Identifying Inspiration between Book and Film Dialogue

Heloïse Kverndal

4th Year Project Report
Artificial Intelligence and Computer Science
School of Informatics
University of Edinburgh

2023

Abstract

This project introduces the concept of *inspiration*. This is when two texts are similar enough in either phrasing or meaning that if one came first, we could determine that the second had been written based on the first. Our goal is to develop a system that can identify this idea of inspiration between film and book dialogue in films that are based on a book and therefore may display some inspiration. We aim to do this to both provide a tool for film analysis for students, providing insight into the process of creating a film based on a book, as well as contribute to the field of Natural Language Processing by further expanding and developing the practice of similarity measurement. We evaluate this system by testing on J.R.R. Tolkien's *The Lord of the Rings* trilogy, developing the system initially on dialogue from the first book *The Fellowship of the Ring* and subtitles from the film of the same name, and then testing on the held-out third book/film *The Return of the King*. We finally test an automated version of the system, using Automatic Speech Recognition techniques to generate the film dialogue directly from the film audio.

Research Ethics Approval

This project was planned in accordance with the Informatics Research Ethics policy. It did not involve any aspects that required approval from the Informatics Research Ethics Committee.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Heloïse Kverndal*)

Acknowledgements

I would like to thank my project supervisor, Dr. Peter Bell, for supporting me throughout this project, for giving me invaluable advice and for always providing me extensive feedback on quick turnaround even up to the last day.

I would also like to thank my friends and family for their love and support and in particular my fellow computer scientist flatmates for their endless encouragement and reminders to see the sun once in while.

Table of Contents

Chapter 1

Introduction

Adaptation and inspiration have been a part of storytelling since humans first began to pass on stories via word of mouth, with each new storyteller naturally rephrasing and reimagining stories as they are passed down from person to person [1]. Similarly, this process has woven its way into literature, with writers being inspired by the stories they have heard or read and putting them into their own writing [2]. Even the process of multi-media adaption can be traced back hundreds of years, with the adaption of stories into books, plays, parodies, illustrations, and more [3]. And along with these adaptations, come the criticisms that the new version does do justice or does not follow faithfully enough to the original content, even as far back as 1796 [2]. With the dawn of film in the 20th century this concept has further extended into film [4] with the adaptation of books into films now seen as a popular practice in the entertainment industry. Though often, as in the 1700s, these films do not stay entirely faithful to the book content, there are many, particularly classic, films that are highly inspired or draw strongly from their book source material.

This project attempts to automatically identify the moments in a text that are directly "inspired" by another source. Specifically focusing on the example of a film adaptation of a book and identifying the dialogue in the film that has been "inspired" by the book. Humans can easily use their own intuitions to determine whether a piece of text is based on another, but this is a laborious and time-consuming task, that in large texts becomes unrealistic. Even between a single book and film, this would take an extensive amount of time and most likely a lot of prior knowledge about both the book and the film. In this project, we seek to offer a way to quantify this idea of *inspiration*, so we can apply these human intuitions automatically. We define two texts as being *inspired* when two texts are similar enough in either phrasing or meaning that if one came first, we could determine that the second had been written based on the first. For example, if we look at the phrases:

```
Phrase 1: "It's a dangerous business, Frodo, going out of your door,"
he used to say.
Phrase 2: "Remember what Bilbo used to say: It's a dangerous
business, Frodo. Going out your door."
```

We can determine that if Phrase 1 was written first, Phrase 2 has been *inspired* by Phrase

1. Humans can do this with relative ease, but how do we quantify this automatically? We can identify parts of the sentence that are identical, but even in this example, only five words align exactly - *"It's a dangerous business, Frodo"*. The sentence is rephrased - *"going out of your door"* to *"going out your door"* - and reworded with implied context - *"he used to say"* and *"Remember what Bilbo used to say"* - causing it to be harder to quantify algorithmically. A sentence can be reworded and structured differently, but still have the same semantic meaning, but at what level does this cause it to no longer be classified as inspired? Contextual information also influences our decision, if a relatively common sentence, for example, *"All right!"*, appears in two texts in entirely different contexts, we cannot really say it has been inspired, but if it appears in the same contexts, then we are much more likely to determine it as inspired. How can we identify all these different similarities and then decide how similar is similar enough to have been inspired? Then if we have a way to identify these matches, how can we find them in two large documents of text of different sizes, that may or may not contain any matches?

If we can provide a tool to do this, not only would this give critics a tool to accurately measure faithfulness to the source material, but it also offers a way to quantify this idea of *inspiration*, which to our knowledge has previously not been researched in the field of Natural Language Processing (NLP). NLP is an area of research that essentially enables machines to "understand" human language. One of the significant challenges in NLP is to measure the degree of similarity between two or more pieces of text, which arises in several applications, including information retrieval, which requires similarity in order to find the most relevant document to a query, text summarization, which requires similarity to help rewrite text into a summary while retaining the same meaning, and plagiarism detection, which is most similar to our task, requiring similarity to detect when text has been copied from an alternate source. There are also many benefits in wider computer science research by producing an alignment of book text and film video, we can extract by finding cases of inspiration. This is because it provides a link between audio-visual data and a textual description of the same scene, as subtitles are synchronized to the film via timestamping, which can be very useful in providing machine learning algorithms examples of the same content in different mediums. It also has numerous applications in the film and publishing industries, as well as for critics, including audio description, and content adaptation, as well as providing tools for film analysis for students, as it provides insight into the process of creating a film based on a book, or as we will refer to it: screenwriting.

1.1 Contributions

In this project I hope to contribute to the existing NLP research on similarity by developing a system using a combination of previously developed methods for similarity comparison, as well as novel ones to quantify our introduced idea of *inspiration*; in which we can strongly predict that a target sentence has been inspired by a query sentence, producing a mapping between inspired dialogue in a film and its corresponding book text. We are also looking to create a lightweight system that can generalize to new books/films, and is not weighed down by methods with long training or searching times so that we can efficiently compare every sentence in two large documents - the film subtitles and book dialogue.

1.2 Structure

In this dissertation I will present an automated system for identifying inspired sentences between film dialogue, drawn from film subtitles, to their corresponding dialogue in the book text, leveraging recent advances in Natural Language Processing (NLP) and machine learning. The proposed system uses deep neural networks, specifically transformers, for semantic comparison [5], and established methods in text comparison developed through years of research [6][7], as well as novel methods developed in this project. The background of previous research done in this area and the strategies that have been previously used are detailed in Chapter 2. The details of these methodologies will be covered in Chapter 3, while the implementation of the system itself will be detailed in Chapter 4.

The system has been developed and initially evaluated on a single book/film, specifically J.R.R. Tolkien's *The Fellowship of the Ring* from the trilogy *The Lord of the Rings*, as this is considered a good example where many matches occur and has ground-truth labels for evaluation. We have also evaluated it on a second, held-out book in the trilogy, *The Return of the King*, to test how well the system generalizes. This achieved promising results in all areas, demonstrating the success of the proposed approach. The details of these evaluations are found in Chapter 5. As an extension of the project, we also experiment with extracting the dialogue directly from the film's audio, detailed in Section 5.4. Rather than simply from the already transcribed subtitles. This is to demonstrate the concept of a fully automatic pipeline of the system, not requiring a subtitle file, which may be unavailable for less popular films, as well as testing the system's ability when the provided data contains errors - which an automatic speech recognition transcription will naturally have. These experiments also produced promising results, demonstrating the feasibility and potential of this addition.

The overall success of the system and evaluation of the performance of the combination of experiments, as well as possible ways this project could be taken forward is then discussed in Chapter 6.

Chapter 2

Background

As we have established the goal of this project is to automatically capture inspiration between the dialogue in a book with its corresponding dialogue in a film adaptation. Due to the specific nature of this task, it falls at an intersection between many disciplines, including text alignment, semantic matching and information retrieval, as we are both trying to search through large documents of text, compare their sentence level similarity and softly align them. And as such, we will need to use a combination of previously developed techniques to solve this problem. Despite the fact that there is a vast wealth of literature on related topics, our specific topic is not highly explored. As such I will provide a background of existing techniques for similarity measuring and information retrieval to give a general idea of the field, before outlining previous research that has been done that follows a similar task to our own.

2.1 General Background

2.1.1 Similarity Measures

There are various and extensive techniques for similarity comparison, so we will give a broad overview and describe some of the most commonly used measures. These measures can be divided into three broad categories:

- String based Similarity
- Edit-distance based Similarity
- Semantic Similarity

String-based measures are based on comparing the character strings of two words or phrases. The simplest string-based measure is exact matching, where two strings are considered similar only if they are identical. Other string-based measures include Jaccard similarity and cosine similarity. Jaccard similarity measures the similarity between two sets by comparing their intersection and union, while cosine similarity measures the angle between two vectors representing the occurrence of words in a document. While both these methods could be useful in our task as they are not affected by the length of sentences, Jaccard similarity is unlikely to be useful to us, as it ignores both ordering and frequency of words in a sentence, which are more important when measuring inspiration. It is also possible that there are only a few similar words between our inspired sentences, so we want to utilise as many other similarities as possible, including ordering and frequency. Cosine similarity however is a measure we want to use if we can find informative vector representations of our sentences.

Edit-distance based measures are based on calculating the minimum number of operations (insertions, deletions, or substitutions) needed to transform one word or phrase into another. They include Levenshtein distance [8], which is the simplest form based on just the number of insertions, deletions or substitutions, Damerau-Levenshtein distance [9], and Jaro distance [10]. Damerau-Levenshtein distance is similar to Levenshtein distance, but it also allows for transpositions (swapping adjacent characters or words). The Jaro distance takes into account the number of matching characters between two strings as well as the number of transpositions needed to make the strings match exactly. Edit-distance based measures however are unlikely to be useful in measuring inspiration as they are better at measuring very similar sentences and we do not always expect inspired to use exactly the same wording and ordering.

Both string and edit-distance based simple metrics tend to be very well tested as they have been present in research for decades, and present tried and true methods [11].

Semantic measures are based on comparing the semantic similarity of words, aiming to capture the similarity between different words that have similar meanings. The most recent advancement in this area has come in the form of word embeddings based on deep learning models, the most prevalent of these being BERT [12] and ELMo [13]. Semantic word embeddings are a type of NLP technique used to represent words in vector space, where each dimension of the vector corresponds to a different feature of the word's meaning. These embeddings are created by analyzing large amounts of text data using state-of-the-art neural network algorithms.

Semantic word embeddings have several advantages over traditional methods for representing words in language processing tasks. Firstly, they can capture subtle nuances in word meaning and associations between words, closer to the human concept of similarity. Secondly, they are more computationally efficient than traditional methods because they can be used to represent large vocabularies with relatively small vector sizes. Finally, they can be trained on large amounts of data and can be reused across multiple language processing tasks. And, as they are essentially still vectors, we can use traditional methods to compare, with cosine similarity being the most commonly used.

On assessing these methods, we can see that our goal comes somewhere between string-based similarity and semantic similarity, as edit-distance based similarity, as we have previously mentioned, is more useful in exact alignment, so we will need to use a combination of both techniques to accurately capture inspiration.

