Use of algorithms in detecting intrinsic and extrinsic plagiarism through NLP and machine learning

Kerry Xu, Adrien Nurboja

Research paper for

The British Science Association's CREST AWARD scheme

at Tiffin School

03/07/2017

Table of contents

List of tables and figures

List of acronyms and abbreviations

NLP	Natural Language Processing
WWW	World Wide Web
NLTK	Natural Language Toolkit (Python library)
POS	Part of speech
AI	Artificial Intelligence
NYSE	The New York Stock Exchange
CbPD	Citation Based Plagiarism Detection
BC	Bibliographic Coupling

1. INTRODUCTION

Plagiarism is the unacknowledged use of someone else's work or ideas [1]. Advances in technology is increasing the widespread accessibility of data and has resulted in increased text reuse leading to real-world consequences, for example plagiarism in academia is a serious issue. In the pre-technological era, plagiarism was almost difficult to detect and prove however the development of the WWW has resulted in readily available plagiarism detection software which use various algorithms. This include extrinsic plagiarism detection where the suspected text is compared to a database of millions of files and to a lesser extent, intrinsic plagiarism detection.

This paper will analyse the current plagiarism detection methods in both extrinsic and intrinsic plagiarism detection and develop a try develop a new method in detecting plagiarism.

2. RESEARCH FOCUS

Plagiarism exists in several forms ranging from word to word copying to paraphrasing. The aim of this paper is to develop a plagiarism detector that can be used in detecting all forms of plagiarism.

2.1 RESEARCH PROBLEM

Extrinsic plagiarism detectors have made considerable progress in the last few decades, the new technological era has enabled computers to process and compare billions of documents in seconds. Unfortunately, there are also huge limitations to extrinsic plagiarism detection, not only must there exist a large corpus (database of text) containing every single article and document for a word to word plagiarism detector to work but it must also detection-verbatim plagiarism (involves rewriting text) which is aimed to trick plagiarism detection software. This is an example of a problem that humans such as teachers can detect and determine the likelihood of non-verbatim plagiarism (not word for word plagiarism) whilst a computer would have difficulty by using extrinsic methods.

Another problem with plagiarism detection is the detection of false positives, given the number of common phrases in the English language, strings of concurrent text between documents are inevitability cause many plagiarism detectors to falsely detect a coincidence.

Source code plagiarism

Source code plagiarism, the unlawfully stealing other person source code or program code is a type of plagiarism which is currently being researched on where people apply code obfuscation techniques to make the code look different and then claiming that it is one's own code in a way violating the terms of original license. The increase in software plagiarism is also becoming more prominent as websites like SourceForge.Net have over 430,000 open source projects to download. We will focus on the broader topic of textual plagiarism as it occurs in all academic discipline.

Consequences of plagiarism

The consequences of plagiarism can be severe and has personal, professional, ethical, and legal impacts. Anyone can plagiarise however it is particularly common for students, authors, journalists and academic professionals who plagiarise for several reasons including pressure to meet deadlines and difficulty in integrating source material. Not all plagiarism is intentional and accidental plagiarism can occur, this includes the author failing to cite a fact that they thought was common knowledge.

Whilst plagiarism itself not illegal, it often violates intellectual property rights which can lead to serious legal repercussions and the original author has the right to sue a plagiarist regardless of the type of plagiarism. "In 2005 there was an intellectual property lawsuit filed by Compuware against IBM. "As a result, IBM paid $140 million in fines to license Compuware's software and an additional $260 million too", [2] Not only did this lawsuit have monetary repercussions of $400 million but it also resulted in a damage in reputation for the company as seen when IBM's NYSE stock prices fell from $90.70 in March 2005 to $76.70 in the following month.

Plagiarism results in a loss of student, professional and academic reputation and can result in students being suspended/expelled and can destroy a career. Plagiarised research is usually poorly researched is can be dangerous for example a plagiarised medical paper could result in the loss of lives.

The overall consequences of plagiarism are far-reaching and immense and the best way to get people to avoid plagiarising it to teach people about the severe consequences that it can result in.

3. BACKGROUND READING

Extrinsic Plagiarism Detection

Extrinsic plagiarism detection is the process of comparing a single document with a large database of documents (corpus) using NLP to identify if this document has been plagiarised. This may include copying and pasting or paraphrasing sections of documents on the WWW. This involves extracting various data from the document and comparing it with the documents in the corpus and can be detected through methods such as fingerprinting and string matching.

These plagiarism detection methods tend to have a higher level of accuracy as they can identify the actual source that is copied however the disadvantage of extrinsic plagiarism is that it demands a large amount of computational power and time and a large database for all the substrings for the document to be compared.

Intrinsic Plagiarism Detection

Intrinsic plagiarism detection is the process of detecting plagiarism in a document without the use of a reference collection. This can usually be done via analysing the language use of the suspected document for example teachers can suspect that a student is plagiarising even without using any reference as when students plagiarise from other sources as their style of writing can often change when plagiarising from multiple sources. Our research will be focussed on stylometry as an intrinsic method to identify how people change their style of writing when plagiarising, this includes changes in syntactic features e.g. increased use in the passive voice and changes in structure.

Comparison

Extrinsic plagiarism detection and intrinsic plagiarism detection both have its own distinct advantages and disadvantages. Extrinsic plagiarism detection can accurately detect and pinpoint plagiarism back to a source for copy and paste plagiarism but suffers from computationally demanding tasks and face difficultly when detecting plagiarism with heavy modification. On the other hand, intrinsic plagiarism detectors are more versatile as they can detect all sorts of plagiarism even without a reference corpus however the accuracy of intrinsic plagiarism detectors are much lower and often result in the suspected document being checked by a human.

Python

Python is an open source high-level programming language which the source code of python is easily accessible and it has syntax closer to human languages and further from machine languages making it a simple yet productive and powerful language.

Python is the most popular dynamic language and allows you to put any value into any variable making the code faster to write and clearer as there is no type checking prior execution however this comes at the expense of the potential misuse of objects resulting in code being more prone to runtime errors. It is also a strongly typed language as every object in the language has a definite type and there is and there is no way to circumvent that type

We will be using python as it is a general use programming language which comes with a great standard library and includes powerful third-party libraries such as the Natural Language Took Kit, used for natural language processing, NumPy and SciPy which are used in scientific computing. This means that python is very flexible and adaptable so it is suitable for a range of tasks ranging from data science to visualization.

Writing in python results in more readable code that is typically 3-5 times shorter than equivalent Java programs and 5-10 times shorter than equivalent C++ code. [3] The trade-off to shorter code is that programs will typically run slower statically typed languages (type checking during compile time rather than run time) such as Java and C#, as the object type must be inspected at run time.

We considered alternatives such as C# and Java which are very similar however Python has a must better set of third-party libraries which are essential for use in NLP and machine learning. The benefits of these third-party libraries such as NLTK and StatsModels.api in our project will outweigh the increased accessibility of debugging tools and performance available in C# and Java.

```cpp
#include <iostream>

int main()
{
    std::cout << "Hello, world!\n";
    return 0;
}
```

C++

```csharp
public class Program
{
    public static void Main()
    {
        System.Console.WriteLine("Hello, World!");
    }
}
```

C#

```python
print("Hello World!")
```

Python 3

Figure 1. Comparison of “Hello World” Program in various programming languages

Natural Language Processing

Natural language is language that humans use to communicate with each other, this exists in both written and oral forms. “Natural Language Processing (NLP) is an area of research and application that explores how computers can be used to understand and manipulate natural language text or speech to do useful.”[4] This can enable computers to derive meaning from natural language and can result in many real-world applications ranging including spam detection, machine translation and autocomplete. It is a very powerful tool as there is a lot of text data on the internet such as articles where meaning can be extracted using NLP algorithms and be used for research.

Natural Language Toolkit (Python)

NLTK is a python library that provides tools for natural language processing, we will primarily be using it to pre-process unstructured text e.g. Wikipedia articles so it can be processed through a machine learning algorithm to be classified.

Tokenization

Tokenization is the process of breaking text down into words and sentences, this allows the text to be more split into understandable chunks. NLTK provides a sentence tokenize and a word tokenize method. A sentence tokenizer may appear simple to program but is a lot more complex than it appears. A sentence boundary is hard to determine as although exclamation point and the question mark are almost always unambiguous examples of such punctuation, the period is an extremely ambiguous punctuation mark as by itself, it is insignificant to decide whether it is used as a full-stop or as part of an abbreviation e.g. Mr. In the Brown corpus, there are 48885 sentences and 3490 (1 in 14) contain at least one non-terminal period. [5]

Punkt Sentence Tokenizer

PunktSentenceTokenizer is a module contained in the NLTK which is designed to split text into sentences by using an unsupervised algorithm for sentence boundary detection and can therefore accurately tokenize a piece of text into sentences. “It is based on the assumption that a large number of ambiguities in the determination of sentence boundaries can be eliminated once abbreviations have been identified.” [6]. This will increase accuracy for sentence tokenization.

Stop word Removal

Stop words are extremely common words in a language which add little meaning to a document other than for grammatical purposes or to add structure. These often include short function words such as *the, is* and *at* as well as common lexical words such as *want.* Removing stop words help focus on analysing the more meaningful data rather than the stop words. The NLTK has a stop words Corpus by *Porter et al* which contains 2,400 stop words for 11 languages, we use this corpus to pre-process text by removing the stop words. Stop word removal is useful in plagiarism detection as people try to modify stop words rather than the actual content in plagiarism.

Stemming

Stemming is the process of reducing a word back into its base/root form. This allows for the words to be treated as the same despite having different suffixes. This can be useful in plagiarism detection as people may change the suffix of a word in order to avoid plagiarism detection.

Example: *Affect, Affected, Affecting, Affects, Affection, Affections → Affect*

Stemming Algorithms – Porter's algorithm

There are several algorithms for stemming including using a lookup table which is fast but all forms of a word must be listed in the lookup table. Suffix stripping is another algorithm which relies on rules rather than suffix stripping e.g. if a word ends in 'ed', 'ing' or 'ly', remove the suffix as example of a suffix stripping algorithm is Porter's algorithm which is the most common English stemmer however this can have difficulty stemming exceptions e.g. 'ran' and 'run'. [7] It also has difficulty dealing with languages with more complex language roots such as in Turkish with the word *uygarlastiramadiklarimizdanmissinizcasina* which will require a complicated rule set in order to stem however the English language is a lot easier to stem so this problem is trivial. [8]

(F)

Rule			Example		
SSES	→	SS	caresses	→	caress
IES	→	I	ponies	→	poni
SS	→	SS	caress	→	caress
S	→		cats	→	cat

Figure 2. Porter algorithm first phase rule groups [9]

Stemming Algorithm - Lancaster Stemmer

NLTK includes several stemming algorithms including the Porter stemmer as well as the Lancaster (Paice/Husk) stemmer which is a fast-aggressive stemmer which utilises a single table of rules, each of which may specify the removal or replacement of an ending [10]. The aggressive algorithm will reduce the trim down the word by a large amount which can be advantageous or disadvantageous depending on the situation. A key advantage to the Lancaster stemmer is that it is flexibility as it allows you to add a new set of rules without extensive code change.

