Imperial College London

INDIVIDUAL PROJECT REPORT

DEPARTMENT OF COMPUTING

IMPERIAL COLLEGE OF SCIENCE, TECHNOLOGY AND MEDICINE

Detecting Plagiarized Automatically Translated Documents

Author:
Calin-Andrei Alexandru

Supervisor:
Dr. Thomas Lancaster

June 13, 2021

Abstract

This report documents the development of a cross-language plagiarism detection algorithm, hosted on a website with an easy to use UI. CL-PD is a subset of plagiarism, which involves translating one's work and taking credit for it. Therefore, the algorithm aims to detect texts that were taken from the indexed part of the World Wide Web and translated into English using automatic translation tools.

The website was developed with the help of React and Typescript and consists of a front-end and back-end. The back-end contains the main algorithm, developed in Python. The algorithm has three stages: Heuristic Retrieval of candidate documents, Detailed Analysis and Post Processing. Those stages' implementation is thoroughly documented and one can easily reproduce these steps by going through the report. The background section features a detailed summary of the different state-of-the-art methods and models for performing cross-language textual similarity analysis.

In addition to this, a full evaluation of the algorithm was performed to assess the success of the project. The evaluation was performed through an experiment, which consisted of gathering 450 chunks of text from 450 randomly selected non-English Wikipedia pages. The chunks were translated into English using the Google Translate API and passed as arguments to the algorithm. The obtained results were generally good, with the algorithm achieving an accuracy of 46% and a precision of almost 75%.

Finally, a conclusion of what went well and what could have gone better is presented, alongside some suggestions and explanations about possible extensions.

Acknowledgements

Firstly, I would like to thank my project supervisor, Dr. Thomas Lancaster, for his advice, suggestions and for the fact that he listened to everything I had to say during our meetings. It was great knowing that he approves of my ideas and that I am not straying off the right path.

Secondly, I would like to thank Mrs Li, Yingzhen for reading my report and giving me important feedback related to time management.

Thirdly, I would like to thank all the staff at the Department of Computing at Imperial College London for everything they taught me during my three-year stay. It was a great experience that helped me grow and develop into a mature individual.

Finally, I would like to thank everyone else that was around me, my friends and family who have continuously showed their support along the way and without whom I would not have been able to see this project through.

Contents

Chapter 1

Introduction

Automatic translation, also known as machine translation is a ”translation carried out by a computer”. It is sometimes referred to as being a **Natural Language Processing** technique that builds language and phrase models used to translate, with the help of bilingual data sets and language assets [12]. Since its creation, machine translation rapidly advanced and in this current age, there are plenty of useful and powerful services such as **Google Translate** or **DeepL** that provide their users with accurate automatic translation mechanisms. When automatic translation tools first appeared, it was easier to distinguish a manually produced translation from one produced by a machine since the latter was less accurate than the former [26]. However, this has changed, and along with it, issues that were not encountered beforehand have emerged, such as cross-language plagiarism in academic institutions [8]. This project aims to tackle the aforementioned issue with the help of an algorithm that detects whether or not a given document has been plagiarized with the use of automatic translation tools.

1.1 The Problem

Plagiarism has been an important issue for academic institutions since their inception, but it became even more common since the rise in popularity of the World Wide Web. This happened because the internet grew rapidly and it currently contains an impressive collection of resources [14]. Plagiarism can take many forms, from extracting and paraphrasing information from another person's work without giving credit, to straight out copying parts or even another person's entire work. However, in recent years, a new form of plagiarism has emerged, a form which is hard or even impossible to automatically detect: cross-language plagiarism or translation plagiarism [23]. The biggest challenge with this problem derives from the fact that whether the translation is performed manually or automatically, some degree of rewriting always occurs. Therefore, when translating the obtained text back to the original language, it would no longer resemble the original text since two layers of rewriting were applied.

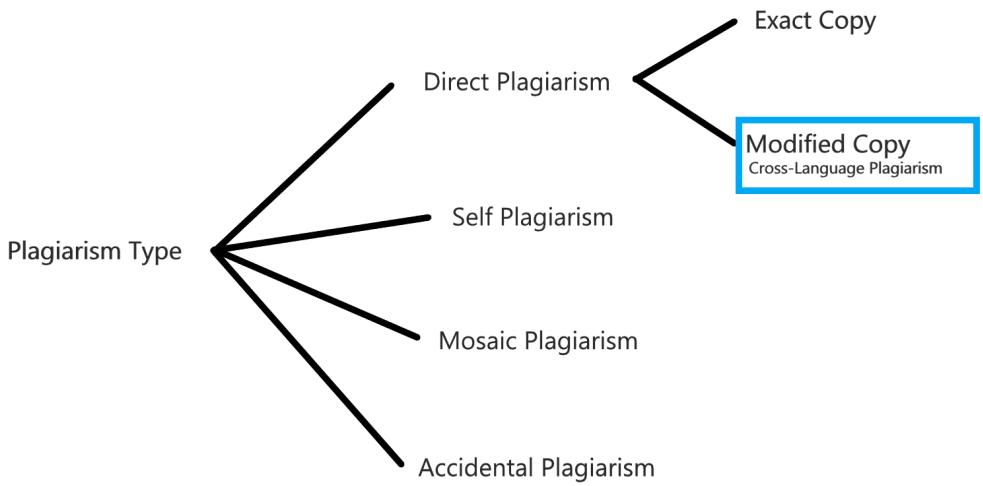

Figure 1.1: The different types of plagiarism

As observed in Figure 1.1, Cross-language plagiarism is a form of direct plagiarism that leads to a modified copy of the original text. This form of cheating occurs when one takes work written in a different language and translates it into another language manually or with the help of automatic translation services such as **Google Translate** or **DeepL**. Since academic submissions are generally passed through a plagiarism check, individuals who engage in cross-language plagiarism try to cheat this filter, which might not signal any issues for work that has been translated from another language. [23].

In addition to facilitating cross-language plagiarism, automatic translation services may also lead to people who hire professionals to produce a translation for a document not receiving the service that they paid for. Since the client might not have enough knowledge of the language of translation, that would lead to them not having a way of knowing whether the translation was performed automatically or manually. The client could try and translate the document back from English into the original language, but it would be nearly impossible to conclude if the differences are not obvious enough.

For example, if the following sentence from Romanian is taken: ”Clientul nu are cum să știe dacă a fost păcălit sau nu”, translated into English and then back into Romanian, the result will be: ”Clientul nu are de unde să știe dacă a fost păcălit sau nu”. The semantic of the sentence is the same, the only notable difference being a small syntactic change, which comes from replacing one of the words in the original sentence with a synonym. Therefore, in this specific example, there is no obvious reason to suspect the use of automatic translation. But perhaps, at a closer inspection, certain machine translation patterns might be uncovered, patterns that would make the process of detection of automatically translated documents more effective [22]. However, the focus of this project is not to develop an algorithm that detects if

a certain text has been automatically translated or not, but to check whether or not the text has been plagiarized with the help of automatic translation tools.

1.2 The Proposed Solution

It becomes clear that both the problem of cross-language plagiarism in academic institutions and human translation services not delivering what was requested of them might arise from the use of automatic translation software. This project will focus on the former since the latter would involve the use of advanced **Natural Language Processing** techniques and would therefore require a very complex or even impossible to implement a solution. Therefore, the project aims to create a mechanism that allows people to easily check whether a piece of academic work has been plagiarized with the help of automatic translation tools. Ideally, this mechanism will be hosted on a website such that it can be accessed by any person. Moreover, the website will be intuitive and easy to navigate through to not confuse the users.

Since this website will be part of the public domain and available for use to any person, members of the academic community will have at their disposal a mechanism that will allow them to perform an extra check for plagiarism.

Figure 1.2: The process of Cross-Language Plagiarism Detection

The mechanism that will be at the core of this website, presented in Figure 1.2, will have three important steps: *Heuristic Retrieval*, *Detailed Analysis* and *Post Processing*. The first step will involve combining several **Natural Language Processing** techniques, such as *Machine Translation* and *Keyword Extraction*. After the *Heuristic Retrieval* step is completed, the *Detailed Analysis* step may begin. In this stage, several of the models proposed in the Background section can be used, such as **CL-CNG**, **CL-ESA**, etc. While the former requires no training, since it uses n-gram word decomposition, the latter can take the form of a *Neural Network* which will perform a classification task since it has to answer the question of whether a given text was plagiarized with the help of automatic translation tools or not. In the training

stage, the model will make use of several cross-language data sets available online, which contain numerous documents that have been manually translated by professionals. The network will be provided with thousands of examples of both manually and automatically translated documents with the end goal of being able to correctly identify texts that are textually similar to one another. Finally, after the second step is completed, the last step involves analyzing the suspicious candidate documents, that arise from the previous step, in detail, to filter out any possible false positives.

While the *Heuristic Retrieval* and *Post Processing* steps will generally be very similar, the *Detailed Analysis* step will differ depending on the chosen model. For the models that require training a neural network, publicly available training data sets already exist and they contain a vast number of data points. Due to this, the data will most likely be split into three sets: one for training, one for validation and one for testing, with an 80-10-10 ratio. The largest part of the data will be used for training, to create the model. The validation data set will be used for hyperparameter tuning, to improve the performance of the previously generated model, while the testing one will be used for determining the accuracy when run on previously unseen data. The use of a training data set that contains data points that have not been used for training or validation will provide a general idea of the performance of the model, ensuring its robustness and that it does not overfit the training data. The goal will be to adjust the design in a way that maximizes the accuracy while minimizing the computation time.

1.3 Ethical Considerations

Since the website that was developed during this individual project will be part of the World Wide Web, humans will interact with it. However, this does not pose many problems from an ethical point of view, since the only way the users will engage with the website will be by submitting a piece of text or a document and getting some sort of feedback. This means that no account creation is required and therefore no personal data or information about the users will be collected or processed. In addition to this, the texts that the people want to get checked for plagiarism will not be stored for further use after processing, since there are enough publicly available data collections that can be used for testing the algorithm. Moreover, when performing the candidate selection step of the cross-language plagiarism detection process, only pages from the indexed part of the World Wide Web are retrieved. Therefore, no major ethical issues can arise regarding the human use or protection of personal data.

The most obvious ethical problems that were considered are the legal issues, which emerged from the use of software with copyright licenses. During the development of the three stages of the cross-language plagiarism detection algorithm, various modules and packages were used and were credited accordingly. This was done on the website, in a screen dedicated to crediting, since that is the part of the project that is available to the public.

In conclusion, although the project did not present many ethical issues, those that arisen were treated carefully by generally giving credit where credit was due. In addition to this, a final issue that was considered was what happens when the detection algorithm gets the wrong answer. However, this was more of a moral issue rather than an ethical issue. Nevertheless, to prevent such a problem, the algorithm only provides a list of suspicious URLs and chunks of text. Therefore, it will remain up to the users to ultimately decide whether or not the submitted text has been plagiarized with the use of automatic translation mechanisms or not by inspecting the feedback provided.

Chapter 2

Background

In this section, an overview of the different *State-of-the-Art* existing methods for performing the *Detailed Analysis* step will be presented. These mechanisms are not used for directly detecting translation plagiarism, but are part of the second stage of the cross-language plagiarism detection process. Their purpose is to detect textual similarities between documents or fragments of them. Five retrieval models present different approaches for performing the task of textual similarity detection. Each of them has the same goal, of deciding whether or not two or more pieces of text written in different languages are textually similar.[1]

2.1 CL Translation Detection Models

In this section, the five aforementioned models will be presented, to offer a general idea regarding the different approaches that can be taken for combating cross-language plagiarism.

- **Syntax-Based** models distinguish themselves by the fact that multilingual documents can be compared without the need of being translated, by just looking at the syntactic structure of the sentences. The best performance is obtained when working on pairs of languages that share a similar syntactic structure, for example, the *Romance* languages [7]. The most common approach of this class is the *Cross-Language Character N-Gram* method or in short **CL-CNG**.

- **Dictionary-Based** models make use of thesauri, dictionaries or other concept spaces [7], such as the *JRC-Acquis Multilingual Parallel Corpus* developed by *Eurovoc*, which establishes connections between texts through *language-independent anchors*. Those *anchors* are pairs of words from numerous languages denoting entity names, locations, dates etc [2]. The most common approach of this class is the *Cross-Language Conceptual Thesaurus-Based Similarity* method or in short **CL-CTS**.

- **Comparable Corpora-Based** models look into comparing the bodies of texts. Unlike **Parallel Corpora-Based** models, **Comparable Corpora-Based** models do not make use of sentence-aligned translations. However, this concept is

best illustrated through examples such as **Wikipedia** and similar data sources, by taking advantage of the texts with the same topic that have a common vocabulary. Although noisier than the parallel model, the comparable corpora is more flexible [7]. The most common approach of this class is the *Cross-Language Explicit Similarity Analysis* method or in short **CL-ESA**.

- **Parallel Corpora-Based** models make use of sentences being aligned, as mentioned above, by use of the **IBM Model 1** [15]. In addition to this, the position of the words is taken into account and statistical dictionary probabilities are calculated with the help of an *expectation-maximization* algorithm [7]. The most common approach of this class is the *Cross-Language Alignment Similarity Analysis* method or in short **CL-ASA**.
- **MT-Based** models first determine the language from which the suspicious text was translated with the help of a language detector. Afterwards, the fragment is translated and monolingual analysis is performed [7]. The most common approach of this class is the *Translation + Monolingual Analysis* method or in short **T+MA**.

