selection frequency into the search engine, the frequency of use for a given phrase is document dependent and will not always be relevant to every SDS.

# Chapter 6

## Conclusions

In this chapter we begin by reviewing whether the original aims and objectives outlined in Chapter 1 have been successfully implemented and explored in this paper. We then outline future research areas for this project. We then outline the deviations from the original project proposal and how the agile methodology was employed to experiment with various prototype models as opposed to creating a single proposed system. Finally we conclude with the key lessons learned from this project and the notable challenges we encountered and had to solve during the project.

### 6.1 Review of Aims and Objectives

The key aim outlined in Chapter 1 was to explore an array of strategies available to aid in the development of a smart predictive search model prototype Yordas could utilise in the production version of HAZEL. This was outlined in Chapter 2, which detailed the comprehensive research undertaken to explore the range of relevant techniques to create these models. An overview of the proposed architecture in Chapter 4 outlined some of the relevant features following feedback from the authoring experts to enable a bespoke system architecture to be developed. The following overarching objectives were also outlined in Chapter 1:

- **Research appropriate techniques used to implement smart search models.** Following the research conducted in Chapter 2 it was decided to experiment with model features through a series of prototype models which were outlined in Chapter 4. This objective was achieved and following the prototypes, a proposed architecture was developed for Yordas to continue developing following completion of this project. However an online learning model which incorporates user behaviours was not accomplished. This was mainly due to the data limitations outlined in Chapter 3. It is envisaged a future iteration of the system can utilise these methods once the appropriate data is stored and a larger corpus of query data is generated. This would enhance HAZEL's predictive capabilities further.

- **Implement a series of prototype models.** As previously mentioned this was successfully accomplished in Chapter 4. This enabled constructive feedback to be obtained from the authoring experts and helped to identify the suitable features which aligned to the users search behaviours.

The following sub-objectives were also outlined in Chapter 1 in order to realise an effective UX:

1. **Relevant search results for a given query:** This was found to be a difficult problem to solve. Various methods explored in Chapter 2 dealt with determining relevance in different ways. BM25 and the TF-IDF cosine similarity search models were deemed more appropriate solutions than simple fuzzy text matching to determine relevant search results for a given query and facilitated ranking of the results. In Chapter 5 we successfully conducted a quantitative evaluation of the extent to which the authoring experts agreed on their assigned relevance labels. It was found the experts generally agreed on the relevance labels when binary relevance was considered, with less agreement for a graded relevance scale. Further quantitative evaluation was conducted to assess the rank of relevant phrases in the search results through a series of commonly used evaluation metrics. A qualitative evaluation was also conducted and presented in Chapter 5, which included general prototype feedback and a questionnaire to solicit feedback and suggestions on search engine features.
2. **Spelling correction:** Both automatic and manual spelling correction techniques were explored in Chapter 4. Following feedback from the authoring experts, the implementation of ‘did you mean’ suggestions was more desirable compared to automatic correction as the latter provided a lack of clarity.
3. **Identification of suitable factors and choice of algorithms for use in the new search engine:** we implemented and experimented with approximate string matching, TF-IDF cosine similarity search and the BM25 models in Chapter 4. It was found both the TF-IDF and BM25 models were more appropriate for the proposed implementation. Learning to rank methods were explored in Chapter 2, although these methods generally require a large amount of data and will not necessarily produce intended results.
4. **Easy to access and use the new search model:** we implemented an easy-to-use toggle feature to ensure accessing the regular and prototype search models were easy to access for the users (see Appendix A.4). This prevented bias towards a particular procedure and provided a fallback when the prototype system did not yield intended phrases (for example when using prototype 1).
5. **Development of testing suite for prototype model testing.** This was implicitly developed in Chapter 4. The IT team at Yordas maintain a test suite to test various aspects of the software and this was extended to support testing of the

prototype search models. Unit tests were developed to test the search suggestion feature produced an intended result, ensure intended phrases were present in the search results and ensure exact phrase matches were present at the top of the results.

6. **Ensure privacy and security-related matters relating to user phrase searches and selections are considered in each prototype.** As each prototype was designed to experiment with a subset of features found during the research conducted in Chapter 2, no external dependencies regarding the storage of phrases were considered. This would have been noticeable if software such as Elasticsearch was used however our prototype and proposed implementations are built from a foundational level. Hence Yordas has complete control over phrase and user data.

## 6.2 Future Work

In addition to the proposed implementation outlined in Chapter 4 we present some interesting areas one could explore in future research.

1. **Multilingual search.** Although only English phrases were considered in this project, one could also research techniques to implement multi-language search support. This could be configured by the user to enable phrases of certain languages to be considered for searching.
2. **Phrase selection location in an SDS.** During discussions with HZC it was discovered certain phrases would be selected more often in specific areas of an SDS. Once the location index of selected phrases is implemented into the `hazel_sds_requests` database, commonly selected phrases at the user's current cursor location could be suggested as a possible phrase without typing a single character. This could be achieved by utilising a clustering model.
3. **Positional inverted index.** As discussed in section 2.6 term proximity information can be included in the search model. However this requires the construction of an appropriate positional index to facilitate lookups of phrases containing query terms which are close to each other. We envisage this would improve ranking scores of phrases containing query terms close to each other, although development time would be increased in order to replace the proposed TF-IDF  $n$ -gram index.
4. **Learning to rank models.** As discussed throughout this paper the lack of data and its limitations made it difficult to construct a suitable machine-learned ranking model. Storing the search results for a given query alongside their ranks would enable a range of these methods to be explored once a larger corpus has been generated. Relevance judgements can also be incorporated into the model to adapt to the behaviour of the collective set of users.



5. **Multi-word prefix search.** As discussed in section 5.3.4 a potential feature to implement in a future iteration of the search engine is the support for prefix searching of complex words. Here the search model should not only search for phrases which contain the user-entered prefix, but also suggest the most likely phrase completion of that prefix.

