Computer-based Recognition of Intertextuality within the Hebrew Bible

Bert Lobbezoo

June 16, 2015

Abstract

Intertextuality can be defined as the influence of other texts on the constructing of new texts. For example, it is well-known that Shakespeare rarely invented his own stories, but in fact they are based on familiar tales or classical writers. It would be very interesting to find the sources he used. This may be done manually, but that is very time-consuming. Another possibility might be to set up a computer-based solution which automatically detects his sources. This solution could also be applied to less well-known authors and writers to detect their sources. It would be very interesting to create a tree or graph of the intertextuality between the work of famous authors over the centuries, which will be possible when the detection of intertextuality becomes very fast and accurate.

The aim of this research is to find a computer-based solution to detect intertextuality. A very good dataset for this problem is the Hebrew Bible. In the tradition of Christianity, a well-known way to interpret the Bible is by comparing certain verses with other relevant verses. So there are a lot of lists with verses that are related in some manner to other verses. This is in fact intertextuality, which is defined as the influence of other texts in the constructing of new texts. Another reason for using the Hebrew Bible is because a lot of research has been done on it. There is a database which contains all kind of grammatical characteristics of the language. It also contains the structure of the language, so the sentences, clauses and phrases of each verse are available in that database. This can be used to find the exact intertextuality.

This research examines three approaches which are intended to recognize intertextuality. Two of them are novel because of their involving of the special features available in the database of the Hebrew Bible. The first approach uses multiple distance metrics. These distance metrics measures the similarity of the verses. The second approach uses high-level grammatical features. These features are based on grammatical characteristics of the text. These features are manually annotated. For us it is very special to get this data, because it is very time-consuming work to annotate all these grammatical features. The third approach involves the scale of language. Each sentence can be divided into lower scales, first into clauses, then into phrases and finally into words. This approach measures the distance of verses on these scales to find the exact intertextuality. The plagiarism detector Copyfind also evaluates the intertextuality between texts in order to compare the last approach with existing solutions.

The results are promising. It is possible to detect intertextuality between texts. We have found that a simple string metric produces good performance. This performance can be slightly improved by the use of multiple string metrics. The results of the high-level grammatical approach are not satisfactory, so this is not a good way to detect intertextuality. However, the results of the multi-scale distance approach are comparable with those of distance-based approach.

The results of these two approaches definitely outperform the results of the plagiarism detector Copyfind.

Chapter 1

Introduction

Intertextuality means that no text can be considered in isolation, but instead that a text is constructed from the point of view of other texts. Two examples of intertextuality are that text rely on the beliefs, issues, ideas, statements generally circulated among the readers, and that a text use recognizable kinds of language, phrasing, and genres. Manually recognizing intertextuality between two texts is a time-consuming job, especially when the texts are long. It is for example hard to manually inspect all the texts of Aristotle in order to find intertextuality with the texts of Plato. When a computer is able to do this job, it becomes a very simple operation to find the intertextuality between all texts of classical Greek philosophers. Then it is probably that the computer finds some unknown relatedness between the work of Greek philosophers, because the computer is able to find all relations between the texts. By hand, this is very difficult and faults are easy made by humans. Not only for Greek philosopher, but also the computer might find all relations between all writers over the history of earth. It becomes a simple job to construct a sort of dictionary which contain all writing that historical writers have written about a specific word or event. Another application for an intertextuality detector is that it can suggest intertextuality. On the internet, there a lot of collections of texts and books. An example is Google Books, which has scanned on April 2013 more than 30 million books. An intertextuality detector can make suggestions what other authors have said about a specific event, or can suggest what the first occurrence is of a certain statement. If this was done manually it would take years of research to read all these books and find related parts, but a computer is able to do it in a few minutes.

The problem considered in this report is how to detect intertextuality? There are a few studies that have interfaces with this question. An example is Coffee et al. [6] who developed a tool to automatically detect allusions in Latin poetry. But what preciously is intertextuality? Can it be defined more exactly? Bazerman [3] has defined six levels of intertextuality. According to him, texts may draw from (1) authoritative sources, (2) social events, (3) text that are used from statements as background, support, and contrast, (4) beliefs, issues,

ideas, statements generally circulated among readers, (5) recognizable kinds of language, phrasing, and genres and (6) available language resources. So there are many forms of intertextuality. From this list, it may become more clear that no text ever is fully invented by an author, but that it always depends on what others have said, on what the reader knows, on the cultural customs of both writer and reader, etcetera.

Although not much work has been spent on the detection of intertextuality, more research is done on recognizing semantic text similarity or plagiarism. These are two terms that are being related to intertextuality, but in fact they are far from equal. The term semantic in semantic text similarity stands for the meaning of words. Semantic similarity means that people agree that something has roughly the same meaning. This field deals more with human agreement and semantic relatedness than with text relatedness. The difference with plagiarism is this: plagiarism is a part of intertextuality, but intertextuality can also occur in other forms, for example, it is no plagiarism if someone is referring to an event, or when a text is based on well-known beliefs and ideas. These examples are clearly a form of intertextuality but are definitely no intertextuality.

In the tradition of Christianity have always been searched for cross-references of verses in the Bible. A cross-reference is a pair of verses that are in some way related to each other, which is nearly similar to the definition of intertextuality. In our research, we refer to three lists of these cross-references, from three leading translation of the Bible into the English language. The intersection of these three would omit obscure or rare cross-references and select only the lucid ones. This combined list contains more than 10.000 cross-references between verses. The Hebrew Bible contain approximately 23.000 verses, so roughly each verse is on average part of one cross-reference.

The Bible is the most widely read book of the world, no book has more translations in different languages, and it is one of the most intensively investigated books of the world. Not only in early times, but it is still a very interesting book for modern literature studies. This is the reason why we choose the Hebrew Bible as our research object to detect intertextuality. We consider only the Hebrew Bible because the whole Bible is written in two languages, the first part in Hebrew and the second part in Greek. In this report, we display verses of the Bible in the translation of the English Standard Version (ESV), which is an English translation of the Hebrew Bible.

Our solution to the problem mentioned above is threefold. We examined three approaches each with a different solution to detect intertextuality.

The first approach is based on the literature about plagiarism detection and semantic text similarity. We examined different string metrics to find out what string metric produces the best result. In the literature, most researchers founds that n-grams are a good way to detect plagiarism and semantic text similarity [2, 1, 30]. We explored the best value of n and we also evaluated what combination of string metrics provides the best performance. The second approach involves the high-level grammatical characteristics of the text. The reason for this approach is that it has never been evaluated in literature. In the related field of the detection of semantic text similarity, all researchers use

only the lexical data itself, no one has considered the syntactic features. We examined the grammatical data to see whether it is a good indicator of intertextuality. Furthermore, we identified the best performing grammatical features for the detection of intertextuality. The third approach uses the tree structure of language. It is perfectly suitable to establish the exact location of intertextuality within the text. The main question in this approach is: what are the 'peaks' of intertextuality? Where are the corresponding parts? This information is involved in the method used to detect intertextuality. To compare the performances of the approaches with those of existing tools, we deployed the open source software Copyfind, which is a plagiarism detector.

The content of the paper is as follows. First, we elaborate on the background to the problem (Chapter 2). Then we clarify the theoretical part of the three solutions (Chapter 3). Next, we describe how we evaluated and validated our work (Chapter 4). After this, we explain the experiment we carried out to measure the performances of the distance-based approach (Chapter 5), the high-level grammatical approach (Chapter 6) and the multi-scale distance approach (Chapter 7). Finally, we come to a conclusion and explain the future work that needs to be done (Chapter 8).

Chapter 2

Background

In this chapter, we explain the background of the research. First, the definition of intertextuality is explained and also the types of intertextuality are described (Section 2.1). Then we elucidate on the related work of the research (Section 2.2).

2.1 Intertextuality

The origin of the word 'intertextuality' comes from the Latin word 'intertexto', which means 'mingle while weaving' [15]. According to Bazerman [3], one can define intertextuality as

> "the explicit and implicit relations that a text or utterance has to prior, contemporary and potential future texts."

Clear examples of intertextuality are allusion, quotation, calque, translation, pastiche and parody. Another clear figure is plagiarism, where the author denies it his work is intertextuality. In Table 2.1, the less familiar figures of intertextuality are explained.

However, intertextuality is broader than these figures. It means that no text is original. The well-known saying of Isaac Newton depicts this clearly: If I have seen further, it is by standing on the shoulders of giants. This holds not only for scientific research, but also for simple texts. Each author uses sayings, phrases, ideas of earlier authors. Sometimes the new author is unknowing of this, other times he is doing it fully knowing, for example when he is writing a parody or when he quotes someone.

Figure	Meaning	Example
Allusion	One refers indirectly to an object or circumstance	Obama refers in his convention speech in 2008 to Martin Luther King with the words 'Now is the time'
Calque	A word or phrase borrowed from another language by literal translation	In Dutch the word skyscraper is translated as 'wolkenkrabber' (literally 'clouds-scratcher')
Pastiche	Imitation of the style or character of the work of artists. The difference with parody is that a pastiche gives a comic spin to the work, and a parody does this to the author.	Since the author's time many stories about Sherlock Holmes have been written as pastiches.

Table 2.1: Less familiar examples of intertextual figures explained.

Bazerman [3] has identified six levels of intertextuality. These are:

1. Text draw on prior texts that are used as authoritative sources, for example a court decision that refers to a constitution.
2. Text may refer to events, for example the attack on the World Trade Center in 2001.
3. Text may use explicit statements as background, support, and contrast. For example: an author cites an encyclopedia.
4. Text may use beliefs, issues, ideas, statements that generally circulated among readers and are familiar for them.
5. Text might have recognizable kinds of language, phrasing, and genres.
6. Text uses available resources of language. For example: an author uses a well-known genre.

The levels start with explicit types of intertextuality, and ends with more implicit types. However, they are all types of intertextuality. For our research it is important that at least the explicit types of intertextuality will be recognized. The detection of the sixth level of intertextuality is not relevant for us, because then the intertextuality is on a higher level. Therefore, it needs another solution than the solutions we propose in this research.

Intertextuality can occur between written text, but it can also occur in spoken text, it can even appear in painting or other kinds of arts. Our research focus only on written intertextuality.

The difference of intertextuality with plagiarism is that plagiarism is just a part of intertextuality. In fact, it belongs with level three of the levels stated above. A difference is that in plagiarism, it is the author's aim to hide the fact that he is using another's text. In contrary to this, intertexuality has not always the aim to hide that fact, for example when an author is citing another, or making a parody of another's work. Another difference is that citing wrongly is also plagiarism.

The difference of intertextuality between semantic text similarity is somewhat more difficult. On first sight, semantic text similarity and intertextuality are interchangeble terms, but that is not true. Intuitively, semantic text similarity is a human cognitive measure [11]. An example is the similarity between *car* and *bike*, these words are more related than *car* and *food*. It has been proven that this hold for most humans [28, 23]. In practice, the semantic text similarity is measured within a taxonomy or ontology, for example Marsi and Krahmer [20] and Maguitman et al. [19]. The distance between nodes and edges of a taxonomy or ontology are measured to calculate the similarity between two words and this measurement is the semantic text similarity. The aim of intertextuality is to recognize text that *sounds* the same, in order to get more information about a text, while the aim of semantic text similarity is to detect texts that *are* indeed the same. For example, when Obama say: Now it is time, he means something else than what Martin Luther Kings have meant when he uses that words in his speeches. King meant that it is time to make a real democracy, in which black people have to the same right as white people, but Obama meant that it was time to address the problems that are faced to the U.S. and the world in that time. So between the same words spoken by Obama and King there is no semantic similarity, but it is clearly an example of intertextuality.

2.2 Related work

Text representation is very important in the field of text processing. The simplest way of text representation is using the words of a sentence, separated by spaces. An example is the work of Islam and Inkpen [13]. A more normalized way of representing words is the stemmed representation. Examples are statistical stemmers [16] and rule-based stemmers [26]. More advanced models are the bag-of-words and Latent Semantic Analysis (LSA). In bag-of-words, each document has a vector. The vector is of the size of all words in the corpus. The number of occurrences of each word in a document is counted and is put into the vector. The main drawback of this method is that it is not very efficient. Only a small part of all words of the corpus are in a document. A more efficient method is LSA. In LSA, a matrix is constructed with on the rows the unique words of the corpus and the columns represent the documents. Then singular value decomposition is used to reduce the numbers of rows of the matrix. Then the similarity between unique words is computed by taking the cosine of the angle between the vector Dumais [7]. However, due to computational limits, only several hundreds of unique words can be used. Example of research that

uses LSA are Foltz, Kintsch, and Landauer [8] and Oleshchuk and Pedersen [24]. Two other representations are a taxonomy and ontology. A taxonomy classifies words into categories. Examples of common-used taxonomies are Wikipedia [25] and Wordnet [22]. Research that use these taxonomies are Hu et al. [12] and Gabrilovich and Markovitch [9]. As representation of the text, they simply use the words. Then the words are enriched by a weighting that is based on a taxonomy. Oleshchuk and Pedersen [24] developed a method to create a graph-based ontology, which contains the relationships between concepts. This ontology might be used to measure the semantic similarity between texts.

A method that is often used for plagiarism detection is using n-grams. Barrón-Cedeño and Rosso [2] uses n-gram comparison to recognize plagiarism on word level. As distance, he used the containment measure [18]

$$dist(a, b) = \frac{|N(a) \cap N(b)|}{|N(a)}$$

where a is a sentence of the text that is candidate of being plagiarized from $b \in B$. Another approach that uses n-grams is the approach of Stamatatos [30]. Stamatatos uses character n-gram profiles, which means that the n-grams are applied on characters and that the frequencies of the n-grams are incorporated.

Some additional improvement of n-grams are examined by Adeel Nawab, Stevenson, and Clough [1]. They use modified n-grams and weighted n-grams by a language model and discovers that these additions enhance the performances. As distance, they use the asymmetric containment measure of Clough et al. [5]:

$$dist(A, B) = \frac{\sum_{b \in B} \sum_{a \in A} |N(a) \cap N(b)|}{\sum_{b_1 \in B} \sum_{b_2 \in B} |N(b_1) \cap N(b_2)|}$$

The Levenshtein distance is a well-known string metric, invented by Vladimir Levenshtein [17]. The Levenshtein distance between two strings is defined as the minimum number of edits that are necessary to transform one string to the other string. Edits are insertions, deletions or substitutions. The Levenshtein distance is very effective for short strings, but for longer strings it takes far more computational time.

Another simple string metrics is the Longest common substring [10]. This string metric seeks for the longest common substring within a set of strings. It is especially developed for DNA-sequences, but it can also be applied on strings. The difference with the Levenshtein distance is that the longest common substring uses only insertions and deletions, and not substitutions.

The Jaro distance also measures the similarity between strings. The main part is equal to the Levenshtein distance, but it also considers the number of transpositions between the strings [14]. It further uses another algorithm than the number of edits of the Levenshtein distance. Winkler adjusted the Jaro-distance in such a way that the characters at the beginning of the string are more significant than differences at the end of the string [33]. With this adjustment, the Jaro-Winkler distance is best suitable for the comparing of smaller strings like names and words.

In information theory, most frequent K characters is a string metric that quickly can estimate how similar two ordered strings are. Seker et al. [29] invented this metric. It simply counts the frequency of the most frequent k characters in two strings and divides that number by the sum of the lengths of the strings. For our research, the ordering does not matter because the texts are not ordered in any way.

A research that approaches our work is that of White and Joy [31]. They investigated plagiarism detection on sentence level by using a word count metric. With this metric, the detector is able to detect not only substrings, but also obfuscation, like reordering, splitting sentences and merging. The difference with our approach is that we use a syntax count metric. We can conclude that n-grams are good features for the detection of intertextuality.

Now we describe the research of others on semantic text similarity, and the differences with our research. Metzler, Dumais, and Meek [21] try to detect intertextuality on short sentences, something what we also need to do with our dataset. They examined a lexical, probabilistic and hybrid approach and found that the hybrid approach outperform the other.

Chapter 3

Approaches

This chapter describes the approaches that are invented and evaluated for the detection of intertextuality. First, we elaborate on the data we have used (Section 3.1). Second, we elucidate on the representation of the text (Section 3.2). Third, we describe the distance-based approach (Section 3.3). Fourth, we explain the high-level grammatical approach (Section 3.4). Finally, we discuss the multi-scale distance approach (Section 3.5).

3.1 Data

This section explains the data we used for our experiments. It starts with an explanation of the text that is used, in Section 3.1.1. Then we explain which features we use besides the lexical data available in the text, in Section 3.1.2.

3.1.1 Hebrew Bible

In this research, the Hebrew Bible is used as training text. The main reason for this choice is that it has a lot of known intertextuality in itself. In tradition of Christianity is always searched for pairs of verses that have some relation to each other, because a well-known explanation method of the Bible is comparing a verse with another verses to come to the best explanation of a verse.

Furthermore, in Hebrew similar words have the same consonants, so the recognition of synonyms is roughly the same as the recognition of exactly the same words. For example, the consonants of the Hebrew translation of 'He wrote' and 'He dictated' are both 'KTB'. The Hebrew Bible is transmitted with only the consonants. The vowels are added later and are not absolutely necessary for the meaning of Hebrew, so this research is done on the consonants of the Hebrew Bible. In Table 3.1 the original language is shown at the left. The consonants are the square-looking signs, the vowels are the small signs up and below these consonants. In the middle, the transliterated text is displayed. This

transliterated text contains no vowels. The > stands for a particular consonant in Hebrew. On the right an English translation is shown.

Genesis 1:1 (BHS)	Genesis 1:1 (transliterated)	Genesis 1:1 (ESV)
בְּרֵאשִׁית בָּרָא אֱלֹהִים אֵת הַשָּׁמַיִם וְאֵת הָאָרֶץ׃	B R>CJT BR> >LHJM >T H CMJM W >T H >RY	*In the beginning, God created the heavens and the earth.*

Table 3.1: Example 1: A verse in the original Hebrew language (left), in a transliterated form of the original Hebrew language (center) and translated in English (right)

3.1.2 WIVU-database

A large database that contains all grammatical characteristics of the Hebrew Bible is the WIVU database. LAF-fabric is a good method to process the data of this database [27]. This database contains all words of the Hebrew Bible, with a lot of grammatical characteristics of each word. On higher language scales, the database contains also grammatical features. More exactly, the grammatical characteristics of the phrases, clauses and sentences of the whole Hebrew Bible are also annotated in the database.