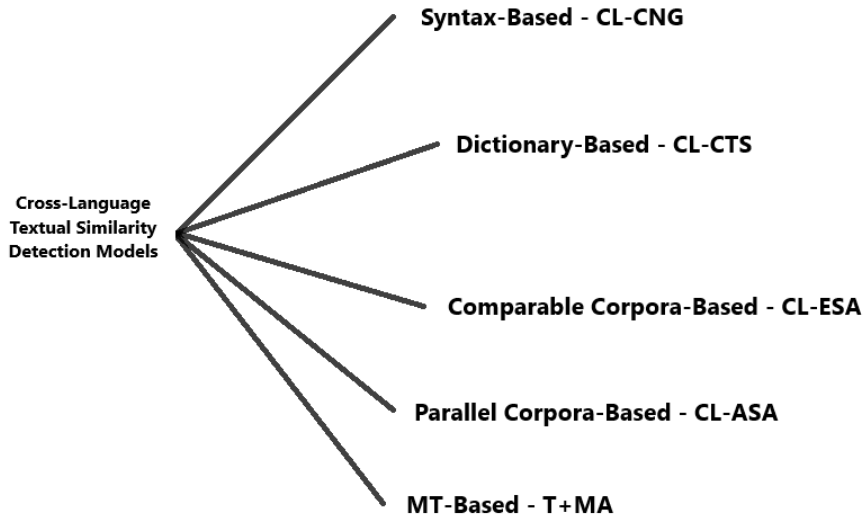

Figure 2.1: The different retrieval models

2.2 CL Translation Detection Methods

Figure 2.1 displays the different techniques that can be used for performing the task of cross-language translation detection. However, this section will present one approach from each aforementioned model, alongside the approaches that could be

most relevant to the development of the algorithm mentioned in Chapter 1.

The most important step of cross-language translation detection is *Information Retrieval*, which given two documents, d_1 and d_2, compares them with the help of a *Retrieval Model* R, which is a tool for computing representations D_1 and D_2 of the aforementioned documents, alongside a *similarity function* f. This function receives as parameters D_1 and D_2 and its result is a real number, which offers information about the textual similarity between d_1 and d_2 [17]. One of the most commonly used functions is the cosine similarity, which returns a value in the interval $[0, 1]$. The closer the value is to one, the higher the similarity between the two given documents is.

2.2.1 CL-CNG

As its name suggests, the *Cross-Language Character N-Gram* approach makes use of *n-grams*, which are a reunion of n arbitrary chosen characters. Since a dictionary is needed for information retrieval and the size of it may be very large or in theory even infinite, a method that would lead to its effective reduction had to appear. Once the n-gram approach was applied to the task of information retrieval, the number of items in the collection drastically reduced, since it was bounded by the number of letters in the alphabet to the power of n. Therefore, for an n that is quite small, in the range $[1, 5]$ for example, the number of possible *n-grams* becomes tractable in the context of memory handling [11].

For example, in the case of the Romanian alphabet, which contains 32 characters, including the whitespace and *2-grams*, the size of the dictionary would be 1024, which is quite a small number. In this same case, no more than 32.768 *3-grams* could be found. Although a significantly larger number, it is still not a big challenge, computationally-wise for most modern-day CPUs.

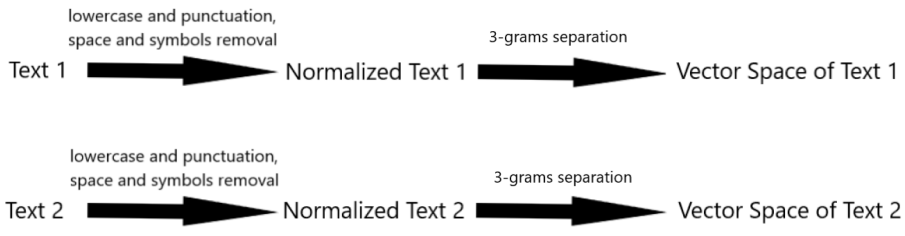

Figure 2.2: The first steps of the CL-C3G approach

CL-CNG is the oldest, being developed in 2004, and arguably the simplest of the models described in this section. Since the variant that produces the best results is **CL-C3G**, from this point forward, *3-grams* will be used as the standard when offering details about this approach. It uses a simplified alphanumeric alphabet, which does not contain any special characters or symbols. As a result, before performing the

separation into *3-grams*, any given text must first have all its characters transformed into lowercase and have its punctuation marks, blank spaces and symbols removed. After all these preliminary steps have been performed, the text can be separated into *3-grams* [17]. The aforementioned steps can be observed in Figure 2.2.

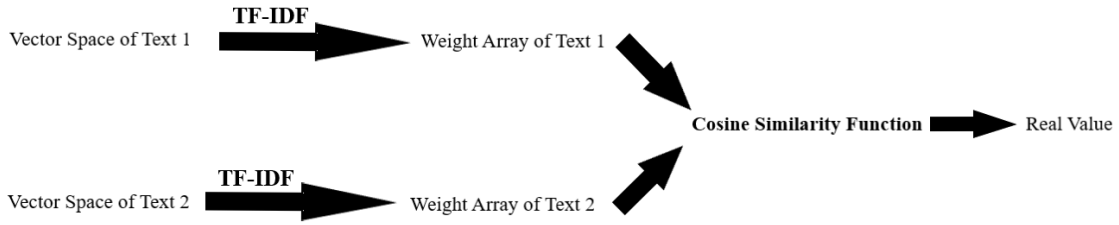

Figure 2.3: The final steps of the CL-C3G approach

Figure 2.3 showcases the final steps of **CL-C3G**. The given document is represented as a vector with a dimension of the size of the alphabet to the power of three. Since in a given text only a part of all the possible *3-grams* combinations occur, the vector space can be sparsely populated. Because of this, its elements have to be weighted following a standard weighting scheme, which in this case is the *term frequency-inverse document frequency* or in short **TF-IDF** [17]. This metric is calculated by multiplying two different measurements: term frequency and inverse document frequency. The former relates to simply counting the number of occurrences of a given word in a text and possibly adjusting this number to the length of the document. The latter relates to calculating how rarely or often is a word encountered in a given set of documents. **TF-IDF** takes real values in the interval $[0, 1]$. If the number is very close to 0, it implies that the word is common and comes up in numerous documents [27].

After these steps are performed for the two candidate texts, the only remaining step is to assess the textual similarity. This is done using the *cosine similarity* measure since the documents have been previously represented as vector spaces. The advantage of using this metric is that it doesn't take into account the size of the documents when computing the similarity of two documents. From a mathematical point of view, it measures the cosine of the angle between two vectors projected in a multi-dimensional space. In the case of **CL-C3G**, the two vectors are the arrays that contain the *3-grams* of the two documents. If plotted on a multi-dimensional space, with each dimension being represented by a *3-gram*, the *cosine similarity* will showcase the angle of the words, rather than their dimension. This is preferable in the given case since a smaller angle will result in a higher similarity between the two documents [19].

In conclusion, **CL-C3G** is a simple, but powerful approach for computing textual

similarity between two given texts. However, in some certain cases, it might be too slow. In addition to this, smaller documents have fewer words, which results in a low number of *3-grams*. Therefore, most *3-grams* in the dictionary will have a value of 0 when constructing the vector space which will lead to inefficient use of memory.

2.2.2 CL-CTS

The *Cross-Language Conceptual Thesaurus-Based Similarity* method aims to calculate the textual similarity of two given documents by a measure of *shared concepts*, which is assigned using a **Conceptual Thesaurus** and **Named Entities** [16].

English	Romanian	French	German
commercial law	drept comercial	droit commercial	Handelsrecht
foreign policy	politică externă	politique extérieure	Außenpolitik
working conditions	condiții de muncă	condition de travail	Arbeitsbedingungen

Table 2.1: Multilanguage samples extracted from Eurovoc

A **Conceptual Thesaurus** incorporates words or structures of words and tries to cover all the concepts of various domains. One of the most popular and useful CTs is *Eurovoc*, which was mentioned in the previous section. It was created by the European Parliament and is currently maintained and constantly updated by the Publications Office of the European Union, with new words or languages [16]. It contains more than 6000 multilingual concepts which have been categorised through identification numbers. In addition to this, the part that makes it valuable is that the words that it contains have been translated into 27 languages and include more than 20 domains that the European Parliament activates in. Table 2.1 presents three examples of word constructions extracted from *Eurovoc*.

Similar to **CL-C3G**, the documents that are checked for textual similarity are represented as vectors. However, instead of creating a vector space from the words in the documents, this approach creates it using the concepts present in *Eurovoc*. However, the problem of word assignment is not easily solvable, since directly assigning a word based on its direct occurrence proved to produce insignificant results [3]. Due to this, a concept is assigned to a document through the following function:

$$v(e,d) = \sum_{t \in e, T_e} f(t,d) \tag{2.1}$$

In this function, e is a *Eurovoc* concept and d is one of the documents that need to be analyzed. The function f is used for calculating how frequently t, the stemming of e, appears in the text and T is the set of all *Eurovoc* words. A given concept e is assigned to document d, with the weight v(e,d), if the function v(e,d) produces a value larger than 0 [16].

Due to its multilingual structure, the CT can be exploited based on the idea that two or more terms combined can be domain-dependent, but each term alone can be domain-independent. For example, a construction such as ”working conditions” is more specific than just the terms ”working” or ”conditions”, which may be present in a vast number of domains. As a result, not all terms are useful when computing the similarity estimation between two given documents. The function of document frequency or *df* was introduced to combat the noise introduced by the similarity increase of non-relevant documents when df(t) is very high. A new set was created from T, in short **RC**, from reduced concepts, which includes only those terms which satisfy the following equation, with the variable β being an arbitrarily small value [16]:

$$0 < df(t) < \beta \tag{2.2}$$

In a previous paragraph, it was mentioned that the concepts in the *Eurovoc* CT receive an identification number. Due to this, when constructing the vector space of the document, each id represents one dimension. to compute the textual similarity between two given documents d_1, in language L_1 and d_2, in language L_2, the following equation is used [16]:

$$\omega(d1, d2) = \frac{\alpha}{2} * \left(\frac{\overrightarrow{c_{d1}} \cdot \overrightarrow{c_{d2}}}{|d1||d2|} + l(d1, d2) \right) + (1 - \alpha) * \xi(d1, d2) \tag{2.3}$$

This equation contains two terms: the CT component and the *named entity* component. The c vectors correspond to the conceptual vectors of the two documents that are compared, while $||$ denotes the size in words of d_1 and d_2. The ξ function returns the *cosine similarity* of the character *3-grams* of the *named entities* of the two documents, while the function *l* is called the length factor penalty for the given documents and is defined in [18]. $(1 - \alpha)$ is a weighting factor used to normalize the output of the function between the interval [0,1] [16].

The use of *named entities* and implicitly of the second component of the equation emerged from the idea that those **NEs** can be considered as distinguishing features when identifying diverse documents on similar conceptual topics. Since the *named entities* usually have more or less significant variations through different languages, *character n-gram* is used to compute textual similarity. In addition to this, parallel documents have specific length distributions [18], which helps with the process of integrating the information about length for parallel document pairs. Therefore, by using the *length factor*, this information is induced in the textual similarity estimation process [16].

In conclusion, although **CL-CTS**, outperforms **CL-CNG** from a computational and temporal point of view, it produces better results only on particular data sets. However, its strongest advantage is that it produces high stability across all the data sets while performing consistently [16].

2.2.3 CL-ESA

The *Cross-Language Explicit Similarity Analysis* method is a multilingual retrieval model used for computing cross-language textual similarity. This approach makes use of the multilingual alignment of Wikipedia documents. The main goal of this method is to generate two vectors, for two documents written in two different languages, and to compare them using a measure such as the cosine similarity. The most important feature of this retrieval model is that it performs semantic analysis without the need for automatic translation capabilities. The precursor of this approach is the retrieval model called *Explicit Semantic Analysis*, or in short **ESA** [13].

The idea of **ESA** is to use a collection of documents to encode the specific knowledge of a given document with respect to it. Therefore, for each specific document in the set, a single concept is created, which the given text is compared to. Due to this, the *Cross-Language Explicit Similarity Analysis* approach represents documents as n-dimensional concept vectors using the following formula [13]:

$$\mathbf{d} = \big(\varphi(v, v_1^*), \varphi(v, v_2^*), ..., \varphi(v, v_n^*)\big)^T \tag{2.4}$$

This approach uses a collection of texts D^* called index documents, which has a size of n. From D^*, v_i^* is extracted, representing the vector space model representation corresponding to the *i*th entry in D^*. In addition to this, v is the vector space model representation for the document that needs to be analyzed. The dimensional vector **d** is created by applying the *cosine similarity* function φ to all the (v,v_i^*) pairs. In the case in which the *cosine similarity* of such a pair is too small, the value is set to 0 [13].

After calculating the n-dimensional concept vectors **d1** and **d2** for two documents that need to be analyzed for textual similarity, the *cosine similarity* function is applied again, to **d1** and **d2** this time, with the result being defined as the similarity between the two documents under ESA [13].

The method presented above can naturally extend to multiple languages due to its nature, by having document index collections in different languages. Due to this, there is no need for any translation technology, just a comparable set of texts about similar topics written in various languages. One such set of documents is Wikipedia, which contains a variety of concepts written in many different languages and one such approach that makes use of it is **CL-ESA**. [17].

The *Cross-Language Explicit Similarity Analysis* method features 3 different sets: **L**, D^* and **C**. **L** represents the collection of languages, D^* is the set of index document

collections, with each D_i^* containing texts in the language L_i and **C** contains several *concept descriptors*. If D^* has the property that *i*th document of all the index document collections in D^* describes c_i in the language l_i then it is called a *concept-aligned comparable corpus* [17].