## 6.3 Deviations from the Proposal

In the initial project proposal, emphasis was placed on the creation of a single prototype search model and the exploration of machine learning approaches in the research phase. Although the latter was explored, the data limitations outlined in Chapter 3 prevented a more thorough experimentation with machine learning models and hence a more foundational approach was taken. As opposed to a single prototype model, we decided to implement a series of prototype models to experiment with different search engine features and ascertain which were more appropriate for the problem at hand. Therefore the initially planned timeline had to be adjusted to compensate for the agile methodology employed in this project. As the prototype models were iteratively implemented into HAZEL throughout the project, no explicit documentation outlining details for integration was required.

## 6.4 Lessons Learned

Given the difficulty of this project in terms of the wide variety and diversity of available techniques, it was necessary to first experiment with a subset of these features to evaluate their applicability and usability in HAZEL. Prototype 1 demonstrated the need for a more sophisticated approach to phrase searching, as simple fuzzy text matching in isolation, although was found to work for short phrases, generally does not perform as well for longer phrases. As such we proceeded to experiment with more complex features in prototypes 2 and 3 such as frequent and rare word search, preprocessing methods and tolerating grammatical mistakes in user queries.

In general we learned that it is difficult to produce reasonable search results given only a 1 or 2 character search query. This meant more effort on the users behalf, as not only can these prefixes be ambiguous, but many phrases also contained the same words. Therefore ensuring users invest more effort into their search queries will inevitably increase the probability of yielding relevant search results.

An interesting point learned in this paper is users may prefer deterministic behaviour for the generated search results. For example if a phrase was recently selected on multiple occasions, one could suggest this phrase is more relevant than other similar phrases and thus promote the rank of this phrase. However in the context of this project it was found users generally want similar results for similar queries and do not want suggested results to take priority over expected results. As such it was paramount a core search mechanism (notably the TF-IDF cosine similarity and BM25 search models) provide a

solid foundation for the search engine before proceeding to experiment with potentially undesirable suggestion generation mechanisms.

## 6.5 Final Remarks

Overall, we are extremely proud of the result of this project and we believe it was a great success. We successfully achieved the key aims and objectives outlined in this paper and implemented a final working prototype search engine currently used in HAZEL's production environment (prototype 3). A wide variety of different techniques were explored in this paper and recommendations were given for further development. Following implementation of the proposed search engine architecture we hope this project will continue onto something bigger in the future.

# References

- [1] Baeza-Yates, R., Ribeiro-Neto, B., et al. (1999). *Modern information retrieval*, volume 463. ACM press New York.
- [2] Carterette, B. and Jones, R. (2008). Evaluating search engines by modeling the relationship between relevance and clicks. In *Advances in neural information processing systems*, pages 217–224.
- [3] Ceri, S., Bozzon, A., Brambilla, M., Della Valle, E., Fraternali, P., and Quarteroni, S. (2013). *Web information retrieval*. Springer Science & Business Media.
- [4] Cohen, J. (1960). A coefficient of agreement for nominal scales. *Educational and psychological measurement*, 20(1):37–46.
- [5] Connelly, S. (2018). Practical bm25 - part 2: The bm25 algorithm and its variables.
- [6] Craswell, N. (2009). *Mean Reciprocal Rank*, pages 1703–1703. Springer US, Boston, MA.
- [7] Cummins, R. (2016). Lecture 1: Introduction and the boolean model.
- [8] Dalianis, H. (2002). Evaluating a spelling support in a search engine. In *Proceedings of the 6th International Conference on Applications of Natural Language to Information Systems-Revised Papers*, NLDB '02, page 183–190, Berlin, Heidelberg. Springer-Verlag.
- [9] Garbe, W. (2017). Symspell vs. bk-tree: 100x faster fuzzy string search spell checking.
- [10] Grotov, A. and de Rijke, M. (2016). Online learning to rank for information retrieval: Sigir 2016 tutorial. In *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval*, pages 1215–1218.
- [11] Joachims, T. and Radlinski, F. (2007). Search engines that learn from implicit feedback. *Computer*, 40:34 – 40.
- [12] Jones, K., Walker, S., and Robertson, S. (2000). A probabilistic model of information retrieval: development and comparative experiments: Part 2. *Information Processing Management*, 36:809–840.
- [13] Kasyoka, P. (2014). A framework for aggregating and retrieving relevant information using tf-idf and term proximity in support of maize production. *International Journal of scientific and technology research*, 3:205–209.

- [14] Kukich, K. (1992). Techniques for automatically correcting words in text. *ACM Comput. Surv.*, 24(4):377–439.
- [15] Li, H. (2011). A short introduction to learning to rank. *IEICE Trans. Inf. Syst.*, 94-D:1854–1862.
- [16] Liu, B. (2011a). *Web Data Mining: Exploring Hyperlinks, Contents, and Usage Data*. Springer Publishing Company, Incorporated, 2nd edition.
- [17] Liu, T.-Y. (2011b). *Learning to rank for information retrieval*. Springer Science & Business Media.
- [18] Magdy, W. (2019). Text technologies for data science - ir evaluation (2).
- [19] Magdy, W. (2020). Preprocessing.
- [20] Manning, C. D., Raghavan, P., and Schütze, H. (2008). *Introduction to Information Retrieval*. Cambridge University Press, Cambridge, UK.
- [21] Martins, B. and Silva, M. (2004). Spelling correction for search engine queries. volume 3230, pages 372–383.
- [22] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.
- [23] Plachouras, V. (2009). *Term Proximity*, pages 3036–3036. Springer US, Boston, MA.
- [24] Rasolofo, Y. and Savoy, J. (2003). Term proximity scoring for keyword-based retrieval systems. In *European Conference on Information Retrieval*, pages 207–218. Springer.
- [25] Sakai, T. and Kando, N. (2008). On information retrieval metrics designed for evaluation with incomplete relevance assessments. *Information Retrieval*, 11(5):447–470.
- [26] Sanderson, M. (2010). Test collection based evaluation of information retrieval systems. *Foundations and Trends in Information Retrieval*, 4:247–375.
- [27] Sevenster, M., van Ommering, R., and Qian, Y. (2012). Algorithmic and user study of an autocompletion algorithm on a large medical vocabulary. *Journal of Biomedical Informatics*, 45(1):107 – 119.
- [28] Song, F. and Croft, W. B. (1999). A general language model for information retrieval. In *Proceedings of the eighth international conference on Information and knowledge management*, pages 316–321.
- [29] Song, R., Yu, L., Wen, J.-R., and Hon, H.-W. (2011). A proximity probabilistic model for information retrieval. *Microsoft Research*.