2.1.2 Information Retrieval

Despite many methods being available to compare the similarity of sentences, to be able to capture inspired sentences between two texts, we must find them. To do this efficiently, we must draw on the field of Information Retrieval (IR). IR is the process of retrieving relevant information from a collection of documents, given a query or a user's information need. One of the key tasks in IR is efficiently computing similarity comparison in order to find relevant documents. This task has been studied extensively in the field of IR, and there are many techniques that have been developed for similarity comparison over large collections of documents.

One of the earliest techniques for similarity comparison was the vector space model (VSM) [14], which represents documents as vectors in a high-dimensional space, where each dimension corresponds to a term in the document. Documents are compared based on the cosine similarity between their corresponding vectors. The VSM has been widely used in IR and has been shown to be effective for many types of queries and documents.

Another popular technique for similarity comparison is latent semantic analysis (LSA) [15], which is based on the idea that there are underlying latent semantic dimensions that capture the meaning of words and documents. LSA uses singular value decomposition (SVD) to identify these latent dimensions and represents documents as vectors in this reduced dimensional space. Similarity is then computed based on the cosine similarity between the vectors in the reduced space. However, this is a method better suited to broad matching over a huge database of documents, with its benefits coming in the form of robustness to large data and conceptual similarity, rather than finding the exact matches we are trying to extract in our project.

More recently, deep learning techniques have been applied to similarity comparisons in IR, such as using convolutional neural networks (CNNs) [16] and recurrent neural networks (RNNs) [17] to learn representations of documents and queries. These techniques have shown promising results in various IR tasks, including document ranking and question answering. However, these may be suitable for our system as we are looking to keep it lightweight and avoid large models.

2.2 Related Research

Now we have discussed the general research base in this area, we will now look at previous research done that is most similar to capturing inspiration between book and film dialogue. The most similar of these was performed by Zhu et. al [18] and Tapaswi et. al [19], who in their papers developed systems to match the scenes of books and films. We can see that this aim is slightly different, as they were attempting to align visual content from films to their textual description in their corresponding book. As such the focus is more on the visual/text matching, matching not only the dialogue but the context, how what is described in the book can be matched to what appears on the screen. However, we can still take inspiration from their methodology.

One similarity to our own task is that both of these papers also use dialogue matching to align scenes, using the subtitles contained in the films already, as they provide a way to align the audio-visual data with the book dialogue, as well as provide an accurate transcription of the dialogue. This task is very similar to our own, except that we are looking for broader or partial matches, that can still be described as inspired despite being quite different, rather than the more directly aligned dialogue that these papers are looking for. However, we can still adapt some of the similarity/matching techniques used in these papers, as they have a similar purpose. Both papers use the common information retrieval technique of TF-IDF, which we also decide to utilize in our project and is described in full in Section 3.1, enabling fast matching, while still finding near-match occurrences, though it is important to note that the drawback of this method is that it cannot capture semantic similarity. Zhu et. al combats this by also using semantic word embeddings learned from the book, in order to find semantically similar matches, where the dialogue may have been paraphrased or reworded slightly. Though this may partially capture semantic similarity, semantic embeddings are most powerful when trained on very large datasets, larger than a single book. Another potential issue with this approach is that training, encoding and calculating the similarity between word embeddings can be slow and resource intensive, so it may be better to use more efficient methods. Zhu et. al also uses the BLEU similarity measure [7], as a fast evaluation method to find very similar matches. This method is usually used in the assessment of machine translations, but can also be used to assess similarity. The main rationale behind this measure is to use a weighted average of variable length phrase matches, by modified n-gram precision and comparing sentence length, against alternative translations, or possible matching sentences in this case. This scored very highly when compared to human judgements on the same translations, making it a strong candidate to use in our project, which we do indeed go on to experiment with and is described in full in Sections 3.2, 4.4 and 4.5.4. On top of these measures, Zhu et. al also further extends their approach, by using a deep convolutional neural network (CNN) to measure the scene contextual similarity around the dialogue to ensure accurate matching, rather than simply that the same strings are said. For example, they saw that "I love you" appeared many times, but obviously each occurrence may match a different occurrence in the text, so using the visual context of the scene aids in determining which occurrence it matches. However, as we are not developing a visual element in our solution, though this would aid in finding accurate matches, this extension is beyond the scope of this task. Instead, we may want to use other methods of ensuring matches are relevant using the surrounding textual context and comparing for similarity instead. This concept is the inspiration for the methodology described in Section 3.4 and implemented in Section 4.5.

As we can see there are many strategies we can take from these papers in finding and measuring the similarity of dialogue in a book/film. However, we also hope to advance these techniques to be more sophisticated, as although both these papers use advanced vision and multimedia alignment techniques, the textual matching is still mostly rudimentary. It is also more important in our task that we do not get false matches, as we are specifically trying to capture when text has been inspired by another text.

Chapter 3

Methodology

Now that we have established the context surrounding this project, including its motivation and the relevant research, we can turn our attention to the methodologies we employ to solve our task. In this chapter, we will explore the various techniques and tools that were used to achieve our research objectives.

The problems that we have to face in our project include:

- Finding matching sentences in two large documents of text, where matches may be sparse or non-existent
- Finding matching sentences where the syntactic structure has changed
- Finding matching sentences where words have been replaced by semantically similar ones
- Finding matching sentences supported by the surrounding context
- Finding matching sentences with long identical subphrases

The methodologies we can use to address these issues include ranked retrieval using TF-IDF [20], TF-IDF cosine similarity [14], SBERT cosine similarity [5], Bleu similarity [7], and context. Each of these methods plays a critical role in achieving the project's goals.

Ranked retrieval is a popular information retrieval technique that involves ranking documents based on their relevance to a query. In this project, we employed a variation of this technique that utilizes Term Frequency Inverse Document Frequency (TF-IDF) to calculate relevance. This technique involves computing a score that reflects the similarity between a query and a document, and then ranking documents based on this score. We use this to solve the problem of finding matches within two large documents, using each sentence in the smaller document as a query to find its possible matches in the larger document. This allows us to narrow down our matches before applying more fine-grained similarity metrics. The full details of this methodology are defined in Section 3.1.1, and its implementation into our system is described in Section 4.3.

TF-IDF is also applied as a similarity metric, using the TF-IDF score to represent sentences in vector space. These vectors can then be compared for similarity using cosine similarity. This methodology in our project targets the basic matching of sentences, capturing when two sentences share similar words, where words are scaled based on their rarity. The details of this are described in Section 3.1.2, its implementation is described in Section 4.5.1.

Sentence-BERT (SBERT) is another method used to represent sentences in vector space. SBERT is based on the well-known transformer architecture for semantic embedding Bidirectional Encoder Representations from Transformers (BERT). SBERT is used to encode the semantic meaning of a sentence as a vector, which can then, similar to TF-IDF vectors, be compared using cosine similarity. We use this method to solve the problems of finding matches where the syntactic structure has changed and finding matches where words have been replaced by semantically similar ones, as SBERT allows us to compare the underlying semantic meaning of sentences. We describe the full methodology of this process in Section 3.3 and demonstrate its implementation into our system in Section 4.5.3.

The Bilingual Evaluation Understudy (BLEU) similarity is a metric commonly used in NLP to measure the similarity between two texts. Though this technique is designed for evaluating the quality of machine translation, it can also be useful as a simple similarity measure and is one of the methods used previously in aligning books and movies [18]. It is used in our project as a fast metric to calculate similarity and as an attempt to solve the problem of finding matches with identical subphrases. The full description of this methodology can be found in Section 3.2, and its implementation into our system in Sections 4.4.1 and 4.5.4.

Finally, we employ a method novel to this paper as a strategy to deal with the problem of context influencing similarity. This attempts to address the intuition that matches are more likely when the surrounding context is the same. The method involves combining the calculated similarity between sentences with that of the similarity between the corresponding surrounding sentences. The full details of this method are described in Section 3.4, and its subsequent implementation is described in Section 4.5.2.

In the following sections, we will delve deeper into each of these methodologies, explaining what they entail and how they can be applied to identify *inspiration*. We will also discuss the advantages and limitations of each method, and how they can be adapted for use in other projects.

3.1 TF-IDF

3.1.1 TF-IDF Ranked Retrieval

The first approach we will look at in this project is Ranked Retrieval [6]. This is a common technique used in Information Retrieval when dealing with large corpora of text such as ours. This method is often used in search engines and other applications where the goal is to identify documents that are most relevant to a given query.

In this specific version of ranked retrieval, we utilise the TF-IDF metric [20] to rank the documents - or possible sentences. The theory behind TF-IDF is that some words will have greater relevance than others. TF-IDF gives us one way to reflect this, assigning a weight to each term in a document that reflects how important that term is to the document relative to the entire corpus. The weight is based on two factors: the term frequency (TF), which measures how often the term appears in the document, and the inverse document frequency (IDF), which measures how rare the term is across the corpus.

To compute the TF-IDF score for a term in a given document, we multiply the term frequency in that document by the inverse document frequency across the entire corpus. The resulting score gives us a measure of how important the term is to the document in the context of the entire corpus.

Given a query(q) tokenized into terms(t) we can use TF-IDF ranking to identify the documents that are most similar to the given query sentence. To do this, we first calculate the TF-IDF scores for each of the query terms(t) using the equation:

$$w[t][d] = (1 + \log_{10} tf(t,d)\ )\log_{10}(\tfrac{N}{df(t)})$$

Where:

- $tf(t,d)$ is the frequency of term t in document d (the number of times t occurs in d)
- $df(t)$ is the number of documents in the entire collection that contain term t
- N is the total number of documents in the collection

Using the logarithm of N over the DF allows us to calculate the inverse document, therefore creating high values when a term is rare and low when a term is common. We also utilise a variation of TF-IDF scoring called sublinear scaling. This is where we take one plus the logarithm of the TF (which is set to 0 if TF is 0 to avoid a mathematical error), to dampen the effect of very frequent words, as we do not expect high occurrences of a term in a document carry the same weight as a single occurrence, particularly in the case of text with hyperspecific language like that of a book or film with low occurrence tokens like names and places.

Once each of the scores for each query token has been calculated, these tokens are summed for each document containing them. Giving each document a score in relation to the query which we can use to rank them:

$$Score(q,d) = \textstyle\sum_{t \in q \cap d} w[t][d]$$

It is also important to normalize these scores by sentence length to ensure that long sentences do not receive unfairly high scores, particularly when we are matching sentences to sentences, which may be of varying lengths.

Overall, TF-IDF ranking is a highly useful technique for information retrieval that can be used to quickly identify relevant documents or sentences within a large corpus of text. It is particularly useful in narrowing down a set of search results and is often used as an initial step in Information Retrieval before using more sophisticated, more computationally expensive techniques for refinement [21]. And despite the emergence of newer techniques, it is still widely used in industry and academia and is a valuable tool for any researcher working with textual data, and, though it has its limitations remains the most popular technique for ranking documents in information retrieval.

3.1.2 TF-IDF Vectors

This concept of the TF-IDF can be extended further to represent sentences in vector form [14]. This becomes very powerful as it allows us to represent sentences in vector space and therefore perform vector operations, such as using the cosine similarity to compare vectors.