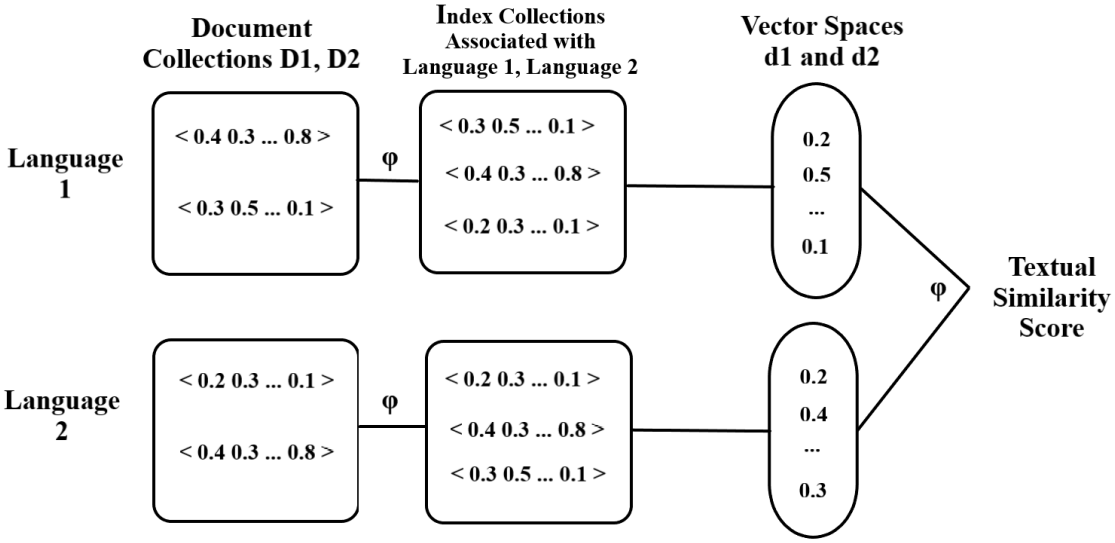

Figure 2.4: CL-ESA approach

Figure 2.4 displays the main steps of the **CL-ESA** approach. Given two documents written in two different languages, they are both represented as **ESA** vectors with the help of the index document collections corresponding to the languages of the documents. The similarity analysis is performed in the concept space, by computing the cosine similarity between the vectors associated with the two documents. In this step, **CL-ESA** utilizes the reasoning of a *comparable corpus alignment* that if all the concepts in C are described well and complete for all the languages in the set **L**, the two documents are represented in *comparable concepts spaces*, with the use of the index document collections associated to them. For this approach to yield results, each index document set must meet the requirements of **ESA** [17].

The *Cross-Language Explicit Similarity Analysis* approach can be tuned to obtain a higher or lower retrieval quality, depending on the number of desired languages. For example, if the former is desired, the documents that need to be analyzed have to be represented as 10^5-dimensional concept vectors. However, in this case, the downside of having high accuracy is that computation time becomes quite significant and no more than 2 languages can be represented at the same time. If high multilingualism is needed, the texts should be represented as 10^3-dimensional concept vectors, for the approach to be feasible from a computational point-of-view. However, the retrieval quality is highly affected by this representation. A decent balance between computational time and retrieval quality is achieved by representing the documents as vector concept spaces with dimensions between 10^3 and 10^4 [17].

2.2.4 CL-ASA

As mentioned before. **CL-ASA** has at its core statistical machine translation technology. It initially functioned by combining two measures, the *translation model probability* and *language model probability*. However, those two models were replaced with ones that produced better results: the *Translation Model* and *Length Model*, which will be presented next [17].

The *Cross-Language Alignment Similarity Analysis* method entails the creation of a *bilingual statistical dictionary* called *Cross-Lingual Plagiarism Analysis system* or in short **CLiPA** on the assumption that the parallel corpus is aligned and using the *IBM Model 1* [7].

Model 1 is a probabilistic generative model that functions inside a benchmark which assumes that any given source sentence **S** of length **l** translates into a target sentence, **T**. This translation is ensured through a process that consists of 2 steps. The first step is generating a length *m* for the target sentence. The second step is done for each target sentence position $j \in 1, ..., m$ and consists of selecting a generating word from **S** and creating a target word t_j at position *j* concerning the generating word from **S** [15].

However, *Model 1* functions on a simpler variation of this framework, which assumes that all the possible lengths for the target sentence have a uniform probability ϵ, generating words for source sentence are equally likely and the translation probability, defined as $tr(t_j|s_i)$, of the target language generated word depends only depends on the generating word from the source language. The probability that a target sentence **T** is derived from the source sentence **S** is given by the following equation [15].

$$p(T|S) = \frac{\epsilon}{(l+1)^m} \prod_{j=1}^{m} \sum_{i=0}^{l} tr(t_j|s_i) \tag{2.5}$$

For the creation of a bilingual dictionary, *IBM Model 1* expects a *sentence-aligned parallel corpus*. Therefore, the probability of two texts being translated from one to another was defined as [17]:

$$p(d_1|d_2) = \prod_{x \in d_1} \sum_{y \in d_2} p(x, y) \tag{2.6}$$

In this formula, $p(x, y)$ represents the probability that the word x is a translation of the word y. Although it generated good results for translating sentences, **CL-ASA** has to function for entire texts, which present various lengths. Due to this, the model was adapted to use weights instead of probabilities, changing from the *Translation Model Probability*, to simply the *Translation Model* [17]:

$$\omega(d_1|d_2) = \sum_{x \in d_1} \sum_{y \in d_2} p(x, y) \tag{2.7}$$

The other important component of the *Cross-Language Explicit Similarity Analysis* approach is the *Length Model*, which is based on the expectation that a pair of translated documents d_1 and d_2 are unlikely to have the same length, but their sizes should be closely related by a length factor for every language pair. Due to this, the model has been defined for a pair of documents d1 and d2 as [17]:

$$\varrho(d_2) = exp\left(-0.5\left(\frac{|d_2|/|d_1| - \mu}{\sigma} \right)^2 \right) \tag{2.8}$$

The variables μ and σ represent the average and the standard deviation of the character lengths produced by document translations from the language of d_1 to the language of d_2, while $|d_1|$ and $|d_2|$ represent the lengths of the two texts [17].

As mentioned at the beginning of the subsection, **CL-ASA** was initially based on a *translation model probability*, written as $p(d_1|d_2)$ and *language model probability*, written as $p(d_2)$. Given a document d_1 written in the language l_1 and a document d_2 from a collection of documents $D^{'}$ written in language $L^{'}$ the approach measured the probability that d_2 is a translation of d_1, using Bayes' rule [17]:

$$p(d_2|d_1) = \frac{p(d_2)p(d_1|d_2)}{p(d_1)} \tag{2.9}$$

Since the purpose of the model is to retrieve the possible translation of d_1 for the target language $L^{'}$ and not to simply translate d_1 into $L^{'}$, the formula was adapted to use the *Translation Model* and *Length Model* instead. Therefore, the similarity of two given documents in the context of **CL-ASA** is now measured by the following model:

$$\varphi(d_1|d_2) = \varrho(d_2)\omega(d_1|d_2) \tag{2.10}$$

It is observed that this formula is not normalized, unlike other similarity measures, however, the partial order generated amid the documents matches the order of other similarity measures. From formula 2.7 it results that if valid translations from word x to word y appear in the language vocabularies, the weight increases. In addition to this, from 2.8 it can be observed that if the translation of document d2 from d1 does not have the expected length, the similarity measure reduces.

2.2.5 Translation + Monolingual Analysis

Unlike any of the previously described approaches, which only use principles of *Machine Translation* to compute the textual similarity between two documents, **T+MA** methods produce actual machine translations of one of the two given texts. This is the first step of the process and it is performed to reduce the complexity of the problem, by transforming it from multilingual into monolingual. This idea gained popularity quickly and therefore, different methods were proposed in the following years [5].

One such method is *Language Normalisation*, which implies the creation of a representation where all the documents are written in the same language. This is considered to be one of the most popular preprocessing strategies for the information retrieval step of textual similarity detection. The language proposed as the base for this representation is English since the majority of the content available on the World Wide Web is written is available in this language and automatic translation mechanisms for a language pair are most likely to exist when English is one of the two languages [5].

The first step of this method is to determine which is the most likely language the two given documents were written in, with the help of a language detector. In the case in which one of the texts is not written in English, it is translated into it. The second step is to perform the textual similarity detection, which can be done through various methods since the process is monolingual. One such method, which is considered to be a good option due to its simplicity is the *Character n-Gram Profiles* or in short **CnGP** [5].

In **CnGP**, the two given documents are compared through their *profiles*, which are bags of *tf-weighted character 3-grams*. Different fragments s of the document are chosen and to compute their profiles, *sliding windows* of length m and step n are used. The dissimilarity between every profile p_s and p_d is computed with the aid of one of the normalised documents using the following formula [5]:

$$nd_1(p_s, p_d) = \frac{\sum_{t \in p_s} \left(\frac{2(tf_{t,p_s} - tf_{t,p_d})}{tf_{t,p_s} + tf_{t,p_d}} \right)^2}{4|p_s|} \tag{2.11}$$

In this formula $tf_{t,p}$ represents the normalised term frequency of *t*, which is a *character n-gram* of s or d, while $|p_s|$ represents the size of the profile of the fragment $s \in d$. The values of the formula are bound between 0 and 1, with the maximum similarity being achieved at 0. A potential case of plagiarism arises when a higher than expected standard deviation appears between the profile of a fragment and the mean concerning all the possible fragment profiles p_s and p_d [5].

Another method is to use *Web-based cross-language models*, which builds upon the same principles of *Language Normalisation*. The first such model was developed in 2009, to detect plagiarism from Malay into English and made use of the *Google Translation* and search APIs [5]. The method features several steps: translation of the given document into English, removal of the stop words, stemming of the words, identification of similar texts in the collection of documents, comparison of similar pattern and displaying a summary of the result.

Since this is a **T+MA**-based model, the translation of the document into English is the first step. This is done to improve the effectiveness of the textual similarity detection process since the document collection that is used is the World Wide Web, which contains a vast number of texts in the English language. The Google Translate API was used for this task since it produces accurate translations and was freely distributed at that moment in time. After this step is completed, the process of removing *stop words* begins. This is done to reduce the complexity of the next steps since *stop words* do not add any value or meaning to the sentences of the documents. In addition to this, they make up almost 40 to 50 per cent of the number of words in a document collection and their removal would save computation time and space, while not hindering the effectiveness of the retrieval steps [24].

For example, the most common *stop words* include the, our, a, an, by, all, what, ever, do, is etc, and a sentence such as ”What is plagiarism?” will reduce to simply ”plagiarism?” if the process of removing *stop words* occurs.

After the removal of the *stop words* from the documents, the next step is to perform stemming on the remaining terms. This process is performed on all the words and consists of obtaining their roots by removing the prefixes and suffixes. This is done to improve the effectiveness of the information retrieval step by using the root terms for pattern matching. One of the most common algorithms used for stemming is *affix stripping*, which is based on a set of rules for removing the suffixes and prefixes of the words [24].

For example, one such rule for prefix removal could be: ”If the term begins with *auto*, remove the *auto*” and for suffix removal: ”If the word ends in *-al*, remove the *-al*”. As a result, after the process of *affix stripping* is completed, words such as ”autobiography” would reduce to ”biography” and words such as ”regional” would reduce to ”region”.

After the translation and preprocessing of the document is complemented, the monolingual analysis of the generated text follows. Since the original document has been translated into English, the algorithm uses the World Wide Web as the collection of documents that it works with, in particular the *Google AJAX Search API*. This search engine is used for looking up keywords and certain sentences from the suspect text and it returns the most similar documents from the World Wide Web. Moving forward, the actual analysis is performed between the given text and the similar doc-

uments returned by the search engine. This is done with the help of a *fingerprint matching technique*, which fragments the text into *character n-grams*, compares the fingerprints associated with the suspect documents and the documents in the corpus and then provides a textual similarity score [24].

2.3 Translation Detection Methods Comparison

In this section a comparison will be made between the models that are considered to be the most relevant for the development of the algorithm proposed by this individual project. **CL-CTS** was discarded because no API was found for *Eurovoc* or other *Conceptual Thesauruses*. In addition to this, **T+MA** approaches are also discarded since they are computationally expensive, requiring a translation step and a preprocessing step before the actual textual analysis is performed.

The three approaches that will be compared are **CL-C3G**, **CL-ESA** and **CL-ASA**. This choice was made due to the fact that APIs that are publicly available exist for comparable and parallel document collections that are required by **CL-ESA**, such as *Wikipedia* and **CL-ASA**, such as the *JRC-Acquis Multilingual Parallel Corpus*, which contains legal documents from the European Union which have been translated and aligned in 22 different languages [29]. **CL-C3G** is selected for the comparison due to its nature of not being computationally expensive while also performing well on various document collections. *Wikipedia* is used by **CL-ESA** due to the fact that it contains documents in more than 200 languages that are also linked with one another when describing the same topic [13]. In 2010, an evaluation of these three models was performed. For both document collections, texts that do not have aligned versions in all of the languages required for the comparison are discarded and as a result, 45.984 and 23.564 documents remain in the *Wikipedia* and *JRC-Acquis* collections respectively. The selected texts from both collections were split into a testing set, which contained 10.000 documents and training set for the retrieval model, which contained the remaining texts [17].