- 
- [30] Svore, K. M. and Burges, C. J. (2009). A machine learning approach for improved bm25 retrieval. In *Proceedings of the 18th ACM conference on Information and knowledge management*, pages 1811–1814.
  - [31] Tao, T. and Zhai, C. (2007). An exploration of proximity measures in information retrieval. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 295–302.
  - [32] Trevisiol, M. (2017). Learning to (retrieve and) rank - intuitive overview - part iii.
  - [33] Tunkelang, D. (2017). Evaluating search: Measuring searcher behavior.
  - [34] Yang, Z. and Xiaobing, Z. (2013). Automatic error detection and correction of text: The state of the art. In *2013 6th International Conference on Intelligent Networks and Intelligent Systems (ICINIS)*, pages 274–277.
  - [35] Zhang, E. and Zhang, Y. (2009). *Average Precision*, pages 192–193. Springer US, Boston, MA.

# Appendix A

## HAZEL UI

### Template Safety Data Sheet

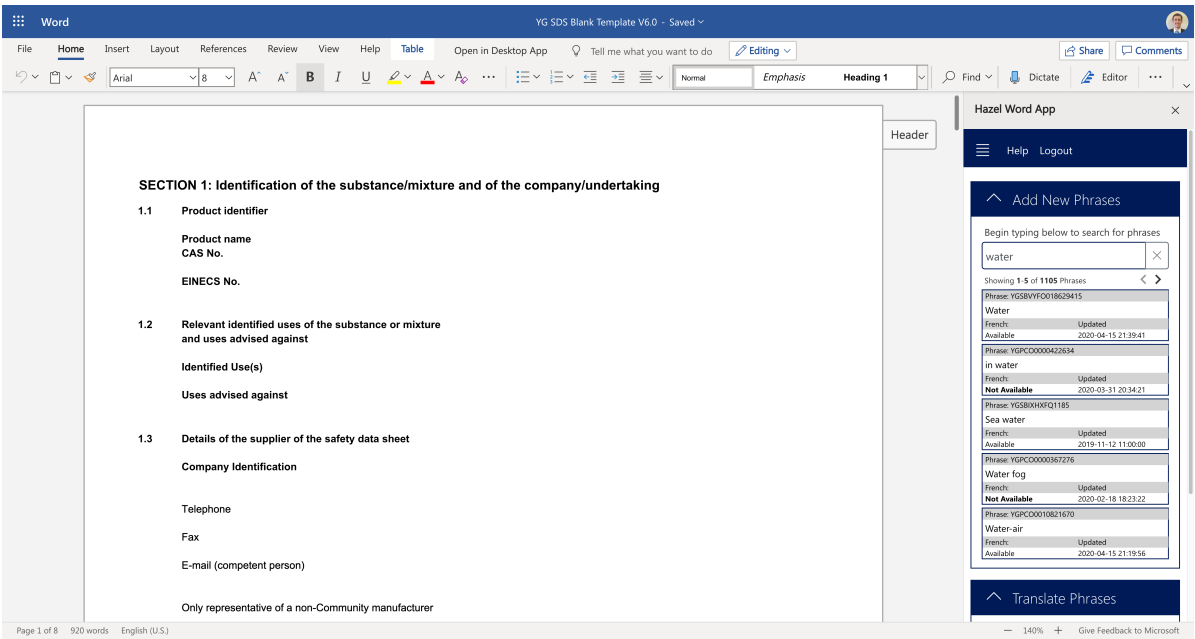


Fig. A.1 Safety Data Sheet blank template (left) with HAZEL add-in (right).

# HAZEL Phrase Search

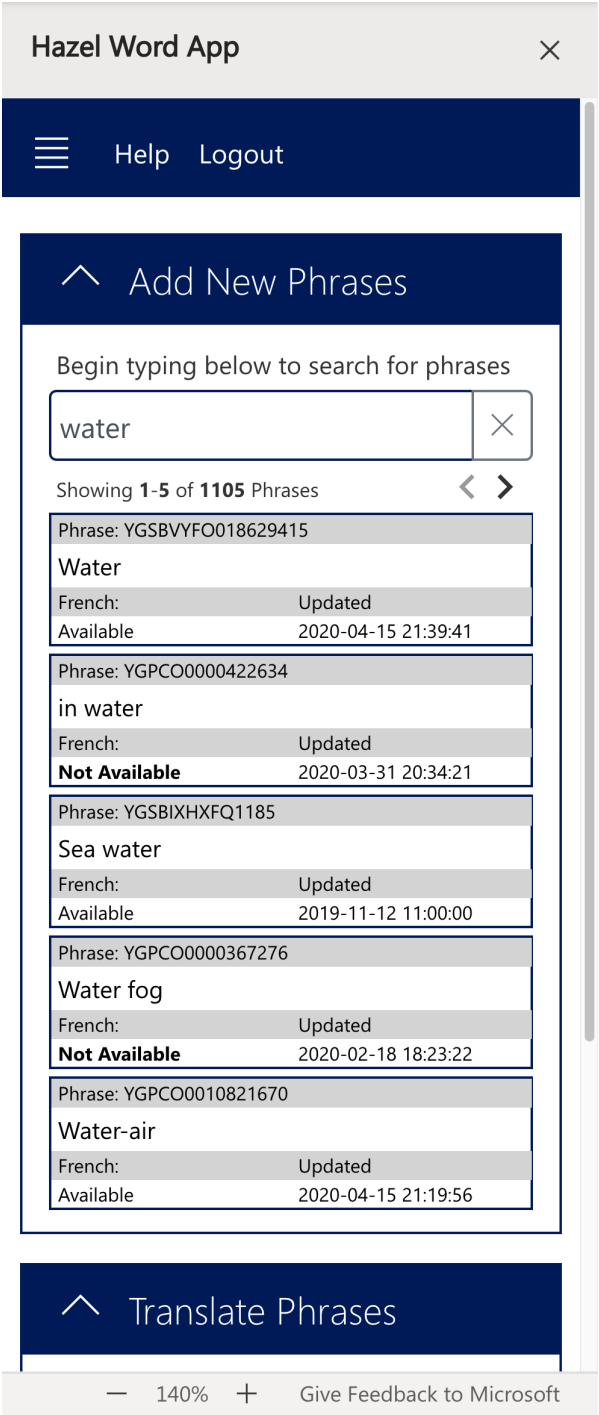
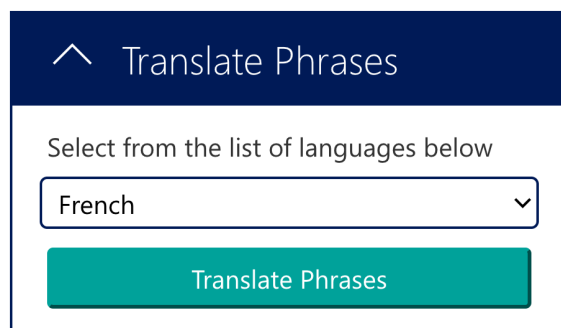


