TakeLab

Laboratorij za analizu teksta i inženjerstvo znanja
Text Analysis and Knowledge Engineering Lab
Sveučilište u Zagrebu, Fakultet elektrotehnike i računarstva
Unska 3, 10000 Zagreb, Hrvatska

UNIVERSITY OF ZAGREB
FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING

BSc THESIS No. 5331

Cross-Lingual Plagiarism Detection from Wikipedia

Ivan Tokić

Zagreb, July 2017

UNIVERSITY OF ZAGREB
FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING
BACHELOR THESIS COMMITTEE

Zagreb, 3 March 2017

BACHELOR THESIS ASSIGNMENT No. 5331

Student:	**Ivan Tokić (0036487991)**
Study:	Computing
Module:	Computer Science

Title:	**Cross-Lingual Plagiarism Detection from Wikipedia**

Description:

Plagiarism detection is an authorship analysis task that aims at determining the originality of text using natural language processing techniques. In extrinsic plagiarism detection, the plagiarized text is detected by computing the semantic similarity between two texts. Ideally, such systems are capable of discovering not only text fragments that are identical to each other, but also fragments that are not identical but semantically similar, i.e., paraphrased. Furthermore, systems for cross-lingual extrinsic plagiarism detection can analyze texts written in different languages -- a case that often arises in practice.
The topic of the thesis is the cross-lingual plagiarism detection with English Wikipedia as the source and students' theses in Croatian as the target. Do a literature survey on methods for extrinsic plagiarism detection, including monolingual and cross-lingual methods. Devise and implement a cross-lingual semantic similarity model and a method for cross-lingual plagiarism detection based on machine learning. For cross-lingual semantic similarity, you may rely on publicly available machine translation services or online dictionaries. Choose a couple of topic from computer science and compile a suitable test collection for system evaluation. Additionally, you should evaluate the system on a collection of student theses. All references must be cited, and all source code, documentation, executables, and datasets must be provided with the thesis.

Issue date:	10 March 2017
Submission date:	9 June 2017

Mentor:

Associate Professor Jan Šnajder, PhD

Committee Chair:

Full Professor Siniša Srbljić, PhD

Committee Secretary:

Assistant Professor Tomislav Hrkać, PhD

I would like to thank my mentor, Jan Šnajder, for his patience and help provided, as well as all my friends, family, and my significant other for their support during the making of this thesis.

Contents

List of Figures

List of Tables

1 Introduction

The problem of plagiarism has plagued the academic community ever since its inception. The University of Oxford defines plagiarism as presenting the work or ideas of others as your own without proper acknowledgement.[1] They further expand into eight forms of plagiarism:

- uncredited verbatim quotation,
- unreferenced copying of information from the Internet,
- paraphrasing the work of others without acknowledgement,
- collusion with other students,
- inaccurate citation of other's work,
- failure to acknowledge assistance of others,
- use of material written by professionals or other people, and
- self-plagiarism.

Of these eight, the system described here will be taking into consideration the first three forms, with certain limitations.

Plagiarism is an issue that's at the same time both widespread and hard to uncover. Manual detection, which was for a long time the only possible defense against plagiarism, is time consuming and requires extensive domain knowledge to even perform correctly. Thus, with the increasing computing power available and the advancements of artificial intelligence, many have tried, with varying success, to automate this arduous task. The prime example of this is the PAN series of authorship tasks competitions.[2]

Obtaining real world datasets for training plagiarism detection models is almost an impossible task due to various legal and ethical issues accompanying plagiarism cases. As such, custom syntetic datasets are often built for these purposes. One example of this is PAN's source retrieval dataset, however datasets from related problems, such as Microsoft's paraphrase detection corpus, are also

[1] `https://www.ox.ac.uk/students/academic/guidance/skills/plagiarism?wssl=1#`
[2] `http://pan.webis.de/`

frequently used.[3][4]

Getting hold of such datasets for cross-lingual plagiarism is an even bigger challenge, especially for languages without a large number of speakers, such as Croatian. Wikipedia, the largest online encyclopedia, is a vast source of free knowledge in today's world. It comes at no surprise that this also makes it one of the most common targets for plagiarism. It is with this in mind that a custom dataset based on Wikipedia articles in the domain of machine learning was built.

The system presented here attempts to detect plagiarism cases in Croatian papers with English Wikipedia articles as sources. The model compares potential plagiarism targets and their respective sources by applying various syntactic and semantic features and then classifying them as plagiarisms or not. A local Wikipedia search engine is used to reduce the otherwise unwieldy search space, and commercial translation services are used to translate Croatian texts into English.

The thesis is split into six chapters, along with an appendix covering supplementary material. The next chapter examines the current state of both the mono-lingual and the cross-lingual plagiarism detection, and how the approach presented in this thesis supplements and/or differs from the outlined methods. In chapter 3, we go into a detailed analysis of the dataset creation, its further processing, and finally its representation and use in the training and the evaluation of the system. Chapter 4 describes the syntactic and semantic features used, the system's interaction with the dataset, as well as the methods used to train and evaluate the model. Finally, chapter 5 takes a deep look into the resultant data and analyzes the common errors encountered. The appendix A covers some of the more important implementation details specific to the exact platforms and libraries the system was built upon.