Now we explain the structure of the language in the database. The lowest scale of a text is the word scale, then the phrase scale follows. Another higher scale is the clause scale and finally the sentence scale follows. The highest scales are half-verses and verses, but verses and sentences are not always in parallel. An example is when a sentence start in a previous verse and ends in the next verse. Because of the structure of the Hebrew Bible, verses can mostly be separated into two half-verses. Figure 3.1 shows an example of the language and database structure. Table 3.2 you see the average and the standard variation of the number of sentences, clauses, phrases and words per verse.

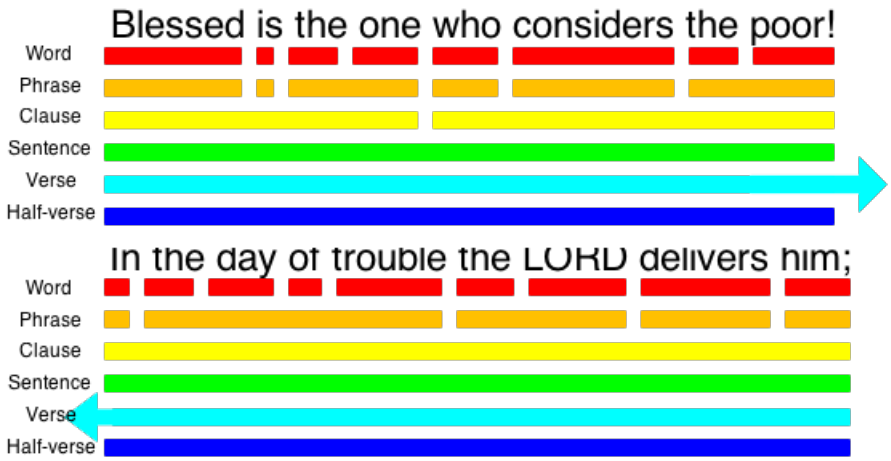

Figure 3.1: An example of the structure of words, phrases, clauses, sentences, verses, half-verses (Psalm 41:2 - English Standard Version)

Granularity	**Mean**	**Standard variation**	**Mean + std**	**Mean + 2std**
Verses	1	1	2	3
Sentences	$2.87 \approx 3$	$1.73 \approx 2$	$4.61 \approx 5$	$6.34 \approx 7$
Clauses	$3.88 \approx 4$	$2.02 \approx 2$	$5.90 \approx 6$	$7.92 \approx 8$
Phrases	$11.54 \approx 11$	$5.59 \approx 6$	$17.13 \approx 17$	$22.71 \approx 23$
Words	$18.37 \approx 18$	$8.82 \approx 9$	$27.20 \approx 27$	$36.02 \approx 36$

Table 3.2: Mean and standard variation of the number of occurences per verse for each granularity.

As text representation, it is usually to remove irrelevant words, like articles. Table 3.3 shows the different representations that are used. Further, it displayed the all translations that are used in our research. A representation that contains the Hebrew vowels is available, however, we did not use it because that representation provides poor results.

In the text representation, we used the Hebrew lexemes. Lexemes can be seen as a way of stemming text. A lexeme is the root of a word. The lexeme of a verb is the root of that verb, and did not contain the suffices of that verb. The lexeme of a noun is the noun in single, so without '-s' at the end of the noun. Example 2 shows an English example of lexemes. As default translation, we use the ESV, so all verses in this report are from the ESV translation.

By justice a king builds up the land, but he who exacts gifts tears it down.	*By justice king build up land, he exact gift tear it down.*
Proverbs 29:4 (ESV)	Proverbs 29:4 (Lexemes small of ESV)

Example 2: An English example of lexemes

Abbreviations	**Language**	**Remarks**
BHS	Hebrew	Only the consonants of the Hebrew text
Lex moderate	Hebrew	The lexemes of the Hebrew text that are verbs, nouns, proper nouns, personal pronoun, adjectives, adverbs, interjection, negative particle and interrogative particle
ESV	English	
NIV	English	
NAS	English	
SVV	Dutch	Translation which is known as a literally translation

Table 3.3: Explanation of the different translation of the Hebrew text

3.2 Representation

In this section, we provide a mathematical explanation of the features we have.

Words The smallest units in the database are words. Let us denoted a word as D_i. Each word has some features: $\forall d : d = \begin{bmatrix} X_{D,1} \\ X_{D,2} \\ \vdots \\ X_{D,\delta} \end{bmatrix}$.
An example is the feature *number*. Each word has a certain number, in Hebrew it might be a singular, dual or plural. In English, only the numbers singular and plural exist. The word 'tree' is singular, and the word 'trees' is plural.

Phrases A phrase is denoted as C_i and it holds that $C_i = \{D_1, \cdots, D_n\}$. Each phrase has some features: $\forall c : c = \begin{bmatrix} X_{C,1} \\ X_{C,2} \\ \vdots \\ X_{C,\gamma} \end{bmatrix}$.
An example of a phrase feature is the *phrase type*. Each phrase has a certain type. The sentence 'The angry king is chasing the frightened enemy' has two

phrases of the type nominal phrase and one phrase of the type verbal phrase. ‘ The angry king’ is a noun phrase, and ‘the frightened enemy is also a noun phrase. The phrase ‘is chasing’ is of the type verbal phrase.

Clauses Each clause contains one or more phrases. For clause B_i it holds that $B_i = \{C_1, \cdots, C_n\}$.

Each clause has some features: $\forall b : b = \begin{bmatrix} X_{B,1} \\ X_{B,2} \\ \vdots \\ X_{B,\beta} \end{bmatrix}$.

An example of a feature that each clause has is relation with its context. An example of a relation is a relative clause. In the sentence The king, who was angry, chased the frightened enemy’, the clause ‘who was angry’ is a relative clause.

Sentences Each sentence contains one or more clauses. For sentence A_i it holds that $A_i = \{B_1, \cdots, B_n\}$.

Verse A verse is defined as V_i. A verse consist of some sentences and it holds that $V_i = \{A_1, \cdots, A_n\}$. The set may be empty.

Pair features Let us denote a pair of verses as $P_{x,y} = \{V_x, V_y\}$ with $x \neq y$ and $P_{x,y} = P_{y.x}$. Both V_x and V_y have the features described above. On top of that, the pair $P_{x,y}$ has also some pair features. A pair of verses may be a cross-reference, but can also be not a cross-reference. A pair that is a cross-reference is called a positive object, a pair of verses that not a cross-reference is called a negative object. An example of a feature over a pair is the Levenshtein distance between the verses of the pair.

The sentences of the pair $P_{x,y} = \{V_x, V_y\}$ have these features:

$$\forall a \forall b : \{a, b\} = \begin{bmatrix} Z_{A,1} \\ Z_{A,2} \\ \vdots \\ Z_{A,\alpha_Z} \end{bmatrix}$$

So and so on for lower levels.

3.3 Distance-based approach

This section explains the details of the distance-based approach. First, the different scales on which the distance metrics are applied are explained. Second, the distance measures itself are described.

3.3.1 Scales

This kind of scales are different from the scales that language has and what are used in the multi-scale distance approach. The scales that are explained in this section are the units that are fed to the distance metrics.

The first scale on which the distance metrics are computed are the character units. Simply the characters are taken as units to compute the distance on.

The second scale deals with n-grams. An n-gram is a contiguous sequence of n characters within a string. For example, the 2-grams of the word 'fight' are 'fi', 'ig', 'gh' and 'ht'. And the 3-grams of that word are 'fig', 'igh' and 'ght'. When taken words instead of characters as units, then the 2-grams of the sentence 'He is a king' are 'He is', 'is a', 'a king', and the 3-grams of that sentence are 'He is a' and 'is a king'.

The third scale is the word scale and the fourth scale is called word n-grams, which are n-grams but then on word level.

Table 3.4 shows an example of the units of the four scale with an example sentence.

Scale	**Units**
Original	He is a king
Character scale	{'H', 'e', ' ', 'i', 's', ' ', 'a', ' ', 'k', 'i', 'n', 'g' }
2-grams	{'He', 'is', 'ki', 'in', 'ng' }
3-grams	{'kin', 'ing'}
Word scale	{'He', 'is', 'a', 'king' }
Word 2-grams	{'He is', 'is a', 'a king' }
Word 3-grams	{'He is a', 'is a king' }

Table 3.4: Units of an example sentence for the four different scales.

3.3.2 Distance measures

In this subsection, all distance measures that are used in our research are explained. All distance metrics are normalized by the length of their inputs.

Levenshtein distance The Levenshtein distance is the number of edits that are required to change one string into another. An edit is an insertion, deletion or substitution. Mathematically, we could define the Levenshtein distance between two strings a and b by $d_l = lev(|a|, |b|)$ where

$$lev(i,j) = \begin{cases} \max(i,j) & & \text{if } \min(i,j) = 0 \\ \min \begin{cases} lev(i-1,j)+1 & \text{insertion} & \\ lev(i,j-1)+1 & \text{deletion} & \text{otherwise} \\ lev(i-1,j-1)+1_{(a_i \neq b_j)} & \text{substitution} & \end{cases} & & \end{cases}$$

where $1_{(a_i \neq b_j)}$ is an characteristic function which is equal to 0 when $a_i = b_j$. Otherwise, it is equal to 1. The i and j are indices that run over a and b.

For example, the Levenshtein distance between 'table' and 'tall' is 2. The 'b' needs to be substituted by a 'l' and the 'e' needs to be deleted. So two edits are necessary to change 'table' into 'tall'.

Jaro-Winkler distance The Jaro-Winkler distance is a based on the Jaro distance, so we first explain the Jaro distance and then explain the Jaro-Winkler distance. The Jaro distance is developed within the area of the linking of records [33]. The Jaro-Winkler distance is designed for short strings like person names. The Hebrew Bible contains a lot of person names and place names, and there is a lot of intertextuality related to similar person and place names. Therefore, the Jaro-Winkler distance is a good choice for the detection of intertextuality. Probably the full verse does not begin with person or place names, but on lower scales, for example phrases, the names could be easily matched with each other.

The Jaro distance d_j of two strings a and b is defined as:

$$d_j = \begin{cases} 0 & \text{if } m = 0 \\ \frac{1}{3}\left(\frac{m}{|a|} + \frac{m}{|b|} + \frac{m-t}{m}\right) & \text{otherwise} \end{cases}$$

where m stands for the number of characters that match and t stands for the half of the number of transpositions. Two characters of the strings a and b only match if they are equal and the distance between them is not farther than $\left\lceil \frac{\max(|a|,|b|)}{2} \right\rceil - 1$.

The Jaro-Winkler distance gives an extra weights p to a prefix, with a maximum length of ℓ. Given two strings s_1 and s_2, their Jaro-Winkler distance d_w is:

$$d_w = d_j + (\ell p(1 - d_j))$$

where d_j stands for the Jaro distance between the strings a and b, ℓ is the length of the common prefixes of the strings. In our case, the length of the common prefix is 3. The default value for the constant p in the work of Winkler is $p = 0.1$, this is the value we also used.

An example: Given the strings a 'king' and b 'knight', then $m = 4$, $|a| = 4$ and $|b| = 6$. There are two mismatching characters, i/n and n/i, so $t = \frac{2}{2} = 1$. Therefore, the Jaro score is

$$d_j = \frac{1}{3}\left(\frac{4}{4} + \frac{4}{6} + \frac{3}{4}\right) = 0.806$$

Now we compute the Jaro-Winkler distance. We calculate $\ell = 1$, so

$$d_w = 0.806 + 1 * 0.1 * (1 - 0.806) = 0.825$$

Longest common substring The longest common substring is the longest string that is a substring of two or more strings. For example, the strings a =‘abcabc’ and b =‘adcbac’ have as longest common substring the string LCS =‘acbc’, with a =‘a-c-bc’ and b = ‘a-cb-c’. The normalized distance is the length of LCS divided by the length of the shortest string, so $\frac{LCS}{\min(|a|,|b|)} = \frac{4}{6} = \frac{2}{3}$.

This metric is very the same as the Levenshtein distance, but has another cost system. In the Levenshtein distance, a substitution or deletion has a cost of 1, but for the longest common substring it has no cost. The longest common substring only consider characters that are the same, and does not consider non-matching characters.

Most frequent k characters The most frequent k characters is a string distance metric and is developed for the quickly estimating of how similar two strings are. It simply counts the number of characters, and selects the k most frequent characters. Then the frequencies of the characters that occur in the most frequent k characters of both strings are summed up.

This technique is developed because the Dice's coefficient and similar functions takes too much time. The most frequent k characters algorithm is very fast and easy and therefore a good candidate for big data studies.

An example of the most frequent k characters is: take the strings a =‘abaabc’ and b =‘aacddd’, and take $k = 2$. The most frequent two characters of string a are ‘a’, which has a frequency of three and ‘b, which has a frequency of two. For the string b, the most frequent two characters are ‘d and ‘a, with a frequency of three respectively two. Only the character ‘a is most frequent in both strings, so the score of the algorithm is in this case the sum of the frequency of ‘a in both strings, so it is five. The score of the most frequent k characters is normalized by the sum of the lengths of both strings.

Dice's coefficient The Dice's coefficient is a statistic used for comparing the similarity of two samples. It is defined as:

$$Q = \frac{2|A \cap B|}{|A| + |B|}$$

with A and B are the characters, n-grams, words or word n-grams of the strings a and b. Because of its definition, it is not necessary to normalize Dice's coefficient.

3.4 High-level grammatical approach

The high-level grammatical approach simply counts the occurrences of features within a verse. For example, the count features of the verse V_i are:

$$count(V_i) = \left[\left[\sum_i X_{A,1}, \ldots, \sum_i X_{A,\alpha} \right], \ldots, \left[\sum_{i,j,k,l} X_{D,1}, \ldots, \sum_{i,j,k,l} X_{D,\delta} \right] \right]$$

There are a few ways to define the features of a pair of verses. Remember that a pair of verses may be a cross-reference if it is a positive object, otherwise it is not a cross-reference. The first way is simply subtracting the features of both verses and take the absolute of the result:

$$count_{sub}(P_{x,y}) = |count(V_x) - count(V_y)|$$

The second way is adding the features of both verses, which is a variant of subtracting and is known that it could also deliver good results:

$$count_{add}(P_{x,y}) = count(V_x) + count(V_y)$$

The third way is concatenating the features of both verses:

$$count_{con}(P_{x,y}) = [count(V_x), count(V_y)]$$

In the previous equation the features of full verses are used. It is also possible to use the features of separate parts of the verses, for example the count features of the phrase. But then the following problem occurred. A verse has a variable number of sentences, clauses, phrases and words, so we cannot simply concatenate all count features of all individual clauses. So we computed the mean and standard variation of the number of sentences, clauses, phrases and words within a verse and uses this to select a fixed number of clauses what I shall use for the count features on clause level. Table 3.2 shows these means and standard variations.

3.5 Multi-scale distance approach

The third approach involves the scales of the text for the detection of intertextuality. As Figure 3.1, shows, a verse consists of sentences. A sentence comprises clauses, a clause consists of phrases, and finally a phrase comprises words. In Hebrew this is as usual as in English.

First, we have to answer the question how to measure similarity between parts of verses. In theory, the metric that works best for the baseline should also work best for the multi-scale distance approach, so we use that metrics for this approach.

The distances between two verses are computed in the same way as the outer product or the Cartesian product is computed. Table 3.5 shows this clearly. In the example in the table, the most intertextuality could be find between phrase 1 of verse 1 and phrase 2 of verse 2. Computing the distances on other scales works on the same way as in this example. The last step of the computation is to convert the matrix to a feature vector. That feature vector could be used as input for a classifier.

Now we give an example of the multi-scale distance approach. In Example 3, two verses are shown that have intertextuality. For simplicity, we use the equal word count as metric in this example.

		Verse 1				
		Phrase 1	Phrase 2	Phrase 3	$\cdots$	Phrase m
	Phrase 1	x	0.29	0.21	$\cdots$	0
	Phrase 2	0.85	x	0	$\cdots$	0
Verse 2	Phrase 3	0.43	0	x	$\cdots$	0
	$\vdots$		$\vdots$	$\vdots$	$\ddots$	$\vdots$
	Phrase n	0	0.17	0	$\cdots$	x

Table 3.5: The distances according to the multi-scale distance approach between two verses. There is a peak of intertextuality between phrase 1 of verse 1 and phrase 2 of verse 2.

In the beginning, God created the heavens and the earth.

Genesis 1:1 (ESV)

Thus says God, the Lord, who created the heavens and stretched them out, who spread out the earth and what comes from it, who gives breath to the people on it and spirit to those who walk in it:

Isaiah 42:5 (ESV)

Example 3: An example to illustrate the working of the multi-scale distance approach

Table 3.6 shows the word count between the full verses (row 1), half verses (rows 2-3) and clauses (rows 4-13). As the table shows in row 2, the exact location of intertextuality for the half-verses is between the only one half-verse of the verse 1 and the first half-verse of verse 2, because this row has the highest value for the equal word count on the half-verse scale. On the clause scale, there is a peak of intertextuality in row 9-11.

#	Scale	Left verse	Right verse	Equal word count
1	Full verse	In the beginning, God created the heavens and the earth	Thus says God, the Lord, who created the heavens and stretched them out, who spread out the earth and what comes from it, who gives breath to the people on it and spirit to those who walk in it:	19
2	Half verse	In the beginning, God created the heavens and the earth	Thus says God, the Lord, who created the heavens and stretched them out, who spread out the earth and what comes from it,	15
3	Half verse	In the beginning, God created the heavens and the earth	who gives breath to the people on it and spirit to those who walk in it:	4
4	Clause	In the beginning	Thus says God, the Lord,	1
5	Clause	In the beginning	who created the heavens and stretched them out,	1
6	Clause	In the beginning	who spread out the earth and what comes from it,	1
7	Clause	In the beginning	who gives breath to the people on it	1
8	Clause	In the beginning	and spirit to those who walk in it:	1
9	Clause	God created the heavens and the earth	Thus says God, the Lord,	3
10	Clause	God created the heavens and the earth	who created the heavens and stretched them out,	4
11	Clause	God created the heavens and the earth	who spread out the earth and what comes from it,	4
12	Clause	God created the heavens and the earth	who gives breath to the people on it	2
13	Clause	God created the heavens and the earth	and spirit to those who walk in it:	1

Table 3.6: Number of half-verses per verse versus the number of occurrences

As described earlier, the main problem of this approach is that language is very variable. There are a different number of sentences within verses, there are also a different number of clauses per sentences. A classifier can only process a feature vector of a fixed length, so we should find a way to map a feature vector

that consists of a variable number of distances to a feature vector of fixed length.