To represent sentences in the corpus as vectors, we must first create a bag-of-words model of all the unique terms in the corpus. A bag-of-words model is a simple method of representing text data by creating a vector system from the corpus vocabulary. A sentence can be represented as a collection or ”bag” of the words in the vocabulary, disregarding grammar and word order, but preserving their frequency of occurrence, where each word is represented by its frequency count. For example, if we have a sentence and a vocabulary of length n, we would create a vector of length n, where each element corresponds to a word in the vocabulary, and the value of each element corresponds to the frequency count of each word in the sentence. To adapt this model to use TF-IDF, we simply replace the frequency count with the word’s TF-IDF score.

As previously mentioned, we can then compare these vectors using cosine similarity. The cosine similarity is a measure of the similarity between two non-zero vectors of an inner product space that measures the cosine of the angle between them. Specifically, the cosine similarity between two vectors d_1 and d_2 is defined as:

$$cosine\_similarity(d_1, d_2) = \frac{(\vec{V}(d_1) \cdot \vec{V}(d_2))}{|\vec{V}(d_1)||\vec{V}(d_2)|}$$

Where the numerator represents the dot product (also known as the inner product) of the vectors V (d_1) and V (d_2), which are the vector representation of the documents d_1 and d_2. While the denominator is the product of their Euclidean lengths.

This has been shown many times [22][6] to be a good method of comparing TF-IDF vectors as it ignores differences in document length and word frequency, which can be important for measuring similarity in high-dimensional and sparse vector spaces, which TF-IDF vectors create as many of the entries in each vector will be zero.

Another extension of TF-IDF is to use n-grams as well as words. N-grams are a technique used to represent text as a sequence of overlapping groups of n consecutive words and aid in identifying combinations of words appearing together. This can be particularly helpful for finding repeated key phrases. For example, the sentence “One ring to rule them all” can be represented as:

- Unigrams (1-grams): `"One", "ring", "to", "rule", "them", "all"`
- Bigrams (2-grams): `"One ring", "ring to", "to rule", "rule them", "them all"`

- Trigrams (3-grams): `"One ring to"`, `"ring to rule"`, `"to rule them"`, `"rule them all"`

To incorporate these into vectorization, we simply include each n-gram as a unique term in the vocabulary. We then compute the TF-IDF weight for each n-gram in the sentence, using the same formula as we do for unigrams - or words. Finally, we combine the TF-IDF weights of all n-grams in the sentence to create a single vector representation of the sentence.

If using a bigram system, we can see that this could be helpful in capturing the phrase “one ring”, a commonly used phrase throughout the Lord of The Rings stories. And by utilizing n-grams we can identify its appearance, and calculate its corresponding rarity.

Using n-grams in this way, not only helps with common phrases, but can also help to capture local context information that may be lost when using only unigrams, and can lead to better performance in tasks such as text classification and information retrieval. However, it also increases the dimensionality of the vector space and can be computationally expensive, where the higher the value of n, the more expensive. The choice of n-gram size is typically determined by the specific task and dataset and may require experimentation to find the optimal value.

Although TF-IDF can be a very powerful tool, it cannot capture semantic similarity. If semantically similar words are used, they are completely separate tokens in the TF-IDF calculation, so we therefore cannot use it to identify when a sentence has been rephrased. However, if only word ordering or syntactic structure is altered between similar sentences, TF-IDF cosine similarity will be able to capture it, as TF-IDF vectors are based on the bag-of-words model, which disregards grammar and word order.

3.2 BLEU

Another method that we use to measure similarity is BLEU. BLEU is a metric designed to evaluate the quality of machine-generated translations by comparing them to human translations. However, it can also be used to compare the similarity between two strings of text.

Similar to TF-IDF, BLEU utilizes n-grams, however in BLEU n-grams are used more directly. The basic idea behind BLEU is to compare n-grams in one string of text with the n-grams in another. The BLEU score is then a measure of how many n-grams in the candidate text overlap with the n-grams in the reference text.

To calculate the BLEU score, the number of times each n-gram occurs is counted in the candidate and reference texts. The precision of the candidate text is then calculated by dividing the sum of the counts of matching n-grams by the total number of n-grams in the candidate text. However, precision alone is not enough to evaluate the quality of the candidate text, as it does not take into account the length of the candidate text. To address this issue, BLEU also calculates the brevity penalty, which penalizes short candidate texts that match the reference text by reducing the BLEU score. The brevity penalty is calculated by taking the ratio of the length of the candidate text to the length of the reference text. If the candidate text is shorter than the reference text, the penalty will be greater than 1, and the BLEU score will be reduced accordingly. The precision and brevity penalty are then combined to calculate the BLEU score. The BLEU score ranges from 0 to 1, with higher scores indicating better similarity between the candidate and reference texts.

In addition to the traditional BLEU score, we can also use sentence-level BLEU with smoothing. Particularly "smoothing method 7", as suggested by Chen and Cherry [23], who systematically compared different smoothing techniques for sentence-level BLEU.

Smoothing method 7, also known as Lin and Och method, is a popular smoothing technique that uses a modified Kneser-Ney smoothing approach [24] to address the issue of zero counts in the n-gram statistics. This method has been shown to be effective in improving the performance of sentence-level BLEU, especially for longer sentences.

To calculate the sentence-level BLEU score with smoothing method 7, we first tokenize the sentences being compared into n-grams. The modified Kneser-Ney probabilities can then be calculated for each n-gram in the first and second sentences, using the modified counts with discounting factor d = 0.75. These probabilities are then used to calculate the precision for each sentence, as well as the brevity penalty as described in the traditional BLEU score.

The idea behind sentence-level BLEU is to calculate the BLEU score at the sentence level and then average the scores across all sentences to get an overall score for the text. This approach can be useful when comparing the similarity between two texts that have different sentence structures or lengths, as ours may have.

Finally, precision scores are averaged across all sentences and the result is multiplied by the brevity penalty to get the sentence-level BLEU score. By using sentence-level BLEU with smoothing method 7, we are able to gain more granular insights in the comparison and it can be especially effective when dealing with longer sentences.

Though the BLEU metric is ultimately simpler than TF-IDF, it is less computationally expensive and is able to be to higher-order n-grams because of this, meaning it can be more helpful for capturing longer continuous matches. However, like TF-IDF it is not able to capture semantic similarity.

3.3 SBERT

As we have previously discussed, none of these methods so far can quite fully capture the human intuition of sentence similarity. For example, two sentences with different wordings and structures we may actually consider to be matching because they have the same semantic meaning, but traditional similarity metrics, struggle to capture this similarity accurately. In order to combat this, we will utilize a variation of the widely-used Bidirectional Encoder Representations from Transformers (BERT) architecture [12], called Sentence-BERT (SBERT) as introduced by Reimers and Gurevych [5].

BERT is a powerful neural network-based model for natural language processing, used to encode a word to represent the semantic meaning of the word in vector space - an embedding. However, unlike BERT, which produces a single, vector representation for each input word, SBERT produces a single, fixed-length vector representation for each sentence in the input text. These sentence embeddings can then be used for comparison.

SBERT has been shown to achieve state-of-the-art results on various sentence-level NLP tasks such as semantic textual similarity [25], paraphrase detection [26], and sentiment analysis [27]; and has been used in various practical applications such as chatbots [28], customer service [29], and content recommendation systems [30].

3.3.1 Model Training

The SBERT model adds to the BERT architecture by adding a pooling operation to produce fixed-sized sentence embeddings and finetunes using the siamese network architecture to ensure the model produces similar embeddings for semantically similar sentences. The siamese network architecture is where two copies of the BERT model are used to encode two input sentences of known similarity separately. The sentence embeddings produced by each BERT model are then compared using the cosine similarity function, and the model is trained to maximize the similarity score for similar sentence pairs and minimize it for dissimilar sentence pairs using Mean Squared Error Loss (MSE). The full process of this can be seen in figure 3.1 This is so the produced sentence embeddings are semantically meaningful and can be compared with cosine-similarity.

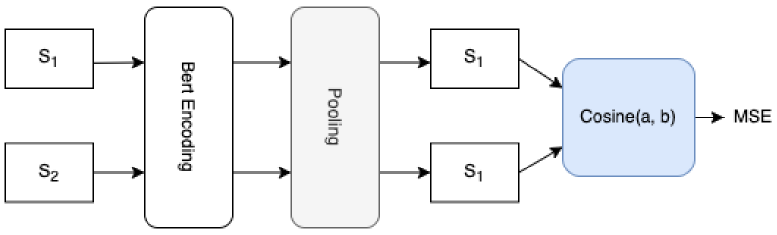


Figure 3.1: Siamese Training

3.3.2 Encoding

To compute new embeddings for a sentence using a trained SBERT model, the BERT architecture is used, which follows the structure of a Transformer. The sentence is first tokenized and each token then is converted into a high-dimensional vector representation using an embedding layer. Tokens are also split into subwords, however for the simplicity of our example we do not demonstrate this. This layer assigns a unique vector representation to each token based on its position in the input text and its context within the surrounding text. So if we take an example input sentence that consists of n tokens (words), represented as a sequence of one-hot encoded vectors $x_1, x_2, ..., x_n$. A one-hot encoded vector, being similar to the previously defined bag-of-words encoding described in Section 3.1.2, but instead only one element is “hot” (has a value of 1), while all other elements are “cold” (have a value of 0). The position of the ”hot” element indicates the word to which the vector corresponds in our vocabulary. Each token, represented as a one-hot vector, x_i is mapped to a d-dimensional embedding vector e_i using an embedding matrix E, such that:

$$e_i = E \cdot x_i$$

Where E is a d x V matrix, where V is the size of the SBERT model vocabulary and d is the dimensionality of the embedding space, a hyperparameter set before training the model. Each column of E represents the embedding for a single token in the vocabulary, which is learned during the pre-training phase of the SBERT model.

Once this embedding vector for each token has been generated, which we will refer to as the token embedding, the encoded representation of the input text is generated using multi-head self-attention [31], which allows the model to capture the contextual meaning of each token in the input text. To apply the multi-head self-attention to each token embedding, we compute a set of attention scores for each token, which determines how much attention to pay to each other token in the sequence, generating a set of context-aware vector representations for each token. Continuing our previous example, we define the set of token embeddings for the sentence as a matrix $X = [e_1, e_2, ..., e_n]$, where each column is a token embedding vector. The set of attention scores A_{self} is then calculated for each token i, given by:

$$A_{self}[i] = softmax(\frac{Q_i K_i^T}{\sqrt{d_k}} \cdot V_i)$$

where Q_i, K_i, and V_i are d_k x d matrices, called query, key, and value matrices, respectively. These matrices are learned parameters of the model that are computed from the token embeddings using linear transformations. The softmax function normalizes the attention scores so that they sum to one. The parameter $sqrt(d_k)$ is a scaling factor that helps to stabilize the attention scores during training.

For simplicity, this only describes calculating the self-attention directly for each token. However in the actual system, multi-head attention is calculated by splitting each token into "heads", where Q, K and V are linearly projected to a smaller dimension, and attention is calculated in parallel for each head, then concatenated and linearly projected back to calculate the output attention, as depicted in figure 3.2.