In the aforementioned evaluation, the three approaches were compared in a ranking task with the help of three different experiments, which are performed on two different test collections for each model. The languages which were paired with English are Spanish, French, German, Dutch and Polish. For these experiments, the query document d_1 is considered from a test collection D_1, alongside the collection D_2, which contains the documents aligned with the ones in D_1. In addition to this, d_2, a document aligned with d_1 is also considered. The three experiments are performed on 1000 randomly chosen texts d_1, with the help of the three approaches **CL-C3G**, **CL-ESA** and **CL-ASA**.

The first experiment is the *Cross-Language Ranking* in which all the documents in the collection D_2 are ranked with respect to their cross-language textual similarity to d_1. The retrieval rank of d_2 is also recorded and it should be on the first or on one of the top ranks in order to be decided if d_2 is a translation of d_1 or not.

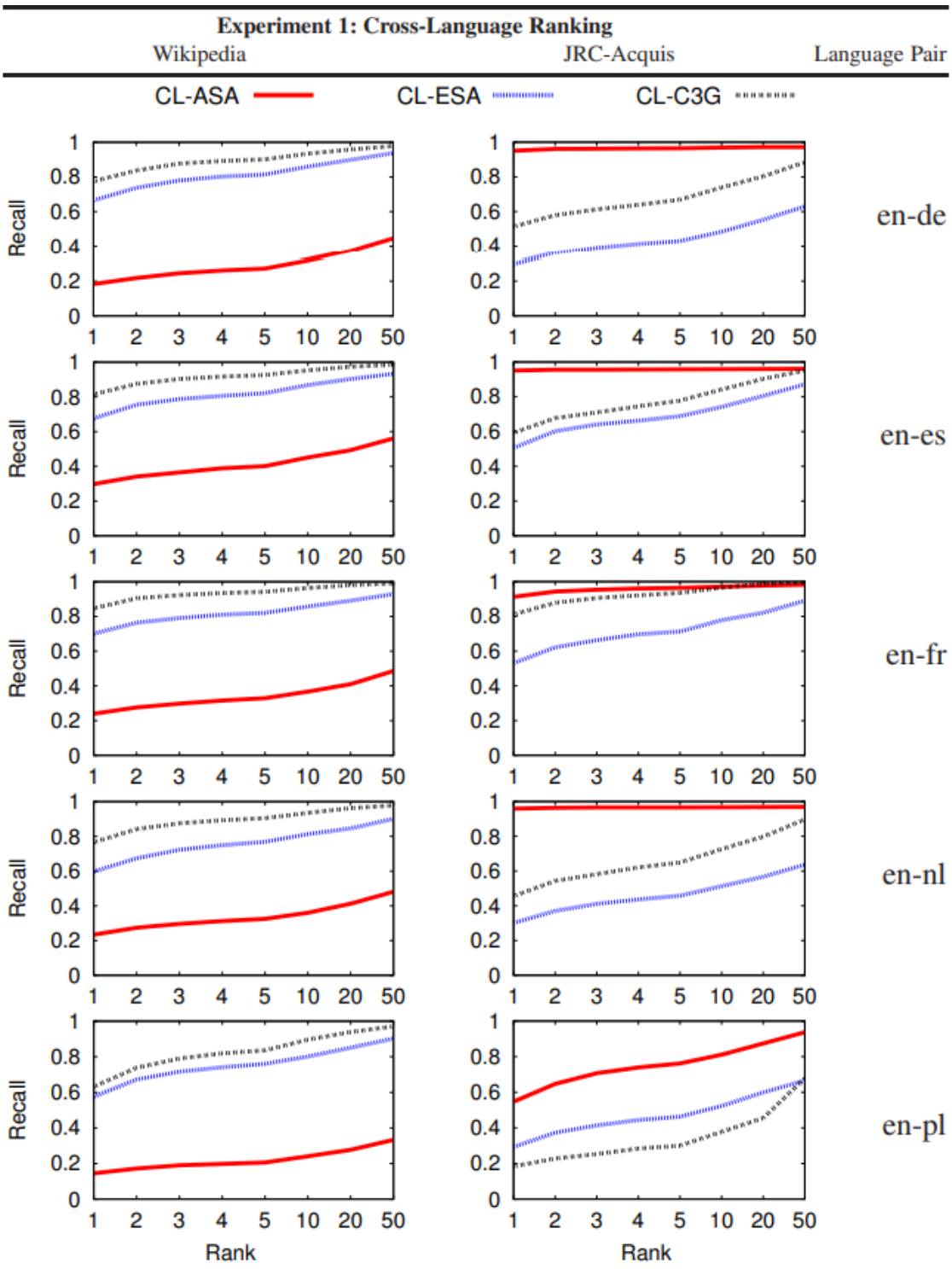

Figure 2.5: Martin Potthast et al. (2010), Results of Experiment 1 for the cross-language retrieval models [17]

Figure 2.5 extracted from [17] presents the results of the experiment through plots of recall-over-rank on the different language pairs. It can be observed that unlike **CL-ASA**, which varies in its performance, the other two approaches achieve stable results on both document collections. This can be explained by the fact that *JRC-Acquis* is a collection of parallel documents and **CL-ASA** has proved to work better on translations rather than on comparable corpus. One of the reasons why **CL-ESA** and **CL-C3G** achieve better results on the Wikipedia collection than on the *JRC-Acquis* one might be that the latter might be biased to some extent since it only contains legislative texts from the European Union and is therefore homogeneous. Due to this, **CL-ASA** appears to be more susceptible to bias than the other two approaches, but it may perform better when a more diverse parallel document collection is used for training [17].

The second experiment is the *Bilingual Rank Correlation* in which given a pair of aligned documents d_1 and d_2 from the collections D_1 and D_2, the texts from the latter are ranked twice. Firstly, they are ranked with respect to their cross-language similarity to d_1, using one of the three approaches and afterwards, with respect to their monolingual similarity to d_2, with the help of the *vector space model*. The **VSM** is an algebraic model used for representing text documents as vectors of identifiers which helps decide whether or not two texts have a similar meaning even when they don't share the same words [20]. After completing the two rankings, the top 100 of both are compared using *Spearman's rank correlation coefficient* ρ which estimates the positive or negative association of the rankings and returns a value in the interval $[-1, 1]$ [25].

Language Pair	**Experiment 2: Bilingual Rank Correlation**					
	Wikipedia			JRC-Acquis		
	CL-ASA	CL-ESA	CL-C3G	CL-ASA	CL-ESA	CL-C3G
en-de	0.14	0.58	0.37	0.47	0.31	0.28
en-es	0.18	0.17	0.10	0.66	0.51	0.42
en-fr	0.16	0.29	0.20	0.38	0.54	0.55
en-nl	0.14	0.17	0.11	0.58	0.33	0.31
en-pl	0.11	0.40	0.22	0.15	0.35	0.15

Figure 2.6: Martin Potthast et al. (2010), Results of Experiment 2 for the cross-language retrieval models [17]

Figure 2.6 extracted from [17] presents the results of the experiment by presenting the ρ values for the different language pairs and document collections. It can be observed that **CL-ASA** performs significantly better on the *JRC-Acquis* corpora than on the *Wikipedia* one. Different from the first experiment, **CL-ESA** has a similar performance to **CL-C3G** and **CL-ASA** on *JRC-Acquis*, however, it continues to outperform **CL-ASA** on the Wikipedia corpus. In addition to this, **CL-C3G** is also outperformed by **CL-ESA** on both document collections. All three models present poor performance on at least one language pairing, but, **CL-ESA** appears to be the most reliable

as a general-purpose retrieval approach, while more care needs to be taken when selecting the language pairings used for the other two models [17].

The third and final experiment is the *Cross-Language Similarity Distribution* which provides an indication of what can be expected from each of the three retrieval models. Therefore, the experiment does not directly compare the models but provides information about the range of cross-language textual similarity values measured when using one of the approaches and in particular, which values relate to a lower similarity and which values relate to higher similarity.

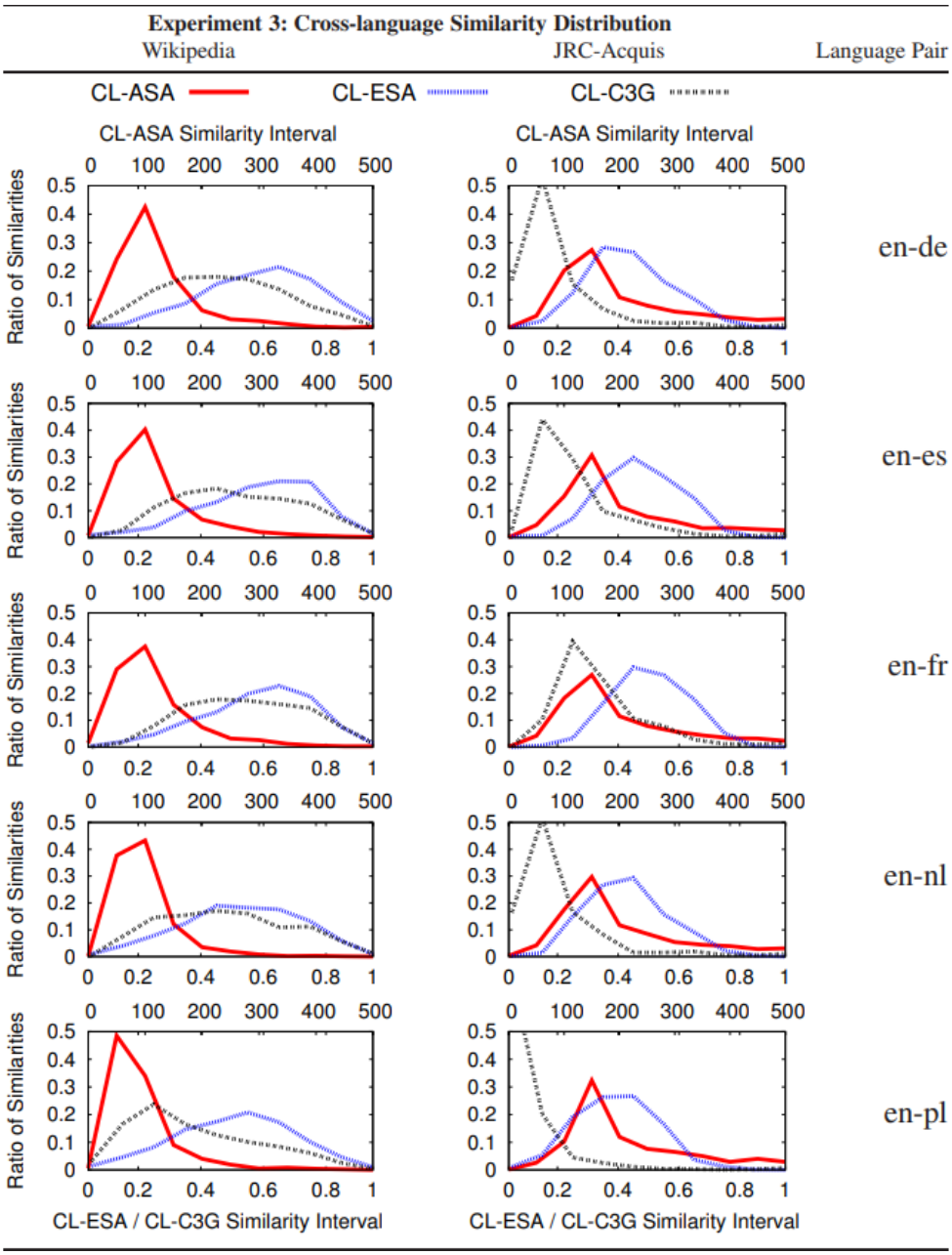

Figure 2.7: Martin Potthast et al. (2010), Results of Experiment 3 for the cross-language retrieval models [17]

Figure 2.7 extracted from [17] presents the results of the experiment through plots of similarity-over-similarities on the different language pairs. Since the textual similarities computed by **CL-ASA** are not normalized like for the other two approaches, the similarity distributions have been plotted on a different scale: the top x-axis of the plots presents the range of similarities measured with CL-ASA, while on the opposite side there are the similarities measured with the other models. As a result, the absolute values of the three approaches are not significant, but the order of the documents that they induce is. Because of this, the comparison of the similarity values produced by two different models is not enough to determine which of them performs better. For example, comparing **CL-ESA** with **CL-C3G** by their similarity distribution will result in the former being better, as it is more to the right. However, if looking at the first experiment, the latter outperforms **CL-ESA** [17].

The conclusion of these experiments is that each of the three models has its strengths and weaknesses and therefore one must be careful when choosing which approach is better in which context.

Chapter 3

Implementation

This chapter documents the implementation of the two main parts of the project, namely the cross-language plagiarism detection algorithm and the website which showcases it. For the former, the three stages: Heuristic Retrieval of candidate documents, Detailed Analysis and Post Processing are thoroughly detailed, while for the latter, the front-end alongside the back-end are presented.

3.1 The Website

The website, which is the actual part of the project that users will interact with, will be presented in this section. This was chosen over having a native application due to the fact that it does not require downloading. Therefore, the users would be more likely to try and use the website, due to the fact that they would only require a browser.

3.1.1 Implementation Details

Front-End and User Interface

The front-end has been created with the help of React and TypeScript, as it was required to develop and release a web application as part of the Individual Project. Therefore, it became obvious that a web development framework would work best for this type of project. In addition to this, the website had to be available for both mobile and desktop users and had to properly function for different types of screens and resolutions. Those are the main reasons why React was chosen, alongside the fact that there are numerous resources and online support related to it, as it is popular among the developer community.