There is a bright escape for this problem, but it has some drawbacks. Most of the Hebrew verses could be divided into two so-called half-verses. In Hebrew poetics, it is normally to use parallelism, a figure of speech in which the first sentence pose a statement, and a second sentence tries to deepen out this first sentence, for example by saying it again in other words. An example is 'The merchant uses dishonest scales; he loves to defraud'. The first half-verse is 'The merchant uses dishonest scales', the second half-verse is 'he loves to defraud'. There are always a fixed number of distances between all half-verses of two verses, so the half-verses are a good candidate for this approach.

There are two problems with this escape. First, not all verses of the Hebrew Bible have such a clear separation inside it. Table 3.7 shows the frequency of the number of half-verses per verse. For the experiments on the half-verses, only the verses that consist of two half-verses are selected, the others are neglected. So this approach could only be applied on a part of the Hebrew Bible. Second, the escape could only be applied on half-verses, because the frequency of the number of sentences, clauses and phrases per verse is too diverse. If the escape is applied on one of these scales, only a very small part of the Hebrew Bible would be evaluated.

Number of half-verses per verse	**Frequency**
1	616
2	20.333
3	1.132
4	1

Table 3.7: Number of half-verses per verse versus the number of occurrences

The first solution for the problem of the variable length of sentences, clauses and phrases is simply adding zeros to make all feature vectors of a fixed length. The main drawback of this solution is that there are a lot of zeros necessary because there is always a very long sentence, clause of phrase, which also needs to be classified. So we get a lot of features in this solution, especially on clause and phrase scale.

The second solution that we used for dealing with the variable length of sentence, clauses and phrases is by a combination method. We used five combination methods, which are: mean, median, minimum, maximum and product. These combination methods could be used in a two-fold way.

The first way is by applying the combination method on each sentence, clause and phrase per verse. For example, we select the maximal distance out of all distances between all sentences of a cross-reference.

The second way is more complicated. It can only be used when multiple distance metrics are used. The combination methods are applied on each distance metrics. For example, we use ten distance metrics, then the maximum values of

all individual distance metrics (that are computed over all distances of a scale) are selected.

Chapter 4

Evaluation

In this chapter, the evaluation and validation of our research is discussed. First is started with the explaining of the label set we used (Section 4.1). Then we elaborate on the different validation datasets we selected for our experiments (Section 4.2). Next, we explain the working of the classifiers that we are using for our experiments (Section 4.3). After that, we describe which performance measurement we used for our experiments (Section 4.4). Finally, we explain the working and optimisation of Copyfind, a plagiarism detector that is used to compare the performances of our approaches with existing solutions. (Section 4.5).

4.1 Label set

Through the centuries, a lot of effort is made to find cross-references of verses in the Bible. This is very important in the Christian tradition, because one of the rule in explaining the Bible is comparing verses with other verses. Especially verses that are difficult to explain can be explained by using other relevant verses.

These cross-references are a good indicator of intertextuality. Intuitively, intertextuality means that two verses are related in some way, so the cross-references are taken as positive examples of intertextuality. However, there might exists some cross-references that are in the list because of theological reasons or because another reason which has nothing to do with intertextuality. Therefore, we combine the multiple lists of these cross-references by taking the intersection.

Table 4.1 shows these translations, and also contain the number of cross-references within each translation. In the last row of that table, the word 'Combined' means the intersection of the cross-references of the three translations.

Translation	Count
ESV	46.841
NAS	52.761
NIV	75.008
Combined	10.778

Table 4.1: Number of cross-references

An important question is: how to sample the negative examples? The Hebrew Bible has 23.213 verses, so there are $\frac{n(n-1)}{2} = \frac{23213 \cdot (23213-1)}{2} = 269.410.078$ possible pairs of verses. This means in the case of the combined lists we have 10.778 positive examples and 269.399.300 negative examples. Due to time limits, we chose to randomly sample the negative objects to get the same number of negative objects as the number of positive objects.

Two examples of cross-references can be found in Example 4 and 5. Two other examples of no cross-reference are put in Example 6 and 7. The intertextuality could clearly be seen in the positive examples, but there is definitely no intertextuality in the negative examples.

What is man, that you make so much of him, and that you set your heart on him,
Job 7:17 (ESV)

O LORD, what is man that you regard him, or the son of man that you think of him?
Psalms 144:3 (ESV)

Example 4: An example positive o intertextuality

And the people of Israel who were present at Jerusalem kept the Feast of Unleavened Bread seven days with great gladness, and the Levites and the priests praised the LORD day by day, singing with all their might to the LORD.
2 Chronicles 30:21 (ESV)

Seven days you shall eat unleavened bread, and on the seventh day there shall be a feast to the LORD.
Exodus 13:6 (ESV)

Example 5: Another positive example of intertextuality

But they soon forgot his works; they did not wait for his counsel.	*Then the boundary goes in another direction, turning on the western side southward from the mountain that lies to the south, opposite Beth-horon, and it ends at Kiriath-baal (that is, Kiriath-jearim), a city belonging to the people of Judah. This forms the western side.*
Psalms 106:13 (ESV)	Joshua 18:14 (ESV)

Example 6: A negative example of intertextuality

When Eber had lived 34 years, he fathered Peleg.	*Then they will know that I am the LORD, when I have made the land a desolation and a waste because of all their abominations that they have committed.*
Genesis 11:16 (ESV)	Ezekiel 33:29 (ESV)

Example 7: Another negative example of intertextuality

4.2 Validation datasets

To evaluate and validate the experiments, we need different datasets. We selected seven datasets. Two of them are easy datasets, two are intermediate datasets and two are difficult datasets. The last dataset contains all books and therefore all available cross-references.

For the dataset of a single book, only the cross-references that are within that book are used. Only that books are selected that have a reasonable number of cross-references in itself. We selected the following easy, intermediate and difficult datasets:

(a) As easy books we selected 'Genesis' and 'Ezra'. Genesis has a larger class size and Ezra has a very small class size.

(b) As intermediate books we selected 'Job' and 'Joshua'. These books have two different type of genres.

(c) As difficult books we selected 'Psalms' and 'Isaiah'. These books have a relatively high number of cross-references. However, the cross-references are considered as more difficult to recognize because of the type of genre

and writing style of the books. Both books have a different type of genre and writing style.

Table 4.2 displays the size and difficulty of the datasets that are selected.

#	**Book**	**Size**	**Difficulty**
1	Genesis	699	Easy
2	Ezra	84	Easy
3	Joshua	283	Intermediate
4	Job	508	Intermediate
5	Psalms	2.225	Difficult
6	Isaiah	965	Difficult
7	All books	10.778	Not relevant

Table 4.2: Details of the standard test datasets. The columns size means the number of positive objects within a dataset.

4.3 Classifiers

For nearly all our experiments, two classifiers are used to verify the results, which are the nearest mean classifier and the logistic classifier. Other classifiers are in the most cases outperformed by the logistic classifier, so the logistic classifier is a good choice. The logistic classifier is a fast and relatively simple classifier which can easily handle high-dimensional data. The nearest mean classifier is involved because of its simpleness and fastness, but it hardly outperformed the logistic classifier in our experiments.

The only classifier that has performances that are a little worse than the performances of the logistic classifier, is the nearest neighbour classifier. However, this classifier becomes very slow when large datasets are used. In the experiments, classifiers needs to train on more than 10.000 objects, so these datasets are far too large for these classifiers.

4.3.1 Nearest mean classifier

We explain first the nearest mean classifier. A classifier has always to phases, a training phase and a testing phase. In the training phase, the classifier learns from examples. In the testing phase, the performances of the classifier are evaluated.

In the training phase of the nearest mean classifier the mean (center) is determined of the objects of each class. In Figure 4.1, the mean of the blue class is $[0.0, -0.2]$, and the mean of the blue class is $[2.0, -0.1]$.

In the next phase, when a new observation has to be classified, the distance between a new object and the means of the classes are calculated. The new object is assigned to the class which mean is the closest. In the figure, a black

line is drawn. This is the classification boundary. New objects on the left of this boundary are assigned to the blue class, and objects on the right are assigned to the red class. If we know the label of these new observation, we can check whether the classifier classified the new observation correct or false.

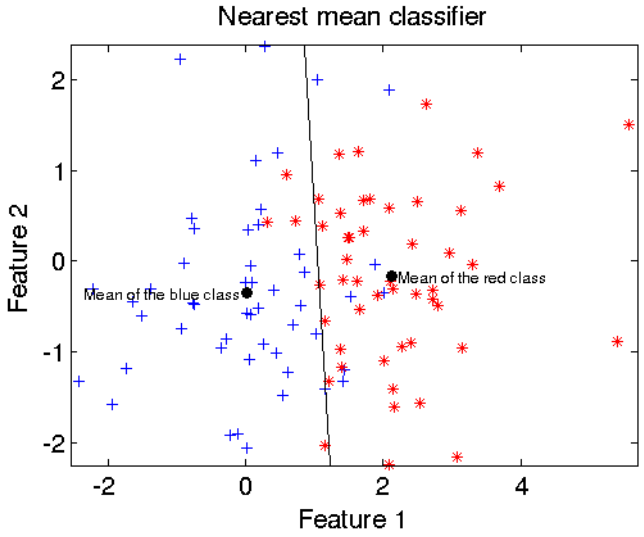

Figure 4.1: The decision boundary of the nearest mean classifier on a simple classification problem.

Let μ_1 be the mean of class ω_1 and μ_2 be the mean of class ω_2. For the nearest mean classification, it holds that $\mathbf{w} = \mu_1 - \mu_2$. Now the decision boundary can be described by

$$g(x) = \mathbf{w}^T\mathbf{x} = 0$$

and the classification can be defined by

$$\text{classify } x \text{ to } \begin{cases} \omega_1 & \text{if } \mathbf{w}^T\mathbf{x} \geq 0 \\ \omega_2 & \text{if } \mathbf{w}^T\mathbf{x} < 0 \end{cases} \tag{4.1}$$

The main advantage of the nearest mean classifier is that it is a very simple and fast classifier. It solves simple problems easily, and has no problems in dealing with enormous datasets. The main disadvantage is that it could not solve complex problems. The nearest mean classifier assumes a Gaussian d An example of this is shown in Figure 4.2.

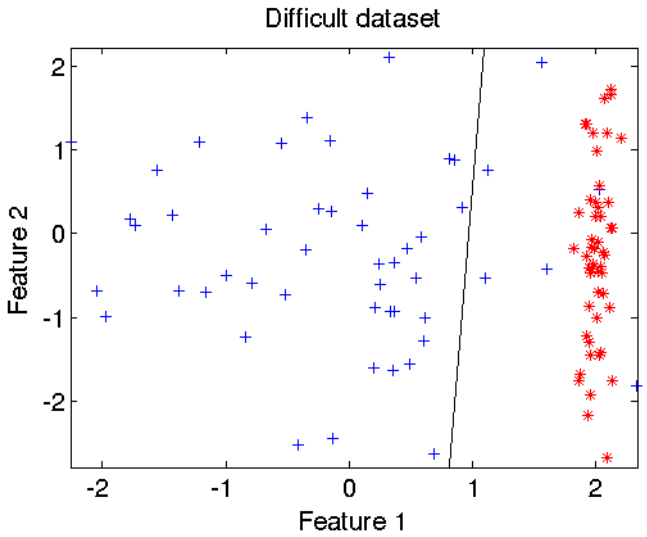

Figure 4.2: The decision boundary of the nearest mean classifier on a difficult classification problem.

4.3.2 Logistic classifier

In logistic discrimination the logarithm of the likelihood ratios is modelled via linear functions. That is

$$\ln \frac{P(\omega|\mathbf{x})}{P(\omega_M|\mathbf{x})} = w_{i,0} + \mathbf{w_i}^T\mathbf{x}$$

This can be simplified to

$$P(\omega_1|x) = \frac{e^{w_0+\mathbf{w}^T\mathbf{x}}}{1+e^{w_0+\mathbf{w}^T\mathbf{x}}}$$

$$P(\omega_2|x) = \frac{1}{1+e^{w_0+\mathbf{w}^T\mathbf{x}}}$$

To estimate the set of the unknown parameters, a maximum likelihood approach is employed. Optimization is performed with respect to all parameters. The optimization formula is equal to

$$L(\theta) = \ln\left\{\prod_{k=1}^{N_1} p(\mathbf{x}_k^{(1)}|\omega_1;\theta)\prod_{k=1}^{N_2} p(\mathbf{x}_k^{(2)}|\omega_2;\theta)\right\}$$

It is too time-consuming to calculate all possible θ so the parameters are optimized by using EM (Expectation-Maximization). This algorithm iterates over the expectation step and the maximization step. In the expectation step a function for the expectation of the log-likelihood is created. This function is

evaluated by using the current estimate for the parameters. In the maximization step, parameters are computed, which maximizes the expected log-likelihood found in the expectation step. These parameter-estimates are used to compute the distribution of the latent variables in the next expectation step.

There is a close relation between the logistic classifier and the LDA (Linear discriminant analysis). The difference is that the LDA assumes the classes have an equal class covariance matrices and are Gaussian distributed, but the logistic classifier can deal with classes that are Gaussian distributed and do not have equal class covariance matrices.

Figure 4.3 displays the decision boundary of the logistic classifier on a complex dataset. It clearly has a better decision boundary than the nearest mean classifier.

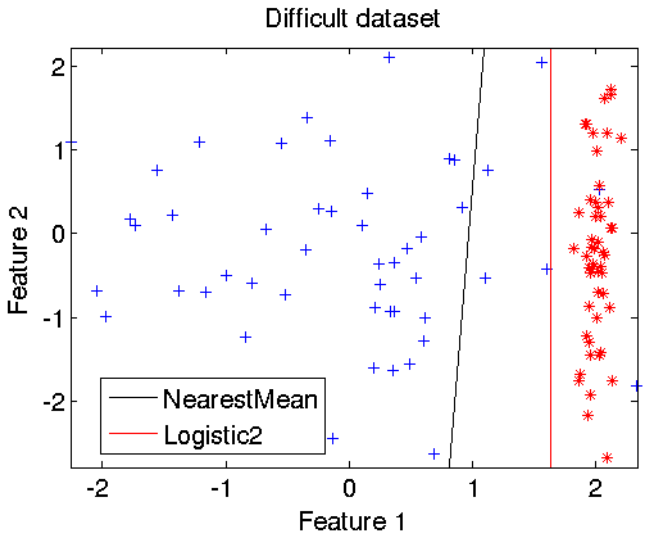

Figure 4.3: The decision boundary of the logistic classifier on a complex problem. It is clear that the logistic classifier provides a better decision boundary than the nearest mean classifier.

4.4 AUC

As explained earlier in this report, there is a high imbalance between the classes. There are a lot more negative examples than positive examples. For this reason, the Area Under Curve (AUC) is used as validation measure. The AUC is known to be insensitive to class imbalance. In all the experiments, a class ratio of 1:1 is used by randomly sample negative object in such a way that the number of positive objects is as large as the number of negative objects. The AUC is a good predictor for the goodness of classifiers when all negative objects are used.

Figure 4.4 displays the Receiver Operating Characteristic (ROC) curve of a

dataset that has two classes. The ROC curve is the trade-off between the two classes. In the figure, it is the black line. The AUC means Area Under Curve, and in the figure it is the grey area. This means that the lower the AUC, the better the classifier. In some literature, the axis are turned, in that cases a higher AUC means a better classifier, but in our research it is used in the way as displayed in the figure.

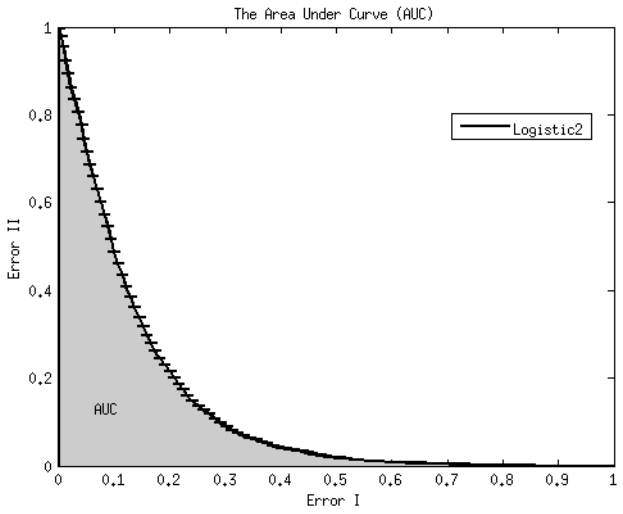

Figure 4.4: The Area Under Curve (AUC)

4.5 Copyfind

The plagiarism detector Copyfind [4] is used to compare the approaches with the work of others. Copyfind compares text with one another to determine whether they share words in phrases.

The essence of Copyfind is that it searches in two documents for sentences that have common phrases. If a certain number of sentences is found, then the documents are marked as plagiarism. Copyfind has a several options to optimize the matching algorithm. These parameters are the following:

1. **Shortest phrase to match**: The number of words that a phrase minimal should have to be a candidate for matching.

2. **Fewest matches to report**: The number of matching sentences that will cause Copyfind to report two documents as plagiarism.

3. **Most Imperfections to allow**: The maximum number of non-matches that Copyfind will allow between perfectly matching portions of a phrase.

4. **Minimum percentage of matching words**: This is the minimum percentage of perfect matches that a phrase can contain to be considered as a match.
5. **Ignore all punctuation**
6. **Ignore outer punctuation**: All punctuation characters that appear to the right or left of a word are ignored.
7. **Ignore numbers**
8. **Ignore letter case**
9. **Skip non-words**
10. **Skip words longer than n characters** This choice allows Copyfind to skip over many non-textual items, including filenames, URL, image data, and other word-processor junk.

We optimized these options so that Copyfind is able to detect intertextuality. Remember that plagiarism is related to intertextuality.

For the optimization of Copyfind, 500 positive examples and 500 negative examples of intertextuality are selected. This is repeated ten times, and the standard deviations are displayed in the figures.