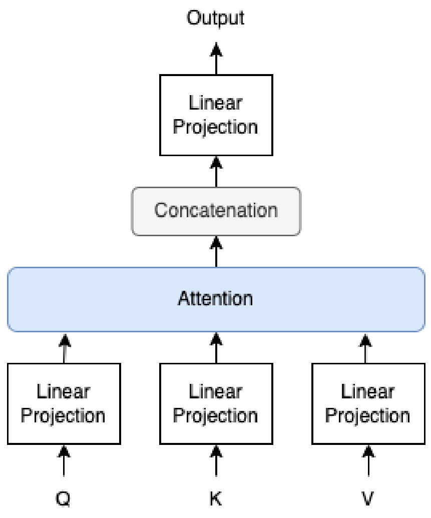


Figure 3.2: Multi-head self-attention

The resulting set of context-aware token embeddings is then combined into a fixed-length vector representation of the entire input text, by using a pooling operation, which is a technique for combining and generalizing features. This produces a single vector that represents the entire input text in a way that captures its semantic and contextual meaning.

3.3.3 Using SBERT

To then compare the similarity between two sentences using SBERT, we then simply compare the encoded vectors using the cosine angle between the vectors, as described in Section 3.1.2. The resulting value will be a measure of the similarity between the two sentences, with a value of 1 indicating perfect similarity and a value of 0 indicating no similarity.

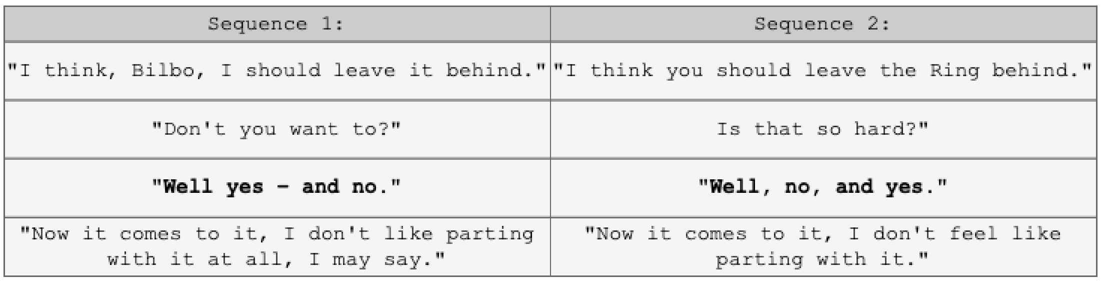
BERT methods have been shown to outperform traditional methods such as TF-IDF and word embeddings for computing sentence similarity in a variety of tasks, including paraphrase identification, question answering, and text classification [32][33]. And SBERT has been shown to provide a significant speed-up in comparison to BERT [5]. One of the advantages of SBERT is that it can capture the nuances of sentence meaning that cannot be captured by simpler methods such as TF-IDF. Additionally, SBERT is better able to handle out-of-vocabulary (OOV) words and can generalize to new sentence structures that were not seen during training, as it uses subword tokenization, as in the original BERT model. This process means that during training words are segmented into subword units, each with its own embeddings, so when a new word is encountered, it can be broken down into subwords and meaning can be inferred from combining subword embeddings in the training set. Though this does not help in all cases if no subwords appeared in training, this is still an improvement to TF-IDF, which does not have a strategy to deal with unencountered words.

However, the drawback of SBERT is that it is more computationally expensive, even when using a pre-trained model. This is because, in contrast to more traditional methods like TF-IDF and BLEU, SBERT requires more complex input processing, including subword segmentation and encoding. SBERT also requires more computational resources during inference, as SBERT's encoding process can take more time and computational resources than computing a simple TF-IDF vector or shared n-gram score. Therefore it may be more useful to use this similarity metric sparingly and for getting more fine-grained matches.

3.4 Context

We have already mentioned that inspiration can be affected by the surrounding context of the query and target sentences, with similar contexts implying a stronger match. In other words, we can intuitively say that if the sentences around a target sentence are similar to the sentences around the query sentence, it is much more likely that the target sentence and the query sentence are a match. For example figure 3.3:

We can identify that these are similar passages and that some inspiration has occurred. However, if the query sentence is *"Well yes - and no."* and the target sentence is *"Well no, and yes."*, as stand-alone sentences, though the two share the same words, they are



Sequence 1:	Sequence 2:
"I think, Bilbo, I should leave it behind."	"I think you should leave the Ring behind."
"Don't you want to?"	Is that so hard?"
"Well yes - and no."	**"Well, no, and yes."**
"Now it comes to it, I don't like parting with it at all, I may say."	"Now it comes to it, I don't feel like parting with it."

Figure 3.3: Two Similar Sequences, Target Matches in Bold

commonly used words and the ordering has been rearranged, so would perhaps not be as conducive to a match. However, by including the surrounding sentences we can see that the surrounding context is the same, suggesting that this is indeed a match. In our system, we want to be able to leverage this concept in our comparison, so we introduce the idea of *context*.

In our project *context* is when the similarity of the surrounding sentences, measured by one of our previously described similarity metrics, is used to "boost" the score of query and target sentence similarity. To capture this context, we first define a window around each sentence in parallel texts, 1-2 sentences long. We can then use a previously discussed similarity method and together the similarity of the target and query sentences, with that of the context window to give us a combined score. This means that the higher the similarity of the surrounding sentences and the sentence themselves, the higher the score. This can also be normalised by the size of the chosen context window for scaling.

Though this does present some potential issues with edge cases - with very similar matching sentences in the context window potentially skewing the results, this is usually what we want and it does a good job at rudimentarily addressing this intuition. And, when used in conjunction with other methods, can be very effective. Though it inherits all the benefits of the chosen similarity metric, it also inherits all the drawbacks.

Though this is particularly useful in our task, it could also be used in various settings when aligning two accounts of a scenario, even in machine translation.

Chapter 4

Implementation

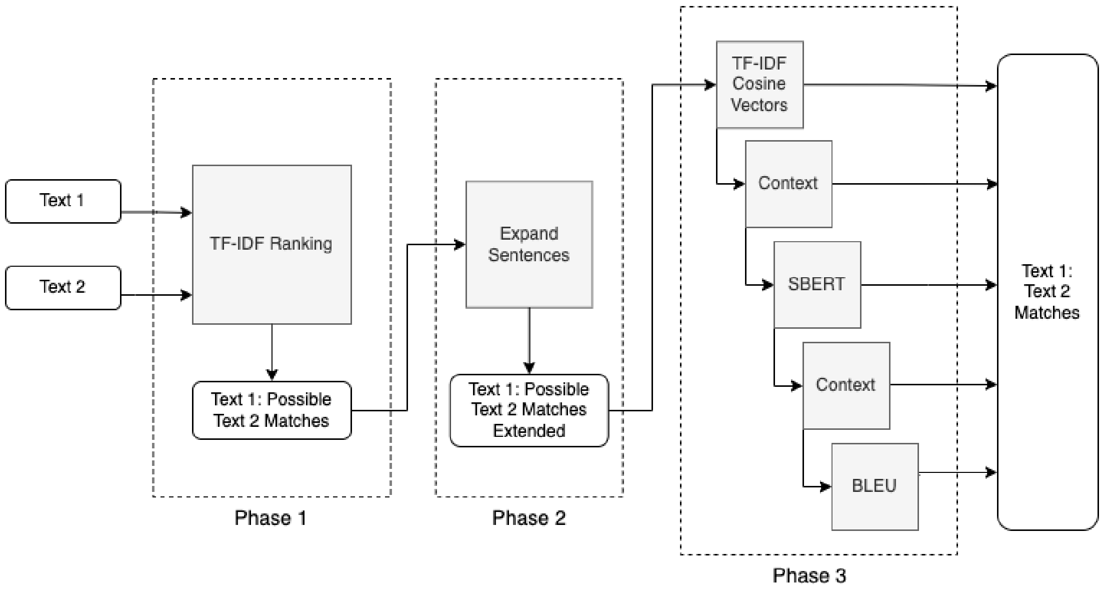


Figure 4.1: System Overview

Now we have detailed the methodologies and the different ways they can be useful, we will move on to describing how we utilised them in our task.

This first involves preprocessing the data, extracting the sentences and normalising the text, described fully in Section 4.2. Then the cleaned data goes through three phases of matching on varying levels to refine the matches, detailed in figure 4.1. Phase 1 is a basic refinement of the sentences using TF-IDF ranking, which is expanded in Section 4.3. In this phase, we use each film sentence as a query to return the top 5 "closest" sentences in the book, provided they are above a certain threshold. This allows us to narrow down the sentences and discard any sentences that have no possible matches. In this Phase we simply aim to maximize the number of true positives that our system produces, as more intensive refining occurs later in the system.

Phase 2, detailed in Section 4.4, addresses the problem that some sentences may have been split into multiple sentences in the film, or been drawn from multiple book sentences. As such, to try to optimize our matching, we want to combine any surrounding sentences that improve the matches, which Phase 2 attempts to do using BLEU as a fast metric to test if adding the surrounding sentences improve the matches.

Once the sentences have been expanded, we go into our third and final phase where we perform more fine grain similarity checking, using a combination of cosine similarity between TF-IDF vectors, "context" of surrounding vectors (as introduced in Section 3.4), cosine similarity between SBERT vectors and finally BLEU as a last check to try to catch any remaining sentences. In this Phase we now aim to balance the true and false positives our system produces, as we are aiming to extract as many "real" matches as possible without also returning many false ones - ones we would not judge to be inspired. We can say our system has been successful if a large proportion of the gold standard matches are found, while not also returning a large proportion of false matches.

4.1 The Data

To develop the system, the data we used comes from book/film of J.R.R. Tolkien's *The Fellowship of the Ring*. This provides many benefits, as it is famously a good example of a film adaptation that takes many lines from the source material. Also due to its popularity, its resources are widely available, including a fan-made list of matches between the book and the film, which we are able to use as a gold standard, which I will detail in section 5.1.

This data comes in the form of the entire original book text, which we will have to extract the dialogue from and the official subtitles used in the film. Using the subtitles also provides many benefits as it contains time-stamped data, allowing us - if desired to link the audio-visual content of the film to our matches as well.

4.2 Preprocessing

To use this extracted data, we first need to preprocess it. Preprocessing is an essential step when approaching any natural language processing task and this project is no different. Preprocessing is vital to ensure text is cleaned and prepared, extracting the meaningful information from the text while removing irrelevant or redundant information, and ensuring all text is normalised to the same level.

4.2.1 Dialogue Extraction

The subtitles are easy to extract dialogue from as they are already only made up of dialogue, but in the case of the book, we want to remove all non-dialogue text, as we are unlikely to find matches between the film dialogue and the descriptive parts of the book. Though it is possible that a scriptwriter has adapted some description from the book into the dialogue, it is unlikely to be a very concrete match and it is much more beneficial to the system to decrease the amount of text we need to search through to find matches.

The most obvious way to extract the dialogue from the book is to use take text surrounded by quotation marks, for example:

```
"Well, if you want my ring yourself, say so!" cried Bilbo.  "But you
won't get it.  I won't give my precious away, I tell you."
```

For this, all we need is a simple regex to extract the text between quotation marks. Regular expressions are powerful tools for pattern matching and can be used to extract specific patterns of text from a larger corpus. We utilized the standard Python python library `re` and the additional module `regex` [34] to perform sentence extraction. However, like much of language processing, it is rarely simple and one rule does not usually cover all. As the text only uses single quotes, there is no way to distinguish between actual quotes in the book, apostrophes and dialogue quotation marks. This is the first example of something that humans can easily intuit, but is hard to fully capture with a computer. The way that I got around this was to use a regex that captured sequences where the first quotation mark is preceded by a space and the ending quotation mark is followed by a space (or a full stop, which covers some edge cases). The text in between is limited to 2500 characters (over the length of the longest speech in the book) to prevent incorrect matches from stacking and accidentally capturing the large sections of the book.