Lemmatization

Lemmatization is the process of reducing a word back into its dictionary form (lemma), the process is closely related to stemming (both are a form of normalization). Lemmatization takes account of the context of the word in the text for example it may use POS (part of speech) tagging for example stemming the word *saw* → *s* whilst lemmatization would find the context of the word use e.g. if used as a noun *saw* → *saw* but if used as a verb *saw* → *see*.

Example: *Am* → *Be*, *Better* → *Good*, *Affected/Affecting* → *Affect* (sometimes same as stemming)

Stemmers tend to be easier to implement and run faster compared to lemmatizers as stemmers usually use crude heuristic (rough approximation) process that chops off the ends of words whilst lemmetizers use a lot of resource doing vocabulary and morphological analysis in order to derive the base/dictionary form of the word.

NLP in Plagiarism Detection

“The methods used for plagiarism detection so far are mostly limited to a very superficial level, for example by comparing suspicious texts and original texts at the string level to check the amount of word overlapping across documents.” [11] Consequently, the accuracy of detection approaches is yet to reach a satisfactory level. [12]. We can use natural language processing to increase the accuracy of detection in intrinsic and extrinsic plagiarism by using text pre-processing which includes tokenization and stop word removal, shallow NLP techniques which includes stemming and lemmatization as well as deep NLP techniques e.g. using WordNet Synset Extraction which is using lexical database which is similar to a computer accessible dictionary and thesaurus.

Different Types of plagiarism

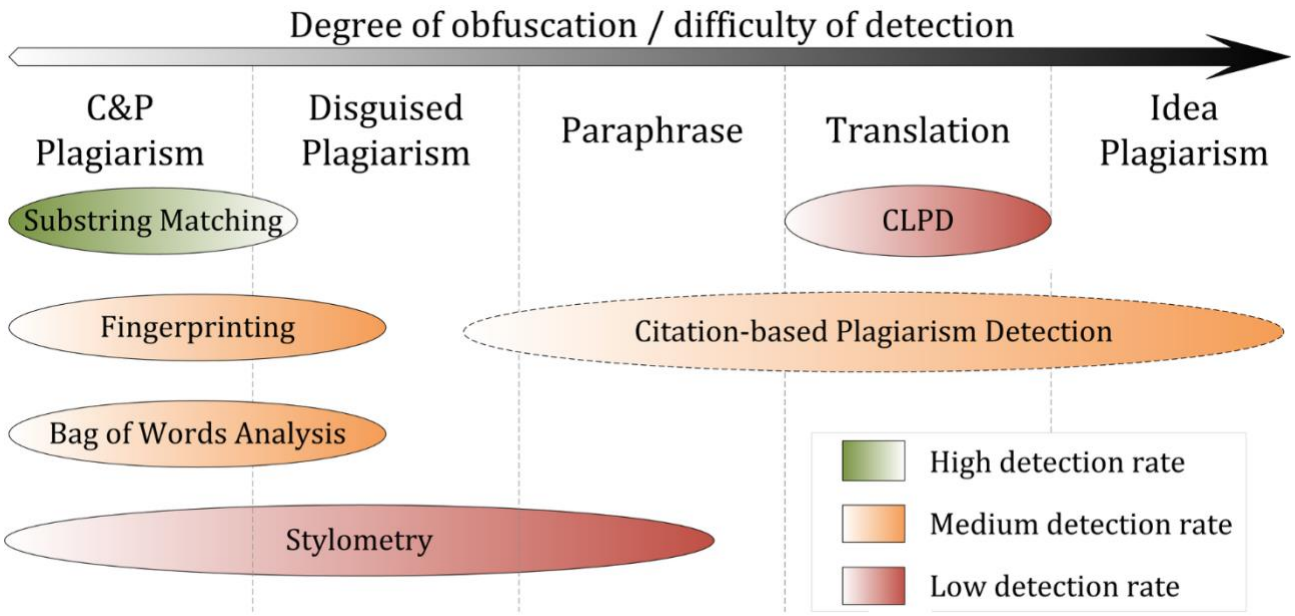

Figure 3. Different approaches to plagiarism detection based on the type of plagiarism [13]

Word for word plagiarism is where phrases, sentences of paragraphs are copied and pasted from a source e.g. book, article or website without citing the source or using quotations. This type of plagiarism is easily detected through many approaches such as fingerprinting and string matching with high degrees of accuracy when there is a large database where phrases from the suspected document can be compared.

Word Switch Plagiarism (Disguised plagiarism) is where sentences or paragraphs are copied and a few words are changed and is still considered plagiarism is it is not correctly cited. This type of plagiarism is slightly harder to detect depending on the degree of modification. Slight modifications can be detected with substring matching and fingerprinting to a lower degree of accuracy however it is harder to detect plagiarism if there is a large amount of modification.

Paraphrasing plagiarism is a form a word switch plagiarism when a writer summarizes an idea taken from another source but fails both to cite the authors and to provide a reference to the article.

Paraphrasing plagiarism and word switch can be difficult to detect and prove however stylometry is an approach that can detect all three types of plagiarism albeit at a low detection rate. We will focus our project on this area so we can make a versatile plagiarism detector.

Regression Analysis - Linear Regression

Linear regression is a regression model where the dependant and independent variables are both continuous (can take an infinite set of values). Linear regression is used when there is a linear relationship between the two variables so there is a change in the independent variable results in a proportional change in the dependant variable.

The data points can be presented on a 2D plane and the linear regression line would be where the sum of the error between the data points and the line (squared) is at a minimum. A linear regression line takes the form where A is the intercept and B is the sensitivity of y after a change in x. A linear regression line cannot be used for a plagiarism index which is bound from 0 to 1, therefore it does not have much use in plagiarism detection.

Regression Analysis - Logistic Regression

Logistic regression is a regression model where the dependant variable is categorical (can only take a finite set of values), in the case of binomial logistic regression, there are only two possible values for the dependant variable, “0” or “1” which is usually represented a probability. The use of logistic regression is to calculate the probability of an event occurring given some causes. Unlike in linear regression, the logistic regression curve looks like an “S” shape and is known as a sigmoid curve. This function has a ceiling probability of “1” and the floor probability is of “0”.

Logistic regression can be used to analyse for many purposes including analysing past events by examining how the causes lead to the effects, allocating resources for example determining the optimal amount of time/money for a project, predicting future events and classifying problem instances into categories.

The regression curve of a logistic regression is in the form, where the values of A (intercept) and B (regression coefficient) are such that the misclassified points are as few as possible. Linear regression lines can be converted into a logistic regression curve by using the logit function as the log of the odds of the probabilities (are linear related to the independent variable.)

$$\ln\left(\frac{P}{1-p}\right) = A + Bx$$

$$S = \frac{1}{1 + e^{-(A+Bx)}}$$

Multiple logistic regression can be used when there is one categorical independent variable but there are more than 1 dependant variables so multiple causes can be used to create a regression line. This can be helpful in detecting plagiarism as there can be many independent variables that we can combine with a logistic regression model to find out if a piece of text could be plagiarised.

The disadvantages of using binomial logistic regression for plagiarism is that it will only give a binomial value, either the piece is plagiarised, or the piece isn't plagiarised but , a piece of text could be only partially plagiarised, The problem with using these regression lines is that it is often hard to find and quantify detectable independent variables for example it is hard to quantify a change in the style of an author's writing (stylometry) directly but we will be using logistic regression in our plagiarism detector in order to try detect the style of author writing in a different manner.

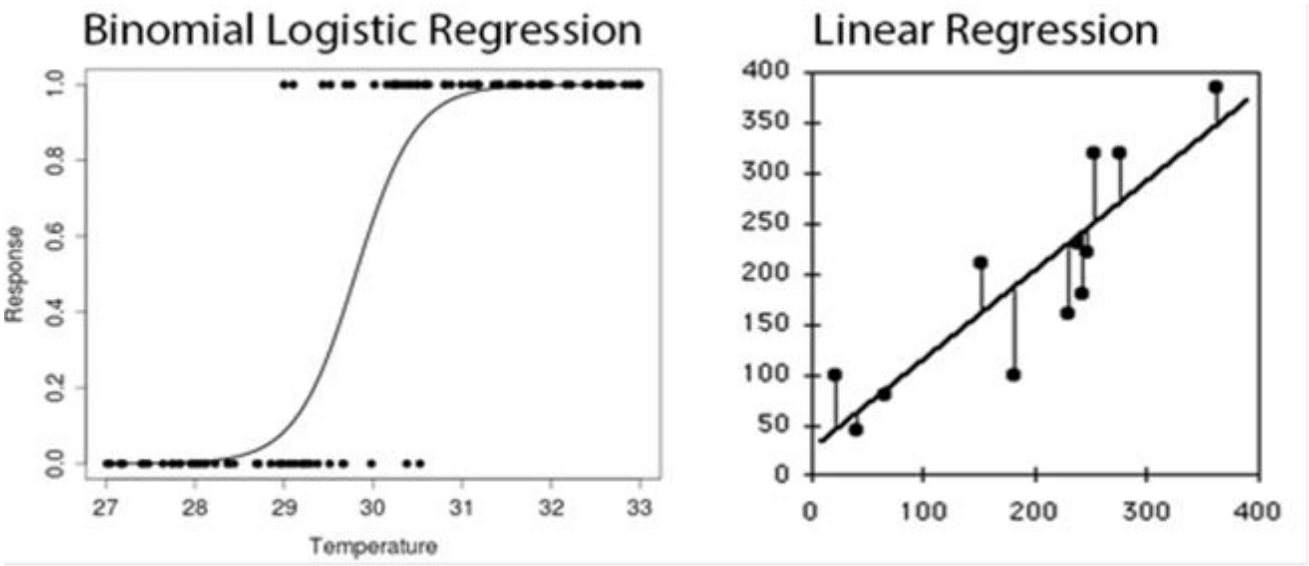

Figure 4, Binomial logistic regression and linear regression comparison

Machine learning can be used in combination with logistic regression or linear regression and is a go to method for binary classification and can predict possibilities. Maximum-likelihood estimation is a common algorithm used by a variety of machine learning algorithms, it can be used to calculate the coefficients of the logistic curve and this can be used to classify an input. By using machine learning, the training corpus can constantly be updated with new data therefore making the logistic regression algorithm more accurate.

Stylometry

Stylometry is the statistical analysis of the variation within a writer's piece of text. Writers usually write in a unique style and this means that stylometry plagiarism detection can detect copy and paste plagiarism if the text was from multiple sources through stylometric analysis. Stylometric analysis includes analysing features of text such as sentence length, typeati of vocabulary and the frequency of various words. We will be doing some further reason and analysis into hapax legomenon which are words that are only used once to determine whether it is suitable for use in plagiarism detection.

Stylometry is very flexible as it can be used on a single piece of text or multiple pieces of text and by using stylometry on an author's recent works, analysing a sudden change in style could indicate plagiarism. The disadvantage of using stylometry is that the extract being analysed must be of sufficient length, at least a few paragraphs and it is best used over tens of thousands of words. It is quick and requires a lot amount of processing and can be a very powerful intrinsic plagiarism detector once more research is done into this area of study however it is limited by the current, lower accuracy levels as suspected text may not be proven plagiarised by stylometry alone.