Figure 4.5 shows the performances for different values of the first parameters, which was the shortest phrase to match. For the English translation, ESV, two words in a phrase that match are the best value, but for the lexemes of the Hebrew text, only one word is sufficient for the counting as a candidate phrase. This can be explained by the fact that in the Hebrew text just lexemes are used, and non-relevant words are omitted into the text. So phrases of one word are relevant. However, the English translation contains all these non-relevant words, like 'a' and 'the', etcetera. So in that case a phrase of a minimal length of can be considered as a candidate phrase.

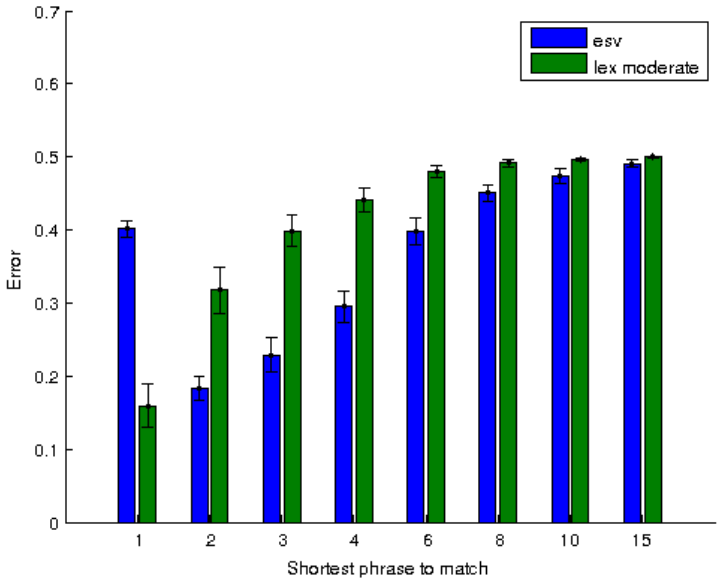

Figure 4.5: Performances for different values of shortest phrase to match.

Next, Figure 4.6 displays the performances for different values of the parameter of the fewest matches to report. The differences between the ESV and lex moderate are caused by the fact that the other parameters are optimized for the ESV and not for lex moderate. Therefore, it is important to compare the results of a translation with itself and not with the other translation. It can be concluded that a value of 1 or 2 for the parameter fewest matches to report are the best values.

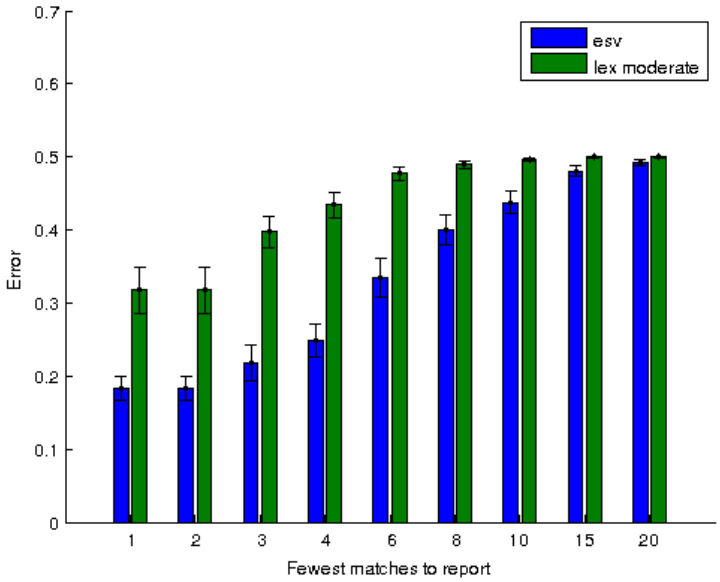

Figure 4.6: Performances for different values of fewest matches to report.

The following parameter that must be optimized is most imperfections to allow. The results can be found in Figure 4.7. It is clear that for both values the best value is 4.

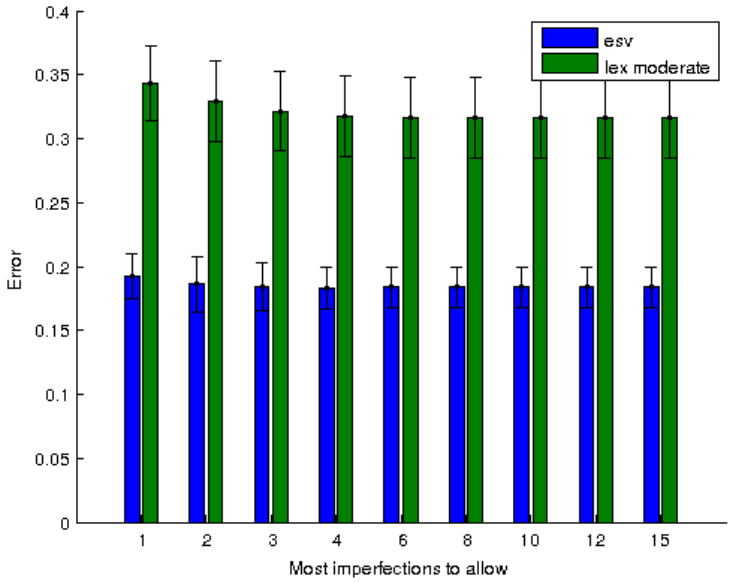

Figure 4.7: Performances for different values of most imperfections to allow.

For optimizing the parameter minimal percentage of matching words, the results for the ESV and lex moderate again are the same. This can be seen in Figure 4.8. Both translations have the best result at a value of 10 for the parameter.

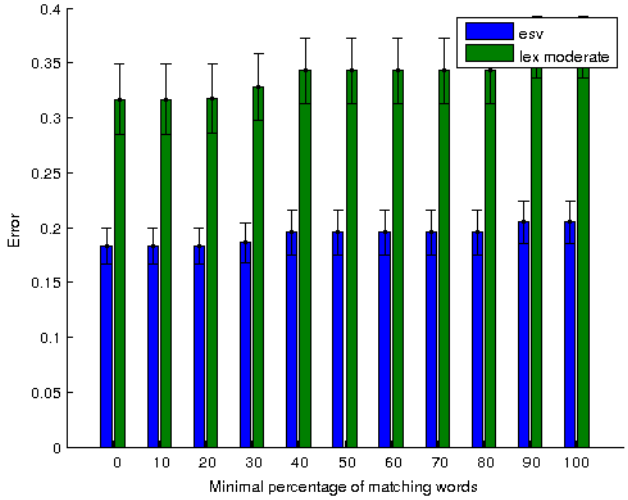

Figure 4.8: Performances for different values of minimal percentage of matching words.

The fifth to ninth parameter could have the value true or false. Turning these parameters on or off does not have significant influence on the performance, so we can neglect these parameters.

Figure 4.9 shows the results of the optimization of the last parameter. The best results are provided when words longer than 7 or 10 characters are skipped. With a higher n, the results remain stable, so it can be concluded that skipping words does not lead to a better performance. For the ESV, a value of 10 performs a bit better than a value of 7, this is because the English language has longer words than the Hebrew lexemes. Therefore, the skipping of long words is turned off when using Copyfind as detector of intertextuality.

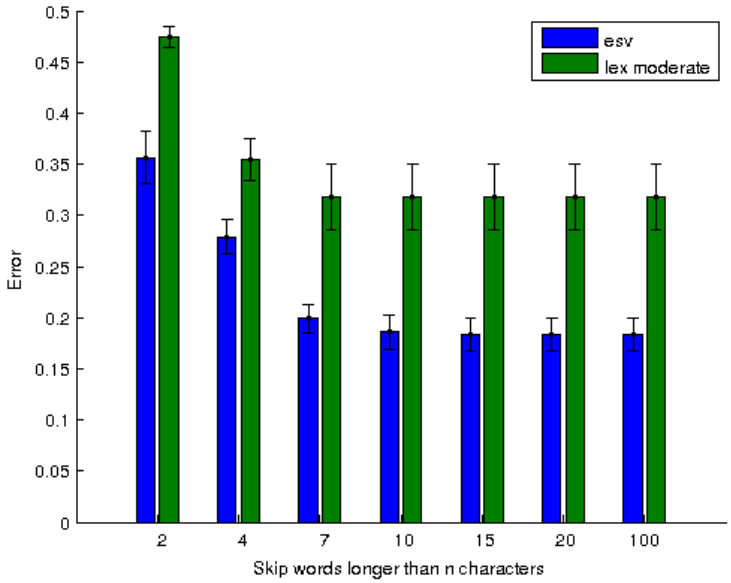

Figure 4.9: Performances for different values of skip words longer than n characters.

Chapter 5

Distance-based approach

This chapter describes the experiments and results for the distance-based approach. First, an experiment that evaluates the performances of single distances is described. Then an experiment that evaluates the performances of all distances is explained. After that, dimension reduction is applied to find the best distances. Next, an experiment is conducted to find the best classifier. Then the influence on the results of using different languages is figured out. Finally, some conclusions are drawn about the distance-based approach.

5.1 One distance

The distance-based approach uses a string distance metric to define the distance between two verses. The string distance can be applied on four scale, namely on character scale, word scale n-gram scale and word n-gram scale.

These distances are calculated from all the cross-references within the Hebrew Bible, and they are also computed from the pairs of verses that are not a cross-reference. The distances are normalized to take the length of the verses into account. For each single distance 10-fold cross-validation is applied to compute the AUC of the single distances. Intuitively, the data is linear separable, because a large distance stands for no intertextuality and a small distance stands for that the verses are related to each other. For that reason we use the nearest mean classifier.

Character scale The string metrics on character scale have average performance comparing to the other scales. This is not surprising because letters are very redundant in sentences. This is also the case in English. For example, the letters of the words THING and NIGHT are exactly the same, but are not related in any way with each other.

Furthermore, the sequence of characters is also not a good indicator for intertextuality. For example, the word BLOOD and BLOOM have nearly the same sequence but have a total different meaning.

The string metrics on character scale are evaluated, but the best performing has an AUC performance of 0.2584 with a standard deviation of 0.0021 so we will not describe the results of the character scale into detail.

Word scale The distance metrics are not able to handle characters, but what are the performances of the word scale? Words are a better indicator of intertextuality. When two sentences contain the same word, there is a higher chance that there is intertextuality than if two sentences share a character. Figure 5.1 shows the performances of the distance metrics on word scale.

Dice's coefficient is performing very well on word level. The reason is that similar words are very indicative for intertextuality. The sharing of the sequence of words is less important than the fact that two verses share the same words. So the distances in which the sequence of words is important are outperformed by the Dice's coefficient. Remember that the distances in which the sequence are important are the Levenshtein distance, Jaro-Winkler, longest common substring and most frequent k characters.

The number of equal or nearly equal weighted words (the grey bars) have average performance. A reason for this might be that the manner of weighing is not correlated with intertextuality. So the parts of the verses that contain intertextuality occur not more or less within the corpus of the Hebrew Bible, so then the weighting does not lead to better recognition of intertextuality.

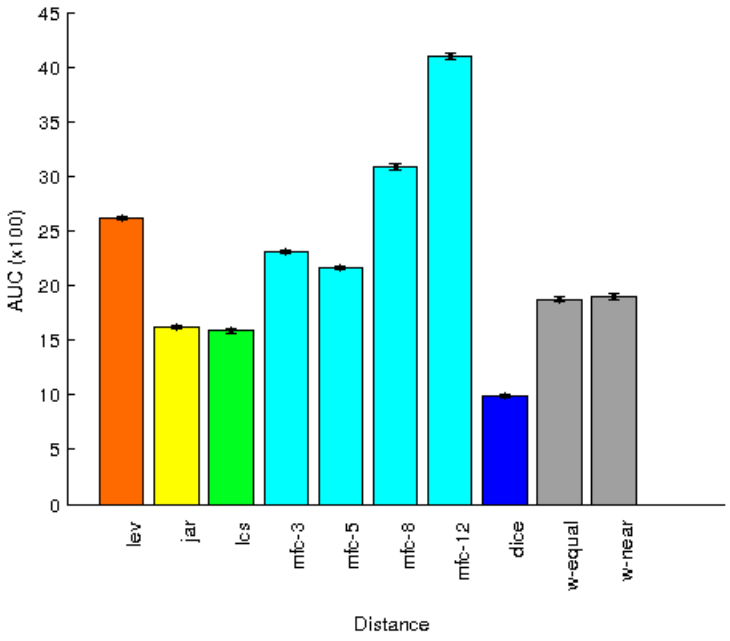

Figure 5.1: The performance of each individual distance feature on word scale. The figure contains the distances Levenshtein (lev), Jaro-Winkler, (jar), longest common substring (lcs), most frequent k characters (mfc, with after the hyphen the value of k, Dice's coefficient (dice), the sum of the weighted equal words (w-equal) and the sum of the weighted words that are nearly equal (w-near)

n-gram scale The n-gram scale is the best performing scale. Almost all distances have a good performance on this scale. Figure 5.2 shows the results of the distances for n-grams with multiple n.

The figure clearly shows that a $n \geq 4$ do not lead to good performances. In the most cases, $n = 3$ is the best option, although a $n = 2$ has also good performances. The only exception is Dice's coefficient with $n = 4$ and space representation space-1, which has performances comparable with the same metric but then for 2-grams or 3-grams. The reason for this is that the space is included within its n-grams, so a $n+1$-gram of space-1 contains the same information as an n-gram of space-3.

The n-gram scale is comparable by the word scale, although there are difference. Most Hebrew lexemes have three characters, so a word consists mostly of two 2-grams or one 3-gram. When all the lexemes in a verse have three characters, then the distance on n-gram scale are the same as the distance on the word scale. We will see that the n-gram scale combines the advantages of the word scale with own advantages so that the n-gram scale outperforms all other

scales.

The Levenshtein distance on n-gram scale outperforms the Levenshtein distance on word scale. Considering the performance on word scale and that a word consists of n-grams, it can be concluded that n-grams add additional information for the string metric comparing to the word scale.

The Jaro-Winkler distance has the same performance(for $n = 3$ as on word scale. A reason might be that the number of transpositions increases more than the number of matching n-gram decreases (see the definition of the Jaro-Winkler distance).

The longest common substring also has the same performance as on word scale. This is not strange because Hebrew words have most times a length of three vowels. So the n-grams are often similar to the words.

The performances of the most frequent k n-grams are comparable to the word scale. However, the most frequent k 3-grams have worse performances than the most frequent k 2-grams, which is not the case by the other metrics. There is no clear explanation for this conspicuity.

The Dice's coefficient measure have the best performance on this scale, and they are comparable to the performance of the Dice's coefficient on word scale. Three different manners of dealing with spaces are evaluated. All three methods have comparable results, so it does not matter how to deal with spaces. The most intuitive manner is space-3, in which the spaces are ignored and the n-grams are taken from full words.

The weighted Dice's coefficient do not outperform the standard Dice's coefficient metric. As already said, the frequency of words and n-grams are not related to the fact that two verses are intertextuality or not.

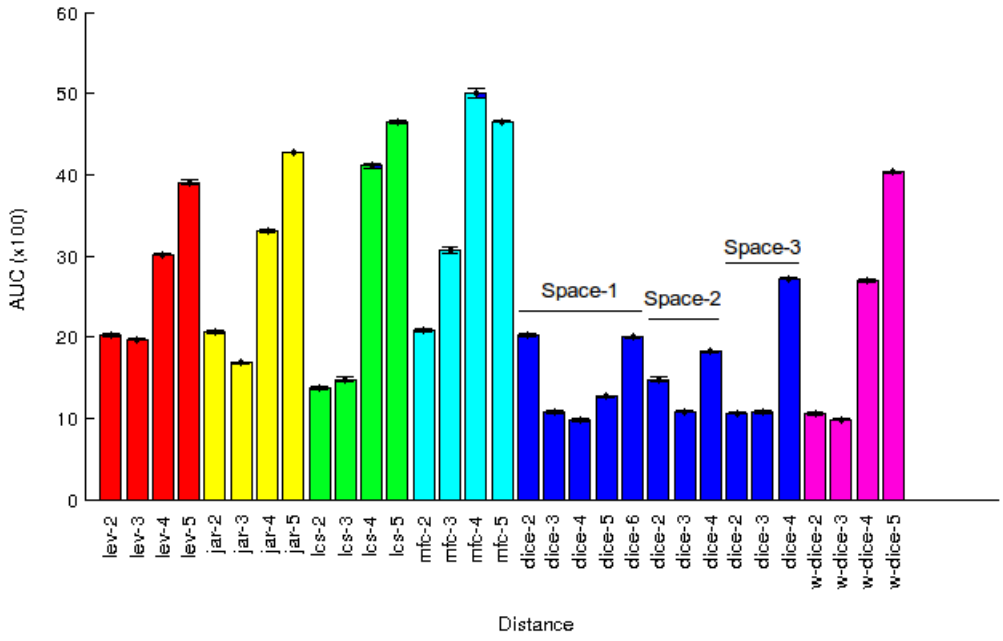

Figure 5.2: The performance of each individual distance feature on the n-gram scale. Each distance has a name and a number. The number stands for the parameter n. The figure contains the distances Levenshtein (lev), Jaro-Winkler (jar), longest common substring (lcs), most frequent k characters (mfc, with $k = 10$), Dice's coefficient (dice) and the weighted Dice-coefficient (w-dice). The Dice's coefficient has three different manner to deal with space. Space-1 means that the spaces are included in the n-grams. Space-2 means the spaces are neglected and the words are concatenated, and space-3 means that the spaces are neglected and there is a stop between words. For the other distances, space-3 is used.

Word n-gram scale The performances on word n-gram scale are the worst comparing to the other scales. The reason is simply that two subsequent words are not a good indicator for intertextuality. When the n parameter increases, the performances entirely decreases, so that random guessing have equal results. Figure 5.3 shows this clearly. The only cross-references in which the word n-gram are performing, are that one that contains very explicit intertextuality, for example when they are citing each other. The chance that two verses that have intertextuality contain two words in the same sequence is very small. Intertextuality could also occur when two words are the same but in another sequence, or when a synonym is used.

So it is reasonable when the higher n, the rare it occurs in two verses. In that case intertextuality is only detected when two verses contain exactly the same

words in the same sequence, this is only the case in very explicit intertextuality.

Dice's coefficient has the best performances on this scale. The reason for this is that the sequence of the word n-grams does not matter for the Dice's coefficient, but is does for the other metrics. So when there is yet one word n-gram that has been share by both verses, this is a small indicator for intertextuality. However, it still has an AUC error of 0.3014 with a standard deviation of 0.0003.