[3]http://pan.webis.de/clef15/pan15-web/plagiarism-detection.html

[4]https://www.microsoft.com/en-us/download/details.aspx?id=52398

2 Related Work

The area of plagiarism detection can be roughly split into two problems, intrinsic and extrinsic plagiarism detection. Intrinsic plagiarism detection deals with stylistic changes in a suspicious document undern an assumption that it's hard to plagiarise a source document while maintaining the document's stylistic patterns, the level of sophistication, and other such features. Extrinsic plagiarism, on the other hand, deals mainly with source retrieval, i.e., it attempts to discover plagiarism cases by comparing a suspicious text against potential sources, and making predictions based on a combination of syntactic and semantic features that appear in the compared texts. Extrinsic plagiarism detection is tightly related to the problems of paraphrase identification, string similarity, and string relatedness. In this chapter, we'll take an overview of the current state of the field of extrinsic plagiarism detection and the various methods employed.

University of Weimar's *PAN* competition offers a variety of plagiarism-related tasks, from source retrieval and intrinsic plagiarism detection, to authorship identification and profiling. As such, it has certainly played a big role in the advancement of the field. In the area of mono-lingual plagiarism detection, PAN's synthetic extrinsic plagiarism corpus is, due to its large size, the most widely used dataset.[1]

Madnani et al. (2012) explore the use of eight different machine translation evaluation metrics in the setting of paraphrase identification. We borrow a number of features from this work, namely the BLEU, NIST, and TER-Plus metrics (see sections 4.2.4, 4.2.5, and 4.3.1, respectively), as well as the general idea of applying machine translation metrics to NLP tasks other than machine translation evaluation itself. Interestingly, they make use of both the PAN's source retrieval dataset, and the Microsoft's paraphrase detection corpus, despite the PAN dataset not being specifically focused on paraphrase identification. One of the more interesting findings of this work is that TER-Plus by itself is capable of

[1] `http://pan.webis.de/clef15/pan15-web/plagiarism-detection.html`

outperforming many older paraphrase identification systems, most of which are substantialy more complex.

Ji and Eisenstein (2013) achieve state-of-the-art performance in the task of paraphrase identification by applying latent semantic analysis in combination with discriminative feature weighting. What makes this approach even more interesting is the use of *transductive* learning in the matrix decomposition phase. Transductive learning here signifies the use of both the training and the test dataset in that specific phase. Ji and Eisenstein utilize latent semantic analysis for classification by converting latent semantic representations of compared sentences, i.e., their LSA vectors, into a sample vector of the form $[\vec{v_1} + \vec{v_2}, |\vec{v_1} - \vec{v_2}|]$. We have taken a very similar approach, extending the sample vector with the cosine difference of the two latent vectors.

Hassan and Mihalcea (2009) explore the use of multi-language encyclopedic knowledge for the task of cross-lingual semantic relatedness. To accomplish this, they utilize the links between the Wikipedia articles' translations to build the concept vector representations of words through *explicit semantic analysis*. While it's a novel idea which shows great results, its usefulness greatly diminishes for our task due to the Croatian Wikipedia's lacking coverage. Instead, we'll be basing our dataset on the English Wikipedia alone, while taking the papers tested for plagiarism from Croatian into English through the use of commercial translation services.

Another interesting approach, which greatly differs from ours and the previously mentioned ones, is that of Franco-Salvador et al. (2013), who apply knowledge graph analysis to the task of cross-language plagiarism detection. For each suspicious target–source pair they build a knowledge graph using a multilingual semantic network, such as *BabelNet*, and compute the probability of it being plagiarised by applying graph similarity metrics to the obtained graph.

Our particular model utilizes a combination of syntactic and semantic features to compare potential plagiarised texts and their respective sources. The input texts are first translated from Croatian into English by one of commercial translation services, a comparison of which is also made. The potential plagiarism sources are obtained through a latent semantic indexing–based search engine of text fragments from Wikipedia articles. Finally, binary classification backed by a support vector machine with the RBF kernel is performed, determining whether pairs of potential plagiarisms and their respective sources are in fact plagiarisms.

3 Dataset

As with all natural language processing tasks, the dataset is one of the most crucial parts of the system. With that in mind, the following sections will describe in detail the process of obtaining and working with the data used throughout this work.

3.1 Paper Generation

Good, realistic, plagiarism detection datasets are incredibly hard to come by. When extended to cross-lingual plagiarism, this problem becomes even more pronounced.

Since our source corpus is a subset of Wikipedia, it would only make sense to also base our suspicious papers dataset on the same subset. A selection of around 40 text fragments were extracted from the said subset to be used for paper generation. These fragments were selected on the basis of how likely they are to be plagiarised by actual students. It was noted that students most frequently plagiarise the introduction and the highly technical parts, as both of these require substantive knowledge to write, or even discreetly plagiarise, in a meaningful way. With this in mind, most of the fragments selected came from these sections of the articles.