There are many indicators that a piece is plagiarised that can be stylometrically analysed including:

• "Use of advanced or technical vocabulary beyond that expected of the writer."

• "A large improvement in writing style compared to previous submitted work."

• "Inconsistencies within the written text itself, e.g. changes in vocabulary, style or quality."

• "Incoherent text where the flow is not consistent or smooth, which may signal that a passage has been cut-and-pasted from an existing electronic source."[14]

String Matching

String matching algorithms can be used to identify the longest pair of identical strings within two documents and these strings are used as indicators for potential plagiarism. If two documents exceed a certain chosen threshold then the documents are deemed to be plagiarised. These algorithms perform well at detecting word for word plagiarism but face difficulty when facing paraphrased plagiarism although they can perform better post NLP text processing e.g. stemming.

Citation Based Analysis

Citation based analysis is the examination of the frequency and patterns of citations in documents ultimately creating links between documents and revealing properties of the documents based on the citations. Citation based analysis can significantly improve detection for more sophisticated forms of plagiarism such as paraphrasing or idea plagiarism. Citations also have a valuable fingerprint that can be used to detect plagiarism, research have shown that the “order of citations in a document often remains similar even if the text has been strongly paraphrased or translated in order to disguise plagiarism.”[15]

The flaw in many plagiarism detection software is that they do not consider the citations that might have been displayed when using copy and paste. Citation based analysis (CbPD) allows for identifying these citations and comparing with the citations in the original source. This can be done using many algorithms. The simplest is using a Bibliographic Coupling (BC) score which provides a value for the number of identical citations in both documents. This can be made more accurate by considering other factors such as the position of citations and sequences of citations. This can be implemented by other algorithms such as Citation Chunking (CC), Greedy Citation Tiling (GCT) and Longest Common Citation Sequence (LCCS). CbPD is a new method used to detect plagiarism and therefore research into it has only begun and prototypes have been made by some professors from the Department of Statistics at Berkley in the University of California and the Department of Computer Science at Otto-von-Guericke-University Magdeburg.

Bag of words

A bag of words is just all the words in a document ignoring grammar and positioning. It is used to extract frequencies of words for use in NLP in information retrieval. This can be used in plagiarism detection such to compare frequencies of words. We have used this in some many of our programs such as our hapax legomenon frequency and string matching program. Bag of words can be used to find the cosine similarity of texts. This is a measure of similarity between the frequencies of common words in the documents.

Fingerprinting

Fingerprinting in NLP is a method of comparing the fingerprints of 2 documents. A fingerprint is a small piece of the document which uniquely identifies the document. This can be compared with another document to find out if it has been plagiarised. The most common method used for fingerprinting is n-grams, this process involves splits the document substrings of length n.

'If we compare a document A of size |A| against a document B, and if N is the number of substrings common in both, then the resemblance measure R of how much of A is contained in B can be computed as follows" [15]:

$$R = \frac{N}{|A|} \qquad 0 \leq R \leq 1$$

The closer R is to 1, the greater the resemblance between the 2 documents.

Overlapping n-grams are the document are split into all possible combination of n-gram tokens whilst non-overlapping is where each n-gram token appears once. Overlapping n-grams produce more accurate results for plagiarism detection because it compares more substrings than non-overlapping n-grams however this will require more computing power and will take a long longer, especially when comparing against a huge database due to the increased number of comparisons. [16]

Types of N-Gram Plagiarism Detection

Stop word n-grams create substrings using only the stop words. This allows for identifying simple paraphrasing however text with high obfuscation will need a smaller n-gram and therefore resulting in more substrings and more false positives. Another type of n-gram is named entity n-grams which can detect high levels of obfuscation because named entities are a lot less common than stop words. As named entities such as names tend to be rarer, there may be few comparisons causing little contribution to the overall detection of plagiarism. Finally, there is all word n-grams which uses all the words in the document. This results in a very thorough analysis to detect plagiarised text however the consequence of thorough analysis is substantial amounts of false positives and therefore using natural language processing to remove common words is vital to reduce the quantity of comparisons and to decrease the amount of false positives.

Fingerprint analysis

We decided to create our own code to test these methods to find out which one is more accurate. We used the plagiarised text from the samples we collected previously and obtained the non-plagiarised samples using random text generators online such as http://randomtextgenerator.com/ and http://www.johno.jsmf.net/knowhow/ngrams.

```
#code for stopword n-gram analysis

import time
import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from string import punctuation

def Main():
    start = time.time()
    #get the files containing the original and plagiarised text
    original = word_tokenize((open('original.txt','r').read()).lower())
    plagiarised = word_tokenize((open('plagiarised.txt','r').read()).lower())

    Resemblance(Stopwords(original),Stopwords(plagiarised),Length(original))
    finish = time.time()-start
    print('time taken = ',finish)

def Length(textList):
    #find the length of the text
    length = 0
    for i in textList:
        if i not in list(punctuation):
                length += len(i)
    return length

def Stopwords(text):
    #get a list of all stopWords
    stopWords = set(stopwords.words('english'))
    #retrieve all stopwords from the text
    stopWordsO = [word for word in text if word in stopWords]
    return Ngram(stopWordsO,2)

def Ngram(text,n):
    #creates a list of n-grams
    newList = []
    for i in text:
        for j in i:
            newList.append(j)
    ngrams = []
    for i in range(len(newList)-n+1):
        ngrams.append(newList[i:i+n])
    return ngrams

def Resemblance(original,plagiarised,A):
    #calculates the resemblance
    n = 0
    for i in original:
        if i in plagiarised:
            n += 1
    r = n / A
    print('Resemblance, R = ',r)

Main()
```

Figure 5 Stop Words N-Gram Code

Stop Word N-Gram Analysis

We used the Word Tokenize function from NLTK to split the files into an array of words and then by using a corpus of stop words provided by NLTK, we compared the text against the list of stop words and recorded all the stop words in the text and split them into n-grams. The program then uses this data along with the number of substrings in 'original.txt' as the length to calculate the resemblance using the formula.

For our results for the stop word n-gram method, we calculated the resemblance of 5 samples of plagiarised text with the original text. Resemblance in the non-plagiarised text was also collected. This was repeated for different values of n up to 4 because any higher value resulted in a dramatic decrease in accuracy.

n	plagiarised text	non-plagiarised text
1	1	1
	1	1
	1	1
	1	1
	1	1
Average	1	1
2	0.973	0.838
	0.995	0.883
	0.985	0.862
	0.993	0.844
	0.983	0.86
Average	0.9858	0.8574
3	0.89	0.485
	0.945	0.492
	0.917	0.43
	0.954	0.544
	0.917	0.542
Average	0.9246	0.4986
4	0.738	0.53
	0.823	0.583
	0.83	0.507
	0.854	0.541
	0.797	0.562
Average	0.8084	0.5446

Figure 6. Stop Words N-Gram Table

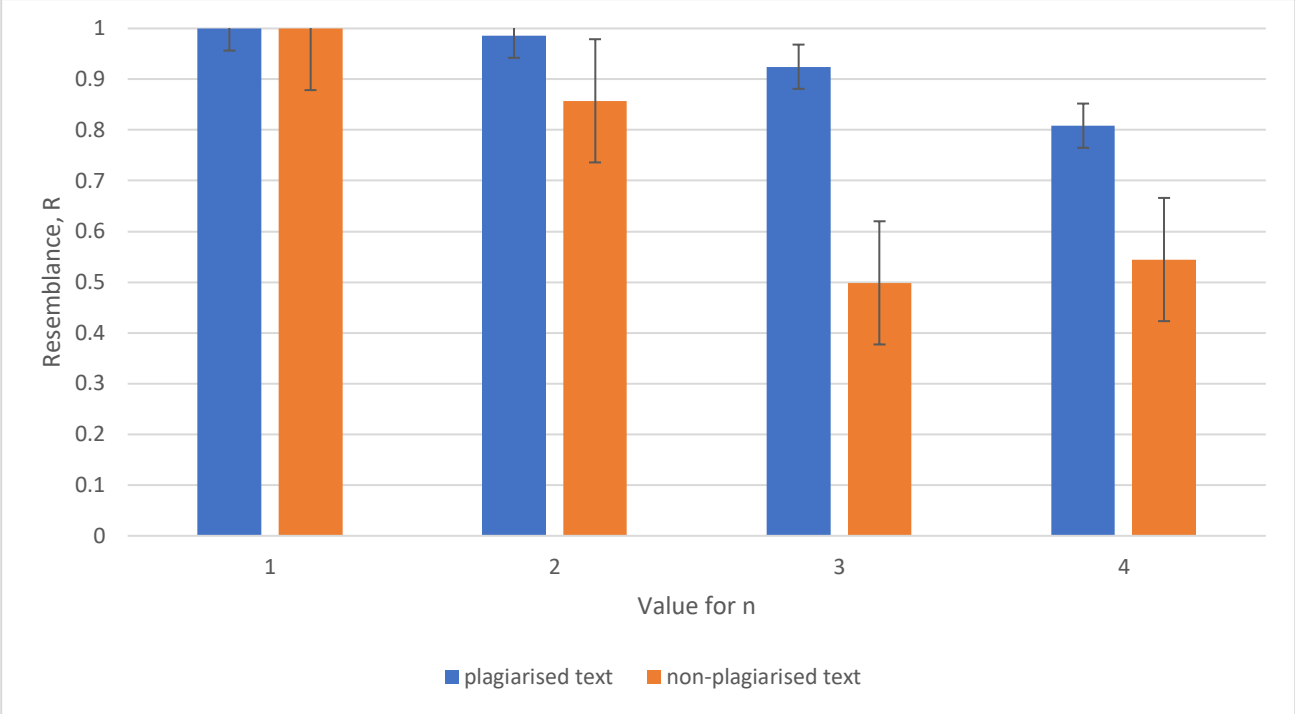

Figure 7. Stop Words N-Gram Graph

By observing the graph, we came to the conclusion that trigram (where n = 3) was the most accurate at detecting plagiarism. This is because it showed a much greater significant difference compared to the other values for n.

Named Entity N-Gram Analysis

We located a large document containing a lot of common named entities in English (http://www.cnts.ua.ac.be/conll2003/ner/lists/eng.list) and used this to retrieve all the named entities from the texts. We then used the re module to split the named entity into a list named corpus. The common named entities in both texts were acquired and used along with number of named entities in 'original.txt' as the length to calculate the resemblance.