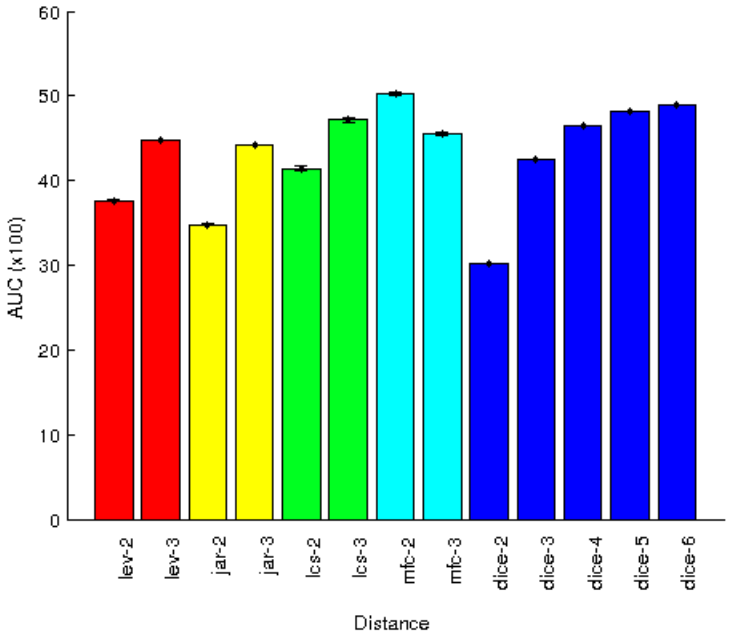

Figure 5.3: The performance of each individual distance feature on the word n-gram scale. Each distance has a name and a number. The number stands for the parameter n. The figure contains the distances Levenshtein (lev), Jaro-Winkler (jar), longest common substring (lcs), most frequent k characters (mfc, with $k = 10$) and Dice's coefficient (dice).

5.2 Multiple distances

The next step is to put all single distances into one dataset and apply the classifiers onto it. Figure 5.4 shows the performances of all datasets. The figure includes the standard variation over ten repetitions.

The idea behind combining multiple distances is that each distance adds some additional information to the classifier. For example, the cross-references that are not recognized by the distances of the word scale, could be recognized by distances from the n-gram scale. So each distance could potentially lead to

a small improvement.

According to the figure, it is clear that combining all distances deliver better performance than using one single distance. Using all distances provide an AUC result of 0.063 with a standard deviation of 0.0011, and the best performing single distance has an AUC of 0.0978 with standard deviation of 0.0013.

The second remark is the standard variation is large when the number of objects is low. The datasets Ezra and Joshua has the least number of objects, and have the greatest standard variation. The dataset that contains all object, called All books, has the smallest standard variation.

Another thing that attracts attention is that not always a single classifier has the best performance. For the books Genesis, Ezra, Joshua, Job and Isaiah, the nearest mean classifier has the best performances, but for the other books the logistic classifier is the best. The logistic classifier has also the best performances on the most important dataset, that contains all objects. An explanation might be that the nearest mean classifier cannot deal with higher object sizes, or that the logistic classifier improves it performances when there are more objects to train on. It can be noticed that the differences are not so large as it seems, the AUC difference is just around 0.02 for the datasets Genesis, Ezra and Psalms comparing to All books.

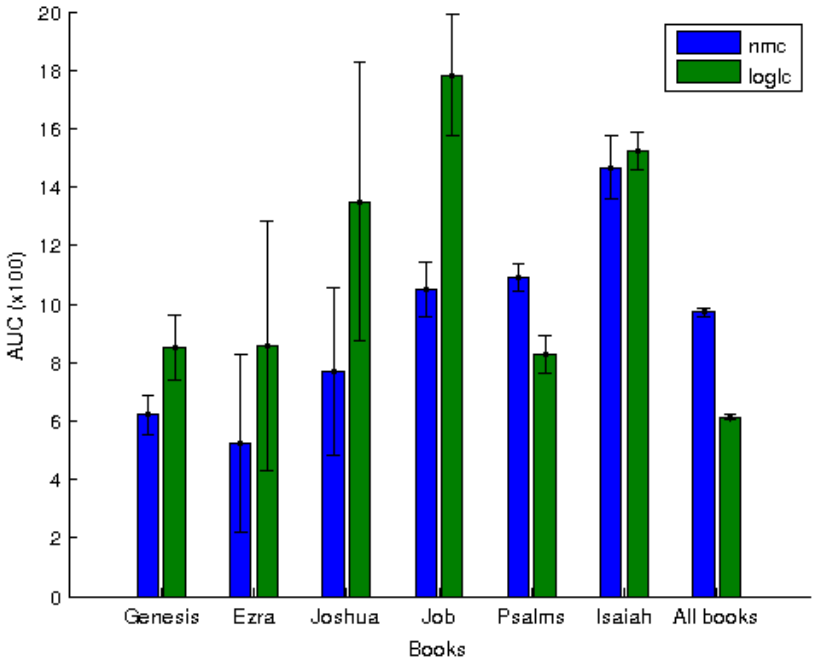

Figure 5.4: The performance of all distances per dataset using the nearest mean classifier and the logistic classifier.

5.3 Classifiers

Figure 5.5 shows the performances of several classifiers on the different datasets. Nine different classifiers are used, these are described in Table 5.1. The experiment is repeated ten times and the standard deviation is shown in the figure.

Classifier	**Abbreviation**
Nearest Mean classifier	nmc
Linear discriminant classifier	ldc
Quadratic discriminant classifier	qdc
Fisher's linear classifier	fisherc
Logistic classifier	loglc
k-nearest neighbour classifier (using $k = 1$)	knnc
Parzen classifier	parzenc
Decision tree classifier	treec
Support Vector Machine	svc

Table 5.1: The classifiers that are evaluated and their abbreviations.

As the figure shows, the results of the linear discriminant classifier, the quadratic discriminant classifier, the Parzen classifier and the decision tree classifier poor enough to be ignored. The other classifiers have comparable results. The three classifiers that have the best results, are the nearest mean classifier, the logistic classifier and the support vector machine. From these three, the logistic classifier performs overall the best.

So this experiment points out that using the logistic classifier is the best choice. The support vector machine is also a good choice, but it takes a lot more time to do experiments with this classifier, in particular when the number of objects is greater than 10.000. The nearest mean classifier is also a good choice, because it is very fast and simple.

The tree classifier has reasonable results on some datasets, but on the fifth and seventh dataset the performances are very bad. This may be due to the size of the datasets, applying pruning might prevent this.

Fisher's classifier has bad results on the dataset Ezra, but on the other dataset it performed quite well. The reason for this is that the Dataset Ezra has just 84 positive and 84 negative objects, so the classifier underfits. On the larger datasets, Psalms and All books, the fisher classifier has very good performances.

This experiment also reveals that complex classifiers are not accurate candidates for the distance-based approach. Indeed, simple and fast classifier outperforms the complex ones.

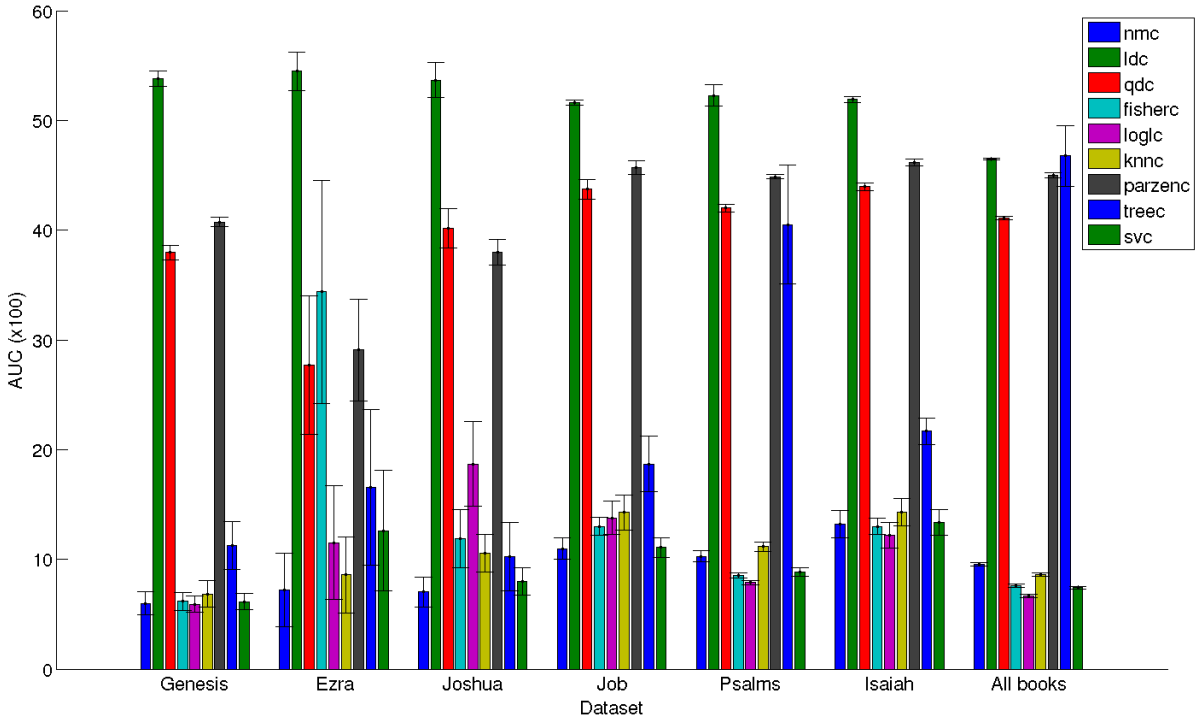

Figure 5.5: The performances of different classifiers on the different datasets using the distance-based features.

5.4 Dimension reduction

The next question that needs to be answered is: is it possible to decrease the AUC error even further with dimension reduction? To answer this question, two types of dimension reduction are evaluated, namely PCA and feature forward selection. The nearest mean classifier and logistic classifier are applied on both types of dimension reduction. Figure 5.6 displays the results. The blue line is the performance of the nearest mean classifier using 1 to 100 principle components of the data, the red line shows the results of the logistic classifier that is applied on 1 to 100 principle components. The green line displays the performance of the feature forward selected features using the nearest mean classifier, and finally the black line stands for the same features but now a logistic classifier is applied onto it. The experiment is repeated ten times and the standard deviations are denoted in the figure.

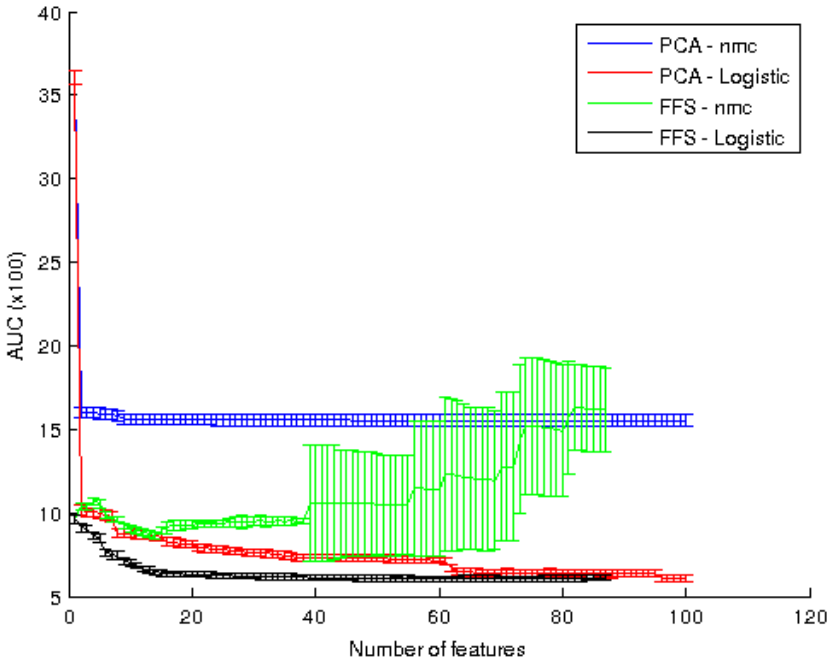

Figure 5.6: The feature size curve using dimension reduction. FFS means feature forward selection, nmc means nearest mean classifier and logistic means logisic classifier.

As the figure shows, the nearest mean classifiers (blue and green lines) are outperformed by the logistic classifiers (red and black lines) for each feature size. Further, the performances of the nearest mean classifiers do not improve on a greater feature size. The reason for this is that the nearest mean classifier is a simple classifier, that is not able to handle this kind of problem, which need a more complex classifier. Another reason is that the features are very correlated, so adding another feature do not really improve the performances because it contains no additional information. The figure shows that dimension reduction does not lead to better performance. There is no overfitting for the logistic classifier. Feature forward selection is the best option to select the best performing features.

The six string metrics that are selected first by the feature forward selection algorithm are

1. Weighted Dice's coefficient over 3-grams
2. Jaro-Winkler distance over strings
3. Dice's coefficient over 3-gram words

4. Weighted Dice's coefficient over 4-grams

5. Sum of words that are nearly equal

6. Dice's coefficient over characters

These six string metrics are not really surprising. We already knew that the weighted Dice's coefficient over 3-grams has the best performances comparing to other single distances. The other string metrics make only some little improvement, as can be seen in Figure 5.6, and they are selected because they are the least correlated with the string metrics that are already selected. A Dice's coefficient is selected four times, but only by two of them the Dice's coefficient deals with units over the same scale, namely over n-grams.

It seems that only the fourth string metric is very correlated by the first, but that is not the case because the Hebrew lexemes have in the most cases a length of three characters, and words that are longer are most of the times a contraction of two words. So then the 4-grams become interesting, because the 4-grams neglects words that have less than four characters. It is more probably that two sentences have intertextuality when they have one long word in common than when two sentences have intertextuality when they have one short word in common. So that might be reason why the fourth feature is selected.

5.5 Language

The final question for the distance-based approach that needs to be directed is: what are the performances of different translations and languages of the Hebrew Bible? So far, only the lexemes of the Hebrew text is used, and non-relevant words were deleted from the text. But the cross-references comes from English sources, and there exists a lot of other translations of the Hebrew Bible which can easily be evaluated.

Figure 5.7 shows the performances of four different translations. The performances are evaluated by the nearest mean classifier and the logistic classifier. It is clear that the logistic classifier always outperforms the nearest mean classifier. Furthermore, the results are quit stable. In the figure the standard deviation is shown, which is taken over ten times.

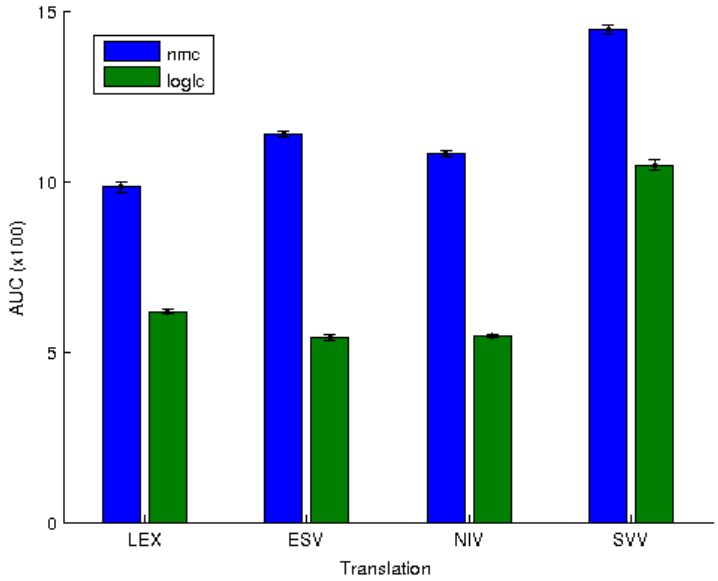

Figure 5.7: The performance of different translations. LEX is not a translation, but are the Hebrew lexemes.

Surprisingly, the lexemes of the Hebrew text are outperformed by the two translations in English, the ESV and NIV. This is surprising because the text of the English translation is not processed in any way, so it may contain a lot of non-relevant words and conjugations. A reasonable explanation is that the cross-references are based on these two translations, and in the English language the cross-reference is clearer than in the Hebrew language. For example, two Hebrew words that have the same semantic meaning are translated in a single English word. In English, the cross-reference can easily be find by human using a concordance, but in Hebrew it is quit more difficult to match the two words.

The fourth translation has the worst results comparing to the others, and it is a Dutch translation which is known as a literal translation of the Hebrew. So it should have comparable results, but that is not the case. It would be very interesting to find out why this translation performs worse, but that question is behind the scope of the research.

5.6 Conclusions

From the experiments described in this chapter, we can conclude the following. First, using one single string metric already very good results can be achieved. Second, the best performing string metric make use of the word scale or the n-gram scale, with a slight benefit for n-gram scale. Third, using multiple distance

outperforms using one single string metric. Dimension reduction does not improve these results. Fourth, the best performing classifier for the distance-based approach is the logistic classifier. Fifth, there are significant different results when different languages are evaluated. The influence of the preprocessing of the texts is not evaluated in this research but slight improvements might be obtained when doing this in a accurate way.

The best result for the distance-based approach is an error of 0.1197 with a standard deviation of 0.0018. This is a big improvement comparing to the software Copyfind, which has an error of 0.1832. The mentioned errors are error counts, because with Copyfind it is not possible to measure the AUC performance.

Chapter 6

High-level grammatical approach

In this chapter, the experiments to optimize the high-level grammatical approach are described. First, the results of each individual high-level grammatical features are evaluated. Second, the performances of all high-level grammatical features are figured out. Third, an experiment is conducted to find the best classifier for this approach. Fourth, different representations of the high-level grammatical approach are evaluated. Fifth, the performances on other scale for the high-level grammatical approach are studies and described. Finally, the experiments in which these multiple scale are combined are outlined.

6.1 One feature

First, the performances of each individual feature are evaluated. Remember that there are 162 different grammatical characteristics, which can be found in Appendix A. However, the results are quite disappointing.

Each individual feature is tested ten times, and the mean is mentioned in the following sentences. Only 16 features have an AUC error lower that 0.45. The other 146 have an AUC error above the 0.45, so that is really close to random guessing. The best performing feature has just an AUC error of 0.42331.

The best five features can be found in Table 6.1. The features belong with two main categories, namely part of speech and a characteristic of a verb.