As the regex used is greedy, often non-dialogue is also captured in favour of capturing longer sentences, usually, he said/she said descriptors. So in a broad attempt to capture this, I also used another regex to remove rogue dialogue descriptors.

Quotation marks were also not only signifiers of dialogue in the book, indents, italics etc. were also used, however, these were less consistent, but be trivially captured.

4.2.2 Normalisation

4.2.2.1 Document Division

Once the dialogue is extracted, the next step is to perform text normalization, which is the process of converting text to a standard format to make it easier to process. The first step here is to split the dialogue into usable chunks. I chose to use sentences, as they are the most natural way to split up language, even if it produces documents of varying lengths.

During my development, I experimented with the idea of using a rolling window to extract matches: comparing the “window” of text in the film with a “window” of text in the book. Where a window is a list of a fixed number of words that moves along the text one word at a time. This would be able to exactly capture similar segments, picking out snippets of the text that match, provided we are able to find a suitable window length. However, not only does this become very computationally expensive, as we must at least compare every word in the film to every word in the book, but it is also not as useful, as it simply picks out snippets. In our task we are trying to do something slightly more complex - identifying when a sentence has been inspired, whether or not all the words in the sentence match. So we may only have a small section of a sentence that matches, but we still want to be able to identify that the sentences are inspired.

4.2.2.2 Text Cleaning

Once the sentences were extracted, all punctuation was removed using Python's built-in `string.punctuation` library. Removing punctuation is a common preprocessing step, as it helps to ensure that the analysis is focused on the content of the text rather than its formatting.

We also converted all text to lowercase. This step is important for several reasons. First, it helps to ensure consistency in the text data by treating all words as the same regardless of their case. Secondly, it can improve the accuracy of text retrieval by matching queries to documents more effectively. In this step, however, we chose not to remove stopwords. Stopping is often performed in normalisation, however, stop words were not removed in this project as they may be important in finding matches between book and movie dialogue. We also chose not to stem the words as this may alter their meaning unexpectedly, instead using SBERT semantic similarity to account for small changes in wording.

The output of the preprocessing step is clean, normalized text data that is ready for matching, without additional noise. By extracting sentences, removing punctuation, and converting all text to lowercase, we ensure that the comparisons in the system are focused on the meaningful content of the text while ignoring irrelevant information.

4.3 Phase 1

4.3.1 Ranked Retrieval

Once the dialogue is extracted, for the first step in our system, we treat the problem as an information retrieval problem - effectively each film sentence is a query that we are searching for in a large corpus - the book dialogue. However, our task is slightly different from a standard information retrieval problem, as we don't just want to find the "most relevant document" (where document here refers to sentence), we want to only return "documents" that we consider relevant to the query. To do this we applied the methodology of ranked retrieval detailed in section 3.2.1, combined with thresholding using the TF-IDF score to ensure only sentences that may be possible matches are returned. We use the film sentences as our query terms, as not only is the corpus smaller but it is also written based on the book. For each film sentence, we are trying to find the sentence that has inspired it. This then becomes the base results for our next step to refine.

4.3.1.1 Inverted Index

To enable efficient retrieval we can calculate the TF-IDF score with the aid of an Inverted Index, another commonly used Information Retrieval Technique. This means that rather than simply storing each document, we use a dictionary-like object with each word (or token) in the corpus as the key and all the documents containing that token as the values. This allows us to easily calculate the TF-IDF weighting of a token in a query that, immediately retrieving all the documents that contain that token and discarding any that do not, rather than searching through the entire corpus each time for a specific token, which would greatly increase runtime.

In our system, we implement this by turning the book text into an Inverted Index. For each term, a list of the sentences in which it appeared is stored, along with the positions of the term in those sentences. We also stored the frequency of each term in the entire dialogue corpus.

4.3.1.2 TF-IDF Weighting

Once the inverted index is created, we use the film sentences to query the index. The film sentences are represented as a set of terms and each "document" - book sentence - can be ranked on their relevance to the query using the TF-IDF weighting scheme as described in 5.2.

For each query, the top 5 are selected as possible matches, as long as they have a score over 0.5, considered to be a value of at least vague similarity. Giving each query a range of possible matches, while discarding any film sentences that do not even slightly match any book sentences.

4.4 Phase 2

4.4.1 Sentence Expansion using BLEU

The next phase of our system expands the possible matches gained from TF-IDF ranking, to try to address the problem that some sentences may have been split into multiple sentences in the film, or been drawn from multiple book sentences. For example, if we look at figure 4.2, we can see an example of a match that is split into multiple sentences: in the book, this match is contained in one sentence, but in the screenwriting process, this has been turned into two, presumably to give more direction to the actor on how the sentence should be delivered. However, we still want to be able to capture this as a match. To do this, we take our refined matches: `film_sentence: [possible_book_sentences]` and treat each film sentence as the "input sentence" and each possible book sentence as the "target sentence". We first check if the sentences adjacent to the target sentence, contain any words similar to the input sentence using a set intersection. This allows us to very quickly filter out any adjacent sentences that have no relation to the input sentence. We then combine any adjacent sentences that are included in the set intersection, with the target sentence and take the combination with the highest BLEU score, as mentioned in Section 3.2, to the input sentence. This should then return the closest combination of sentences to the input sentence. These matches are then passed on to the next phase for a more computationally expensive, granular similarity comparison

As this problem can occur in both directions - the book sentence can be split into multiple film sentences, or vice versa - we also want to perform this in reverse: where each book sentence is treated as the input sentence, and each film sentence is treated as the target sentence. We then take the combination of book/film sentence combinations with the overall highest BLEU score to return the most likely match.

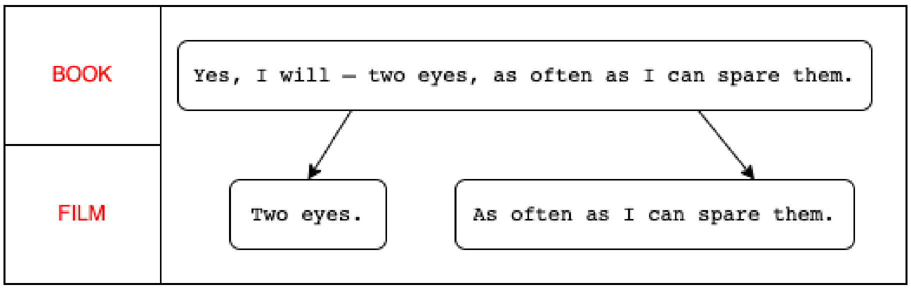

BOOK
FILM
Yes, I will — two eyes, as often as I can spare them.
Two eyes.
As often as I can spare them.

Figure 4.2: Sentence Expansion

4.5 Phase 3

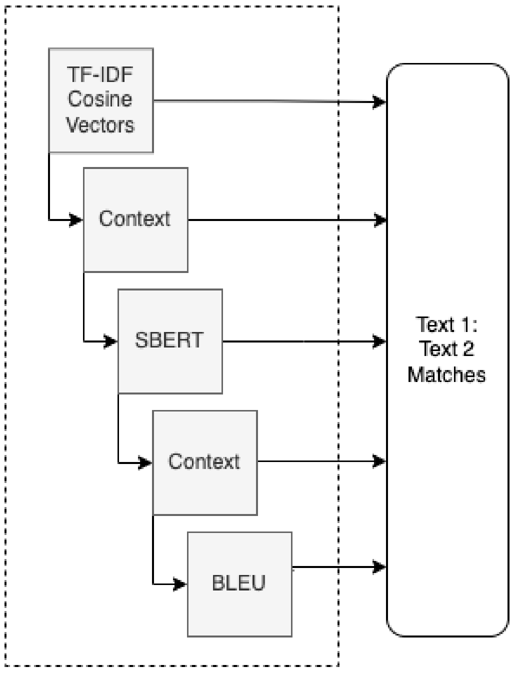


Figure 4.3: Phase 3

For the final phase of the algorithm, once the results have been refined down to what we think are the most likely matches, we use a combination of the techniques sequentially: TF-IDF cosine similarity, context, SBERT and BLEU, as described in Chapter 3, to extract the different types of matches we expect; a breakdown of which can be seen in figure 5.2. Sentences go through each similarity test - testing if a sentence is over a certain score of that metric - and those that pass the test are counted as matches, while those that do not, go on to be tested using the next metric. Sentences that pass none of the similarity tests are then discarded. Using these different tests allows us to be very modular in targeting different types of matches, particularly in deciding the order they are used and deciding the thresholds of each metric. Experimentation on which of these produces the best results is detailed in section 5.2.5.

4.5.1 TF-IDF Cosine Similarity

The first of these metrics we used in this phase was TF-IDF cosine similarity between the vector representations of the TF-IDF weighted scores as described in Section 3.1.2. To use TF-IDF vectorization in our project, we used the implementation provided by the Python library `sklearn` [35] so as to utilise its n-gram capabilities.

The TF-IDF weights for each word or n-gram in the text are computed using the sklearn TfidfVectorizer class. This class takes as input the preprocessed text data and a number of parameters, including the n-gram range and whether or not to use sublinear scaling, which we utilize in our system, as described in 3.1.1. The n-gram range determines the length of the n-grams used in the TF-IDF calculation. In our project, we chose the n-gram range of (1,3), which includes n-grams from 1 to 3. This is to find a balance between allowing higher-order n-grams and not bloating our vocabulary.

Once the TF-IDF weights have been computed, the cosine similarity between the two texts can be calculated using the sklearn cosine similarity function. This function takes as input the TF-IDF weight matrices for the two texts and returns a similarity score between 0 and 1. This score is then thresholded and sentences with a cosine similarity over this threshold are counted as matches, while sentences under the threshold move on to be tested using the other similarity metrics in the phase. I experimented with tuning this threshold in Section 5.2.4.

TF-IDF cosine similarity is chosen as our initial comparison in this phase, as we expect the majority of matches to be collected in this step, as we are using TF-IDF to return our “generally similar” sentences, where the words may have been rearranged, but the words used in the sentences are mostly the same, for example:

```
"I know I don’t look it, but I’m beginning to feel it in my heart."
vs.
"I don’t look it, but I am beginning to feel it in my heart of hearts."
```

4.5.2 Context

In the previous step of Phase 3, I described how I used TF-IDF similarity to refine the search for potential matches in the dialogue. The next step is to catch any matches that are a little under our previous threshold, but the surrounding sentences of the film are very similar to the surrounding sentences of the book, which therefore makes the sentences more likely to be similar. This is done using the idea of Context introduced in Section 3.4. By incorporating contextual similarity into the search process, we aim to further refine the search and reduce the number of false positives, as well as catch any borderline similarity matches that can be called matches due to the context.