For the user interface, the goal was to keep it simple, but attractive at the same time, such that users do not lose track of what they visited the website for. This has been achieved by constantly getting feedback and making continuous adjustments. One such example would be the button that was added in the *About* section, that

takes the user to the *Detection* page. After making this change, the users said that the application became more intuitive to use and easier to traverse.

Common Implementation Details

To ensure the best possible functionality and the robustness of the application, some ground rules were set from the beginning:

- If a component is used multiple times, it should be made into a reusable component. By doing so, code duplication is prevented and if changes need to be made, they are easier to do. One such example is the navigation bar, which is persistent throughout the entire application but is slightly different on the *Home* screen.
- As the React documentation [10] suggests, for each different screen, a particular component must be created in order to display it. Going one level down, each component would have its relevant parts displayed by other subcomponents.
- Hooks should be used to simplify the logic, by adding state to otherwise functional components [21].
- The navigation should be done with the help of React Router, in order to create a better experience for the mobile users, by enabling them to move through the application using the back button. In addition to this, particular paths can be typed in the URL and users can therefore navigate to the *Detection* screen and save time if they desire.
- To ensure consistency in design, Material UI is used, which offers a vast number of components that have the advantage of also being responsive and scaling well at different resolutions.

The Application Structure

Figure 3.1 presents the landing page of the project, also called the *Home* screen, which is built using a React component. Users can start navigating through the application by interacting with one of the buttons present in the button group. By pressing or tapping on one of those, the user is redirected to a different page, which contains a particular feature, such as *Detection*, *About* or *Credits*.

The *About* page is made out of a Card component that holds general information about the project. The user can also find out from this page what the application is about and how to use it. In addition to this, a copy of this project report can be downloaded and the user can easily find the details about the plagiarism detection algorithm. At last, a shortcut to the *Detection* page has been added to the bottom of the card, so that the users can go directly to the core of the application after reading about it.

Figure 3.1: The landing page of the web application

Another page that only contains information is the *Credits* one, where users can find information about the authors and the licenses for the used images. In addition to this, a list of used modules is displayed, alongside the corresponding versions.

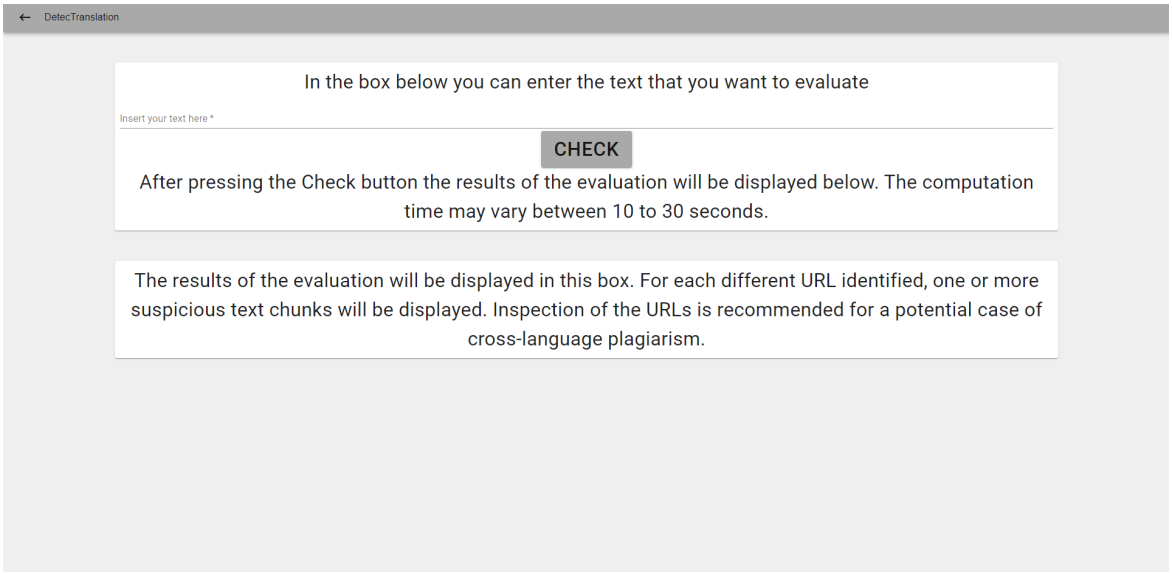

Figure 3.2: The Detection page

The last and most important page is the *Detection* one, presented in Figure 3.2, which contains the core functionality of the application. Inside the page, the user can observe a Card that contains some written information about what they should do, a text box to fill with the text to be analyzed and a button for evaluation which cannot be pressed if the text box is empty. In short, people can write or paste a text and then press a *Check* button. After doing this, the cross-language plagiarism detection algorithm is called having as the argument the text submitted by the user.

Since the process of CL-PD is time-consuming, the user is informed of the fact that it may take up to thirty seconds. A loading circle appears when the evaluation begins, which keeps spinning until the results from the back-end are received. After the whole process is completed and the loading circle disappears, the result of the analysis is displayed in the card below the text box. If no plagiarism was detected, only a short text would appear which would inform the user of this. Otherwise, one or more suspicious passages would be displayed alongside the URLs of the websites they have been found on.

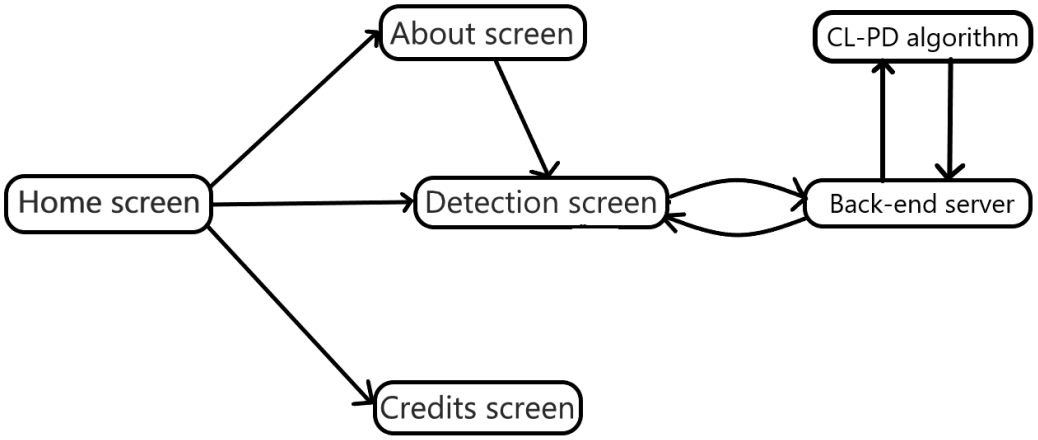

Figure 3.3: The Web Application Architecture

The Back-End

The back-end of the web application is a Python Flask server, which is deployed on Heroku. The front-end communicates with the server with the help of axios, by sending a POST HTTP request to the $/result/$ route of the server. The back-end then processes this request by taking the $text$ value sent as the payload of the request and running the CL-PD algorithm with it as an input. The results are then collected inside a Python list and transformed into a string at the end of the process to be returned to the front-end of the website, to be displayed to the user. The entire architecture of the web application is showcased in Figure 3.3

3.2 The CL-PD Algorithm

In this section, the three stages of the Cross-Language Plagiarism Detection Algorithm will be presented in detail. In short, the three steps include gathering a number of candidate documents that are compared with the given to be evaluated document and filtering out any false positive. After the algorithm is thoroughly explained, a

summary of the most interesting and challenging aspects and how they were overcome will also be presented.

3.2.1 Heuristic Retrieval Step

As previously mentioned, the first stage of the cross-language plagiarism detection algorithm is the *Heuristic Retrieval*. The goal of this step is to generate a set of candidate documents that will go into the *Detailed Analysis* stage. Let d be the to be evaluated document and D' be a collection of documents written in a different language L'. From D', a set of candidate documents D'_q needs to be retrieved, with each document in D'_q being likely to contain a section or sections similar to parts of d.

While this process does not seem complicated at first, it becomes more complex when taking into account the fact that people who engage in cross-language plagiarism exploit the vastness of the World Wide Web. This being the case, the collection D' from which candidate documents are selected for the next stage needs to be the whole indexed part of the World Wide Web for language L'. As a result, the set D'_q of retrieved documents has to have a much smaller size compared to the one of D', since comparing d, with a large number of candidate documents would be infeasible from a computational point of view. Therefore, the set D'_q should contain only the documents that are most likely to contain sections similar to the ones in d.[17]

In order to perform the *Heuristic Retrieval* of the candidate documents, there are three possible approaches that utilize methods such as cross-language information retrieval, monolingual information retrieval and hash-based search. In addition to this, all methods involve performing some sort of keyword extraction or fingerprinting, while two of them also make use of machine translation. [17]

The approach used by this project is the one that makes use of machine translation and information retrieval. This choice was made due to the fact that compared to the approach where no translation is performed at first, the keyword extraction is more accurate and therefore better candidate documents can be retrieved from the World Wide Web. The method that uses hash-based search was not considered since there was no intention of indexing certain portions of the World Wide Web in order to create a dedicated index for plagiarism detection purposes. [17]

Moving to the actual implementation of the aforementioned *Heuristic Retrieval* approach, as shown in figure 3.4, the first step is to perform *Machine Translation*. This is done with the help of the googletrans module of Python which makes use of the Google Translate AJAX API. An instance of the Translator object is created which is used to translate the initial document into nine different languages: Dutch, French, German, Italian, Polish, Portuguese, Romanian, Spanish and Swedish. The languages were chosen based on the number of speakers as a percentage of the European Union population [6]. The process is the same for all nine languages and it is highlighted in the code snippet below:

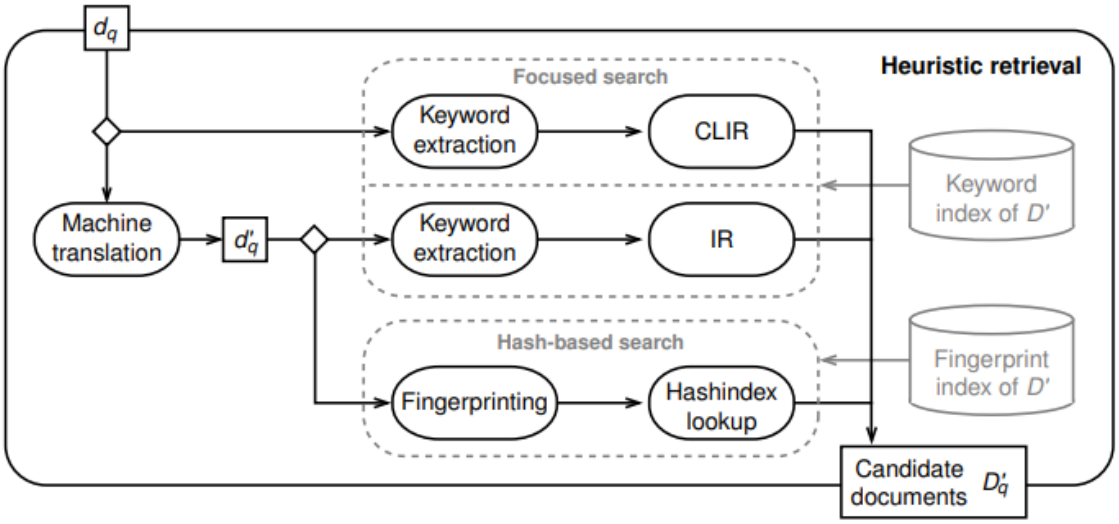

Figure 3.4: Martin Potthast et al. (2010), Retrieval process of the Heuristic Retrieval step for cross-language plagiarism detection [17]

```
# the function receives as input the document submitted for evaluation and returns its translation into French
def get_french_translation(initial_document):
    french_translation = translator.translate(initial_document, dest='fr', src='en')
    return french_translation.text
```

Figure 3.5: Translation of the initial document into French

As shown in Figure 3.5, the $translate$ function takes as input the document to be translated and two other parameters, $dest$ and src corresponding to the language to be translated to and the language of the input document. For each different language, the only difference arises from the $dest$ parameter, which has to be selected accordingly. The nine different functions are called inside another function which is used by the main script, with the purpose of gathering all the translations in a Python list. The whole translation step was organised in a way that permits adding new languages easily if needed or wanted, by just creating a new $get\_anylanguage\_translation$ function and returning it alongside the other languages inside the aforementioned Python list.

After successfully generating the translations of the initial document for all nine languages, the *keyword extraction* step can commence. Initially, for either of the languages, this step was implemented with the help of dictionaries, by counting the number of occurrences of each word. In addition to this, a list of stop words was created for each language, in order to eliminate them from the possible keyword candidates list. As a result, the top three words that occurred the most and were not part of the stop words list were selected as keywords.

However, due to its simplicity, it became obvious after some testing that this method was not producing accurate enough results. The main issue arose from the fact that

this method could only select keywords consisting of a single word and therefore meaningful constructions would get split up and lose their actual meaning. Such an example was illustrated in 2.2.2, where it was mentioned that a construction such as ”working conditions” is more specific than the individual terms of the construction.

As a result, a more sophisticated keyword extraction method was required, which was found inside the Python *Yake* (Yet Another Keyword Extractor) module. Its main advantages are that it makes use of text statistical features extracted from single documents, it does not need to be trained on a specific data set and it does not depend on dictionaries, external-corpus, size of text, language or domain. An experiment carried out on twenty different data sets revealed that *Yake* significantly outperforms ten different state-of-the-art approaches [4].

```
# the function receives as input the German translation of the document submitted for evaluation and returns the
# keywords of the translation
def get_german_keywords(german_translation):
    german_keyword_extractor = yake.KeywordExtractor(lan="de", top=3, dedupLim=0.20, n=2)
    keywords = german_keyword_extractor.extract_keywords(german_translation)

    german_keyword_list = []
    for keyword in keywords:
        german_keyword_list.append(keyword[0])

    return german_keyword_list
```

Figure 3.6: Keyword extraction process for the German Language

For the actual implementation, the process is organised similarly to the one used for translation, the only difference arising from the fact that a different keyword extractor is created for each different language. Therefore, the function highlighted in Figure 3.6, used for performing the keyword extraction task for a text in German, is similar for all the languages.