Category	Feature name	AUC error
part of speech	adverb	0.4233 (0.0020)
verbal stem	nif'al	0.4295 (0.0015)
verbal tense	infinitive (absolute)	0.4310 (0.0015)
part of speech	proper noun	0.4325 (0.0013)
verbal tense	imperfect	0.4333 (0.0017)

Table 6.1: The best five high-level grammatical features with their performances.

6.2 Multiple features

The next step is to combine all the grammatical features. This section explains the performances of combining all 162 grammatical features. In the next section, the performances of different classifiers are explained. In this section we already selected the best performing classifiers. Further, there are three ways to combine all these features, namely by adding, concatenating and subtracting. Later these ways of combining are evaluated, in this section we already incorporated the results of this evaluation that concatenating is the best way of combining.

Figure 6.1 shows the performances of all high-level grammatical features concatenated on the different datasets. The standard deviation over ten times is shown in the figure.

On first sight, it seems that the nearest mean classifier is outperforming the logistic classifier on most of the datasets. This is true, but on the dataset that contains all cross-references (All books), the logistic classifier is better. Further, on datasets with smaller objects sizes, the standard deviation is larger. Ezra is the smallest dataset, and also Joshua is a small dataset. On larger datasets, like Psalms and All books, the standard deviations are small enough to be ignored.

The overall results are quite disappointing comparing to the results of the distance-based approach. The dataset All books provides an AUC error of 0.3540 with a standard deviation of 0.0018 for the logistic classifier. All other datasets have comparable results for the nearest mean classifier.

Only the nearest mean classifier on the dataset Ezra have significant better results, it has an AUC performance of 0.2620 with a standard deviation of 0.0477. However, these results can be explained by the size of the dataset Ezra. It has only 84 positive and 84 negative examples, and there are $162 + 162 = 324$ features, so when each example has one unique feature, then there is a perfect linear separable feature-space. So the results of the nearest mean classifier on the dataset Ezra can be neglected.

A possible explanation for the fact that the logistic classifier for the dataset All books is better than the nearest mean classifier, is that this dataset uses also objects that are between books. Most often one book has one genre, with the same kind of features. For example, the book Psalms will often use first person, because the writer wrote about his own feelings. In contrary to that, the book

Genesis often use third person, because it describes the narratives about the protagonists. So the logistic classifier can better separate the negative examples from the positive examples, assuming that it is the case for positive examples that they have the same features although they are not in the same books.

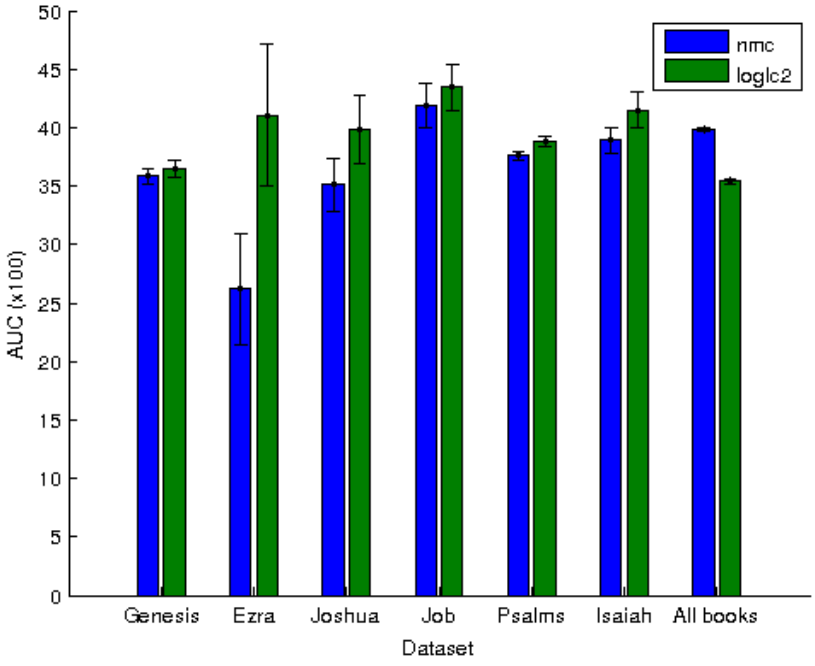

Figure 6.1: The AUC of different datasets on the high-level grammatical features per verse. Two classifiers are used, namely the nearest mean classifier (nmc) and the logistic classifier (loglc)

6.3 Classifiers

Several classifiers are evaluated for the high-level grammatical features. For these tests, the high-level grammatical features are concatenated (see Section 6.4), because this provides the best results.

Figure 6.2 shows the performances of several classifiers on the standard tests. The errorbar on top of each bar stands for the standard deviation, which is computed over ten times repeating the experiment. The dataset All books is not displayed in the figure, because it was too time-consuming to test that dataset for some classifiers. The dataset Ezra is very small, so we neglect the performances of the classifiers for this dataset.

Table 6.2 shows the average performances of each classifier over the six

datasets. The table and figure clearly shows that the linear discriminant classifier, the quadratic discriminant classifier, the Parzen classifier and the decision tree classifier have poor results comparing to the other classifiers.

The best performing classifiers are the nearest mean classifier, the fisher classifier, the logistic classifier, the k-nearest neighbour classifier and the support vector machine. The performances of these classifiers are mostly within the standard deviation of each other. The classifier that has the overall best classifier is the k-nearest neighbour classifier, however, this classifier is very time-consuming, especially when it has to deal with many objects. The same holds for the support vector machine. The logistic classifier has comparable results, and it far more faster. The nearest mean classifier is a very fast and simple classifier. Therefore, in the tests only these two classifiers are used, because evaluating all classifiers for all experiments on the high-level grammatical approach is too time-consuming.

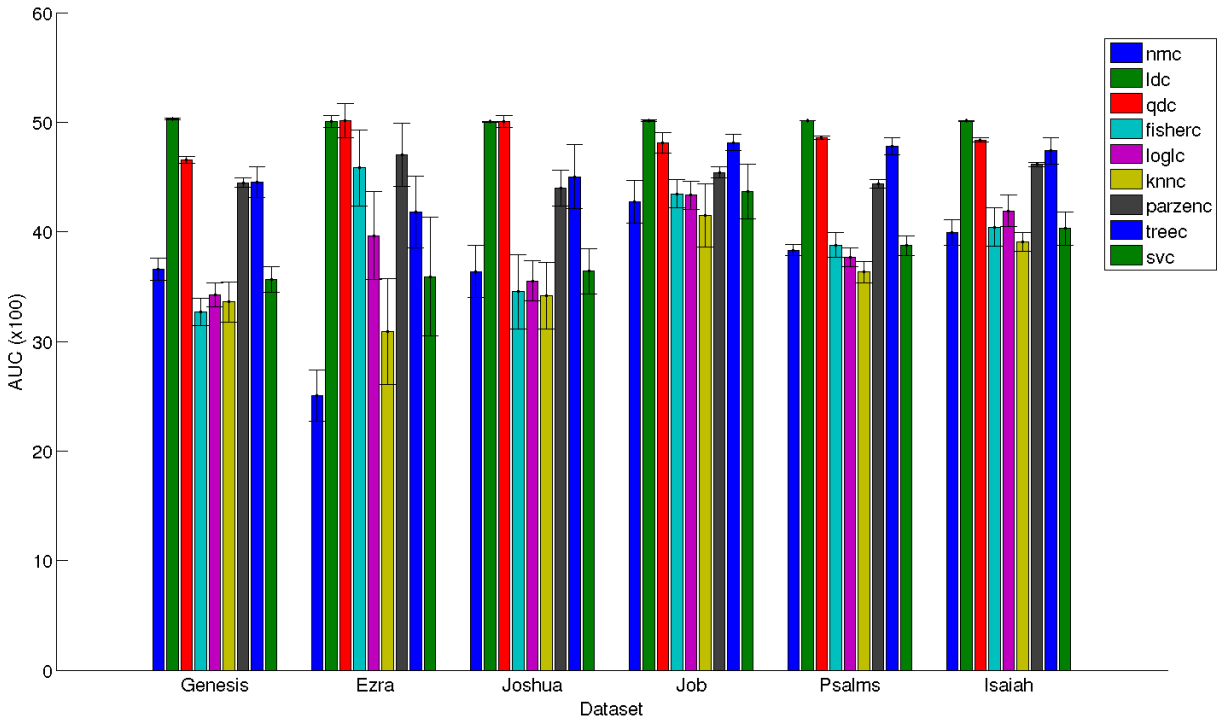

Figure 6.2: The AUC of different classifiers on the high-level grammatical features.

Classifier	Average (std over 6)
Nearest mean classifier	36.5 (0.1)
Linear discriminant classifier	48.6 (1.3)
Quadratic discriminant classifier	39.3 (5.1)
Fisher's linear classifier	39.3 (5.1)
Logistic classifier	38.7 (3.6)
k-nearest neighbour	35.9 (3.9)
Parzen classifier	45.2 (1.2)
Decision tree classifier	45.8 (2.5)
Support vector machine	38.4 (3.1)

Table 6.2: Mean classifiers

6.4 Adding, subtracting or concatenating

The next question that needs to be answered deals with the representation of the features of a pair. As explained earlier, each verse has 162 high-level grammatical features. The question is: how to represent the features of two verses? Intuitive, there are two possibilities. The features can simply be concatenated to each other, or they can be add together. Another solution might be to subtract the features of both verses. This is not intuitive, but it delivered good results in some other pattern recognition problems.

Figure 6.3 shows the results of the three possibilities. As classifier, the logistic classifier is used. The results are ten times repeated and the standard variation is shown in the figure.

The figure shows that concatenating or adding the features is the best solution. For the dataset Genesis, Joshua and Isaiah, adding is the best option. However, for the datasets Ezra, Job and All books, concatenating the features provides the best results.

When looking at the standard variation, the options concatenating and adding are within one standard variation of each other, except for the dataset All books. Further, the dataset All books contain all cross-references, and therefore we can conclude that concatenating the features is the best option to represent the features of two verses into one object.

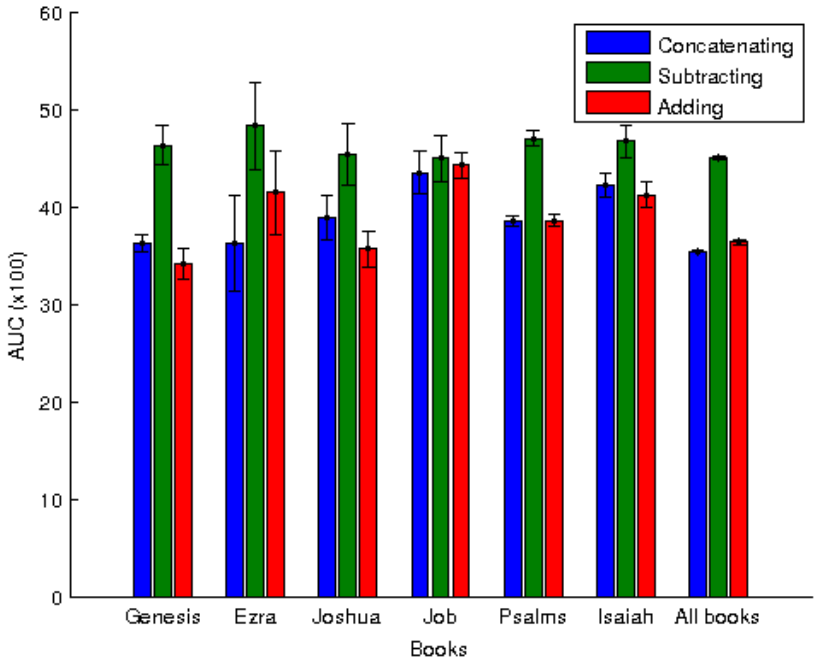

Figure 6.3: The AUC performance of concatenating, subtracting or adding the high-level grammatical features of a pair of verses.

6.5 Dimension reduction

In Section 6.2, we saw that using all features lead to some improvement comparing to using a single feature. However, using all features may overfit the classifier. Reducing the feature may lead to better results. In this subsection, two types of reduction are examined. The first type is Principal Component Analysis (PCA), the other is feature forward selection (FFS).

Figure 6.4 shows the results of the dimensional reduction of the features. The feature forward selection algorithm stopped after 87 features, because then the features have no more valuable information. The experiment is repeated ten times, and the standard deviation is shown in the figure.

The blue line, the PCA feature with the nearest mean classifier, has the worst performance. Using 40 PCA features will deliver the best results, a higher feature size does not increase the AUC performance further.

Then the red line, which are the PCA features using the logistic classifier. It clearly outperforms the PCA feature with the nearest mean classifier, but reaches the best results with 50 PCA components.

Next, the results of the feature forward selection with the nearest mean clas-

sifier, which are depicted by the green line. It has comparable results with the PCA features using the logistic classifier, however, it reaches far more faster the best result. After 25 features, the FFS algorithm using the nearest mean classifier has the best results. Higher feature size just decreases the AUC performance.

Finally, the black line depicts the FFS algorithm using the logistic classifier. It delivers clearly the best results, and gets the best AUC performance at a feature size of 35 features.

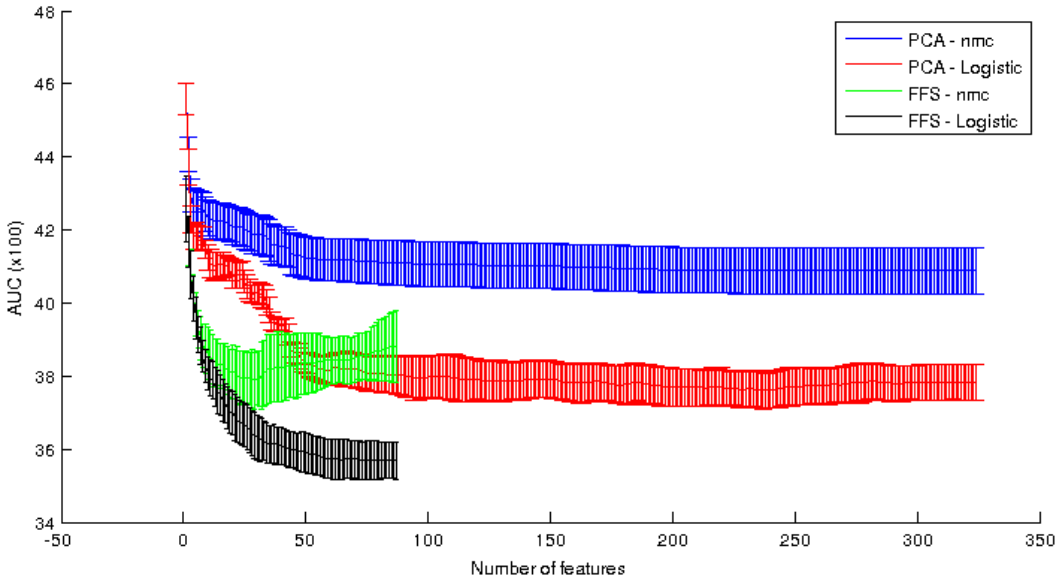

Figure 6.4: The AUC performance of reducing the number of feature by PCA or feature forward selection (FFS). Two classifiers are used, the nearest mean classifier (nmc) and the logistic classifier (Logistic).

6.6 Different scales

There are five scales namely verses, sentences, clauses, phrases and words. Until now, the high-level grammatical features are evaluated on verse scale. Now we examine the performances of the high-level grammatical features on sentence, clause and phrase scale. The word scale is not evaluated because the size of this unit is too small. Figure 6.5 shows the AUC of the high-level grammatical features per scale. The average number of sentences, clauses and phrases per verse is computed, which are displayed in Table 3.2. For example, a verse contains on average seven clauses, so we concatenated the high-level grammatical features of the first seven clauses from the beginning of the verse. The reason for this

is that it is very difficult to deal with the variable length of the language scales per verse.

The figure shows that the logistic classifier always has better performance than the nearest mean classifier, on all scales. Further, the verse scale has the best performances. The performances of the other scales are quite comparable, and the best or the rest is the phrase scale. However, it has still a bad AUC performance of 0.3801.

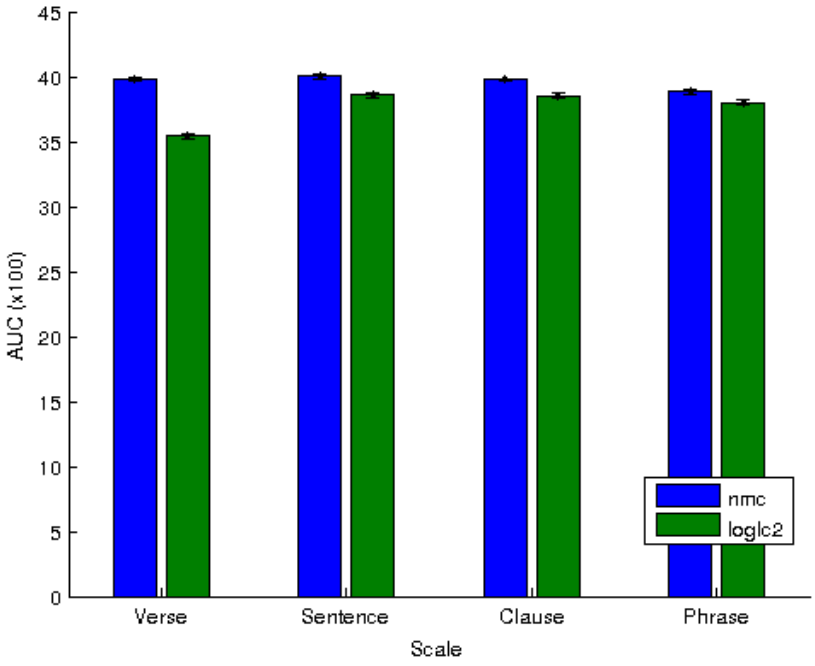

Figure 6.5: The AUC of the scales with concatenating the average number of sentences, clauses and phrases per verse.

Using the mean as average length of a verse might not be a good idea because the length of the verses are very different. Therefore, an experiment is conducted in which not the mean but the mean plus the standard variation is used. Also, the mean plus two times the standard variation as length of an average verse is evaluated. For example, when a verse contains on average 7 clauses with a standard variation of 3, we repeated the experiment with mean+std $= 7+3 = 10$ clauses concatenated and mean $+\, 2 \cdot$ std $= 7+6 = 13$ clauses concatenated. The results are displayed in Figure 6.6.