Fig. A.2 HAZEL phrase search.

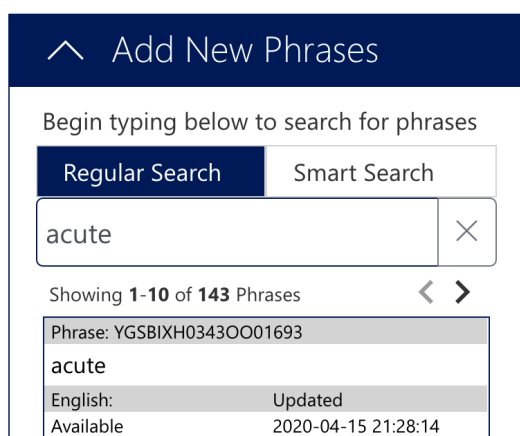
## HAZEL Phrase Translations



HAZEL Phrase Translations interface. The interface has a dark blue header with a back arrow and the title "Translate Phrases". Below the header, it says "Select from the list of languages below". There is a dropdown menu showing "French" with a downward arrow. Below the dropdown is a teal button labeled "Translate Phrases".

Fig. A.3 HAZEL phrase translations.

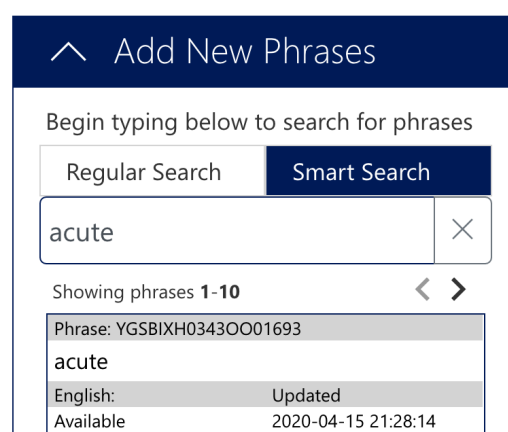
## HAZEL Search Procedures Toggle Component



HAZEL Regular Search Procedure interface. The interface has a dark blue header with a back arrow and the title "Add New Phrases". Below the header, it says "Begin typing below to search for phrases". There are two tabs: "Regular Search" (active) and "Smart Search". Below the tabs is a search input field containing "acute" with a clear button (X). Below the input field, it says "Showing 1-10 of 143 Phrases" with left and right navigation arrows. Below this is a table with the following data:

Phrase: YGSBIXH0343OO01693	
acute	
English:	Updated
Available	2020-04-15 21:28:14

(a) Regular search procedure.



HAZEL Prototype Search Procedure interface. The interface has a dark blue header with a back arrow and the title "Add New Phrases". Below the header, it says "Begin typing below to search for phrases". There are two tabs: "Regular Search" and "Smart Search" (active). Below the tabs is a search input field containing "acute" with a clear button (X). Below the input field, it says "Showing phrases 1-10" with left and right navigation arrows. Below this is a table with the following data:

Phrase: YGSBIXH0343OO01693	
acute	
English:	Updated
Available	2020-04-15 21:28:14

(b) Prototype search procedure.

Fig. A.4 Regular and prototype search procedures toggle component.



# Appendix B

## Phrase Selections

Table B.1 Phrase selection counts from phrase\_selections (as of 11th August 2020).

No.	Phrase	Selection Frequency
1	not applicable	223
2	and	178
3	Safety Data Sheet	125
4	None Known	125
5	Not yet assigned in the supply chain	123
6	CAS No.	117
7	ACCORDING TO EC-REGULATIONS 1907/2006 (REACH), 1272/2008 (CLP) & 2015/830	98
8	Not established	91
9	Unnamed publication	80
10	Date:	76
11	Mixture	75
12	H411: Toxic to aquatic life with long lasting effects.	74
13	Harmonised Classification	73
14	Vinyl Acetate	73
15	Version	68
16	Page:	64
17	No data	62
18	Danger	62
19	Page:	62
20	No data for the mixture as a whole.	60
21	to	60
22	April	59
23	at	58
24	Not applicable	58
25	CAS No.	54