The $extract\_keywords$ function takes as input translated version of the original document and is applied to a $KeywordExtractor$ object. This object is initialised with four parameters: lan, the language used for the keyword extraction process, in the above case ”de”, which is short for German; top, the first x, in the case of Figure 3.6, three most relevant keyword phrases to be returned; $dedupLim$, the duplication limit allowed between the different keywords returned; n, the maximum number of words allowed in the keyword phrases.

After the extraction is completed, the keyword phrases are added to a list which is returned by the function. As in the translation step, the nine different functions are called inside another function which is used by the main script. The only difference is that for each language, the list of keyword phrases is joined together into a single keyword phrase, which will act as the search input for the next stage. Similarly to the previous step, the code has been organised in a way that allows the addition of new languages easily, by just creating a new $get\_anylanguage\_keywords$ function

and returning it alongside the other languages inside the aforementioned Python list.

After the translation and keyword extraction stages are completed, the only remaining step is the actual retrieval of the candidate documents. In order to do this, three processes must take place: getting a list of Uniform Resource Locators (URLs) associated with the previously extracted keyword phrases, returning the HyperText Markup Language (HTML) associated with each of the URLs and lastly extracting the visible text parts from the HTML pages.

The first part of this stage is done with the help of the googlesearch-python module which uses the Google Search Engine API in order to return URLs. The most important feature of this module is the $search$ function, which takes as input 3 parameters: $term$, which represents the phrase used for querying the Google Search Engine; $lang$, the language used by the Search Engine; $num\_results$, the top x URLs to be returned by the query.

```
# function used for returning the list of urls associated with a given search phrase
def get_urls_from_keyword_search_term(keyword_search_term, language):
    url_list = []

    # the search function is part of the googlesearch module; it returns the first num_results urls associated to a given
    # term in a given lang (language)
    for url in search(term=keyword_search_term, lang=language, num_results=3):
        url_list.append(url)

    return url_list
```

Figure 3.7: The URLs associated to keyword phrase retrieval process

The URLs returned by the $search$ function are collected inside a Python list which is returned by the function showcased in Figure 3.7. For each different language, the function is called with the corresponding language shortcode in order to get the most relevant URLs associated with each keyword phrase built in the second stage of the *Heuristic Retrieval* process.

The second part of the document retrieval stage involves fetching the HTML page contents associated with each of the previously obtained URLs. For this task, a function has been created which loops through each URL inside a list of URLs and tries to establish an HTTP connection with the help of the get function inside the $requests$ Python module. In other words, a GET request [9] is sent and the response is stored inside a variable. If a connection error arises during one such request, an exception is thrown and if not, the content of the response is stored inside a Python list which will be returned by the function. The main script will then be able to get a list of all the HTMLs associated with each language.

The part that completes the *Heuristic Retrieval* process is the HTML page content filtering. This is required due to the fact behind all the visible text displayed, there are numerous hidden tags (or elements), which are the building blocks of any web

page [30]. This part was done with the help of $BeautifulSoup$, a Python package for parsing HTML and XML documents. Its purpose is to create a parse tree for parsed pages that is used to extract data from HTML.

The filtered text is obtained by looping through each HTML content page inside the list of HTML pages and creating a different parse tree for each, using the $BeautifulSoup$ constructor with the $lxml$ features. The next step is to completely remove the parts that are inside any $script$ or $style$ tags since those contain only PHP code or CSS code respectively and no visible text. This is done by looping through all the aforementioned tags in the parse tree and using the $extract$ function of the pandas package.

```
# search for script or style tags and remove them completely
for removable_elements in parser(["script", "style"]):
    removable_elements.extract()
```

Figure 3.8: The removal of the script and style tags

However, removing the $script$ and $style$ tags, as presented in Figure 3.8, is not enough. The actual visible text of the web page exists between other tags, such as: $< b > actual\_text < /b >$. The removal of those is facilitated by the get_text function of the $BeautifulSoup$ package, which when applied to the previously created parse tree, traverses it and returns all the visible text strings. After some testing was performed, it was observed that the non-breaking character $xa0$ often appeared and was therefore removed in order to get a clearer text for the next steps. The resulting text is then split into lines and the leading and trailing white spaces are removed. The lines are further split into smaller chunks in order to break the multi-headlines into a single line each. The final step involves recombining the chunks in order to get a whole text, which represents the entire visible text part of the web page.

The HTML page content filtering is done inside a function which is then called for each set of HTMLs corresponding to each language in order to create a Python list of texts. The main script then gets this list and ends the *Heuristic Retrieval* process. Six different candidate documents have therefore been selected for each language and will go into the *Detailed Analysis* step.

3.2.2 Detailed Analysis Step

The second step of the cross-language plagiarism detection algorithm is the *Detailed Analysis* one, which involves computing the textual similarity measure between candidate documents extracted in the *Heuristic Retrieval* step and the previously computed translations of the submitted for analysis document. The goal of this stage is therefore to create a textual similarity matrix which will then be analyzed in the

Post Processing step. The approach taken for this stage is similar to a Translation + Monolingual Analysis model, as presented in Section 2.2.5

Since the candidate documents are generally considerably larger than the text submitted for analysis, the first task performed in this step is to split the candidate documents into chunks of equal size. Since the reference text for the textual similarity analysis is the submitted document, the length of the chunks was chosen as the number of characters of the submitted text. However, there is a downside to this. By splitting on characters, there is a possibility that the first and last word of the chunk might be lost, due to its parts ending in two different chunks.

The split could have been performed on the number of words, but it would have been time-consuming, since it would have required to iterate multiple times through multiple texts of possibly large size. Therefore, the main reason why the splitting on characters count approach has been chosen is that losing two words in every chunk and therefore losing a small bit of accuracy is more favourable than increasing the computation time by up to even 15-20 seconds.

```
def split_text_into_chunks(text, character_count):
    return [text[i:i + character_count] for i in range(0, len(text), character_count)]


def get_array_of_text_chunks(language_texts, character_count):
    chunks_array = []

    for language_text in language_texts:
        chunks_array.append(split_text_into_chunks(language_text, character_count))

    return chunks_array
```

Figure 3.9: The chunk splitting algorithm

The aforementioned process is presented in Figure 3.9. The $get\_array\_of\_text\_chunks$ function is called for each of the nine different languages and receives as arguments the candidate texts for one of the languages alongside the character count of the text submitted for analysis. For each text in the list of candidate texts the helper function $split\_text\_into\_chunks$, which iterates through chunks of the length of the initial document and appends them to a new list, is called. The result returned to the main script is therefore a list of lists of chunks, since each different candidate text has its own set of chunks.

After the chunks have been created, the next step is to iterate through each set of chunks for each language. A list that will be transformed into a vector representation is created, by appending all the pieces from a set of chunks to the translated submitted text. Therefore, the result will be a new list which contains as a first item the

```
def get_list_to_be_vectorized(translated_text, language_text_chunk):
    list_to_be_vectorized = [translated_text]

    for chunk in language_text_chunk:
        list_to_be_vectorized.append(chunk)

    return list_to_be_vectorized
```

Figure 3.10: The list to be vectorized creation algorithm

translated text and following it the text chunks of one of the candidate documents, as shown in Figure 3.10.

The creation of this list is necessary for the next step of the process, where the chunks will be analyzed one against another and the translated text will therefore be compared to each of the chunks of the candidate documents. In addition to this, the translated text is placed first in the list for convenience, in order to easily get the results from the first row or column of the results matrix.

```
def create_cosine_similarity_matrix(list_to_be_vectorized, language_stop_words):
    tf_idf_vectorizer = TfidfVectorizer(stop_words=language_stop_words)

    term_matrix = tf_idf_vectorizer.fit_transform(list_to_be_vectorized)

    cosine_similarity_matrix = cosine_similarity(term_matrix, term_matrix)

    return cosine_similarity_matrix
```

Figure 3.11: The cosine similarity matrix generation algorithm

Figure 3.11 displays the actual textual similarity analysis process, which may begin after all the aforementioned preparatory steps are completed. The function $create\_cosine\_similarity\_matrix$ is called for the previously computed list to be vectorized alongside a specific list of stop words associated with each language. The stop words for each of the nine languages have been extracted from the $stopwords\text{-}json$ library on GitHub [28]. The first part of this function consists of creating a tf-idf Vectorizer, which has the role of converting a collection of documents into a matrix of tf-idf features. The only parameter passed to the constructor is the $stop\_words$ one which has the role of ignoring the list of given words when creating the vector representation.

The tf-idf Vectorizer is part of the $sklearn$ python package and is equivalent to creating an instance of Count Vectorizer and applying a TfidfTransformer to it. The tf-idf measures the originality of a given word by comparing the number of occurrences of the word in the given document (term frequency) with the number of documents in the collection of documents the given word appears in (inverse document frequency). The Count Vectorizer only returns the number of appearances of all the words in a document, while the tf-idf Vectorizer also takes into account how frequently the word appears in the entire collection of documents. Therefore, the tf-idf Vectorizer has been chosen over the Count Vectorizer, since it returns a more accurate representation of how similar two texts are in the context of this project since the candidate documents are generally split into smaller chunks.

After the tf-idf Vectorizer is created, the next step involves fitting and transforming the data from the previously created list to be vectorized. This is done by applying the $fit\_transform$ function to the tf-idf Vectorized with the aforementioned list as an input parameter. The function yields the same result as applying fit and then $transform$, but is implemented more efficiently. Essentially, fitting the data implies using the given as a training set and learning its vocabulary and inverse document frequency. After this, the transformation of the documents into a document term matrix is performed by using the vocabulary and inverse document frequencies learned by fitting. Therefore, the result of the $fit\_transform$ function is a sparse tf-idf weighted document term matrix.

The last part of the *Detailed Analysis* step is computing the cosine similarity matrix, which holds the results of the textual similarity analysis between the translated text and the candidate documents chunks. This is achieved with the help of the $cosine\_similarity$ function of the $sklearn$ Python package. The function receives as input the previously calculated tf-idf weighted document term matrix twice, to compare the translated text with each individual chunk. The resulting matrix is then returned by the $create\_cosine\_similarity\_matrix$ function. The function is called for each different candidate document and therefore there a cosine similarity matrix will be created for each candidate. The first row will be extracted from each matrix and will go into the *Post Processing* step, the final stage of the cross-language plagiarism detection algorithm.

3.2.3 Post Processing Step

Finally, the last step of the cross-language plagiarism detection process is the *Post Processing* one. In this stage, the results from the *Detailed Analysis* are processed and outputted in a way that is relevant to the user. As previously mentioned, for each different languages' candidate documents a cosine similarity matrix is computed. Therefore, each of these matrices' first row is checked.

Figure 3.12 presents how the results are checked and displayed to the user for the Dutch language. The process is similar for the other eight languages. The first part of

```
i = 0
flag = 1
for cosine_sim in results_row:
    if cosine_sim > 0.20 and round(cosine_sim) != 1:
        if flag == 1:
            results.append(urls_array[Language.DUTCH.value][j])
            flag = 0
        results.append(dutch_list_to_be_vectorized[i])
    i = i + 1
j = j + 1
```

Figure 3.12: Outputting the results for the Dutch language

this process is to iterate through each of the cosine similarity values. The nth value represents the similarity between the nth chunk of the currently analyzed candidate document and the Dutch translation of the initial document. If the value is greater than a 0.2 threshold for the nth chunk, it means there are reasonable grounds for suspecting that the initial text or part of it has been cross-language plagiarized from the nth chunk. Therefore, if the flag has the value of one, meaning that the current chunk is the first suspect one for the current candidate document, its value becomes 0 and the URL associated with the website that the candidate document has been extracted from is appended to the list of results that will be displayed to the user, alongside the actual suspicious chunks. After the process is completed for each of the nine languages and all the suspicious URLs and chunks are provided to the user, the cross-language plagiarism detection algorithm is finalised.

The cosine similarity threshold of 0.2 is chosen arbitrarily, however, the choice will be explained in more detail in the Evaluation section of the report, alongside the possibility of adding different threshold categories, each with its own recommendations, depending on how high the textual similarity score is.

3.2.4 Challenges and Overcoming Them

This section documents the biggest challenges faced when implementing the cross-language plagiarism detection algorithm. Since Python was not a very familiar programming language at the start of the project, numerous challenges arose along the way. To ensure the success of the project, these challenges had to be overcome. This involved a process of trial and error, constantly changing and gradually improving parts of the code along the way. In general, there were three key implementation aspects that generated these challenges: having an algorithm with an accuracy as high as possible, reducing the computation time as much as possible and keeping

the code as clean as possible.

Constantly Improving the Accuracy

In the earlier stages of the project, implementation was done in a naive way, not taking advantage of the numerous existing Python packages. Therefore, processes such as keyword extraction or extracting the visible text parts from the HTML pages were performed in inefficient or even incorrect ways. As a result, if the keywords were not properly extracted, the chances of fetching good candidate documents was drastically reduced. On top of this, not retrieving the entire visible text parts from the candidate HTMLs further reduced the chance of finding any suspicious candidate documents.