To incorporate contextual similarity into the search, we first define a context window around each sentence in the book and movie. I experimented with different window sizes and found that a window size of 2 sentences before and after the sentences provided the best results. We then compute the TF-IDF cosine similarity, of each sentence within the context window for the film and book sentences. This cosine score is then added to the cosine similarity of the initial TFIDF cosine score of the book sentence to “boost” it to be a possible match, provided the initial cosine score is over a similarity of 0.6. Putting this in place ensures that the target book sentence has at least some degree of similarity to the film sentence and doesn’t just have very similar surrounding sentences. The boosted score is then also thresholded, and sentences over this threshold are counted as a match. The exact values of this final threshold are experimented with in Section 5.2.4.

Context is used twice in Phase 3, once after calculating the TF-IDF cosine similarity and once after calculating the SBERT cosine similarity, which will be described in the next section. In this second usage, the initial TF-IDF cosine similarity is replaced with the SBERT cosine similarity in order to boost SBERT cosine similarity of sentences that fall slightly under the initial SBERT cosine threshold using the similarity of the surrounding sentences.

4.5.3 SBERT

The next metric used in Phase 3 is the cosine similarity between SBERT semantic embedding vector representations of the sentences, as described in Section 3.3.

To incorporate SBERT into the search, we first generate the SBERT embeddings for each sentence in the book and movie. We used a pre-trained SBERT model to generate these embeddings [36]. A pre-trained model is utilized as training our own would require a huge dataset, which we do not have in our example, and the process is computationally expensive and time-consuming when we already have access to state-of-the-art fine-tuned models trained on billions of sentence pairs, giving us the ability to take full advantage of SBERT’s semantic capabilities. Once these embeddings are created, we then compute the cosine similarity between each pair of sentence embeddings. Again producing a similarity score between 0 and 1, where a score closer to 1 indicates high similarity, while a score closer to 0 indicates low similarity. Similarly to the TF-IDF cosine similarity step, we then threshold the SBERT cosine similarity, taking all sentences over a certain value and passing any under it on to the next step.

This metric attempts to capture matching sentences where the wording of the sentence may have been changed or been paraphrased, but the meaning remains the same and is still recognisable as matching the original sentence. For example:

```
"Already the writing upon it, which at first was as clear as red flame,
fadeth and is now only barely to be read."
                                   VS.
"The writing, which at first was as clear as red flame, has all but
disappeared."
```

Though there is an identical subphrase - *“which at first as clear as a red flame”* - the rest of the sentence has been rewritten, but is still clearly identifiable as being inspired. This kind of match would be discarded just by using TF-IDF, as this sentence scores a low 0.58 in this metric, whereas using SBERT cosine similarity it scores a strong 0.81.

We are also able to capture matches such as:

```
"Today is my 111th birthday."
VS.
"Today is my one hundred and eleventh birthday."
```

Where TF-IDF cannot capture that *"111th"* is the same as *"one hundred and eleventh"*, SBERT is able to capture that these are equivalent.

As we can see this is a very powerful measure of similarity, however it is more computationally expensive, as we must encode the semantic embeddings. We therefore use it towards the end of Phase 3 to avoid uneccessary computation that can caputured by TF-IDF similarity.

4.5.4 BLEU

In the final part of Phase 3, we again use the BLEU metric, described in Section 3.2, to try to catch any remaining matches that have not been caught in the previous steps. We use BLEU, as it is a useful metric in targeting similar phrases.

To calculate the BLEU score, we used the `nltk` Python library [37], using the function `sentence_bleu`, using smoothing method 7 introduced by Chen and Cherry that we also described in Section 3.2; which is particularly useful when comparing sentences of different lengths. The BLEU similarity is calculated between each film and book sentence, and the pairs over a certain threshold are taken as matches.

BLEU can sometimes capture sentences that have nearly identical words, but may not mean the same thing, for example:

```
"It is already begun."
VS.
"It is already begun."
```

BLEU scores this highly, as it has the shared phrase *"it is already"*, and though the last word is different, because it is only one word, the score is only lightly affected. For this reason we only test the BLEU score of sentences over a certain SBERT score, as SBERT embeddings can capture that these sentences have very different meanings; so the sentences have to be at least marginally semantically similar to be tested using BLEU to avoid these kinds of errors. This is also why the BLEU metric is used after the SBERT embeddings have been calculated in the system.

Chapter 5

Experiments

In this Chapter we describe the experiments undertaken to test the performance of the system we have implemented in how well it can capture sentences inspired sentences. The first experiment, detailed in Section 5.2, shows the process of tuning all the parameters described in Chapter 4 to produce the optimum output. While the second experiment, described in Section 5.3 was to test the performance of the system on a held-out book/film combination to test if the system has been overtuned to the *The Fellowship of the Ring*. In this experiment the system is tested on the held-out book in the *Lord of the Rings* trilogy, the film/book of J.R.R. Tolkien's *Return of the King*, as we have similar gold standard data to *The Fellowship of the Ring*. The third experiment, detailed in Section 5.4, was to test the performance of the system when using automatically transcribed text from the film audio rather than the film subtitles, to test if the system can be fully automated.

5.1 Gold Standard Evaluation

To evaluate whether or not we have extracted all the possible sentences, we can compare the output of our system with our gold standard. In this project, we are lucky enough to have a human complied gold standard due to the popularity of the franchise. This compilation was created by an enthusiast who had read the book and watched the movie multiple times. They identified and compiled all the instances where the dialogue in the movie matched the book's dialogue.

This gold standard is used throughout the evaluation of the experiments. Using a human gold standard was important in this process as it allowed me to evaluate the accuracy of the system's output. It also helped to identify areas where the system could be improved during development by demonstrating which matches are not being found.

Specifically, this gold standard is used in evaluation to identify the number of true positives, false positives, true negatives, and false negatives, from which we can then calculate the precision, recall and F1 scores for our system. A true positive occurred when our system identified a match that was also present in the gold standard. A false positive occurred when our system identified a match that was not present in the gold standard. A true negative occurred when our system correctly identified a non-match. A false negative occurred when our system failed to identify a match that was present in the gold standard.

Precision measures the proportion of true positives, drawn from our gold standard, to all identified positives. This is particularly useful as a metric to evaluate how many of the matches returned are false. It is calculated using the equation:

$$Precision = \frac{TruePositives}{TruePositives+FalsePositives}$$

Recall measures the proportion of true positives to all actual positives in the gold standard. This is particularly useful as a metric to evaluate how many of the gold standard matches we are managing to capture. It is calculated using the equation:

$$Recall = \frac{TruePositives}{TruePositives+FalseNegatives}$$

The F1 score is the harmonic mean of precision and recall and is often used as an overall performance metric [38], as it balances both precision and recall. It is calculated using the equation:

$$F1 = 2 * \frac{Precision*Recall}{Precision+Recall}$$

However, it is important to note that through testing we discovered that this gold standard was not exhaustive, and there were many matches which we would consider to be true positives that our system was able to identify that were not in the gold standard. This means that it is not a true gold standard as we initially thought, and we ultimately cannot fully evaluate our system. However, it is still a valuable tool in evaluation, as it provides a baseline and allows us to at least measure the percentage of matches in the gold standard our system can find.

5.2 Parameter Tuning

5.2.1 Dialogue Extraction

The first step of the system that we tested was dialogue extraction. As this step is not trivial and does not always extract all the sentences due to varying indicators of dialogue in the book text and some that have no indication, it is important that we test the system to check how many of the gold standard sentences we have managed to extract.

To assess the dialogue extraction from the text, we directly compared the gold standard with the extracted sentences. Showing that my methods had achieved extraction of all 185 gold sentences (sentences that we consider correct matches) possible to extract out of the 207 gold standard sentences. This is a strong result as 10 of the gold standard sentences are from other Lord of the Rings books, and therefore cannot be found, and others have no identifiable textual signals of dialogue, so it is expected that we will not capture all of them. Despite this we manage to capture 94% of the gold sentences, not including those from other texts, therefore I believe this can be considered a very successful extraction, maximising our likelihood of successful matches for the matching phases of our system.

5.2.2 Phase 1

To evaluate the output of Phase 1, aimed at refining the sentences, I used the evaluation detailed in 5.1 to measure the accuracy of the system's output. In this first case, however, we are only interested in the recall metric, as we want to ensure as many matches as possible are found so that none are discarded going into the second phase, and we expect precision to be very low. However, we also want to discard any matches that are complete non-matches, to avoid unnecessary computation, narrowing down the book sentences from a starting number of 11826 and the film sentences of 1846. We tested three different values of thresholding the TF-IDF score: 0.1, 0.3 and 0.5. The results of this can be seen in table 5.1.

Threshold	Recall	Film Sents	Book Sents	No. Matches
0.1	0.99	1664	3796	9133
0.3	0.99	1664	3796	9133
0.5	**0.99**	**1654**	**3762**	**8918**

Table 5.1: Results of Varying the TF-IDF Score Threshold

From these results we can see that thresholds 0.1 and 0.3, perform the same, whereas 0.5 performed the best, still providing a recall score of 99%, as well as effectively narrowing down the number of sentences; for the book from 11826 to 1654, and for the film from 1846 to 3762, meaning we only need to perform 8918 comparisons in the next phase. We, therefore, use the output of this test going forward in the next phases.

5.2.3 Phase 2

Evaluation of Phase 2 is similar to Phase 1, as our main goal still is to ensure no gold standard matches are discarded while reducing the number of matches to compare. Therefore we once again focus on the recall score and reduction in the number of book and film sentences.

No thresholding is done in this step, so we can simply evaluate the output, which produces a recall score of 98% and reduces the number of film sentences from 1654 to 1536, and the number of book sentences from 3762 to 3423 by combining some possible matches. We expect to see the recall score drop slightly, as some gold standard sentences are combined together and therefore no longer match the original sentences.

5.2.4 Phase 3

Now sentences have been sufficiently narrowed down, Phase 3 is the phase in which the most intensive similarity comparisons occur, using various different metrics. This, therefore, requires rigorous testing to find the correct thresholds for each metric to capture the optimum amount of matches.

5.2.4.1 TF-IDF Cosine

The first threshold we tested is for the cosine similarity between TF-IDF vectors. The results of this can be seen in table 5.2.

Threshold	Precision	Recall	F1
0.70	0.71	0.56	0.62
0.75	0.76	0.49	0.60
0.80	**0.79**	**0.43**	**0.55**
0.85	0.75	0.33	0.46
0.90	0.78	0.29	0.42

Table 5.2: Results of Varying the TF-IDF Cosine Threshold

As this is just the initial step in Phase 3, we want to prioritize precision, as recall will rise throughout the other steps. Looking at the results we can see that precision is highest when the threshold is set to 0.80, so we, therefore use the TF-IDF cosine similarity threshold of 0.80.