Results:

The value for n in this case is the number of named entities per substring. However, due to named entities being very uncommon, we decided to just use a unigram (where n = 1). 7 plagiarised samples were used here along with 7 non-plagiarised texts.

```
import re
from string import punctuation
import time

def Main():
    start = time.time()
    #obtain a file containing named entities
    corpus = open('NE_corpus.txt','r').read()
    corpus = re.split(' PER | LOC | ORG | MISC ',corpus)
    #obtain the files containing text to be analysed
    original = open('original.txt','r').read()
    plagiarised = open('plagiarised.txt','r').read()

    Resemblance(NE(original,corpus),NE(plagiarised,corpus),len(NE(original,corpus)))

    #obtain time taken to complete program
    finish = time.time()-start
    print('Time taken = ',finish)

def NE(text,corpus):
    #retrieve all named entities from the text
    namedEntities = [ne for ne in corpus if ne in text]
    return namedEntities

def Resemblance(original,plagiarised,A):
    #calculate the resemblance
    n = 0
    for i in original:
        if i in plagiarised:
            n += 1
    r = n / A
    print('resemblance, R = ',r)

Main()
```

Figure 8a. Named Entity N-Gram Code

	Resemblance (plagiarised)	Resemblance (non-plagiarised)
	0.375	0.031
	0.526	0.158
	0.5	0.3
	0.591	0.045
	0.167	0.063
	0.2	0.4
	0.56	0.08
Average	0.417	0.153857143

Figure 8b. Named Entity N-Gram Table

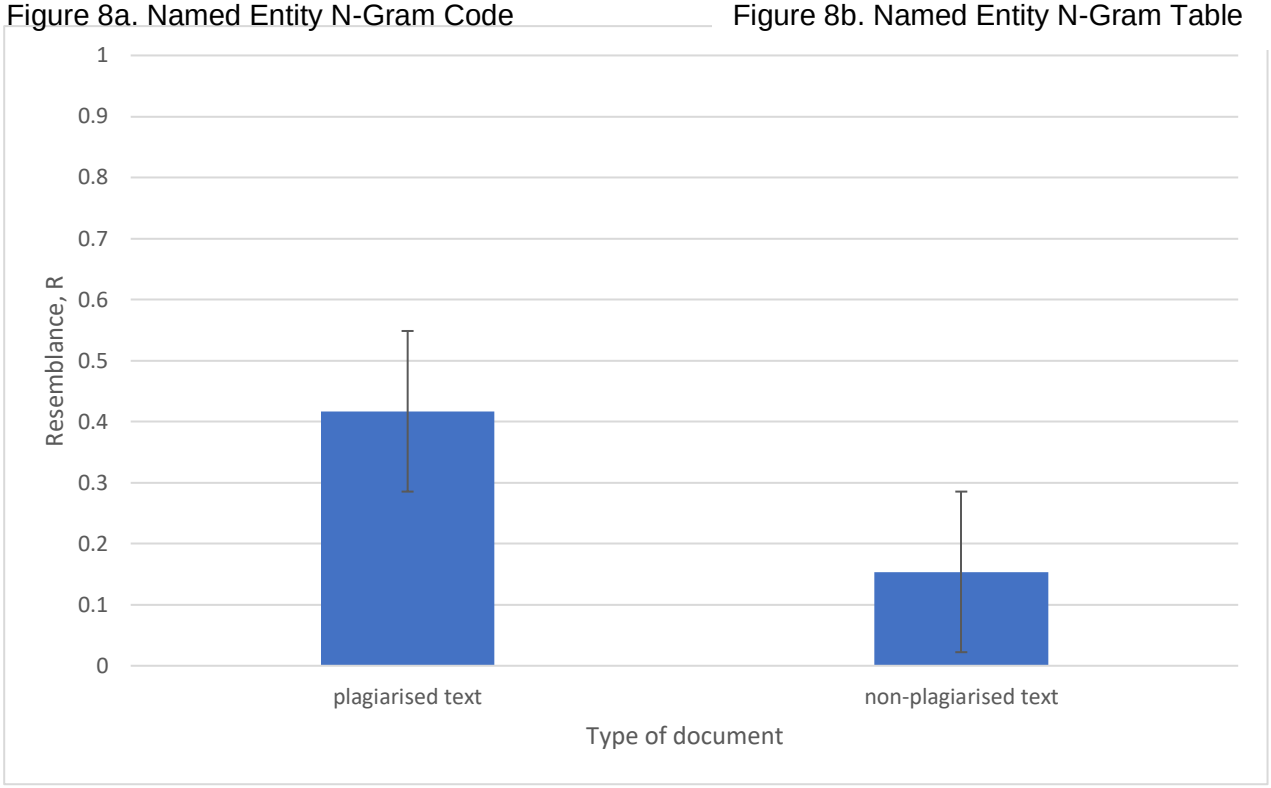

Figure 8c. Named Entity N-Gram Graph

By observing the graph, we can see that the resemblance of plagiarised text is higher than the non-plagiarised text. This can be used for plagiarism detection however as there is no significant difference between the plagiarised text and the non-plagiarised text, this method is not as successful as differentiating plagiarism compared to some of the other fingerprinting methods. In addition, as shown by the graph, there are relatively large error bars so there is a large amount of uncertainty making this a suboptimal method to detect plagiarism however it makes few comparisons so it is one of the fastest n-gram techniques however it can be hard to find a corpus of all named entities.

All Word N-Grams:

```
#code for all words n-gram analysis

import nltk
from nltk.tokenize import word_tokenize
from string import punctuation
import time

def Main():
    start = time.time()
    #obtain lists containing the words found in the texts
    original = Remove_punct(word_tokenize((open('original.txt','r').read()).lower()))
    plagiarised = Remove_punct(word_tokenize((open('plagiarised.txt','r').read()).lower()))

    Resemblance(Ngram(original,4),Ngram(plagiarised,4),len(Ngram(original,4)))

    #obtain time taken to complete program
    finish = time.time()-start
    print('Time taken = ',finish)

def Remove_punct(text):
    #remove the punctuation in the text
    return [word for word in text if word not in list(punctuation)]

def Ngram(text,n):
    #creates a list of n-grams
    newList = []
    for i in text:
        for j in i:
            newList.append(j)
    ngrams = []
    for i in range(len(newList)-n+1):
        ngrams.append(newList[i:i+n])
    return ngrams

def Resemblance(original,plagiarised,A):
    #calculates the resemblance
    n = 0
    for i in original:
        if i in plagiarised:
            n += 1
    r = n / A

    print('resemblance, R = ',r)

Main()
```

This program produces n-grams of all the words located in the texts. Similarly, to the stop words n-gram, We removed the punctuation and used the tokenize functions from NLTK to split the documents into words. The program then creates n-grams and uses this data along with the number of substrings in ‘original.txt’ as the length to calculate the resemblance using the formula.

All Word N-Gram Results

We used the same method of testing for the stopword n-gram to find the optimal value of n for detecting plagiarism.

n	*Resemblance (Plagiarised)*	*Resemblance (Non-plagiarised)*
1	0.996	0.958
	0.998	0.98
	1	0.984
	1	0.972
	0.999	0.935
Average	0.9986	0.9658
2	0.963	0.883
	0.972	0.912
	0.99	0.955
	0.975	0.927
	0.968	0.886
Average	0.9736	0.9126
3	0.798	0.675
	0.883	0.62
	0.91	0.646
	0.911	0.648
	0.832	0.619
Average	0.8668	0.6416
4	0.547	0.238
	0.708	0.245
	0.761	0.262
	0.779	0.252
	0.624	0.232
Average	0.6838	0.2458

Figure 9a. All Word N-Gram Table

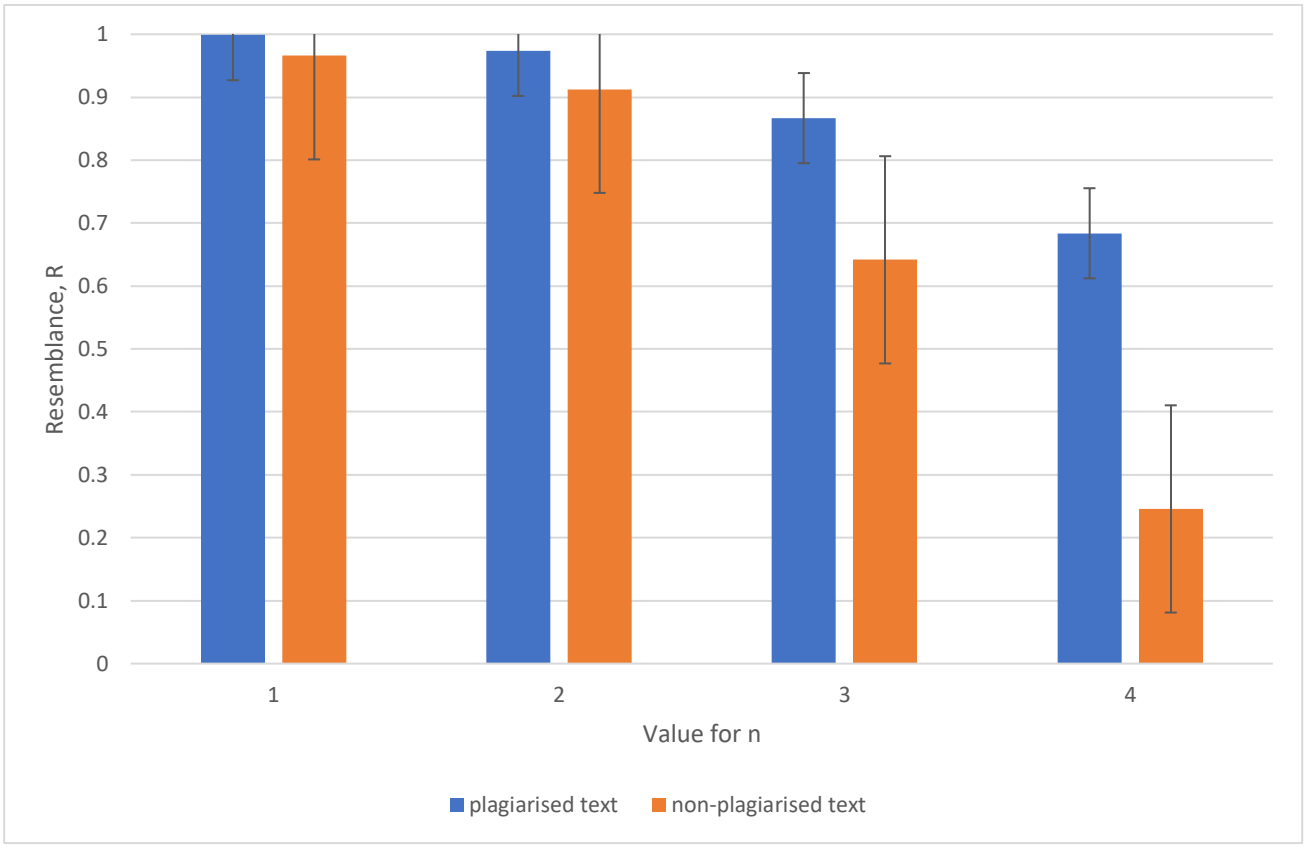

Figure 9b. All Word N-Gram Graph

All Word N-Gram Analysis:

As shown by the graph, the only significant difference found between the texts was when using a quad-gram. Therefore, this was found as the optimal value of n. We did not test values of n greater than 4 because due to the trend of the graph (decrease in resemblance as n increases), if the trend continued then the resemblance would be too low and the data would be of no use.