Although the solution to this problem appeared to be simple, it required a lot of time to get everything right. Python packages are a great resource, however, one has to find the package that best fits the task at hand. Therefore, more time than expected had to be put into finding the best-suited package. For example, there were numerous keyword extractor packages available for installation. However, after reading through the documentation of the most popular ones, YAKE was chosen since it allowed the use of a large range of languages and it allowed tuning parameters such as the number of words in a keyword phrase or how different the keywords should be to one another. This resulted in the potential to gradually improve the accuracy of the algorithm by changing those parameters.

In conclusion, if one wants to improve the accuracy of one's algorithm, reading through the documentation of several packages and choosing the most efficient one for the task at hand is essential.

Improving the Computation Time

Since the algorithm is hosted on a website that is accessible to a general audience, the time required for the algorithm to compute an answer should be as low as possible. Users generally do not want to waste time waiting for results and therefore getting feedback as fast as possible is a must. However, the main challenge was not actually reducing the computation time but ensuring that during this process the accuracy is not affected or it is only slightly affected in comparison to the computation time gained.

The most obvious solution was the same as the one mentioned in the above section. However, although choosing the most efficient packages mitigated the problem to some extent, it was not enough to make a significant difference. As a result, going through each part of the code and attempting to reduce the computation time step by step was required. Two key changes helped reduce the computation time significantly.

After inspecting the code for the HTML extraction algorithm, it was observed that

sometimes a website could not be reached using the requests function and this would result in a timeout. However, instead of just ignoring this and moving to the next URL, the command would retry getting the HTML content from the URL numerous times. This flaw ended up adding more than ten seconds to the computation time for each unreachable URL. By encompassing the request logic into a try and except structure and catching the connection error that derived from the timeout, the algorithm will move to the next URL.

The change that was essential to get the computation time at an acceptable level was reducing the number of extracted URLs per language from ten to five. Subsequently, this implies that the number of extracted HTML pages and filtered visible text pages were also reduced from ten to five. As a result, since these tasks were the most demanding ones, the computation time was halved. This change became possible after testing and comparing the results obtained for the algorithm running with both five and ten URLs. The difference in accuracy was negligible due to the fact that suspicious candidate documents were almost never being identified in the candidate documents from six to ten for either language. This resulted from the improvement of the keyword extraction algorithm, which in turn led to the extraction of better candidate documents.

In conclusion, going through the code is essential when trying to improve the computation time of an algorithm. However, while going line by line is a good practice, it might not be enough. It is important to look at the bigger picture and observe how different parts of the code interact with one another when one wants to ensure that the code behaves in the way it is intended to.

Keeping the Codebase Clean

The last important challenge that arose along the way was trying not to get lost in the code. Since the codebase was growing larger and larger, it became easier and easier to lose track of what was going on. Therefore, it was essential to set out a number of rules in the beginning, in order to keep the code as clean as possible.

One such rule was the coding style. Having consistent indentation and spacing between the lines makes it easier to go through the code when needed. Alongside this, it was important to name variables in explicit ways. By doing this, it was easier to understand and follow the logic of the code. Moreover, commenting on the functions and the more complex parts was essential, since it helped in the latter stages when going over code that was written in the earlier stages. Lastly, refactoring was key in the process of eliminating code duplication. Since a number of tasks were identical for the nine different languages, having one function for all of them instead of one for each significantly reduced the size of the codebase, while also making it neater.

In conclusion, being well organised is not only important for maintaining a high-quality codebase, but it also helps in the process of developing and improving it.

Chapter 4

Evaluation and Results

This section will outline the key conditions that need to be met for the project to be considered successful. Since the project has two big parts, both will need to be evaluated. Since these two components are different, one being a website made in React and the other being a Python algorithm, different benchmarks will be used when evaluating. For the latter, an automatic evaluation mechanism will be created, alongside a data set containing chunks of text extracted from randomly selected Wikipedia pages.

4.1 The Website

For the website to be considered successful, it needs to be able to showcase the detection algorithm that was developed. The users have to be able to easily interact with it and not encounter any unexpected issues. In addition to this, the site needs to be able to handle simultaneous requests, as several users might make use of it at an arbitrary moment in time. The last matter that needs to be addressed is how all the components will scale inside the website, to facilitate use on both desktops and mobile device screens of different resolutions. Testing will be the key to ensuring the correct functioning of the website and it will be split into two essential parts: unit and user testing.

4.1.1 Unit Testing

Due to the use of **React**, unit testing is a perfect fit for ensuring the robustness of the website. Since all the elements, including the pages, take the form of *React Components*, it is convenient to use this form of testing. By checking the functionality of all the *Components*, it can be ensured that up to 100% of the codebase behaves in the way it should.

4.1.2 User Testing

Perhaps the most important part of the website evaluation is constituted by the interaction of real users with it. Individuals were requested to navigate through the web

app and propose suggestions of what can be changed or improved. Since users will be the ones interacting and using the website in their daily lives, it was designed in a user-centred way. Therefore, instead of only evaluating the final product, people have constantly given suggestions along the way. Thus, the addition of new features was guided by the received feedback.

4.2 The Detection Algorithm

This part of the project will use a completely different benchmark for evaluation since the users will not directly interact with it. The algorithm will measure its success through its accuracy. Hence, the higher the accuracy of the final model, the higher the success. The accuracy will be calculated by taking the number of correct predictions and dividing it over the total number of data points used for testing. However, this will not be the only criteria taken into account when deciding the effectiveness of the algorithm.

4.2.1 Precision

The *Precision* of the algorithm is calculated by taking the number of correctly classified positive examples and dividing it by the total number of predicted positive examples. In other words, *Precision* is the probability of an example being classified as positive given that the example is, in fact, positive. Therefore, the higher this metric, the higher the confidence that an example found by the algorithm as positive is indeed positive.

4.2.2 Computation Time

Another important metric that has to be taken into consideration is the time needed to compute the answer to the cross-language plagiarism detection problem. Since users want to get an answer as fast as possible, computational efficiency will need to be achieved in order to reduce the required time as much as possible. The process is long and tedious and therefore there are many steps along the way that can be optimized. However, these improvements must be performed with care as to not affect the accuracy of the algorithm.

One solution to this problem might be to allow the user to select the language or languages they believe the text might have been plagiarized from. By doing this, the computation time can be reduced by up to nine times. However, most users are likely to not have such information and therefore would want to just check for all the nine languages.

4.2.3 The Obtained Results

This section will outline the main parts of the evaluation process alongside the results that were obtained for the cross-language plagiarism detection algorithm. Since

the algorithm attempts to find candidate documents for nine different languages, the data set that was used for testing was created by randomly extracting 450 chunks of text from 450 randomly selected Wikipedia pages, 50 for each of the nine languages used by the algorithm. This was done with the help of a script that went from page to page via a randomly selected hyperlink from the current page and extracted one paragraph for each. Additionally, to better simulate a real case, the chunks of text were also filtered of any citation brackets such as [1] or [2] etc. The data points vary in size, from 20 words up to approximately 300 words and in the subject. The entire selection process was performed in this way in order to ensure that the evaluation does not lead to biased results.

The testing process attempts to replicate a possible real-life situation in which one takes a chunk of text from the World Wide Web, translates it into English and presents it as one's own piece of work. Therefore, each data point from the testing set is first translated into English and then run through the algorithm. For each different chunk of text, the algorithm can either return a correct answer, if the correct Wikipedia URL is identified, a wrong answer, if another URL or another language are identified or no answer, if no URLs are returned. By having this classification, metrics other than accuracy, such as recall and precision can also be calculated. In addition to this, the time for running the algorithm for each chunk of text. As a result, the average computation time can be calculated for the entire data set.

As previously mentioned, the testing process is similar for each of the nine languages. The chunks of text associated with each language are passed one by one to the algorithm after being translated into English. As a result of this and of the way the code was organised, hyperparameter tuning can be performed for each different language. This is important due to the fact that languages are part of different families and may be very similar or different to one another. The main parameters that were adjusted through testing were the cosine similarity threshold in the *Post Processing* stage and the deduplication limit in the keyword extraction stage.

The first parameter that was tuned was the cosine similarity threshold. Finding a good value for this metric was key due to the fact that by setting a value that is too big, the algorithm could miss potential candidate documents and by setting a threshold that is too small, the algorithm could return false candidate documents. The former was preferred, since incorrectly flagging a person's work as potentially plagiarized is worse than not detecting potentially plagiarized work. Five different potential values were selected for the cosine similarity threshold: 0.10, 0.15, 0.20, 0.25 and 0.30. While the last two values were too large and led to missing numerous correct candidates, the first one was too low and returned many incorrect candidate documents. Therefore, the choice to be made was between the 0.15 and 0.20 values. The latter was selected since even though the former helped identify a few more correct candidates, it also returned a few additional incorrect candidates. This applies to all of the nine languages used by the algorithm.

The second parameter on which tuning was performed was the keyword deduplication limit. This metric is passed as a parameter to the keyword extraction constructor and imposes the degree of difference between the keywords. Therefore, the smaller the value, the more different the keyword are. As in the case of the cosine similarity threshold, five different candidate values were selected for the deduplication limit: 0.10, 0.20, 0.30, 0.40 and 0.50. The first and last two values were not a good fit for either of the nine languages. After a more detailed inspection, it became obvious that with these three values the keywords generated were not optimal and therefore it became harder to obtain good candidate documents. However, the 0.20 and 0.30 values were a good fit for all of the nine languages.

After performing hyperparameter tuning on the two aforementioned metrics and obtaining the best possible values for each language pair, a final round of testing was performed and the following results were obtained from a sample of 450 text chunks:

Language Pair	**Precision**	**Accuracy**
English-Dutch	76.47%	52%
English-French	76.92%	40%
English-German	75%	54%
English-Italian	80%	48%
English-Polish	47.61%	20%
English-Portuguese	76.47%	52%
English-Romanian	78.78%	52%
English-Spanish	73.68%	56%
English-Swedish	74.07%	40%

Table 4.1: Results of the cross-language plagiarism detection algorithm

After aggregating the results presented in Table 4.1, the average precision of the algorithm is 74.19% and the average accuracy is 46%. While the results are consistent for eight language pairs, with precision values of above 70% and accuracy values of above 40%, the English-Polish pair performs considerably worse. One reason for this might be the fact that the other languages are part of the same language families, with Dutch, German and Swedish being Germanic languages and French, Italian, Portuguese, Romanian and Spanish being Latin languages, while Polish is the only Slavic language used. As a result, the languages are similar to one another and therefore the results are similar as well. Another reason for this might be the accuracy of the Google Translate API for the Polish language. After translating a chunk of text two times, the original meaning of it might be lost or even changed. Hence, keywords can not be properly extracted resulting in retrieving the wrong candidate documents.

In section 2.3 the performance of three different models that perform cross-language

similarity analysis is compared through three experiments. However, these experiments do not take into account the Detailed Analysis and Post Processing stages and simply feed numerous documents to the models for testing. Therefore, a comparison between these three experiments and the algorithm presented might not be the best fit for evaluating the success of the project.

Since at this moment, there are no published projects that evaluate the whole process of cross-language plagiarism detection, but only the Detailed Analysis step, the results obtained can be used as a future reference when trying to improve the process of detecting plagiarized automatically translated documents. While the accuracy is not very high, sitting at 46%, it is important to note the high precision. As previously mentioned, the goal of the algorithm was to obtain an accuracy value as high as possible while maximizing the precision, since having false positives is not desired. A precision average of 74% is satisfactory when taking into account the fact that a high degree of rewriting is involved when translating and back-translating a chunk of text. There are a few different ways in which the two aforementioned metrics can be improved, which will be detailed in the *Future Work* section.