# Appendix C

## Questionnaire Responses

### Feature 1: Spelling Correction

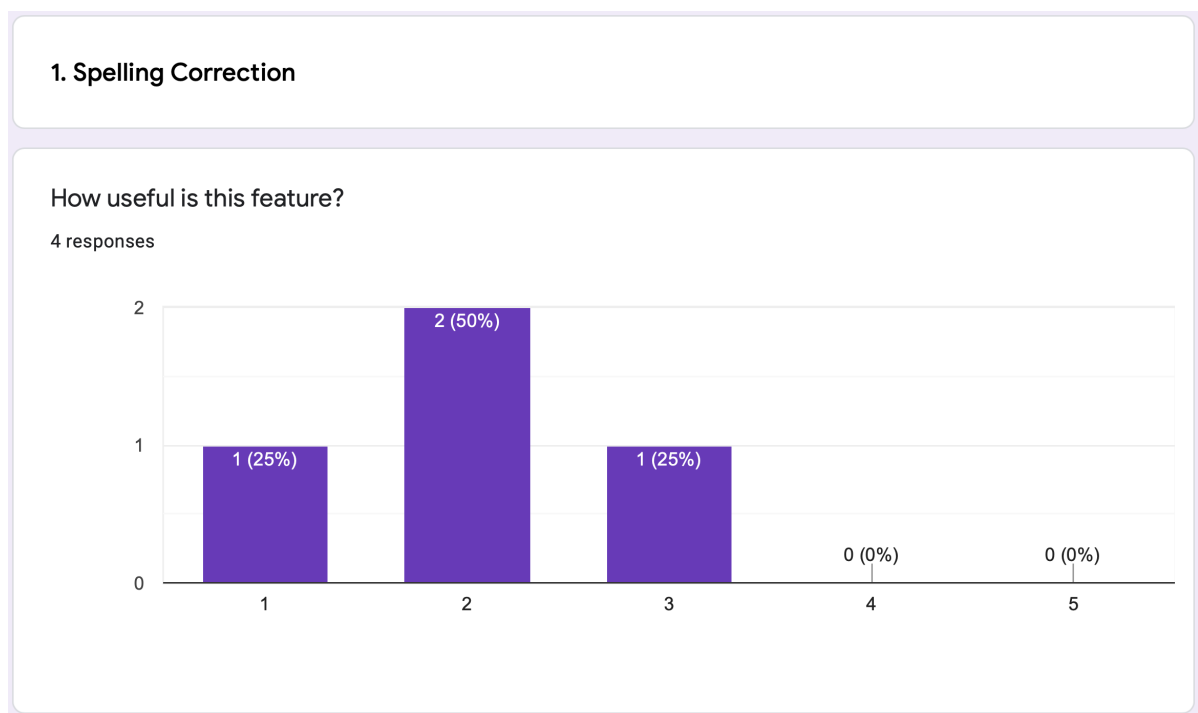


Fig. C.1 Questionnaire responses for feature 1.

## Feature 2: Multi-Word Search

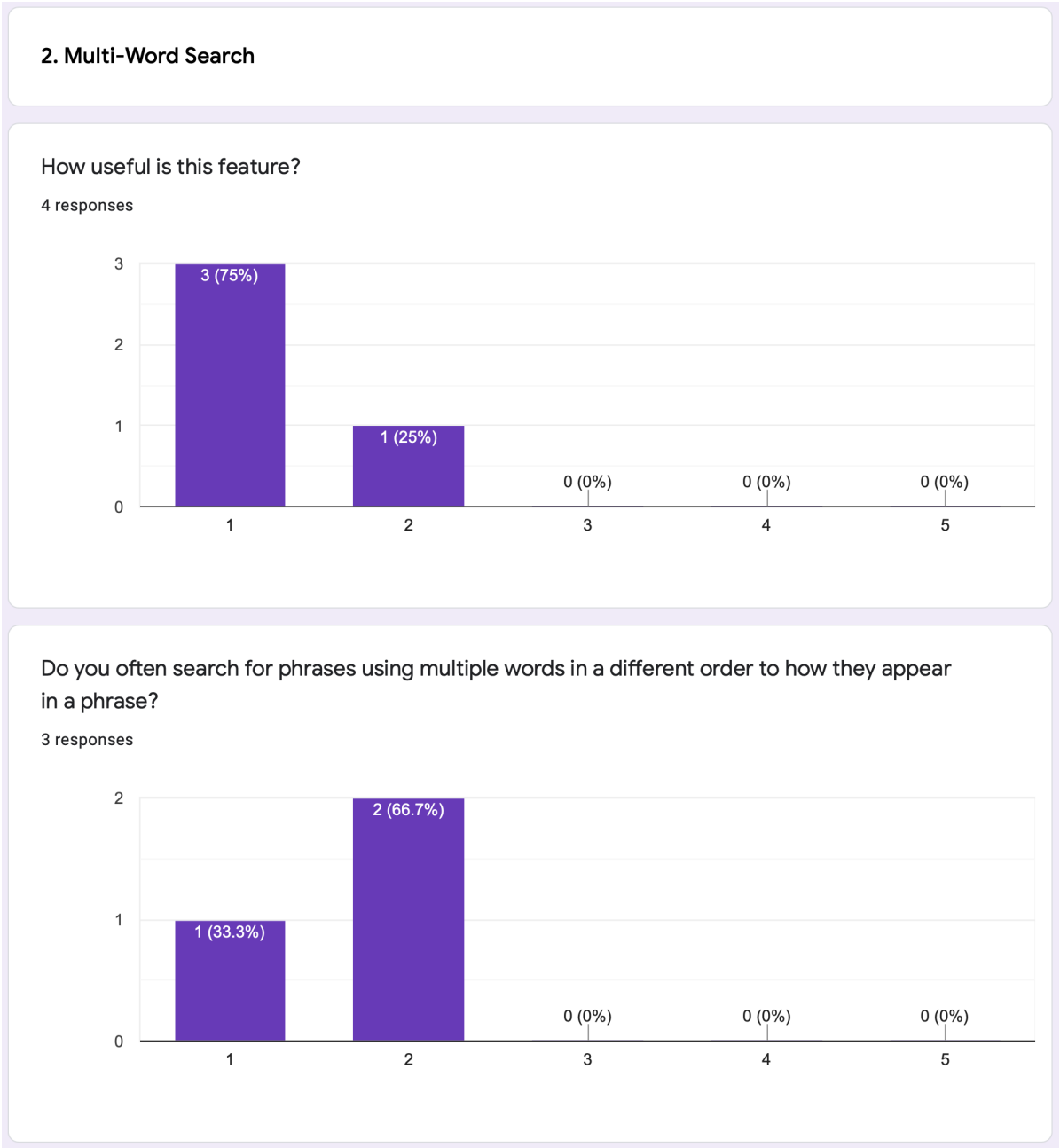


Fig. C.2 Questionnaire responses for feature 2.