Fingerprinting conclusion

Overall, we have experimented three different forms of using n-grams in fingerprinting. The least effective was the named entity n-gram due to no significant difference between the plagiarised and non-plagiarised texts. Both the stopword n-gram and all word n-gram showed positive results. The stopword n-gram had a very high resemblance for plagiarised text however this may be due to many false positives which is a consequence of using this method. However, we found this would be very useful in detecting plagiarism as there is a significant difference. The named entity n-gram was the most accurate providing the largest difference in plagiarism. However, the resemblance for the plagiarism test was under 0.7 due to a large value of n.

Our Program

Overall, we have experimented three different forms of using n-grams in fingerprinting. The least effective was the named entity n-gram due to no significant difference between the plagiarised and non-plagiarised texts. Both the stop word n-gram and all word n-gram showed positive results. The stop word n-gram had a very high resemblance for plagiarised text however this may be due to many false positives which is a consequence of using this method. However, we found this would be very useful in detecting plagiarism as there is a significant difference. The named entity n-gram was the most accurate providing the largest difference in plagiarism. However, the resemblance for the plagiarism test was under 0.7 due to a large value of n.

We have decided to use a stop word trigram and an all word quad-gram program in our final plagiarism detection program as they are very effective in detecting plagiarism as shown by the tests performed on them. We have used these specific values for n to provide optimal performance of the program as these proved very successful.

Machine Learning Based Systems

Machine learning is a type of artificial intelligence (AI), that builds a model from example data to make data driven predictions without having to follow strict static program instructions.

Machine learning can learn from data by using supervised or unsupervised machine learning. Supervised algorithms require humans to provide the input and the desired output and the algorithm analyses the features and determines a relationship between an input and output creating a model which can be used on new data to make predictions. The supervised learning algorithms includes two categories of algorithms, regression algorithms such as using linear regression and classification algorithms which include Naïve Bayes classifier and logistic regression.

Unsupervised machine learning involves algorithms which analyse the input data and tries to identify groups of data with similar traits through inferences from the dataset. Whilst supervised machine learning is used to predict values, unsupervised machine learning is used in clusters analysis where the algorithm will try find hidden patterns, groupings or associations in a piece of data. Algorithms for unsupervised learning include the k-means clustering and the hierarchical clustering.

Rule Based Systems

Rule-based systems is when a program checks a series of static rules to determine an output, these static rules are typically decided from experts in a field of study. These are simpler than machine learning approach and humans can understand how the system works whilst machine learning algorithms which used sophisticated algorithms can result in humans not understanding how their program fundamentally works. Rule based systems do not need data pre-processing and are easier to implement however it can be hard to determine the correct rules to use and machine learning algorithms are dynamic the outputs can change once more training data is adding whilst rule based systems are static unless more rules are programmed.

Supervised machine learning algorithms used to classify documents can be used to identify features of plagiarism and can also determine whether a document is plagiarised whilst rule based systems can also be effective if suitable rules are researched and used in the algorithm. Unsupervised learning can be used to analyse plagiarised text but is less effective at detecting it.

Hapax Legomenon

	Original (%)	*Unacceptable Paraphrasing (%)*	*Acceptable Paraphrasing (%)*
	86.6	88.1	89.2
	84.2	72	75
	78.6	75.9	91.3
	63.6	70.8	71.4
	80.9	95	95
	85	82.4	84.2
	71	74	n/a
	79.2	77.8	73.1
	87.5	81.8	76.2
	86.4	76.2	86.7
Average:	78.6	79.5	82.7

Figure 10. % Hapax Legomenon in Text Sample Table

Hapax legomenon are lexical words (words with meaning) which only appear once in a document. Research from the University of Sheffield states that if the percentage of hapax legomenon in a document is around 40% it can be considered as non-plagiarised whereas if it is around 70%, the document should undergo other methods to identify plagiarism [17]. This could be due to increased word repetition when plagiarising text whilst people tend to consciously avoid word repetition in creating original documents. We can use hapax legomenon to create an intrinsic plagiarism detector if there was a large discrepancy between the plagiarised text and the non-plagiarised text.

We collected a collection of paraphrased text from various sources on the internet along with the original. The unacceptable paraphrased text is where the article is not correctly cited and the writer has only changed around a few words and phrases, or changed the order of the original sentences [18] whilst acceptable paraphrasing involves the use of the writer's own words and informs the reader of the information sources. The reason why we used paraphrased text is that word for word plagiarism would result in the original and plagiarised text having the same % of hapax legomenon.

We wrote our own python script assisted by the Natural Language Tool Kit (NLTK) library to calculate the % of hapax legomenon in a text, we started by stemming the word using a Lancaster Stemmer resulting in reducing the word back into their root form and in this case a hapax legomenon is where the root word only occurs once.

The input is then split into an array of words through the process of tokenization and further NLP processing is done on the individual words. This includes the removal of punctuation and stop words which tend to be functional words with little lexical meaning. This allows us to analyse the number of single occurrence lexical words by using a simple function as shown below.

```
import nltk
from nltk.tokenize import word_tokenize, sent_tokenize
from nltk.corpus import stopwords
from string import punctuation
from nltk.stem.lancaster import LancasterStemmer
from collections import Counter

#get the file containing the original text
original = (open('original.txt','r').read()).lower()
'''print(text)'''

#convert words in text to stem words
st = LancasterStemmer()
stemmedWords = [st.stem(word) for word in word_tokenize(text)]
newText = ''
for i in stemmedWords:
    newText += i + ' '

#get an array of sentences from the text
sents = sent_tokenize(newText)

#get an array of words within the array of sentences
words = [word_tokenize(sent) for sent in sents]

#get a list of all stopwords
stopWords = set(stopwords.words('english') + list(punctuation))

#remove all stopwords in the text
wordsWOStopwords = [word for word in word_tokenize(newText) if word not in stopWords]

#algorithm to calculate number of hapax legomenon
freqs = Counter(wordsWOStopwords)
hapax = 0
for i in freqs:
    if i[0] not in list(punctuation):
        '''print(i,' = ',freqs[i])'''
        if freqs[i] == 1:
            hapax += 1

#output values
print('\n number of hapax legomeno = ',hapax)
decimal = hapax/len(freqs)
percent = decimal * 100
print('\n decimal = ', decimal)
print('percentage = ', percent, '%')
```

Figure 11. Python Code Used To Calculate Hapax Legomenon %

As seen in the source code, we used a Lancaster stemmer rather than lemmatizer. This is due to the simpler implementation and the additional performance of the stemmer and as both are normalizing the text so result should hsave no significant difference regarding which we use.

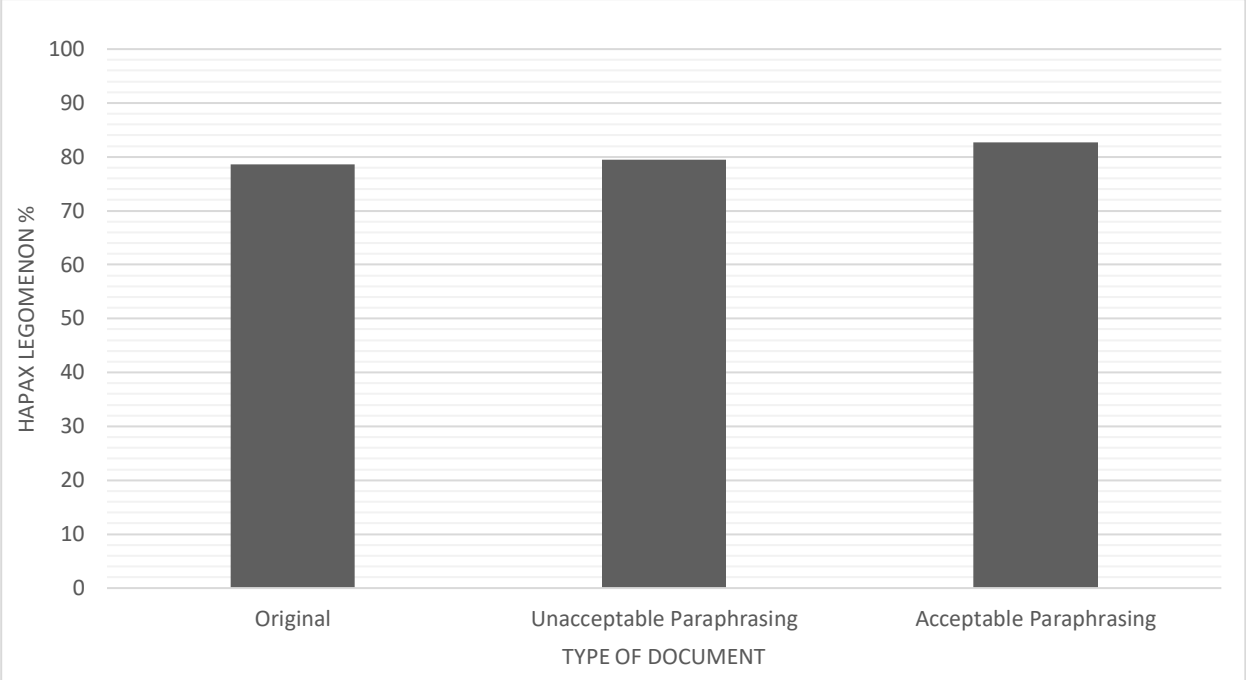

Figure 12. % Hapax Legomenon in Text Samples Graph

Analysis

Contrary to the research conducted at the University of Sheffield, our data suggests that there is a minimal difference of hapax legomenon between the original documents and the plagiarised documents of under 1%. A majority of our sources would be suspected of plagiarising by using their 70% guideline and would require further analysis. In addition, none of the sources would be classified as non-plagiarised with the lowest being 63.6% which is far higher than the 40% guideline for non-plagiarised content and our data does not support the previous paper.

Our research also discovered that acceptable paraphrasing documents generally had an even higher average hapax legomenon % of 82.7%, which should have a lower value as it would be considered original text, the reason for this could be due to the writer is deliberately trying to reduce word repetition to add originality when paraphrasing.

Following the results of our research, we have determined that the hapax legomenon would not be suited for our intrinsic plagiarism detection software as the proportion of documents suspected of plagiarism would be too high and there would be far too many false positives for the detection system to function properly whilst also being unable to detect word for word plagiarism. Our results are not conclusive as we have a small sample size of only 29 samples to test from.

4 PLAGIARISM DETECTOR ALGORITHM DESIGN

We will focus our research on how we can use a machine learning approach to specifically target intrinsic plagiarism. We initially looked at extrinsic plagiarism however after research, we discovered that this area has already been thoroughly researched with efficient hundreds of algorithms to process word for word plagiarism.

We will be focussing on the stylometric aspect where not as much research has been done due to the lower detection rates and our plagiarism algorithm is to be used in conjunction with extrinsic plagiarism so we can increase the detection of disguised and multisource plagiarism where extrinsic detectors have had lower detection rates.