5.2.4.2 First Context

The next threshold we tested is the TF-IDF cosine similarity boosted by the similarity of either the sentences before or after the target book sentence - the first use of context. The results of this can be seen in table 5.3

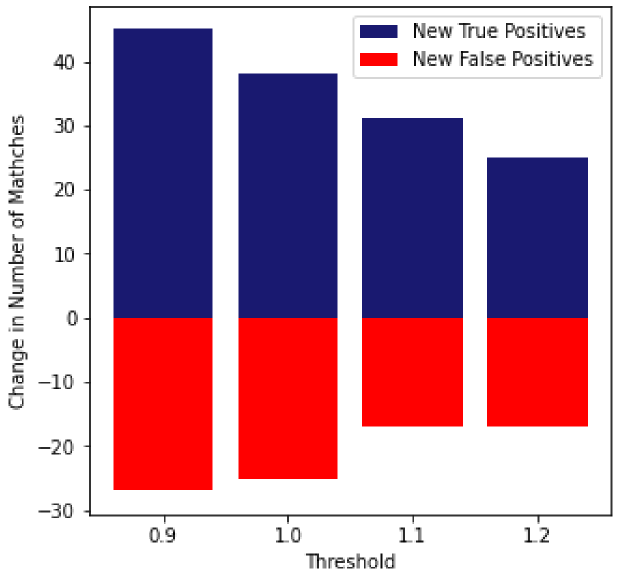


Figure 5.1: Visualising the Additional True and False Positives by Varying the First Context Threshold

As with the TF-IDF Cosine score, we want to prioritize precision, which would suggest that 1.2 is the optimum threshold. However, we can further evaluate these thresholds by analysing the additional true and false positives gained in this step, as shown in figure 5.1. Here we want to maximise the blue bar - the additional true positives - while

Threshold	Precision	Recall	F1
0.9	0.72	0.67	0.69
1.0	0.72	0.63	0.67
1.1	0.74	0.59	0.66
1.2	0.75	0.56	0.64

Table 5.3: Results of Varying the First Context Threshold

minimising the red bar - the additional false negatives - above all else. Here we can see that despite the threshold of 1.2 providing a slightly higher precision score, using the threshold of 1.1 returns the same amount of false positives, but more true positives. Therefore we chose to use the threshold of 1.1 for the first context threshold.

5.2.4.3 SBERT Cosine

The next threshold we tested is the SBERT cosine similarity used to test the semantic similarity of sentences. The results of this can be seen in table 5.4

Threshold	Precision	Recall	F1
0.70	0.51	0.85	0.64
0.75	0.58	0.79	0.67
0.80	0.64	0.74	0.69
0.85	0.70	0.67	0.68
0.90	0.72	0.63	0.67

Table 5.4: Results of Varying the SBERT Cosine Threshold

At this stage, though we still want to prioritise precision, we want to begin to consider recall, as after this step, we expect the majority of matches to have been found, with the second use of context and the BLEU metric being used as a catch for matches that have not quite met the previous thresholds. Therefore, though the highest value of precision was produced by setting the threshold to 0.90, giving a result of 72%, setting the threshold to 0.85 still maintains high precision at 70%, while producing a better recall of 66%. It is important to also consider that setting the threshold to 0.95 - the upper limit of cosine being 1 - means that we would only accept sentences that are very nearly identical, which is not the objective of our task. However, we do want to set it to a high value like 0.85, as we do not want to consider vectors that are too broadly semantically similar as if the content is just vaguely similar, it is hard to say that the film sentence is really inspired rather than just following a similar plot.

5.2.4.4 Second Context

The next threshold we tested is the SBERT cosine similarity boosted by the TF-IDF cosine similarity of either the sentences before or after the target book sentence - the second use of context in our system. The results of this can be seen in table 5.5. Initially in this experimentation, sentences were only tested for context if they were over an SBERT similarity of 0.60, as in the first use of context. However as described in the previous section, we want to exercise greater selectivity in testing sentences for semantic similarity. We therefore also experiment with raising this threshold to 0.70.

Initial Cosine	Threshold	Precision	Recall	F1
0.6	1.1	0.51	0.84	0.63
0.6	1.2	0.57	0.77	0.66
0.6	1.3	0.61	0.72	0.66
0.7	1.1	0.59	0.79	0.68
0.7	1.2	0.62	0.74	0.68
0.7	**1.3**	**0.66**	**0.72**	**0.69**

Table 5.5: Results of Varying the Second Context Threshold and the Initial SBERT Cosine Threshold

As we are nearly at the end of Phase 3 and do not expect to get many more matches, we are now looking to maximise the F1 score - the harmonic mean of the precision and recall - in addition the precision. Therefore we selected the thresholds with the highest F1 score and precision scores, initial SBERT Cosine at 0.7 and the Context Threshold at 1.3.

5.2.4.5 BLEU

In the final stage of Phase 3, we tested the threshold for the BLEU similarity metric, which is used as a final test to try to capture any remaining matches that have slipped through that have similar phrases. The results of this test can be seen in table 5.6.

Threshold	Precision	Recall	F1
0.2	**0.62**	**0.82**	**0.71**
0.3	0.64	0.78	0.70
0.4	0.65	0.77	0.70

Table 5.6: Results of Varying the BLEU Threshold

At this final stage, we must now fully prioritise the F1 score and therefore chose the threshold for BLEU of 0.2. However, this is also supported by analysing the output results, as not only have we managed to capture 82% of the gold standard matches, but the majority of false positives are also what we would call matches, as we can see from an example of the “false positive” matches below:

```
1:  Film sentence:  "welcome legolas son of thranduil"
Book sentence:  "welcome son of thranduil"
2:  Film sentence:  "gandalf the grey did not pass the borders of this
land"
Book Sentence:  "gandalf the grey set out with the company but he did
not pass the borders of this land"
3:  Film sentence:  "the writing which at first was as clear as red flame
has all but disappeared"
Book sentence:  "already the writing upon it which at first was as clear
```

```
as red flame fadeth and is now only barely to be read"
4:  Film sentence:  "every league you travel south the danger will increase"
Book sentence:  "the danger will increase with every league that we go
south under the naked sky"
5:  Film sentence:  "i hope the others find a safer road"
Book sentence:  "we will go and may the others find a safe road"
```

5.2.5 Results

We can see how many matches each metric managed to extract in fig 5.2. Showing that TF-IDF is used to find the majority of matches as it is aimed to do.

Overall our system achieved a precision score of 62%, a recall score of 82% and an F1 score of 71%. Alone these are already successful results, however, they are enhanced further by looking at the extra matches found, as mentioned in the previous section. This shows that the false positive matches are in fact in the majority of cases, extra matches that were not found in the gold standard, though this is very difficult to quantify other than manually. To outperform the human gold standard is exactly what we are aiming to do. Humans will always make mistakes and can easily miss things, particularly at this scale of content, both audio-visual and textual. This demonstrates that our model has been successful in its task, capturing both the majority of the gold standard matches, as well as ones that the gold standard has missed. Though if we did have a more efficient way to quantify the extra true positives, it is possible we might be able to improve our model further with different parameter tunings. 5.2

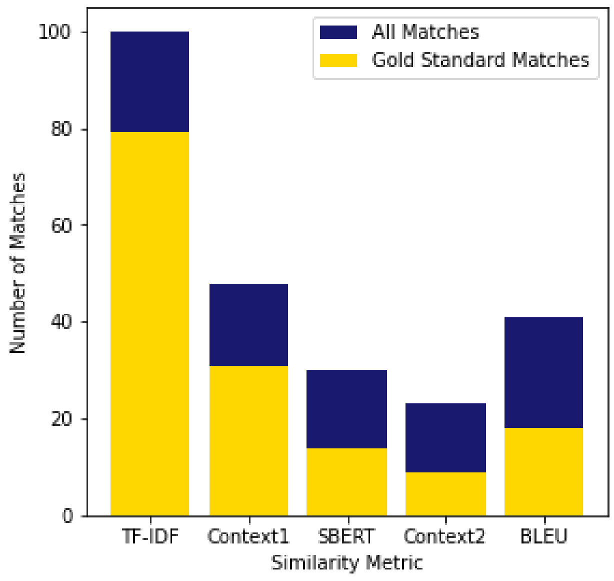


Figure 5.2: Number of Matches Gained from each Similarity Metric

5.3 Experiment with Unseen Book

In this experiment, we tested our now parameter-tuned system on a second new book/film combination to test if the system has been overtuned in Section 5.2. In this experiment, the system is tested on the film/book of J.R.R. Tolkien's *Return of the King*, as we have similar gold standard data to *The Fellowship of the Ring*. This book/film will have similar language to the first one, so we cannot fully test it on an "unseen" example, but it will still contain completely new sentences. Using *Return of the King* also allows us to use a similar gold standard to the first book for this text from the same source.

5.3.1 Results

Using exactly the same preprocessing steps as for the first book, we are able to extract 169 of 176, 96% of the possible gold sentences, extracting a total of 8379 book sentences and 1864 film sentences. The system was then run with these sentences without any changes to the parameters used in Section 5.2. In Phase 1 we got a recall score of 90%, which is a slight decrease compared to the previous recall in Phase 1 in Section 5.2.2, but is still a strong result. In Phase 2 we got a recall score of 94%, again a decrease, but still similar to the previously found recall in Section 5.2.3. We can also see here that the methodology must have been successful, as many sentences have been combined, causing us to increase the recall from Phase 1. However, this decrease can possibly be attributed to more sentences being expanded than in the previous example. In Phase 3 we returned the final output of the system, finding the results shown in figure 5.7.

Precision	Recall	F1
0.66	0.73	0.70

Table 5.7: Results of Experiment on Held-out Book/Film

We can see that the overall F1 score achieved is 0.70, very similar to what we previously found in experiment 1, though the Recall and Precision scores are slightly different, with this experiment gaining a higher Precision, with a lower Recall. However we are still capturing a high percentage of the gold matches. The change in Recall and Precision may suggest that the parameters have been overturned, though we can also speculate, that as the gold standard matches are still quite informal and fan-made, it is possible the author of these matches was more lenient in what they counted as a match in this version.

5.4 Experiments with Audio

In this experiment, we tested the performance of the system when using automatically transcribed text from the film audio rather than the previously used film subtitles. This tests whether the system can be fully automated if subtitles are not available. We expect this to perform worse than the more accurate subtitles, as we expect even a well-trained Automatic Speech Recognition (ASR) system to produce errors in transcription, particularly when the audio we are trying to transcribe uses words that are very specific to the fantasy world of Lord of the Rings, for example, Gollum - one of the character names.

5.4.1 Setup

To transcribe the film audio we utilize the state-of-the-art google cloud speech-to-text API [39], using Google's "most advanced deep learning neural network algorithms". This provides many benefits as well as using a well-trained model: it allows us to use the API's various features to customize the model to our domain, as well as generate punctuation with the transcription, allowing us to split up the sentences normally, as when using the subtitles, it also allows us to transcribe a very large audio file - the full length of the film without having to divide it into smaller chunks which would possibly interfere with the sentence structure.

Even using this advanced model we are not able to fully capture all dialogue in the audio and many sentences are not transcribed or are transcribed incorrectly. This is in part due to the model being very general as it has been trained on a large amount of data. To extend this experiment we would want to implement a language model based on the dialogue from the book that could be used to fine-tune a pre-trained model. However, ultimately this fell out of the scope of this project.

5.4.2 Results

Upon analysing the transcript by comparing it with the previously used subtitles, we can calculate that we have been able to extract 1101 out 1845, only 60% of the original film sentences, leaving us with only 79% of the gold standard matches. The system was then run with these sentences without any changes to the parameters used in Section 5.2. In Phase 1 we got a recall score of 98%, which is very successful compared to the previous recall in Phase 1. In Phase 2 we got a recall score of 99%, again very successful. However in Phase 3 we returned the final output of the system, finding the results shown in figure 5.8.