As the figure shows, taking a larger window definitely improves the AUC performance. A larger window lead to an improvement on each scale. The best AUC performance following from this test is 0.3540 with a standard deviation

of 0.0013.

Further tests pointed out that a window of three times or higher the standard deviation will not improve the results anymore.

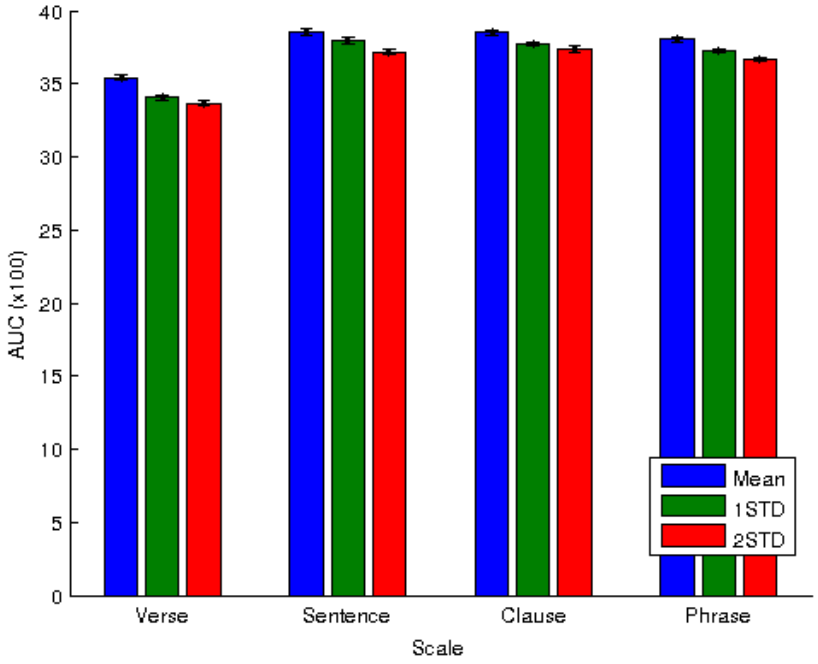

Figure 6.6: The AUC of the different scales with concatenating more than average sentences, clauses and phrases per verse.

6.7 Conclusions

Now the following can be concluded. First, the high-level grammatical approach delivers very disappointing results. Second, the best performing single high-level grammatical feature has just a very bad AUC performance. Third, combining all high-level grammatical feature slightly improve this result. Fourth, the nearest mean classifier and the logistic classifier are the best performing classifiers for the high-level grammatical features. Fifth, concatenating the features of the verses of a pair is the best as representation of the data. Sixth, dimension reduction do not improve the results. Selecting the best 35 features using feature forward selection already reaches the best result, with higher feature sizes no increase could be taken more. Finally, the results can be improved by selecting the high-level grammatical feature over a window of five verses instead of using the features of one verse.

The error (by error counting) that the best solution for the high-level grammatical approach has, is 0.4009 with a standard deviation of 0.0028. Comparing that to the results of Copyfind, which has an error of 0.1832, we can conclude that the high-level grammatical approach could not be used for the detection of intertextuality.

Chapter 7

Multi-scale distance approach

In this chapter the results of the multi-scale distance approach are explained. First is started with the explaining of the experiments on the scale half-verses. Then the experiments on the scales sentence, clause and phrase are described.

7.1 Half-verses

The main problem with the multi-scale distance approach is that the number of sentences, clauses and phrases within a verse is very variable. A classifier can only deal with a fixed feature size, so we have to combine the scale distance is such a way that they will have a fixed length.

However, the half-verses do not have that problem. From the 23213 verses, 20333 can be split into two half-verses. The others can have more or less half-verses, so these verses are neglected in the tests with the half-verses. Then the problem is easier to solve, because a pair of two half-verses has always four distances. These four distances can easily be evaluated with classifiers.

Figure 7.1 shows the performances of four single distances on full verses, on half verses and on both. The classifier that is used is the logistic classifier. The ten distances are chosen from feature forward selection. Selecting more distances was not possible because the tests were highly time consuming. Ten times repeating the experiments is also not done. From the previous experiments, we saw that results on the seventh dataset are always very stable, and always have just a very small standard-deviation. All the experiments on the multi-scale distance approach are applied on the seventh dataset, so we assume that the results are stable.

The figure shows three columns. All columns have ten distances. The left columns display the AUC performances of a single distance on the full verse. The middle columns depict the performances of the four distances between the four half-verses. The right columns show the performances of the distance between

the full verse in combination with the distances between the half-verses. These distances were concatenated and that was used as input for the classifier.

From this figure, it is clear that distances on full verses outperform distances on half verses. Only the third distance has a better performance on half-verses, but its performance is very bad comparing to some other distances.

The final conclusion that can be drawn from the graph is that combining the distance on full verses and half verses lead to a slightly better result. However, this holds mainly for the distances that perform worse. It is very clear that the distances 3 and 10 in the right columns have better performances than in the other columns. For distances that have a good performance, the increase in performance is hardly significant, for example the distances 1, 6 and 8. The experiments are done on the seventh dataset, so all books and objects are included.

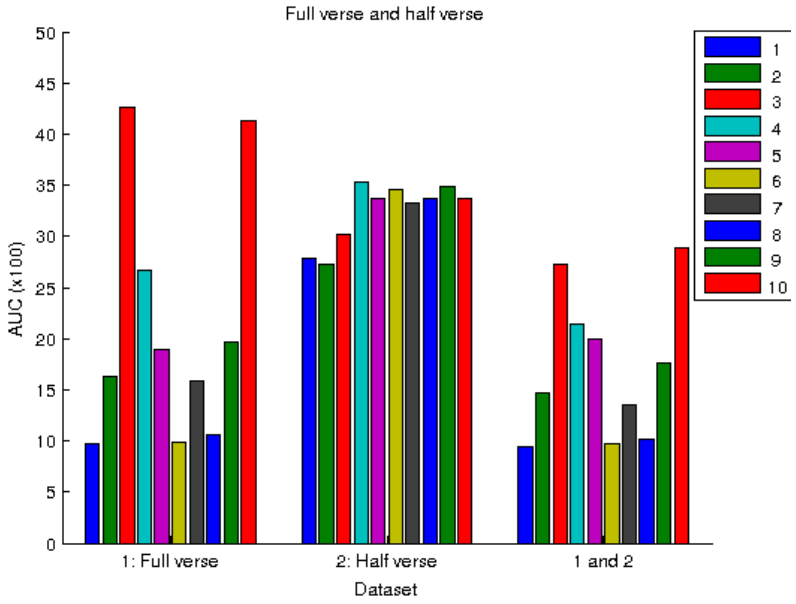

Figure 7.1: The AUC of ten single distances on full verses and half verses.

The next step is to evaluate the performances when multiple distances are used simultaneously. The distances from the ten distance metrics are simply concatenated. For the half-verse distances, this can be done in two ways. The distances can be ordered by distance, so then the first feature is distance 1 of unit 1, the second feature is distance 2 of unit 1, etcetera. The other way is ordering by unit, then the first feature is distance 1 of unit 1, the second feature is distance 1 of unit 2, and so on.

The results are shown in Figure 7.2. As the figure shows, only using half-verses performs worse than using only full verses. Column 1 shows the perfor-

mances of the distances of verse-scale. Column 2 displays the performances of the half-verse distances, ordered per unit. Column 2 depicts the performances of the half-verse distances, ordered per distance. However, combining these methods is slightly better than using only full verses, but this increase in performance is hardly significant (column 4 and 5). Combining both unit ordered and distance ordered features on half-verse scale do not lead to better results (column 6).

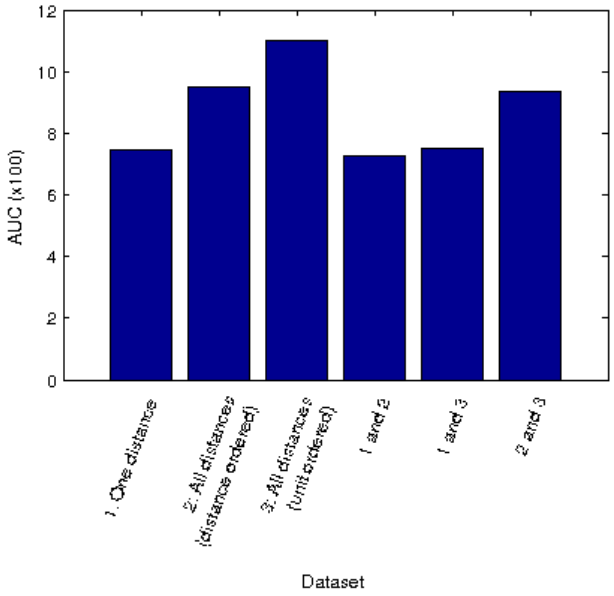

Figure 7.2: The AUC of four combined distances on full verses and half verses.

7.2 Sentences, clauses and phrases

Using half-verse distance, it was not possible to get a better result than the distance-based approach. But how is it for the other scales? The main problem for the other scales is that the number of sentences, clauses and phrases within a verse is very variable, so it does not fit for a classifier, which needs an input of fixed length. So a number of solutions is evaluated to deal with the variable length of the scales. For all the experiments, the same ten distance metrics are used as described in the previous section.

The first solution is taking the length of the longest unit of a scale, and adding zeros to units that are smaller in such a way that all objects have the same length as the longest object. Then the logistic classifier is applied. Now the results of the nearest mean classifier are not taken into consideration because it is too simple to deal with such feature sizes.

Figure 7.3 shows the results of this approach for all scales. Remember that the feature vectors can be distance ordered or unit ordered. Both feature vectors are examined.

As the figure shows, the clause scale has the best performance. Remember the AUC performance of the single ten distances was 0.0745 so the performance of the clause scale ordered by distance has some comparable results, it has an AUC error of 0.0986.

The performances on phrase level are the worst of the scales, this is because a phrase is a too short segment of a verse to get reliable distances. For example, a lot of phrases consist of one word, so then word n-grams are not possible, and also distances on word scale can only be one or zero, which add not new information to other distances.

The sentence delivers average performance, this is due to the fact that sentences are too long. In theory, the intertextuality between two verses comes from just a part of the verses, and it can be assumed that the clause scale covers this parts the best.

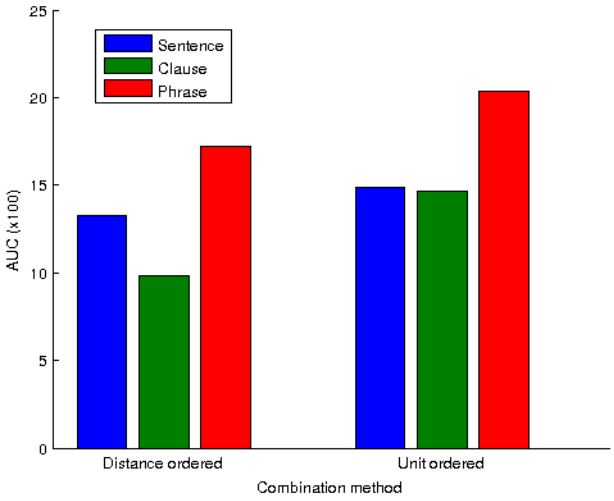

Figure 7.3: The AUC performance using all distances as feature vector.

The second solution to deal with the variable length of the sentences, clauses and phrases is to combine the distances of each unit. Five combination methods are evaluated for each scale, namely the mean, median, minimum, maximum and the product. We are using ten distances, so after the combination step we get ten features.

Figure 7.4 shows the results of this solution. The performances of the combination methods median, minimum and product are comparable to random guessing. This is not strange. The median of a single distance of each unit is mostly zero because there a lot of pairs of units that have no similar characters

and therefore no similar word, n-gram or word n-gram. The minimum is for the same reason not a good choice, because there is always a zero within these distances. The same holds for the product combination method.

So there are two combination methods that have good results, namely the mean and maximum. As already explained, the clause scale have the best performances comparing to the sentence and phrase scale. The best result is on the clause scale using the mean combination method, it has an AUC error of 0.0883. This is the best result for the multi-scale distance approach.

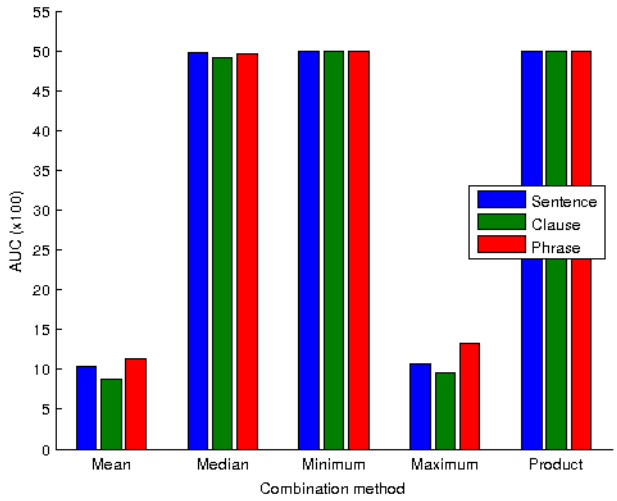

Figure 7.4: AUC performances when combining the distances.

The third solution is to combine the units. Again five combination methods are evaluated for each scale, namely the mean, median, minimum, maximum and the product. On the sentence scale, the first 40 units are considered, on the clause scale, the first 80 units are examined, and on the phrase scale, the first 120 units are evaluated. This numbers are chosen that only rarely the verses contain more sentences, clauses and phrases than the number previously mentioned.

Figure reffig:scale-distance shows the performances of this solution. Again, it is clearly visible that the clause scale performs better than the sentence and phrase scale. The minimum and product combination method has very bad performances, this is because there is always a minimum of zero within the distances of a single unit.

The performances of the maximum method are also poor, this is because the maximum of the ten distances is selected, and this maximum is not very indicative for the presence of intertextuality. Remember that some distance are selected that have individually very bad performance, but in combination with other distances it can lead to a slight improvement.

The results of the mean combination method are average. Probably the mean of the distances of one pairs of units is quite indicative for intertextuality. The ten distances are altogether not very good indicators for intertextuality but in combination, that may be the reason why the mean combination method have average performance.

The median combination method has the best results. It is a bit surprising because the mean combination method has average results, so the median method should have also average results. However, this is not the case. A possible explanation is that there are a lot of outliers that hinders an accurate mean, but these outliers do not affect the median a lot.

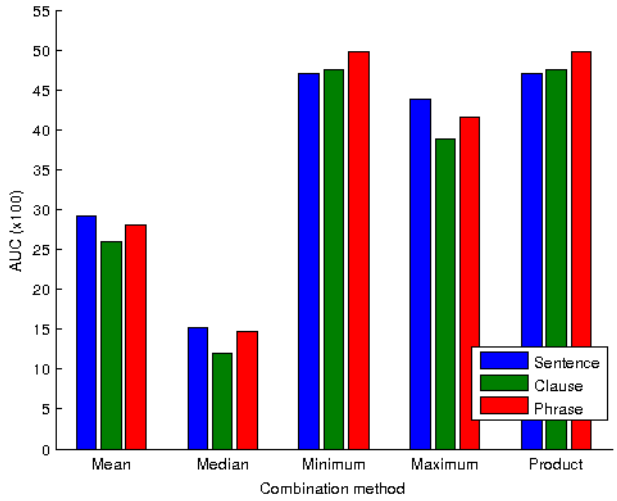

Figure 7.5: AUC performances when combining the units.

7.3 Conclusions

We can conclude the following. First, the multi-scale distance approach do not outperform the performances of the distance-based approach. Second, the half-verses are promising, but the results are quite disappointing. Even combining the verse scale with the half-verse scale do not lead to significant better results. Third, the best results are taken on the clause scale. The intertextuality between two verses are probably located on this scale. This means that in many cases two verses share a related clause, and that related clause causes intertextuality. Fourth, selecting the relevant pair of units out of all possible pairs of units do lead to a small but significant increase in performance. The best result provided when the mean of each distance (of all pairs of clauses) is taken, then a classification error based on error counting of 0.1453 is provided with a standard deviation of 0.0008.

Chapter 8

Conclusions and future work

The problem that is considered in this report is how to recognize intertextuality. Intertextuality means that each text depends on other texts. No text is totally original, but it uses ideas of other writers. In scientific writing, this should be stated explicitly by a bibliography at the end, but in earlier times copying ideas without mentioning the author was generally accepted.

This research uses supervised learning to detect intertextuality. This means that we need training examples of intertextuality. On these examples, a classifier can be trained. This classifier can classify new observations.

The Hebrew Bible is used for the training examples of intertextuality. There are several translations from Hebrew into other languages which contain a list of cross-references. These cross-references mean that the verses of the cross-reference are linked to each other in some way, so these cross-references can be seen as positive examples of intertextuality. The lists of the cross-references of the ESV, NIV and NAS are used in this research. We have taken the intersection of these three lists to be sure that the selected cross-references are accurate examples.

Negative examples mean that there is no relation between two verses. So it is clear that all examples that are not positive could be used as negative examples. However, there are almost infinity more negative examples than positive examples, so we have chosen to select randomly the same number of negative examples as positive examples. The performance measure AUC is used because that measure is insensitive for class imbalance.

We designed three approaches to recognize intertextuality. The approaches are extensively evaluated to find the advantages and disadvantages of each approach.

Seven datasets are selected for the evaluation of the experiments. Six of them are one books. These books have a different number of cross-references, are from a different genre, and are of different difficulty to detect intertextuality. The

seventh dataset consist of all books and all positive examples and is therefore the best dataset for the evaluation.

The plagiarism detector Copyfind is involved to get an indication of the performances of the approaches that are developed in this research. Although Copyfind is meant for plagiarism detection, it can be optimized so that it can be used for the detection of intertextuality.

The first approach that is designed is the distance-based approach. It uses several string metrics to compute the similarity between verses. The string metrics are used in fourfold, they are applied on the characters, words, n-grams and word n-grams of the verses.

From the experiments of the distance-based approach, we can conclude that some single distance metrics already have good performances. Using all distances together slightly increases this performance. This means that the distance metrics are very correlated.

Several classifiers are evaluated for the distance-based approach. It turns out that the logistic classifier is the best performing classifier when all distances are used as input.

Further, there is a small difference in results when different translations of the Hebrew Bible are used. Especially the translations of the cross-reference lists perform better than other translations. This might be an indication that the cross-references are optimized for their own translation. However, the differences are quite small so we can conclude that the cross-references are reliable.