We will be using our research that when plagiarising text, there are many changes in style and one of the key changes is complexity of the text that is used when plagiarising from various sources as some people use disguised plagiarise to steal from many sources. The original context where we got this approach from is the idea that primary school teacher can easily tell if a student is plagiarised based on the complexity of the words that they use and we discovered through our research that different people and websites have unique styles of writing for example BBC bitesize is aimed at school children whilst dissertations are aimed at academics and the style and complexity of the text, in terms of both structure e.g. average word count per sentence, and individual word complexity e.g. average number of letters in a word, should vary.

We will be conducing data collection and analyses to test the hypothesis that “More complex text will have higher average letters in a word and a higher average number of words in a sentence.

4.1 DATA COLLECTION METHOD

For the data collection process, we will get obtain text on the same article from Simple Wikipedia for simple text and Wikipedia for the more complicated text. The aim of the data collection process is to collect data that can be used to test the hypothesis and for our machine learning training data.

We will collect the average number of letters in a word and the average number of words in a sentence from the data by passing it through a piece of python code that we wrote.

Average Sentence Length	*Average Word Length*	*Complexity*
22.5	4.4	0
22.0	4.4	0
18.3	5.0	0
12.0	4.4	0
26.1	4.4	0
15.3	4.3	0
19.9	4.8	0
16.4	4.5	0
16.8	4.9	0
16.5	4.1	0
10.6	5.4	0
18.3	4.0	0
10.3	4.6	0
11.4	4.7	0
14.5	4.4	0
15.6	4.7	0
14.2	4.8	0
12.5	5.0	0
15.8	4.8	0
14.0	4.4	0
13.2	4.6	0
19.5	5.3	1
26.6	5.5	1
18.5	5.1	1
21.0	4.8	1
22.9	5.4	1
24.8	4.8	1
24.9	5.1	1
21.8	4.6	1
19.8	5.3	1
18.6	5.2	1
19.3	5.6	1
21.9	4.9	1
20.0	5.1	1
27.2	5.6	1
23.7	5.6	1
17.1	5.7	1
17.4	5.4	1
19.3	5.4	1
21.2	5.2	1
24.0	4.9	1
21.5	5.8	1

Figure 13. Data Collection Data

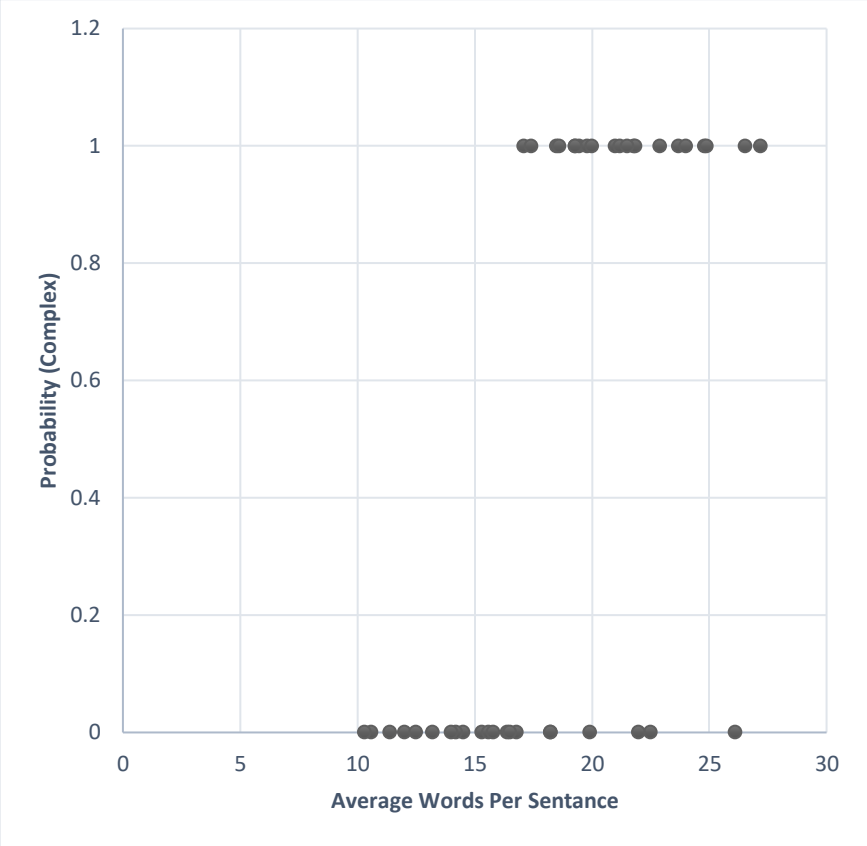

Figure 14a. Average Word Per Sentence vs Complexity

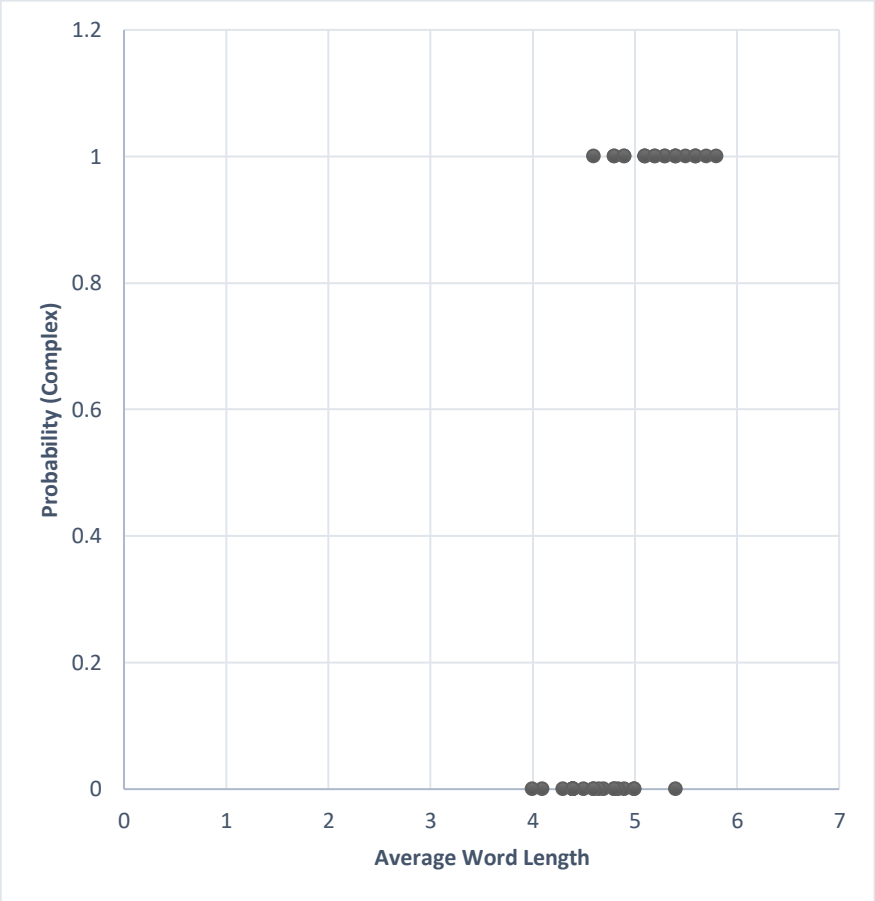

Figure 14b. Average Word Length vs Complexity Graph

4.2 LOGISTIC REGRESSION DATA

Average Sentence Length	*Complex*	*Failure*	*p-Pred*	*Prediction*	*% Correct*
10.3	0	1	0.017	Simple	100
10.6	0	1	0.020	Simple	100
11.4	0	1	0.029	Simple	100
12.0	0	1	0.039	Simple	100
12.5	0	1	0.049	Simple	100
13.2	0	1	0.068	Simple	100
14.0	0	1	0.098	Simple	100
14.2	0	1	0.107	Simple	100
14.5	0	1	0.122	Simple	100
15.3	0	1	0.171	Simple	100
15.6	0	1	0.193	Simple	100
15.8	0	1	0.209	Simple	100
16.4	0	1	0.263	Simple	100
16.5	0	1	0.272	Simple	100
16.8	0	1	0.303	Simple	100
17.1	1	0	0.335	Simple	0
17.4	1	0	0.369	Simple	0
18.3	0	2	0.471	Simple	100
18.5	1	0	0.502	Complex	100
18.6	1	0	0.514	Complex	100
19.3	2	0	0.600	Complex	100
19.5	1	0	0.623	Complex	100
19.8	1	0	0.657	Complex	100
19.9	0	1	0.669	Complex	0
20.0	1	0	0.679	Complex	100
21.0	1	0	0.777	Complex	100
21.2	1	0	0.793	Complex	100
21.5	1	0	0.817	Complex	100
21.8	1	0	0.838	Complex	100
21.9	1	0	0.841	Complex	100
22.0	0	1	0.851	Complex	0
22.9	1	0	0.899	Complex	100
23.7	1	0	0.930	Complex	100
24.0	1	0	0.939	Complex	100
24.8	1	0	0.958	Complex	100
24.9	1	0	0.960	Complex	100
26.1	0	1	0.978	Complex	0
26.6	1	0	0.982	Complex	100
27.2	1	0	0.987	Complex	100
Total	21	20			87.8

Figure 15a. Average Sentence Length Complexity Comparison Table

Average Word Length	*Complex*	*Simple*	*p-Pred*	*Prediction*	*% Correct*
4	0	1	0.006	Simple	100
4.1	0	1	0.010	Simple	100
4.3	0	1	0.030	Simple	100
4.4	0	5	0.052	Simple	100
4.5	0	1	0.088	Simple	100
4.6	1	2	0.145	Simple	67
4.65	0	1	0.184	Simple	100
4.7	0	1	0.230	Simple	100
4.8	2	2	0.344	Simple	50
4.84	0	1	0.396	Simple	100
4.9	2	1	0.479	Simple	33
5.0	0	2	0.617	Complex	0
5.1	3	0	0.739	Complex	100
5.2	2	0	0.833	Complex	100
5.3	2	0	0.897	Complex	100
5.4	3	1	0.939	Complex	75
5.5	1	0	0.964	Complex	100
5.6	3	0	0.979	Complex	100
5.7	1	0	0.988	Complex	100
5.8	1	0	0.993	Complex	100
Total	21	20			80

Figure 15b. Average Word Length Complexity Comparison Table

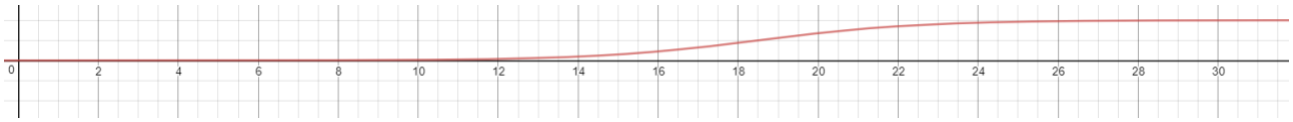

Figure 16a. Average Sentence Length Complexity Logistic Regression Best Fit Line A

$$A = -27.67 \quad B = 5.63 \quad p(x) = \frac{1}{1 + e^{-(-27.67+5.63x)}} \quad p(0.5) = 18.48 \quad X^2 = 26.3$$