Precision	Recall	F1
0.60	0.52	0.56

Table 5.8: Results of Experiment using ASR Transcript

As we can see, though the precision remains similar, the recall has dropped considerably. This is as expected, as though we are able to capture semi-falsely transcribed sentences in the initial Phases, if the transcription is off even by only a few words, it will no longer score as higher as it previously did. However, it may be possible that by lowering the thresholds in Phase these results could be improved, though this will subsequently cause the precision to drop as false positives are accepted under the new threshold. The better way to counteract this would be to

Chapter 6

Conclusions

Our goal in the project was to quantify *inspiration* between sentences and overall, as we have seen through our experimentation, the developed system shows strong potential, able to capture over 82% of the sentences in our text determined to have been inspired. We can also see that our system generalizes reasonably well to unseen data from our experiments on a held-out book/film, reproducing similar results to our original tests, though some parameter tuning may still be required to fully optimize the system. We have also evaluated the system when converted from text-to-text similarity comparison to speech-to-text, transcribing the dialogue directly from the film to fully automate the system for use without subtitles. And though we do not reach the same accuracy as the text-to-text system, the system shows potential if an improved ASR system was developed alongside.

Thus we have set a baseline tool for capturing *inspiration*. This provides a tool that could be used for creative analysis and data extraction, as well as the general methodologies providing contributions in similarity detection, with a variety of extended use cases, by using the concept of inspiration, from plagiarism detection to even comparing victims' accounts of criminals.

6.1 Further Additions

Though the system has achieved strong results, there are many ways it can still be enhanced.

6.1.1 Extend ASR Transcription

As we found in our Audio experiment, the transcription process that we use is not perfect and produces many incorrect results which greatly affect the accuracy of our system. To fully automate our system for use without subtitle data, we would need to extend the ASR methodology to improve transcription. As we have suggested, this could be done via a language model trained on the dialogue in the book text, as this would allow the system to recognize the words and phrases that are most likely to be used in that film context and skewing it towards the dialogue seen in the book. For example, the complex fantasy names could better be captured as the model “expects” to see them.

6.1.2 Use WFSTs in Matching

One type of match that the system is not as good at recognising is in longer sentences, where there is a long phrase that is very similar or nearly identical, but the rest of the sentence deviates significantly. For example:

```
"For in the days of Isildur the Ruling Ring passed out of all
knowledge, and the Three were released from its dominion."
                                  VS.
"And for two and a half thousand years...the Ring passed out of all
knowledge."
```

As humans, we can recognise that the first sentence may have inspired the second, due to the phrase *“the Ring passed out of all knowledge”*. We therefore need a way to try to capture long matching strings, though the BLEU metric attempts to do this, it is limited to the comparison of 4 n-grams, so phrases longer than this are not effectively rewarded. One way to do this would be to utilize Weighted Finite State Transducers (WFSTs) to score sentences. This allows us to directly compare two sentences word by word and for each time this train of matching words remains unbroken, i.e. a long phrase is found, we add to the score and penalize the sentence when the train is broken. [add diagram].

6.1.3 Speaker Identification

Another way we could extend this project further is by extending the speech recognition and utilising which character is speaking to determine more likely matches - i.e. if a character says similar dialogue in the book and the film, it is more likely to be a match than if two different characters say a similar line. We would not want to enforce this as a rule, as there may often be cases when a scriptwriter or director has switched around who speaks what lines, and we would not want to lose these matches, so we would want to add this as an extra component, similar to context or perhaps in conjunction with the context. If the same character says a similar line in the same context in both the book and the film, this has a very strong chance of being a match. In the case of the film, the most straightforward option would be to extract the speaker from the film screenplay. However, though this is a very simple solution, it may be problematic if the screenplay is not available or has been extensively changed in the film itself. Other options include either the extension of the ASR experiment, adding speaker recognition, or using speaker identification from the text - using linguistic style and POS (part-of-speech) tags [40], or a combination of the two [41]. While for the book we may be able to utilize cues in the text, as often it will describe who is speaking - e.g. “said Gandalf”, however, it is also often implied who is speaking, similar to the problem we faced when trying extract the dialogue in [insert a reference to section]. We therefore may need to use a similar method as the film when isolated to just text [40], or utilize methods more specific speaker identification in fiction texts [42].

The drawback however of adding this is it may require extensive training, additional resources, and/or extra time complexity that may detract from the speed and lightweight nature of the current solution, while only adding a small percentage of increase to the matches. However, it may improve the precision.

Bibliography

[1] Josepha Sherman. *Storytelling: An encyclopedia of mythology and folklore*. Routledge, 2015.

[2] Jellenik Glenn and Leitch Thomas. *On the Origins of Adaptation, as Such: The Birth of a Simple Abstraction*. The Oxford Handbook of Adaptation Studies. Ed. Thomas Leitch. New York, 2017.

[3] Lissette Lopez Szwydky. *Transmedia adaptation in the nineteenth century*. Ohio State University Press Columbus, 2020.

[4] Robert Sklar and David A. Cook. history of film. encyclopedia britannica. 2023, March 12.

[5] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv preprint arXiv:1908.10084*, 2019.

[6] Prabhakar Raghavan Christopher D. Manning and Hinrich Schütze. *Introduction to Information Retrieval*, chapter 6, pages 117–132. Cambridge University Press, 2009.

[7] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, pages 311–318, 2002.

[8] Vladimir I Levenshtein et al. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady*, volume 10, pages 707–710. Soviet Union, 1966.

[9] Fred J Damerau. A technique for computer detection and correction of spelling errors. *Communications of the ACM*, 7(3):171–176, 1964.

[10] Matthew A Jaro. Advances in record-linkage methodology as applied to matching the 1985 census of tampa, florida. *Journal of the American Statistical Association*, 84(406):414–420, 1989.

[11] Jurafsky Daniel, Martin James H, et al. *Speech and language processing: An introduction to natural language processing, computational linguistics, and speech recognition*. prentice hall, 2008.

[12] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.

[13] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations, 2018.

[14] Gerard Salton. Introduction to modern information retrieval. *McGraw-Hill*, 1983.

[15] Scott Deerwester, Susan T Dumais, George W Furnas, Thomas K Landauer, and Richard Harshman. Indexing by latent semantic analysis. *Journal of the American society for information science*, 41(6):391–407, 1990.

[16] Qin Yaxue. Convolutional neural networks for literature retrieval. In *2020 International Conference on Computer Vision, Image and Deep Learning (CVIDL)*, pages 393–397, 2020.

[17] Runjie Zhu, Xinhui Tu, and Jimmy Xiangji Huang. Chapter seven - deep learning on information retrieval and its applications. In Himansu Das, Chittaranjan Pradhan, and Nilanjan Dey, editors, *Deep Learning for Data Analytics*, pages 125–153. Academic Press, 2020.

[18] Yukun Zhu, Ryan Kiros, Rich Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *Proceedings of the IEEE international conference on computer vision*, pages 19–27, 2015.

[19] Makarand Tapaswi, Martin Bäuml, and Rainer Stiefelhagen. Book2movie: Aligning video scenes with book chapters. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1827–1835, 2015.

[20] Karen Sparck Jones. A statistical interpretation of term specificity and its application in retrieval. *Journal of documentation*, 28(1):11–21, 1972.

[21] W.B. Croft, D. Metzler, and T. Strohman. *Search engines: Information retrieval in practice*. Addison-Wesley Professional, 2010.

[22] Christian Buck and Philipp Koehn. Quick and reliable document alignment via tf/idf-weighted cosine distance. In *Proceedings of the First Conference on Machine Translation: Volume 2, Shared Task Papers*, pages 672–678, 2016.

[23] Boxing Chen and Colin Cherry. A systematic comparison of smoothing techniques for sentence-level bleu. In *Proceedings of the ninth workshop on statistical machine translation*, pages 362–367, 2014.

[24] Reinhard Kneser and Hermann Ney. Improved backing-off for m-gram language modeling. In *1995 international conference on acoustics, speech, and signal processing*, volume 1, pages 181–184. IEEE, 1995.

[25] Bohan Li, Hao Zhou, Junxian He, Mingxuan Wang, Yiming Yang, and Lei Li. On the sentence embeddings from pre-trained language models, 2020.

[26] Elron Bandel, Ranit Aharonov, Michal Shmueli-Scheuer, Ilya Shnayderman, Noam Slonim, and Liat Ein-Dor. Quality controlled paraphrase generation, 2022.

[27] Klaifer Garcia and Lilian Berton. Topic detection and sentiment analysis in twitter content related to covid-19 from brazil and the usa. *Applied Soft Computing*, 101:107057, 2021.

[28] Mario Rodríguez-Cantelar, Diego de la Cal, Marcos Estecha, A Grande, Diego Martin, N Rodriguez, R Martinez, and L Fernando. Genuine2: An open domain chatbot based on generative models. *Alexa Prize Proceedings*, 2021.

[29] Peiyao Wang, Joyce Fang, and Julia Reinspach. CS-BERT: a pretrained model for customer service dialogues. In *Proceedings of the 3rd Workshop on Natural Language Processing for Conversational AI*, pages 130–142, Online, November 2021. Association for Computational Linguistics.

[30] Li Zhang, Wei Lu, Haihua Chen, Yong Huang, and Qikai Cheng. A comparative evaluation of biomedical similar article recommendation. *Journal of Biomedical Informatics*, 131:104106, 2022.

[31] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[32] Alvin Subakti, Hendri Murfi, and Nora Hariadi. The performance of bert as data representation of text clustering. *Journal of big Data*, 9(1):1–21, 2022.

[33] Santiago González-Carvajal and Eduardo C Garrido-Merchán. Comparing bert against traditional machine learning text classification. *arXiv preprint arXiv:2005.13012*, 2020.

[34] Python regex module. https://pypi.org/project/regex/.

[35] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[36] Wenhui Wang, Furu Wei, Li Dong, Hangbo Bao, Nan Yang, and Ming Zhou. Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers, 2020.

[37] Edward Loper and Steven Bird. Nltk: The natural language toolkit, 2002.

[38] David Powers. Evaluation: From precision, recall and f-factor to roc, informedness, markedness correlation. *Mach. Learn. Technol.*, 2, 01 2008.

[39] Google cloud speech-to-text. https://cloud.google.com/speech-to-text/.

[40] Amitava Kundu, Dipankar Das, and Sivaji Bandyopadhyay. Speaker identification from film dialogues. In *2012 4th International Conference on Intelligent Human Computer Interaction (IHCI)*, pages 1–4, 2012.

[41] R. Turetsky and N. Dimitrova. Screenplay alignment for closed-system speaker identification and analysis of feature films. In *2004 IEEE International Conference on Multimedia and Expo (ICME) (IEEE Cat. No.04TH8763)*, volume 3, pages 1659–1662 Vol.3, 2004.

[42] Kevin Glass and Shaun Bangay. A naive salience-based method for speaker identification in fiction books. In *Proceedings of the 18th Annual Symposium of the Pattern Recognition Association of South Africa (PRASA'07)*, pages 1–6. Citeseer, 2007.