## Feature 3: Phrase Autocompletion

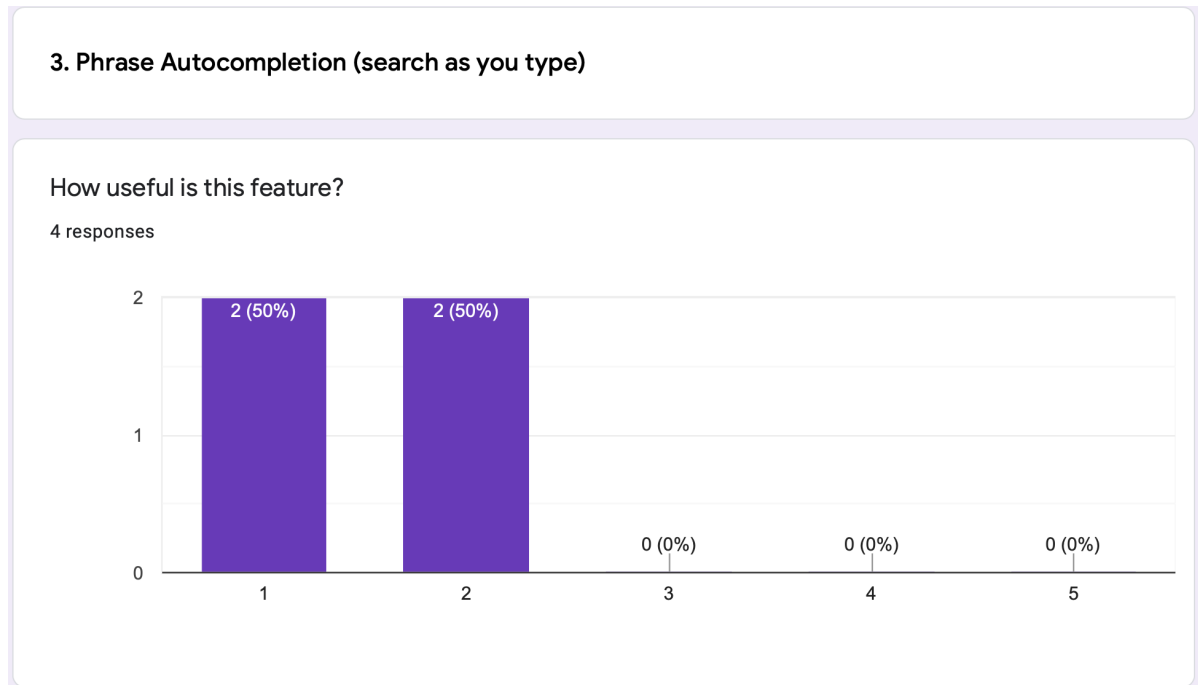


Fig. C.3 Questionnaire responses for feature 3.

# Feature 4: Multi-Word Prefix Search

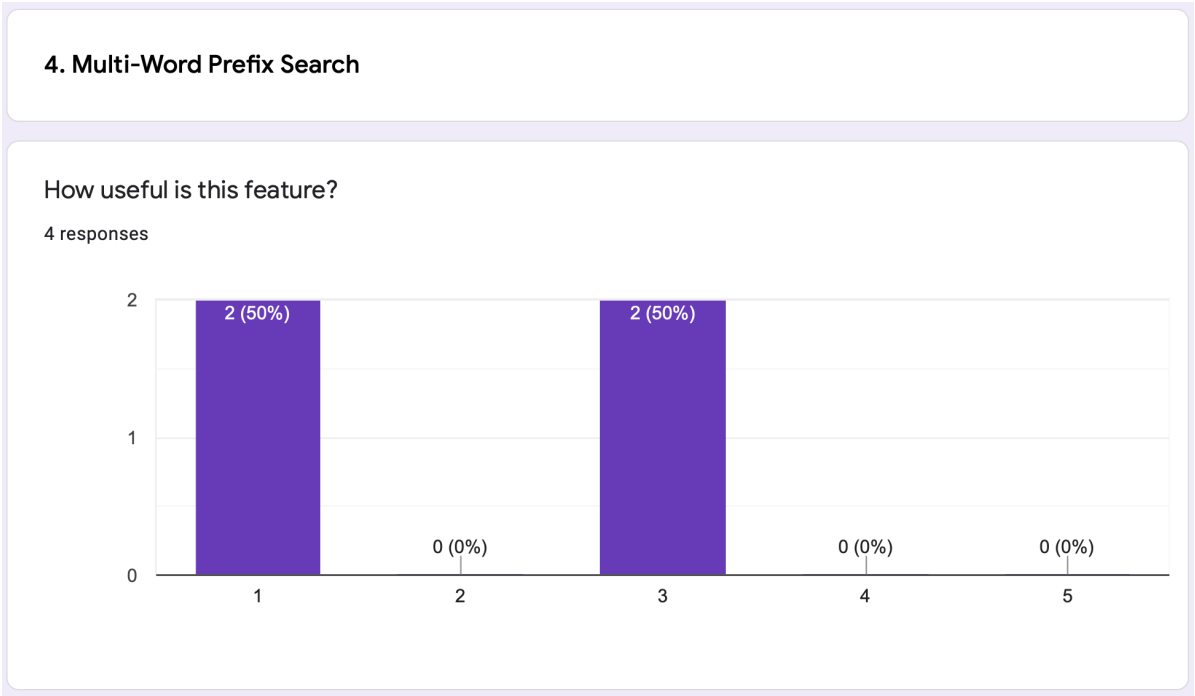


Fig. C.4 Questionnaire responses for feature 4.