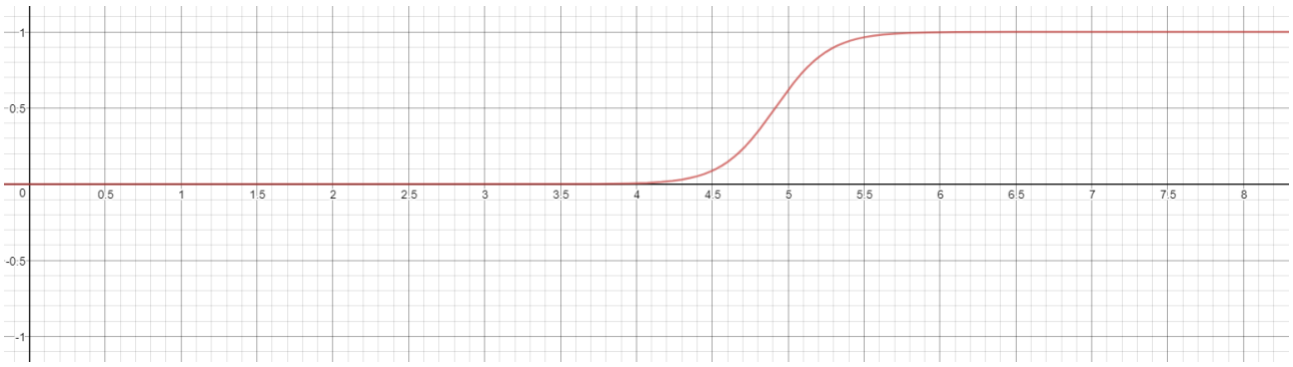

Figure 16b. Average Word Length Complexity Logistic Regression Best Fit Line

$$A = -9.16 \quad B = 0.50 \quad p(x) = \frac{1}{1+e^{-(-9.16+0.50x)}} \quad p(0.5) = 4.9 \quad X^2 = 22.0$$

4.3 DATA ANALYSIS

The data we collected fits a typical binomial distribution S curve and therefore logistic regression is a safe method of measuring our data. We created the logistic regression data set by using the real statistics add-on in Microsoft Excel which gave us an accurate A value and B value by using maximum likelihood estimation and it provides large amount of other statistics useful for data analyses including a chi-squared test which we will use to prove our hypothesis.

The Chi Square test is used to establish if there is a significant objective difference average word length/sentence length and the complexity of the word.

The chi-squared value was calculated in Excel but it can be manually calculated using the formula:

$$X^2 = \sum \frac{(o-e)^2}{e}$$

Percentage Points of the Chi-Square Distribution

Degrees of Freedom	Probability of a larger value of x^2								
	0.99	0.95	0.90	0.75	0.50	0.25	0.10	0.05	0.01
1	0.000	0.004	0.016	0.102	0.455	1.32	2.71	3.84	6.63
2	0.020	0.103	0.211	0.575	1.386	2.77	4.61	5.99	9.21
3	0.115	0.352	0.584	1.212	2.366	4.11	6.25	7.81	11.34
4	0.297	0.711	1.064	1.923	3.357	5.39	7.78	9.49	13.28
5	0.554	1.145	1.610	2.675	4.351	6.63	9.24	11.07	15.09
6	0.872	1.635	2.204	3.455	5.348	7.84	10.64	12.59	16.81
7	1.239	2.167	2.833	4.255	6.346	9.04	12.02	14.07	18.48
8	1.647	2.733	3.490	5.071	7.344	10.22	13.36	15.51	20.09
9	2.088	3.325	4.168	5.899	8.343	11.39	14.68	16.92	21.67
10	2.558	3.940	4.865	6.737	9.342	12.55	15.99	18.31	23.21
11	3.053	4.575	5.578	7.584	10.341	13.70	17.28	19.68	24.72
12	3.571	5.226	6.304	8.438	11.340	14.85	18.55	21.03	26.22
13	4.107	5.892	7.042	9.299	12.340	15.98	19.81	22.36	27.69
14	4.660	6.571	7.790	10.165	13.339	17.12	21.06	23.68	29.14
15	5.229	7.261	8.547	11.037	14.339	18.25	22.31	25.00	30.58
16	5.812	7.962	9.312	11.912	15.338	19.37	23.54	26.30	32.00
17	6.408	8.672	10.085	12.792	16.338	20.49	24.77	27.59	33.41
18	7.015	9.390	10.865	13.675	17.338	21.60	25.99	28.87	34.80
19	7.633	10.117	11.651	14.562	18.338	22.72	27.20	30.14	36.19
20	8.260	10.851	12.443	15.452	19.337	23.83	28.41	31.41	37.57
22	9.542	12.338	14.041	17.240	21.337	26.04	30.81	33.92	40.29
24	10.856	13.848	15.659	19.037	23.337	28.24	33.20	36.42	42.98
26	12.198	15.379	17.292	20.843	25.336	30.43	35.56	38.89	45.64
28	13.565	16.928	18.939	22.657	27.336	32.62	37.92	41.34	48.28
30	14.953	18.493	20.599	24.478	29.336	34.80	40.26	43.77	50.89
40	22.164	26.509	29.051	33.660	39.335	45.62	51.80	55.76	63.69
50	27.707	34.764	37.689	42.942	49.335	56.33	63.17	67.50	76.15
60	37.485	43.188	46.459	52.294	59.335	66.98	74.40	79.08	88.38

Figure 17. Chi-Squared Distribution Table

As there are two possible outcomes, there is a degree of freedom of 1 and as both values are much greater than the critical value of 3.84, we can reject a null hypothesis that there is no significant difference between average word/sentence length and complexity therefore proving our hypothesis.

4.4 LOGISTIC REGRESSION DATA ANALYSIS

The logistic regression curve A value and B values were inputted into Desmos, an advanced online graphing calculator and the sigmoid curve was plotted and the critical values at which the prediction changed was an average word length of 4.9 and an average sentence length of 18.5. This means that if a sample of text has an average word length of above 4.9 then it will be classified as complex whilst if it was under this threshold then it would be classified as simple. The advantage of machine learning is that these threshold values will be dynamic and will constantly change depending on the training data therefore this piece of code will be very versatile.

If we were to use single binomial logistic regression with only one of these two variables, we would pick the average sentence length as an indicator of complexity, this is because the correct prediction % is 88% compared to the 80% for average word length however as seen in the graphs, the average word length seems to be more spread out and there is a much larger spread for simple text. The logistic regression model in Excel will provide an excellent basis before we implement it in python so that we can test that the machine learning algorithm is resulting in the correct values. Considering we only use 40 samples of text and the prediction % is relatively high, this shows that with even more sets off data from various sources such as kids' revision sites and PhD thesis, we can get an even larger contrast and the prediction % should increase further.

The problem with this approach of detecting complexity of a piece of text is that text is not usually classified as either complex or simple and there is a range of values ranging from very complicated to very simple, however for this project, we will be assuming that the text is classified in two relatively broad categories use the resulting complexity in stylometry. This will be done as we compare the complexity of different paragraphs in a suspected piece of plagiarised work by using machine learning and then use a rule based approach to determine whether a piece of work is plagiarised.

Rather than using one independent variable in our binomial logistic regression, we will increase the accuracy of our model further by using a multiple logistic regression model and the real statistics helped us calculate the A value and the B values for our model.

$$A = -92.0 \quad B_1(word) = 1.12 \quad B_2(sen) = 14.3 \quad p(x) = \frac{1}{1 + e^{-(-92 + 1.12x_1 + 14.33x_2)}} \quad X^2 = 47.3$$

```
from numpy import loadtxt, where
from pylab import scatter, show, legend, xlabel, ylabel

#load the dataset
data = loadtxt('PlagiarismData.txt', delimiter=',')

X = data[:, 0:2]
y = data[:, 2]

cplx = where(y == 1)
simp = where(y == 0)
#Plots the data points for complex and simple
scatter(X[cplx, 0], X[cplx, 1], marker='o', c='b')
scatter(X[simp, 0], X[simp, 1], marker='x', c='r')
xlabel('Average Sentence Length')
ylabel('Average Word Length')
legend(['Complex', 'Simple'])
show()
```

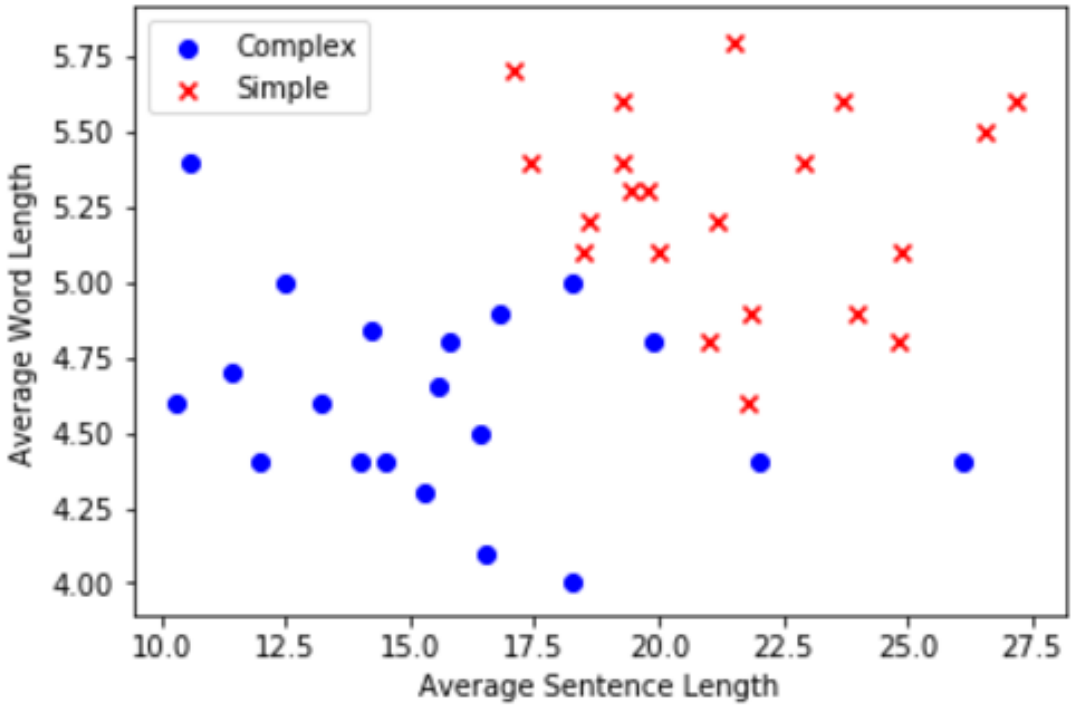

Figure 18 Scatter Graph of Average Word vs Sentence Length

This is a piece of code writing in python and shows the distribution of average sentence length to average word length, this can be used to help instantiate multiple logistic regression.