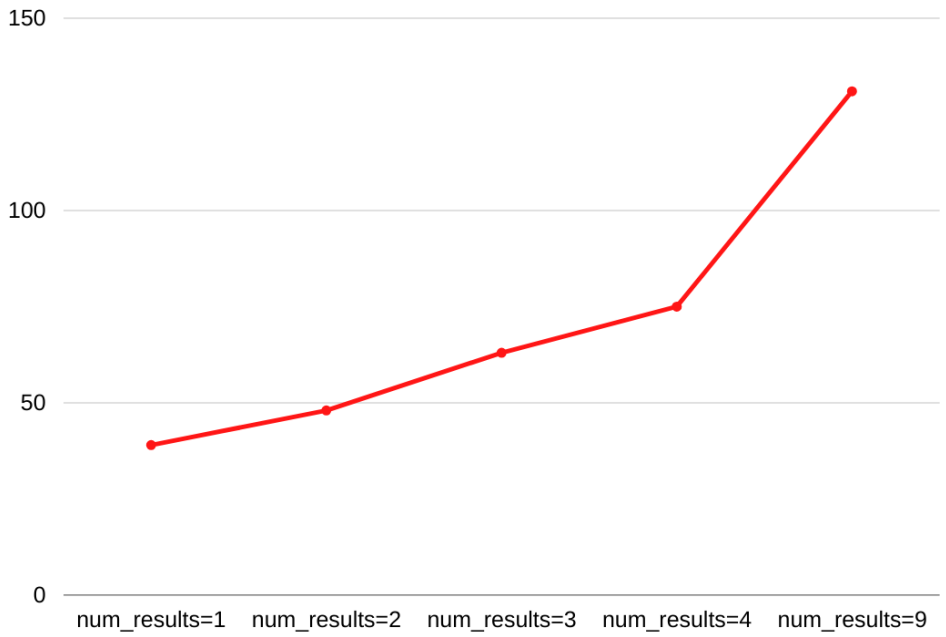

Figure 4.1: The computation time in seconds for the different values of num_results

Another important metric used to evaluate the success of the algorithm was the computation time. A fair balance had to be found between having a good accuracy and a small computation time. Therefore, the main parameter that could be tuned was the $num\_results$ in the search function that returns the top $num\_results + 1$ URLs that match a search phrase. By reducing or increasing this value even by one, the computation time significantly changes. This happens since the HTML page extraction and visible text filtering, which are costly operations from a computational point of

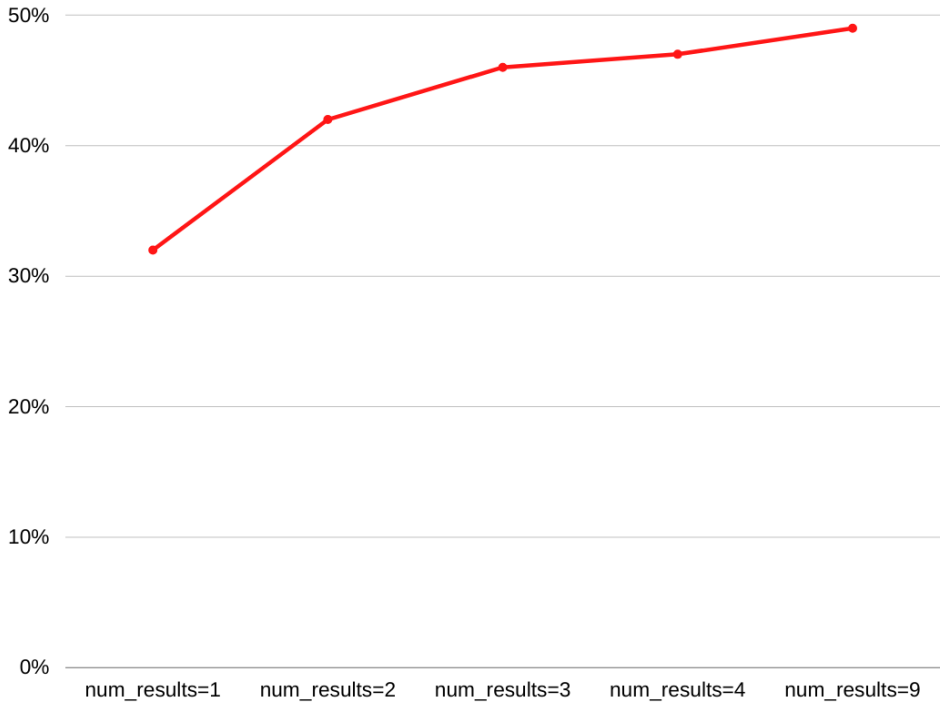

Figure 4.2: The accuracy for the different values of num_results

view, occur nine more or fewer times, once for each language. As a result, the 450 chunks of text data set had to be evaluated for different $num\_results$ values.

After inspecting the values in Figure 4.1 and Figure 4.2, it can be concluded that the optimal value for num_results, that gives the best computation time/accuracy ratio, is three. Choosing a value of one or two would decrease the computation time by a small margin, but would also significantly decrease the accuracy. Choosing a value of four or above would increase the computation time, but would have no effect on the accuracy of the algorithm, increasing it only by a negligible margin.

4.2.4 Conclusion

The main goal of this section was to decide based on the obtained results whether or not the algorithm designed during this project can efficiently solve the task of detecting plagiarized automatically translated documents, without requiring a great amount of computation time. While in terms of its accuracy the algorithm can only be seen as moderately successful, it can be considered a success in terms of precision and computation time. A high precision level significantly rises the confidence level when flagging a candidate document as a suspicious candidate document. Moreover, a computation time of around one minute can be considered an achievement when taking into account the fact that the algorithm has three stages, performs information retrieval, text filtering and computes textual similarity analysis for nine different languages.

These results were not obtained by chance. They were a consequence of careful and thorough testing in combination with performing hyperparameter tuning on key variables of the algorithm. Therefore, the results displayed in this section were obtained by constantly testing and adjusting the parameters, until the best possible combination was found.

Chapter 5

Key Insights and the Future

This chapter summarises the main achievements of the projects alongside the most valuable lessons learned. Moreover, a few possible ideas for future work are presented.

5.1 Summary

The main goal of the project was to develop an algorithm that helps detect plagiarized automatically translated documents. Alongside this, it was important to also develop an easy to use website that would be able to showcase the aforementioned algorithm. Both of these tasks have been completed, and therefore, the project can be considered successful. The website has been made in React with Typescript and it features a simple, but neat interface. The back-end of the website, which also holds the cross-language plagiarism detection algorithm has been with Flask and Python and was deployed on Heroku.

The algorithm has three stages: Heuristic Retrieval of candidate documents, Detailed Analysis and Post Processing. While the first and third stages were done without taking any inspiration from other sources, the Detailed Analysis was based on a Machine Translation model, namely the T+MA (Translation + Monolingual Analysis).

The results obtained were generally good. After conducting an experiment on 450 data points and performing several hyperparameters, the accuracy of the algorithm was found to be at around 46%, while the precision was found to be at around 74%. The computation time was also reduced significantly throughout the project, from 120 seconds to around 60 seconds.

5.2 Possible Extensions

There are several possible extensions that would overall improve the project, however, this section will detail the ones that would have the biggest impact.

Ideas for the Website

In its current state, the Detection screen is not engaging enough while the submitted text is being evaluated. One way of improving this is by constantly getting feedback from back-end about the stage reached by the algorithm and display it to the user. By doing this, the users would know that their text is being evaluated and the page is not stuck.

Another idea that would make using the website easier for the users is to also allow uploading documents, instead of only offering a text box for writing text in it.

Ideas for the Algorithm

The main goal of this subsection is providing ideas for improving the accuracy and precision of the algorithm. One way of doing this is by improving how a candidate text is split into chunks in the Detailed Analysis step. Instead on splitting on the number of characters of the original text, which may lead to losing two words in each chunk, splitting on the number of words of the original text would be better.

A more general way of improving the two aforementioned metrics would be by finding a better way of performing the process of keyword extraction and keyword phrase creation. Having better keywords leads to extracting better candidate documents, which in turn leads to finding more suspicious candidate documents and improving the accuracy and precision of the algorithm.

5.3 Conclusion

Some final thoughts for this project are that time management is important when trying to finish a task. It is very easy to lose track of what is essential and start losing time. It is not a question of whether or not it will happen, but of when it will happen. Therefore, it is key to keep this in mind and always aim to be productive. When getting stuck on one part of the project, it might be a good idea to start working on another part and ideas may eventually come. Also, it is essential to not give up when matters do not go well and continue to focus on the task at hand. In this way, the results will always come and the project will reach its end goal.

Bibliography

[1] Jeremy Ferrero et al. *Deep Investigation of Cross-Language Plagiarism Detection Methods*. 2017. URL: arXiv:1705.08828. (accessed: 19.01.2021).

[2] Ralf Steinberger et al. "JRC EuroVoc Indexer JEX - A freely available multi-label categorisation tool". In: (2013). URL: https://arxiv.org/ftp/arxiv/papers/1309/1309.5223.pdf. (accessed: 20.01.2021).

[3] Camelia Ignat Bruno Pouliquen Ralf Steinberger. "Automatic Annotation of Multilingual Text Collections with a Conceptual Thesaurus". In: (2003). URL: https://www.academia.edu/13517905/Automatic_annotation_of_multilingual_text_collections_with_a_conceptual_thesaurus. (accessed: 23.01.2021).

[4] Campos R.; Mangaravite V.; Pasquali A.; Jorge A.M.; Nunes C.; and Jatowt A. "A Text Feature Based Automatic Keyword Extraction Method for Single Documents." In: (2018). URL: https://doi.org/10.1016/j.ins.2019.09.013. (accessed: 22.04.2021).

[5] Luis Albert Barron Cedeno. "On the Mono- and Cross-Language Detection of Text Re-Use and Plagiarism". In: (2012). URL: https://riunet.upv.es/bitstream/handle/10251/16012/tesisUPV3833.pdf?sequence=1.. (accessed: 29.01.2021).

[6] S Eurobarometer - European Commission. "EUROPEANS AND THEIR LANGUAGES". In: (2012). URL: https://ec.europa.eu/commfrontoffice/publicopinion/archives/ebs/ebs_386_en.pdf. (accessed: 20.04.2021).

[7] Vera Danilova. "Cross-Language Plagiarism Detection Methods". In: (2013), pp. 51–57. URL: https://www.aclweb.org/anthology/R13-2008.pdf. (accessed: 20.01.2021).

[8] Armen Yuri Gasparyan et al. "Plagiarism in the context of education and evolving detection strategies". In: *Journal of Korean medical science* 32.8 (2017), p. 1220.

[9] *GET – What is the GET Method?* URL: https://rapidapi.com/blog/api-glossary/get/. (accessed: 23.04.2021).

[10] *Getting Started - React*. URL: https://reactjs.org/docs/getting-started.html. (accessed: 24.04.2021).

[11] McNamee P.; Mayfield J. "Character N-Gram Tokenization for European Language Text Retrieval. Information Retrieval 7". In: (2004), pp. 73–97. DOI: https://doi.org/10.1023/B:INRT.0000009441.78971.be. (accessed: 22.01.2021).

[12] KantanMT. *Machine Translation*. URL: https://kantanmt.com/documents/Machine_Translation.pdf. (accessed: 11.01.2021).

[13] Maik Anderka Martin Potthast Benno Stein. "A Wikipedia-Based Multilingual Retrieval Model". In: (2008). URL: https://webis.de/downloads/publications/papers/stein_2008b.pdf. (accessed: 24.01.2021).

[14] Izet Masic. "Plagiarism in Scientific Publishing". In: (2012). URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3558294/. (accessed: 03.02.2021).

[15] Robert C. Moore. "Improving IBM Word-Alignment Model 1". In: (2004). URL: https://www.aclweb.org/anthology/P04-1066.pdf. (accessed: 20.01.2021).

[16] Gupta P.; Barrón Cedeño LA.; Rosso P. "Cross-language high similarity search using a conceptual thesaurus. En Information Access Evaluation. Multilinguality, Multimodality, and Visual Analytics." In: (2012), 7488:67–75. DOI: doi:10.1007/978-3-642-33247-0_8. (accessed: 22.01.2021).

[17] Potthast M.; Barrón Cedeño LA.; Stein B.; Rosso P. "Cross-Language Plagiarism Detection. Language Resources and Evaluation". In: (2011), 45(1):45–62. DOI: doi:10.1007/s10579-009-9114-z. (accessed: 22.01.2021).

[18] Ignat C. Pouliquen B. Steinberger R. "Automatic Linking of Similar Texts Across Languages. In: Recent Advances in Natural Language Processing III". In: (2003). URL: https://books.google.ro/books?hl=ro&lr=&id=hZs6AAAAQBAJ&oi=fnd&pg=PA307&dq=Automatic+Linking+of+Similar+Texts+Across+Languages.+In:+Recent+Advances+in+Natural+Language+Processing+III&ots=BVHy1pblOy&sig=23Qd230mZCrkWH-X7r2jLOxqubE&redir_esc=y#v=onepage&q&f=false. (accessed: 23.01.2021).

[19] Selva Prabhakaran. *Cosine Similarity – Understanding the math and how it works (with python codes)*. URL: https://www.machinelearningplus.com/nlp/cosine-similarity/. (accessed: 23.01.2021).

[20] Kurtis Pykes. *Vector Space Models*. 2020. URL: https://towardsdatascience.com/vector-space-models-48b42a15d86d. (accessed: 02.02.2021).

[21] *React Hooks: everything you need to know!* URL: https://softwareontheroad.com/react-hooks/. (accessed: 24.04.2021).

[22] Marek Rei et al. "Artificial error generation with machine translation and syntactic patterns". In: *arXiv preprint arXiv:1707.05236* (2017).

[23] Melissa S.Anderson. "The problem of plagiarism". In: (2011). URL: https://doi.org/10.1016/j.urolonc.2010.09.013. (accessed: 03.02.2021).

[24] Chow Kok Kent; Naomie Salim. "Web Based Cross Language Plagiarism Detection". In: (2009). URL: https://arxiv.org/ftp/arxiv/papers/0912/0912.3959.pdf. (accessed: 30.01.2021).

[25] *Spearman correlation coefficient: Definition, Formula and Calculation with Example*. URL: https://www.questionpro.com/blog/spearmans-rank-coefficient-of-correlation/. (accessed: 03.02.2021).

[26] S Sreelekha et al. "A survey report on evolution of machine translation". In: *Int. J. Control Theory Appl* 9.33 (2016), pp. 233–240.

[27] Bruno Stecanella. *What is TF-IDF?* 2019. URL: https://monkeylearn.com/blog/what-is-tf-idf/. (accessed: 23.01.2021).

[28] *stopwords-json*. URL: https://github.com/6/stopwords-json?fbclid=IwAR2BlbYMr1afhiRyeMuQOks88yJI-9zPd8Si1t1Xd8yiNzuuueEufh-HZjA. (accessed: 18.05.2021).

[29] Ralf Steinberger; Bruno Pouliquen; Anna Widiger; Camelia Ignat; Tomaž Erjavec; Dan Tufiş; Dániel Varga. "The JRC-Acquis: A Multilingual Aligned Parallel Corpus with 20+ Languages". In: (2006). URL: https://arxiv.org/ftp/cs/papers/0609/0609058.pdf. (accessed: 02.02.2021).

[30] *What is HTML? The Basics of Hypertext Markup Language Explained*. URL: https://www.hostinger.com/tutorials/what-is-html. (accessed: 23.04.2021).