The second approach uses high-level grammatical features. These features are very special because they are the outcome of 30 years of analysing and annotating the grammatical characteristics of the Hebrew Bible. No one has tried earlier to use this method in natural language processing because this kind of features are not available for such a large text as the Hebrew Bible.

The conclusions of the high-level grammatical approach are these: each one single grammatical feature has very poor performance. There are only a few grammatical features that perform better than random guessing. Using all the high-level grammatical features do not really improve these performances. The reason is that the few grammatical features that perform better than the rest are quite correlated.

The logistic classifier again has the best performance on the high-level grammatical features. Other classifiers have the same performances, but are slower, especially when the number of training objects is high.

A final small improvement can be made by using the high-level grammatical features of a window around the verse. Including the high-level grammatical features of two verses before and two verses after the cross-reference verses lead to a small but significant increase in performance.

The third approach uses the same distances of the first approach, but then the distances are applied on multiple scales. These multiple scales are half-verses sentence, clause and phrase level. The main problem with this approach is the variable length of the number of sentences, clauses and phrases within a verse. It is not possible to use it directly as input for a classifier, because a classifier needs an input of fixed length. An escape from this problem is using

half-verses. Most of the verses of the Hebrew Bible consists of two so called half-verses. These half-verses are used because the number of distances between these four half-verses are of a fixed length. For the other scales, namely the sentence, clause and phrase scale, another solution is applied. This solution involves combination methods, which summarizes a feature vector of variable length into a feature vector of fixed length.

From the third approach these conclusions can be drawn: using the distances between the half-verses of the examples has no good performance. The verse scale is always outperforming the four distances on half-verse scale. However, combining them with the full-verse distance lead to a very small improvement.

The combination methods on the sentence, clause and phrase scale performs better than the four distances on half-verse scale. It is surprising that the clause scale always outperforms the sentence and phrase scale. It means that the intertextuality between verses are mainly on clause scale. Taking the mean of each distance over all units leads to the best performance for this approach. Again, by combining the verse scale with the clause scale, a small but significant improvement can be taken.

Figure 8.1 shows the performance of each approach and Copyfind. It can be concluded that the distance-based approach has the best performances. The multi-scale distance approach has slightly worse results. Further, the high-level grammatical approach do not lead to good results, because the features do not contain enough information to separate the classes. The plagiarism detector Copyfind provides significantly worse results than the distance-based approach and the multi-scale distance approach. This is expected because Copyfind can only deal with words that are equal, and it recognizes only verses that share exactly the same words. The approaches in this research can also deal with words that are nearly the same, or when verses share recognizable parts of words.

Combining the methods do not improve the results. The reason is that the approaches recognize just a few unique cross-references. If a cross-reference is detected by a different approach than the distance-based approach, the distance-based approach nearly always also recognizes that cross-reference. This means that only a very slight improvement can be made above the performances of the distance-based approach.

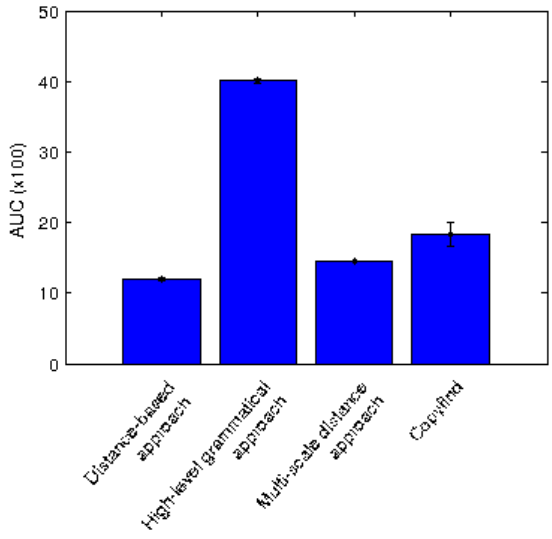

Figure 8.1: All approaches compared

In the future, the distance-based approach should be done on other texts. The Hebrew Bible is a very specific text, and it is not sure that the method will also work on modern texts, or texts that are not as linguistically variable as the Hebrew Bible. The problem is that as far as we know there are no texts available from which the intertextuality is known. The solution is that experts should find all examples of intertextuality in a text. The outcome of this process can be used to evaluate the approach of this research on other texts than the Hebrew Bible. However, this is very time-consuming to find all examples of intertextuality within a text, especially when the text is very long.

Future work may also be done on incorporating all no cross-reference into the solution. In this report, negatives examples of intertextuality are chosen by random resampling of the complement of the positive examples. However, it might be the case that the classifiers are sensitive for a high class imbalance, and therefore they should be tested when all negative examples are involved.

Another very interesting thing that needs be figured out in the future is the involving of a lexical database for the detection of intertextuality. A lexical database contains synonyms of all words of a language. In this research, intertextuality is not detected when synonyms are used, because the distance metrics measure a high distance between two different words with the same meaning. With a lexical database, this disadvantage of distance metrics could be solved.

This research has a unique approach in detection intertextuality, the high-level grammatical approach. It would be very interesting to involve these high-level grammatical features into the distance metrics used in the distance-based

approach. For example, when two words have the same letters, but are from a different part of speech, the distance will be decreased. Or in the other case, when two words have different letters, but are from the same part of speech, then the distance will be increased. This improvement could lead to a better detection of intertextuality. However, it requires a lot of time to optimize the costs and profits when a high-level grammatical feature of two words are similar. Further, there are a lot of high-level grammatical features which all needs to be investigated what the effect is on the performance when it is integrated within the distance metric. That are the reasons why we were not able to realize this method.

In the very future, this research could be applied to all texts of famous early writers, like philosophers, to see a line through the centuries of the surviving of the ideas that are thousands of years old. It would be very interesting to find the influence of for example Plato through the centuries. Whitehead [32] has said "The safest general characterization of the European philosophical tradition is that it consists of a series of footnotes to Plato". This research could give a first move to an easy and simple way for finding the influence of Plato in the European philosophical tradition.

Bibliography

[1] Rao Muhammad Adeel Nawab, Mark Stevenson, and Paul Clough. "Detecting text reuse with modified and weighted n-grams". In: *Proceedings of the First Joint Conference on Lexical and Computational Semantics-Volume 1: Proceedings of the main conference and the shared task, and Volume 2: Proceedings of the Sixth International Workshop on Semantic Evaluation*. Association for Computational Linguistics. 2012, pp. 54–58.

[2] Alberto Barrón-Cedeño and Paolo Rosso. "On automatic plagiarism detection based on n-grams comparison". In: *Advances in Information Retrieval*. Springer, 2009, pp. 696–700.

[3] Charles Bazerman. "Intertextuality: How texts rely on other texts". In: *What writing does and how it does it: An introduction to analyzing texts and textual practices* (2004), pp. 83–96.

[4] Lou Bloomfield. *Copyfind*. Feb. 2015. URL: `http://plagiarism.bloomfieldmedia.com/z-wordpress/software/copyfind/`.

[5] Paul Clough et al. "Meter: Measuring text reuse". In: *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*. Association for Computational Linguistics. 2002, pp. 152–159.

[6] Neil Coffee et al. "The Tesserae Project: intertextual analysis of Latin poetry". In: *Literary and linguistic computing* (2012), fqs033.

[7] Susan T Dumais. "Latent semantic analysis". In: *Annual review of information science and technology* 38.1 (2004), pp. 188–230.

[8] Peter W Foltz, Walter Kintsch, and Thomas K Landauer. "The measurement of textual coherence with latent semantic analysis". In: *Discourse processes* 25.2-3 (1998), pp. 285–307.

[9] Evgeniy Gabrilovich and Shaul Markovitch. "Computing Semantic Relatedness Using Wikipedia-based Explicit Semantic Analysis." In: *IJCAI*. Vol. 7. 2007, pp. 1606–1611.

[10] Dan Gusfield. *Algorithms on strings, trees and sequences: computer science and computational biology*. Cambridge university press, 1997.

[11] Sébastien Harispe et al. "Semantic Measures for the Comparison of Units of Language, Concepts or Instances from Text and Knowledge Base Analysis". In: *arXiv preprint arXiv:1310.1285* (2013).

[12] Jian Hu et al. “Enhancing text clustering by leveraging Wikipedia semantics”. In: *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*. ACM. 2008, pp. 179–186.

[13] Aminul Islam and Diana Inkpen. “Semantic text similarity using corpus-based word similarity and string similarity”. In: *ACM Transactions on Knowledge Discovery from Data (TKDD)* 2.2 (2008), p. 10.

[14] Matthew A Jaro. “Advances in record-linkage methodology as applied to matching the 1985 census of Tampa, Florida”. In: *Journal of the American Statistical Association* 84.406 (1989), pp. 414–420.

[15] Christopher Keep, Tim McLaughlin, and Robin Parmar. *Intertextuality*. 2000.

[16] Robert Krovetz. “Viewing morphology as an inference process”. In: *Proceedings of the 16th annual international ACM SIGIR conference on Research and development in information retrieval*. ACM. 1993, pp. 191–202.

[17] Vladimir I Levenshtein. “Binary codes capable of correcting deletions, insertions, and reversals”. In: *Soviet physics doklady*. Vol. 10. 1966, pp. 707–710.

[18] Caroline Lyon, James Malcolm, and Bob Dickerson. “Detecting short passages of similar text in large document collections”. In: *Proceedings of the 2001 Conference on Empirical Methods in Natural Language Processing*. 2001, pp. 118–125.

[19] Ana G Maguitman et al. “Algorithmic detection of semantic similarity”. In: *Proceedings of the 14th international conference on World Wide Web*. ACM. 2005, pp. 107–116.

[20] Erwin Marsi and Emiel Krahmer. “Classification of semantic relations by humans and machines”. In: *Proceedings of the ACL workshop on Empirical Modeling of Semantic Equivalence and Entailment*. Association for Computational Linguistics. 2005, pp. 1–6.

[21] Donald Metzler, Susan Dumais, and Christopher Meek. *Similarity measures for short segments of text*. Springer, 2007.

[22] George A Miller. “WordNet: a lexical database for English”. In: *Communications of the ACM* 38.11 (1995), pp. 39–41.

[23] George A Miller and Walter G Charles. “Contextual correlates of semantic similarity”. In: *Language and cognitive processes* 6.1 (1991), pp. 1–28.

[24] Vladimir Oleshchuk and Asle Pedersen. “Ontology based semantic similarity comparison of documents”. In: *Database and Expert Systems Applications, 2003. Proceedings. 14th International Workshop on*. IEEE. 2003, pp. 735–738.

[25] Simone Paolo Ponzetto and Michael Strube. “Deriving a large scale taxonomy from Wikipedia”. In: *AAAI*. Vol. 7. 2007, pp. 1440–1445.

[26] Martin F Porter. "An algorithm for suffix stripping". In: *Program* 14.3 (1980), pp. 130–137.

[27] Dirk Roorda et al. "LAF-Fabric: a data analysis tool for Linguistic Annotation Framework with an application to the Hebrew Bible". In: *arXiv preprint arXiv:1410.0286* (2014).

[28] Herbert Rubenstein and John B Goodenough. "Contextual correlates of synonymy". In: *Communications of the ACM* 8.10 (1965), pp. 627–633.

[29] Sadi Evren Seker et al. "A novel string distance function based on most frequent K characters". In: *arXiv preprint arXiv:1401.6596* (2014).

[30] Efstathios Stamatatos. "Intrinsic plagiarism detection using character n-gram profiles". In: *threshold* 2 (2009), pp. 1–500.

[31] Daniel R White and Mike S Joy. "Sentence-based natural language plagiarism detection". In: *Journal on Educational Resources in Computing (JERIC)* 4.4 (2004), p. 2.

[32] Alfred North Whitehead. *Process and reality*. Simon and Schuster, 2010.

[33] William E Winkler. "String Comparator Metrics and Enhanced Decision Rules in the Fellegi-Sunter Model of Record Linkage." In: (1990).

Appendices

Appendix A

High-level grammatical features

The following features are available in the WIVU database:

Category	Granularity	Feature names
Part of speech	Word	Article Verb Noun Proper noun Adverb Preposition Conjunction Personal pronoun Demonstrative pronoun Interrogative pronoun Interjection Negative particle Interrogative particle Adjective
Gender	Word	Masculine Feminine Not applicable Unknown
Number	Word	Singular Dual Plural Not applicable Unknown
Person	Word	First person Second person

Continued on next page

Table A.1 – *Continued from previous page*

Category	**Granularity**	**Feature names**
Person	Word	Third person Not applicable Unknown
Verbal stem	Word	Hif'il Hitpa"el Hof'al Nif'al Pi"el Pu"al Qal Not applicable
Verbal tense	Word	Perfect Imperfect Wayyiqtol Imperative Infinitive (absolute) Infinitive (construct) Participle Participle (passive) Not applicable
Phrase type	Phrase	Verbal phrase Nominal phrase Proper-noun phrase Adverbial phrase Prepositional phrase Conjunctive phrase Personal pronoun phrase Demonstrative pronoun phrase Interrogative pronoun phrase Interjectional phrase Negative phrase Interrogative phrase Adjective phrase
Phrase function	Phrase	Adjunct Complement Conjunction Enclitic personal pronoun Existence with subject suffix Existence Fronted element Interjection Interjection with subject suffix Locative

Continued on next page

Table A.1 – *Continued from previous page*

Category	**Granularity**	**Feature names**
Phrase function	Phrase	Modifier Modifier with subject suffix Negative copula Negative copula with subject suffix Negation Object Predicative adjunct Predicate complement with subject suffix Predicate complement Predicate Predicate with object suffix Predicate with subject suffix Participle with object suffix Question Relative Subject Supplementary constituent Time reference Unknown Vocative
Phrase relation	Phrase	Predicative adjunct Resumption Not applicable
Clause type	Clause	Adjective clause Casus pendens Defective clause atom Ellipsis Infinitive absolute clause Infinitive construct clause Macrosyntactic sign Nominal clause Participle clause Reopening Unknown Vocative clause Wayyiqtol-null clause Wayyiqtol-X clause We-imperative-null clause We-imperative-X clause We-qatal-null clause We-qatal-X clause We-x-imperative-null clause

Continued on next page

Table A.1 – *Continued from previous page*

Category	**Granularity**	**Feature names**
Clause type	Clause	We-X-imperative clause We-x-imperative-X clause We-x-qatal-null clause We-X-qatal clause We-x-qatal-X clause We-x-yiqtol-null clause We-X-yiqtol clause We-x-yiqtol-X clause We-yiqtol-null clause We-yiqtol-X clause x-imperative-null clause X-imperative clause x-imperative-X clause Extraposition x-qatal-null clause X-qatal clause x-qatal-X clause x-yiqtol-null clause X-yiqtol clause x-yiqtol-X clause Zero-imperative-null clause Zero-imperative-X clause Zero-qatal-null clause Zero-qatal-X clause Zero-yiqtol-null clause Zero-yiqtol-X clause
Clause constituent relation	Clause	Adjunctive clause Attributive clause Complement clause Coordinated clause Object clause Predicative adjunct clause Predicative complement clause Referral to the vocative Resumptive clause Regens/rectum connection Specification clause Subject clause Not applicable

Table A.1: The features of the WIVU-database used in the high-level grammatical approach.

Appendix B

Distance metrics

This appendix describes the distance metrics that are used in the distance-based approach.

B.1 Character scale

1. Levenshtein distance
2. Jaro-Winkler algorithm
3. Longest common substring
4. Most frequent 5 characters similarity
5. Most frequent 10 characters similarity
6. Most frequent 15 characters similarity
7. Most frequent 20 characters similarity
8. Dice's coefficient

B.2 Word scale

1. Levenshtein distance
2. Jaro-Winkler algorithm
3. Longest common substring
4. Most frequent 3 words similarity
5. Most frequent 5 words similarity
6. Most frequent 8 words similarity
7. Most frequent 12 words similarity
8. Dice's coefficient
9. Weighted sum of the number of words that are nearly equal
10. Weighted sum of the number of equal words

B.3 n-gram scale

1. Levenshtein distance over 2-grams
2. Levenshtein distance over 3-grams
3. Levenshtein distance over 4-grams
4. Levenshtein distance over 5-grams
5. Jaro-Winkler algorithm over 2-grams
6. Jaro-Winkler algorithm over 3-grams
7. Jaro-Winkler algorithm over 4-grams
8. Jaro-Winkler algorithm over 5-grams
9. Longest common substring over 2-grams
10. Longest common substring over 3-grams
11. Longest common substring over 4-grams
12. Longest common substring over 5-grams
13. Most frequent 10 n-grams similarity over 2-grams
14. Most frequent 10 n-grams similarity over 3-grams
15. Most frequent 10 n-grams similarity over 4-grams
16. Most frequent 10 n-grams similarity over 5-grams
17. Dice's coefficient with 2-grams using space encoding type 1
18. Dice's coefficient with 3-grams using space encoding type 1
19. Dice's coefficient with 4-grams using space encoding type 1
20. Dice's coefficient with 5-grams using space encoding type 1
21. Dice's coefficient with 2-grams using space encoding type 2
22. Dice's coefficient with 3-grams using space encoding type 2
23. Dice's coefficient with 4-grams using space encoding type 2
24. Dice's coefficient with 5-grams using space encoding type 2
25. Dice's coefficient with 2-grams using space encoding type 3
26. Dice's coefficient with 3-grams using space encoding type 3
27. Dice's coefficient with 4-grams using space encoding type 3
28. Dice's coefficient with 5-grams using space encoding type 3
29. Weighted Dice's coefficient with 2-gram
30. Weighted Dice's coefficient with 3-gram
31. Weighted Dice's coefficient with 4-gram
32. Weighted Dice's coefficient with 5-gram

B.4 Word n-gram scale

1. Levenshtein distance over word 2-grams
2. Levenshtein distance over word 3-grams
3. Jaro-Winkler algorithm over word 2-grams
4. Jaro-Winkler algorithm over word 3-grams
5. Longest common substring over word 2-grams
6. Longest common substring over word 3-grams
7. Most frequent 10 word n-grams similarity over word 2-grams

8. Most frequent 10 word n-grams similarity over word 3-grams
9. Dice's coefficient over word 2-grams
10. Dice's coefficient over word 3-grams
11. Dice's coefficient over word 4-grams
12. Dice's coefficient over word 5-grams
13. Dice's coefficient over word 6-grams