5. IMPLEMENTATION

```
#Load file + format
input_file = "ExcelData.csv"
data = pd.read_csv(input_file)
data.columns = ["Complex", "Sentence Length", "Word"]
data['intercept'] = 1.0
train_cols = data.columns[1:]
print(train_cols)
#Apply logistic regression algorith
logit = sm.Logit(data['Complex'], data[train_cols])
result = logit.fit()
result.summary()
```

```
Index(['Sentence Length', 'Word', 'intercept'], dtype='object')
Optimization terminated successfully.
         Current function value: 0.113809
         Iterations 10
```

Logit Regression Results

Dep. Variable:	Complex	**No. Observations:**	42
Model:	Logit	**Df Residuals:**	39
Method:	MLE	**Df Model:**	2
Date:	Sat, 08 Jul 2017	**Pseudo R-squ.:**	0.8358
Time:	02:03:43	**Log-Likelihood:**	-4.7800
converged:	True	**LL-Null:**	-29.112
		LLR p-value:	2.708e-11

	coef	**std err**	**z**	**P>|z|**	**[95.0% Conf. Int.]**
Sentence Length	1.1408	0.549	2.078	0.038	0.065 2.217
Word	14.6515	7.027	2.085	0.037	0.878 28.425
intercept	-93.9492	44.227	-2.124	0.034	-180.633 -7.265

Figure 19. Logistic Regression Model Complexity Implementation Using Python

We implemented this simple logistic regression algorithm to calculate the complexity of text and compare the different paragraphs to see if there is a change in style of writing to detect plagiarism.

Final Program

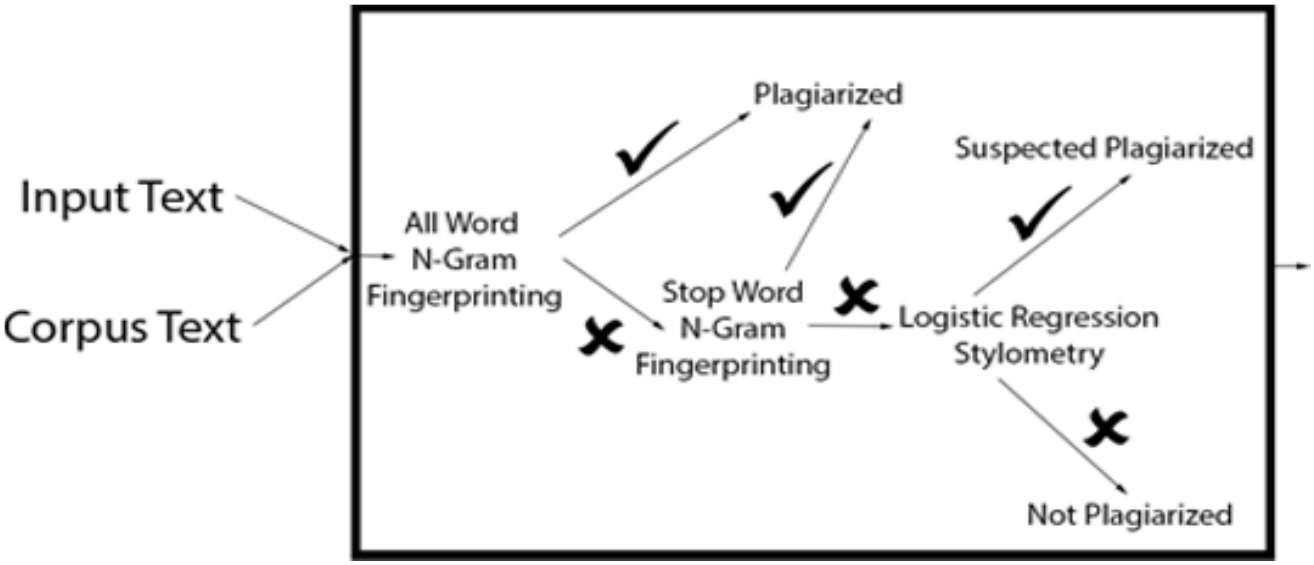

Figure 20. Our Python Program Design

We decided to integrate fingerprinting which is very accurate but requires a lot of computing power with the logistic regression stylometry which tries to analyse a change in writing complexity to detect plagiarism. It will first use all word quad-gram fingerprinting to determine whether it is a word for word or slight word switch plagiarism. If we get a resemblance of greater than 0.6, which is a value we decided on my looking at the data, the graph and the error bars, then we will classify the documents as plagiarised. This is a safe assumption as most non-plagiarised text had a resemblance of around 0.25 whilst plagiarised text was around 0.7 and there was not much overlap.

We will then use a tri-gram stop word fingerprinting algorithm to ensure that all extrinsic forms of plagiarism have been fully detected however during the testing, we found that most of the all word fingerprinting would have already filtered the extrinsic plagiarism so the stop word trigram may not be necessary but it can be a useful precaution. We have set a relatively high threshold of a resemblance of 0.8 as our research indicates that plagiarised text has very high stop word resemblances whilst non-plagiarised text has much lower values. The final stylometry approach uses both a machine learning and rules based approach, it will calculate if each paragraph is complex and then create a complexity index as a ratio and we will use this value to determine changes in style and therefore this can suspect plagiarism like a teacher.

Analysis

Sample	*Original*		*Plagiarized*	*Non-Plagiarized*
1 (med)	All Word N-Gram		0.544	0.335
	Stop Word N-Gram		0.156	0.157
	Complexity Index	0.999	0.999	0
	Overall Detection		Not Plagiarized	Not Plagiarized
2 (med)	All Word N-Gram		0.787	0.507
	Stop Word N-Gram		N/A	0.222
	Complexity Index	N/A	N/A	0
	Overall Detection		Plagiarized	Not Plagiarized
3 (med)	All Word N-Gram		0.76	0.453
	Stop Word N-Gram		N/A	0.178
	Complexity Index	N/A	N/A	0
	Overall Detection		Plagiarized	Not Plagiarized
4	All Word N-Gram		0.595	0.553
	Stop Word N-Gram		0.193	0.237
	Complexity Index	0.001	0.001	0.237
	Overall Detection		Not Plagiarized	Not Plagiarized
5 (med)	All Word N-Gram		0.624	0.341
	Stop Word N-Gram		N/A	0.151
	Complexity Index	N/A	N/A	0
	Overall Detection		Plagiarized	Not Plagiarized

Figure 21. Plagiarism Detection Results Table

Our plagiarism detector was effective at recognising non-plagiarized text and had a relatively high detection rate of plagiarized text. Based on the sample we used, we have an accuracy rate of 80% which is good however there could be a few improvements, if we lowered the threshold for the all word n-gram resemblance e.g. 0.58 then we would have gotten a slightly higher proportion however based on the data, the plagiarism detector we have created is fast and effective.

We used challenging sources of input which consists of more difficult paraphrasing and multiple language translation via google translate plagiarism which is much harder to detect. The all word fingerprinting is by far the most useful detection technique and detected the vast majority of instances. The accuracy would be even higher if we used word for word plagiarism comparison as the all word n-gram is very efficient at comparing these documents.

The stop word n-gram worked less effectively than we would have hoped for as our research indicated values of above 0.8 for plagiarism but the actual values for both plagiarised and non-plagiarized have been much lower which could be due to an error in the code.

The complexity index in linguistics is not very good at detecting the samples that we provided as we used very complicated medical journals which would require complex language regardless of being plagiarised therefore if we have more time then we will try experiment it on other sources of plagiarism however we can compare the complexity index of the original sample to the complexity of the inputted documents to help fingerprinting. This is supported by our data as the plagiarized and the original document had very similar complexity indexes so we can do further comparisons on these to refine our model.

Conclusion

Our program was a plagiarism detector that used various algorithms from extrinsic to intrinsic plagiarism detection. The current extrinsic plagiarism detectors are very powerful and accurate at detecting plagiarism however in this experiment, we only compared the documents to one other document. In future experiments, we need to compare the performance of comparing that document with multiple documents to see whether extrinsic plagiarism detection is limited by computational power or whether technological advances will allow fast accurate plagiarism detection software by using these techniques.

The run time of the program was good although there was some processing time required for the machine learning algorithm. The all word n-gram tends to take a substantial amount of time longer than the stop word n-gram however the performance far outweighs the performance deficit.

Choosing 3 different methods and combining them into our program is an ideal number of functions in retrospect as having more would result in too many false positives whilst having fewer functions to detect plagiarism would result some cases of plagiarism slipping through.

We used both rule based systems and machine learning approaches to our program and found that rule based systems are just as effective if not more effective than machine learning if experts decide on an optimal threshold however the adaptability of machine learning makes it a far greater asset.

Natural language process has played a vital role in helping us evaluate and giving up more options of plagiarism detection with some having even better performance. The Python libraries were very powerful and helpful for this project and without them, our code would increase 10 fold.

6. REFERENCES

[1] Oxford University. Plagiarism. www.ox.ac.uk/students/academic/guidance/skills/plagiarism

[2] Mahalakshmi, S., and S. Kavitha. "Software Program Plagiarism Detection Using Longest Common Subsequence Method."

[3] Python Software Foundation. *https://www.python.org/doc/essays/comparisons/*

[4] Chowdhury, Gobinda G. "Natural language processing." Annual review of information science and technology 37.1 (2003): 51-89.

[5] Grefenstette, Gregory, and Pasi Tapanainen. "What is a word, what is a sentence?: problems of Tokenisation." (1994). Page 4

[6] Kiss, Tibor, and Jan Strunk. "Unsupervised multilingual sentence boundary detection." Computational Linguistics 32.4 (2006): 485-525.

[7] Wikipedia. *https://en.wikipedia.org/wiki/Stemming.* Algorithm

[8] Professor Dan Jurafsky, Stanford University (NLP Course) Word Normalization and Stemming *https://www.youtube.com/watch?v=2s7f8mBwnko*

[9] Stanford University NLP. Stemming and lemmatization. *https://nlp.stanford.edu/IR-book/html/htmledition/stemming-and-lemmatization-1.html*

[10] Lancaster University. The Lancaster Stemming Algorithm (July 2014 Archive) http://web.archive.org/web/20140725114639/http://www.comp.lancs.ac.uk:80/computing/research/stemming/Links/porter.htm.

[11] Bull, Joanna, et al. "Technical review of plagiarism detection software report." (2000).

[12] Lyon, Caroline, Ruth Barrett, and James Malcolm. Experiments in electronic plagiarism detection. University of Hertfordshire, 2003.

[13] Wikipedia. *https://en.wikipedia.org/wiki/Plagiarism_detection.* Plagiarism approaches diagram

[14] Clough, Paul. "Old and new challenges in automatic plagiarism detection." National Plagiarism Advisory Service, 2003; http://ir.shef.ac.uk/cloughie/index.html. 2003. Page 2

[15] Kent, Chow Kok, and Naomie Salim. "Features based text similarity detection." Page 2

[16] Shrestha, Prasha, and Thamar Solorio. "Using a Variety of n-Grams for the Detection of Different Kinds of Plagiarism." Notebook for PAN at CLEF 2013 (2013). Page 2

[17] Clough, Paul. "Plagiarism in natural and programming languages: an overview of current tools and technologies." (2000) Page 12.

[18] Indiana University. *http://wts.indiana.edu/pamphlets/plagiarism.pdf.* Examples of Plagiarism, and of appropriate Use of Others' Words and Ideas (2011) Page 1