## Feature 5: Self-Learning

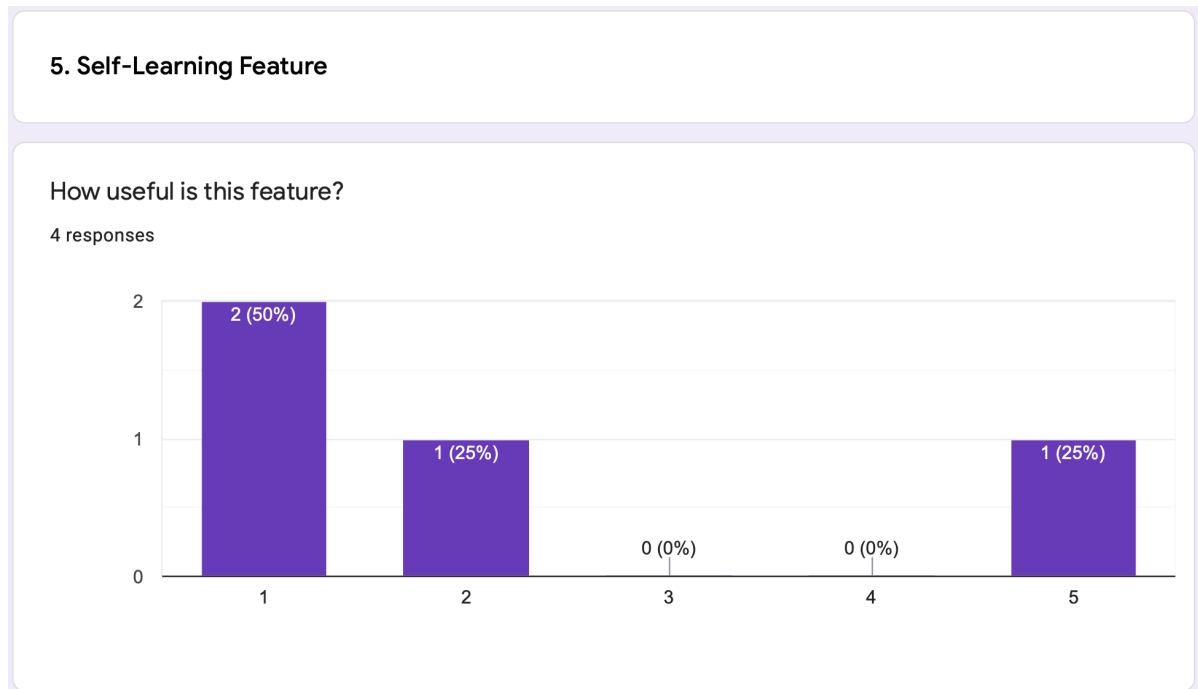


Fig. C.5 Questionnaire responses for feature 5.

## Feature 6: Multiple Search Tabs

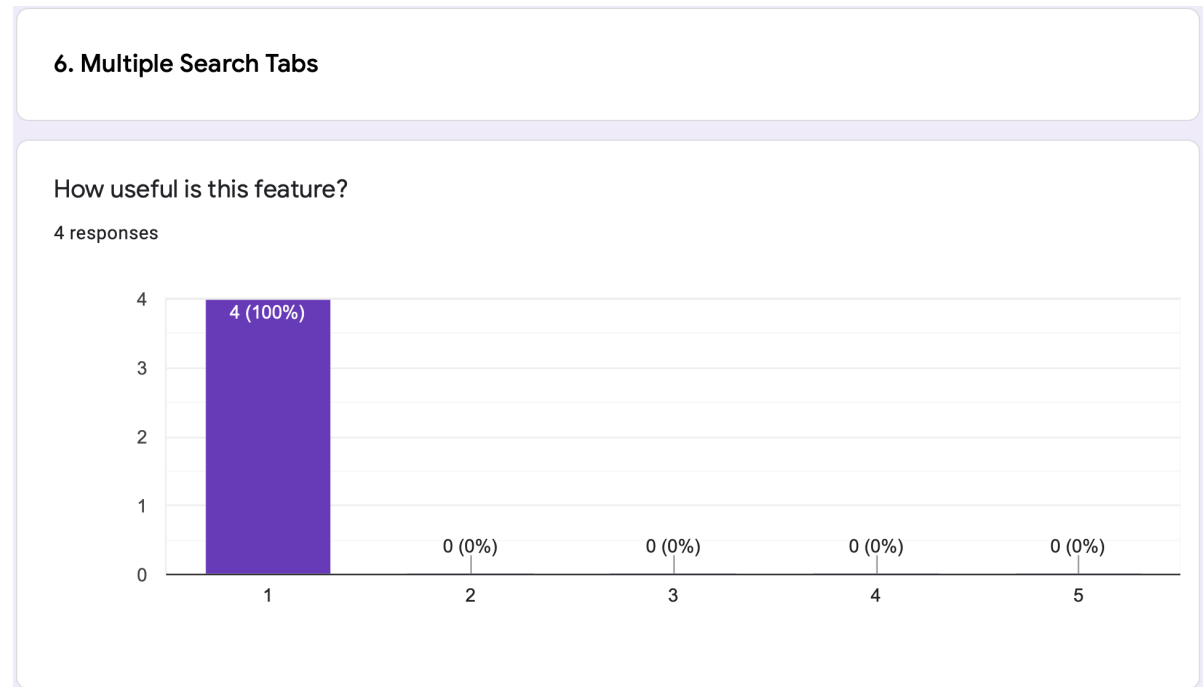


Fig. C.6 Questionnaire responses for feature 6.

# Appendix D

## Project Specification



# Project Specification

Harry Baines

Project hosted by Yordas Group, Lancaster  
MSC DATA SCIENCE, LANCASTER UNIVERSITY

08 June 2020

Industry Supervisor: Tom Hudson

Academic Supervisor: Keivan Navaie

Placement Manager: Simon Tomlinson

## 1 Introduction and Background

Yordas Group are a leading international provider of scientific, environmental, human health, global regulatory and sustainability services. Yordas' Hazard Communications team (HZC) work to author and audit safety data sheets (SDS) for a range of clients with an array of substances for which this documentation is required. An SDS is a structured document containing occupational safety and health data. Specifically, it outlines all legally required information to accompany a product such as the chemicals it contains, any hazards the chemical presents and information on handling, storage and emergency measures in case of an accident. Yordas have implemented a prototype authoring software application (HAZEL) which facilitates the entry of phrases into an SDS, with the added ability to easily convert between different languages with the click of a button. This functionality is achieved via API communication between the authoring software (accessed as a Microsoft Word Add-In) and a list of standardized phrases, represented in different languages, are stored in databases managed on Yordas' servers. Following deployment of HAZEL, data relating to user entry, phrase selection and phrase translation, have all been stored. This has resulted in a significant amount of logged user behaviours detailing which phrases users have selected and translated for a specific SDS at a certain point in time.

## 2 Motivation

The current phrase entry system implemented in HAZEL functions as intended, although is heavily reliant on matching user input with the list of phrases exactly. This becomes particularly time consuming for users of the HAZEL system (currently 6 Yordas experts use the system) entering many phrases of differing lengths, and relies on users typing full phrase strings without grammatical errors. Hence, Yordas envisage an intelligent predictive model utilising contemporary data science and machine learning methods would facilitate entry of both common and complex phrases into safety data sheets, reducing the amount of time spent completing them and enhancing business efficiency. The model should account for minor

spelling mistakes, thus ensuring suggestions are given not only based on how similar the user input is to the list of supported phrases. This project addresses the potential methods that could be employed to elevate HAZEL’s predictive capabilities, including the development of a prototype model, and a well-documented strategy given in the final report outlining different methods Yordas could use along with their relative advantages and disadvantages.

### 3 Data

Yordas store and maintain application data (i.e. phrases, phrase selections and translations, and users) for HAZEL in the following collection of relational databases using MariaDB:

- **hazel\_phrases:** data relating to phrases, such as their textual descriptions, identifiers and language data;
- **hazel\_sds\_requests:** data relating to phrases selected and translated by users, including logs of keystrokes entered by users when searching for phrases logged by timestamp;
- **hazel\_users:** a list of users of the HAZEL system. For this project, I will be working with a test account, with Tom Hudson (industry representative) using an admin account.

The previously mentioned databases are made available at the start of the project through a test account, and can be accessed through a Flask database management application. A list of standardized phrases available are characterised by unique phrase codes, with a mapping to their corresponding phrases represented in one of 43 currently supported languages. All textual entries made by users are stored via API communication between the authoring software and the `hazel_sds_requests` database. Launching a database server (with a username and password provided by Yordas) will enable SQL queries to be executed via the phpMyAdmin interface to facilitate initial data exploration, with more sophisticated queries executed using Python and SQLAlchemy during experimentation.

During initial exploration of the database schemas, I discovered the locations of user-selected phrases were not stored, although translated phrase locations were stored. Following initial discussions with Yordas, this was identified as a limitation of the data and an issue to address in further developments of HAZEL, and may ultimately impact the complexity of the developed prototype as location data for user-selected phrases cannot be considered.

Although not yet known, the choice of machine learning method may require training, validation and test sets to be generated, particularly if supervised/semi-supervised methods are explored. It is expected a significant amount of data wrangling will be necessary to format the data appropriately for use in implemented models found from the research phase.

### 4 Aims and Objectives

The primary aim of this project is to devise a strategy and implement a prototype AI model in which HAZEL will be able to intelligently predict which phrases are most likely to be selected for a given SDS based on current user input and logged user behaviours. The aims

of the project are given below, with each comprising multiple objectives required to be met in order to accomplish the aim.

**1. Devise a detailed and thorough strategy regarding relevant data science and machine learning methods Yordas can utilise to enhance HAZEL’s phrase suggestion capabilities.**

Emphasis here is placed on the creation of a comprehensive strategy detailing viable solutions Yordas can use and implement into the production version of HAZEL, which they intend to productionize in the next few months. The objectives for this aim are as follows:

1. Research and explore common machine learning methods implemented to address phrase suggestion problems.
2. Document all findings in the main report in a concise and succinct format. This will provide Yordas with a comprehensive overview of different strategies they could implement into the production version of HAZEL.

**2. Develop a prototype AI predictive phrase model based on the conducted research.**

Given the short time frame within which the project will be undertaken, only a prototype model will be developed to demonstrate proof of concept. The objectives for this aim are as follows:

1. Development of a range of phrase suggestion models found from the research phase, including a thorough evaluation of each model, and selection of a the model most suitable for deployment. Documentation relating to use of the prototype model, including steps on how to incorporate the model into HAZEL’s codebase, will also be provided.
2. Ensure privacy-related matters relating to user phrase entries are considered in the prototype solution, that is, employing novel cybersecurity techniques to ensure the use of user data is transparent and controllable by the end user.
3. Conclude model implementation by developing a testing suite to enable unit tests to be performed. This will ensure the prototype model works as expected and will help identify any potential issues to be resolved during development of the prototype. This will involve developing a set of methods using Python, similar to that already developed for the HAZEL system, to rapidly and efficiently test various aspects of the developed prototype.

We anticipate significant time will be spent iteratively implementing model prototypes alongside research into the appropriate methods. It is also anticipated the aims and objectives will evolve over the course of the 12 week placement depending on the success, progress made and challenges encountered.

## 5 Project Timeline

The phases of the project to be completed within the 12 week placement period will follow a conventional data science pipeline, starting with problem formulation and requirements gathering, data exploration and processing, implementation of a prototype solution, and concluding with a results analysis and evaluation of the developed prototype and an assessment of the aims and objectives. Due to the inherent nature of the project, we envisage significant time (i.e. the first 5-6 weeks) will be spent in the research and experimentation phases for this project; as previously stated, emphasis is not placed on the development of a production-ready solution per-se, instead a thorough analysis of appropriate traditional and contemporary methods applicable for use in the HAZEL system will be documented in the main report produced by the time the project concludes, in addition to a prototype solution. A summary is given below:

- Weeks 1-6: Research and experimentation of appropriate predictive models to aid phrase suggestion through HAZEL
- Weeks 6-9: Prototype Model Implementation
- Weeks 9-11: Prototype Model Testing and Evaluation
- Week 12: Project Conclusion

During the course of the 12 weeks, the final report will be written concurrently alongside research and the implementation of the prototype model.

## 6 Deliverables

Due to the complexity of the work and the short time frame within which the work will be undertaken, it will not be necessary to produce a production-ready solution, rather a prototype AI model will be developed as a proof of concept, alongside a thorough and well-documented strategy detailing a range of primitive and more sophisticated solutions found during the research phase, which will inform Yordas of how to implement modern data science methods into HAZEL. Presenting a summary of methods found alongside their advantages and disadvantages in the main report will allow Yordas to make an effective business decision of which method to pursue before being productionised upon completion of the project. Additional documentation providing the technical procedures required to incorporate the developed prototype into HAZEL's codebase, as well as code-specific details of the project structure will be provided as markdown files in a GitHub repository in addition to a GitHub Pages website.