As the target language is Croatian, these fragments had to be translated to Croatian, so as to simulate cross-lingual plagiarism. Even though today's translation service are rather advanced, as will be discussed later in chapter 4, the quality of their output on complex sentences still cannot be compared to that of a human being. As such, human annotators were employed to translate these fragments into Croatian.

In order to make the task of detection more realistic, and therefore harder, non-plagiarised cases were introduced into the dataset. These non-plagiarised cases were hand-made by taking the existing Croatian translations and heavily paraphrasing them while at the same time removing existing, or adding new information into the fragment. An example of the introduced plagiarised and non-plagiarised fragments can be seen in the table 3.1.

One great drawback of this approach is the large amount of human labor for the annotation task and subsequently the small dataset. Based on the desired document size and the plagiarism percentage, a requested number of documents are randomly generated from the collected text fragments. The actual evaluation was done these exact documents.

3.2 Wikipedia Corpus

Before we begin devising methods of actual plagiarism detection, we need a way to access the contents of Wikipedia articles. Searching the online Wikipedia while running our system would be incredibly slow and would probably pose quite a few difficulties while training the system itself. Therefore a more employable method would be to download Wikipedia articles beforehand and compose a dataset of articles to perform our detection against.

Since working on the whole Wikipedia corpus would introduce, for the scope of this work, a lot of engineering problems, we'll only consider a subset of Wikipedia. This subset is defined through a Wikipedia category.[1] For the purposes of this work, *"Machine learning"* was chosen for the root category and all the plagiarised paper examples were generated from Wikipedia pages in that area. Note however, that the methods described in this work are category-agnostic and could easily be applied to any other category, irrelevant of its size, or even the whole Wikipedia by supplying *"Main topic classifications"* as the root category.

Once the root category is defined, we need to obtain all of the pages contained in that category, as well as in all of its subcategories, their subcategories, etc. To accomplish this, first the list of all the subcategories included is built by recursively navigating the WikiMedia's public API, namely its `Categorymembers` function.[2] Then, the list of all pages contained in those subcategories is built and any unwanted pages are filtered out. These unwanted pages consist of duplicates, since articles can be, and often are, contained under multiple categories, as well as Wikipedia *meta-pages* such as portals, user pages, and other similar pages which

[1] `https://en.wikipedia.org/wiki/Help:Category`

[2] `https://www.mediawiki.org/wiki/API:Categorymembers`

Table 3.1: Plagiarised and non-plagiarised fragments for paper generation.

Original: *In machine learning, a convolutional neural network (CNN, or ConvNet) is a type of feed-forward artificial neural network in which the connectivity pattern between its neurons is inspired by the organization of the animal visual cortex. Individual cortical neurons respond to stimuli in a restricted region of space known as the receptive field. The receptive fields of different neurons partially overlap such that they tile the visual field. The response of an individual neuron to stimuli within its receptive field can be approximated mathematically by a convolution operation. Convolutional networks were inspired by biological processes and are variations of multilayer perceptrons designed to use minimal amounts of preprocessing.*
Plagiarised: *Kod strojnog učenja, konvolucijska neuronska mreža (CNN, ili ConvNet) je vrsta feed-forward umjetne neuronske mreže u kojoj je spojni uzorak između neurona inspiriran organizacijom vidne kore životinja. Individualni kortikalni neuroni odgovaraju na podražaje u ograničenom dijelu prostora poznatom kao receptivno polje. Receptivna polja različitih neurona se preklapaju tako da popločavaju vizualno polje. Odgovor pojedinačnog neurona na podražaje unutar svog receptivnog polja se može matematički aproksimirati konvolucijskim postupkom. Konvolucijske mreže su bile inspirirane biološkim procesima te su varijacije višeslojnih perceptrona dizajniranih kako bi koristili minimalne količine predprocesiranja.*
Non-plagiarised: *Jedan danas vrlo popularni tip višeslojnih neuronskih mreža su konvolucijske neuronske mreže. Kao što se iz imena može zaključiti, ove neuronske mreže su bazirane na matematičkoj operaciji konvolucije, s kojom na jednoj vrlo pojednostavljenoj razini modeliraju rad vidnog sustava životinja, pa tako i nas samih. Procesom konvolucije se omugućava da svaki pojedinačni neuron djeluje u poprilično uskom dijelu prostora, dok preklapanjem podražajnih područja većeg skupa neurona dobivamo konkretno vidno polje. Kao i druge slične višeslojne, ili duboke, neuronske mreže, konvolucijske neuronske mreže ne iziskuju velike količine predprocesiranja ulaznih podataka.*

serve little purpose to us.[3][4] These *meta-pages* can be easily detected by special prefixes (or as Wikipedia refers to them, *namespaces*) in their page names, in fact categories themselves fall into this category.[5] These prefixes are of the form *"Namespace:"*, e.g., *"Category:"*, *"Portal"*, *"User:"*, *"Textures:"*, etc.; and can easily be filtered out. To get a sense of scale when working with the categories, *"Machine learning"* contains 1078 articles at the time of writing, *"Computer science"* includes around 10 thousand pages, while the whole Wikipedia consists of a whopping 5.4 million articles.

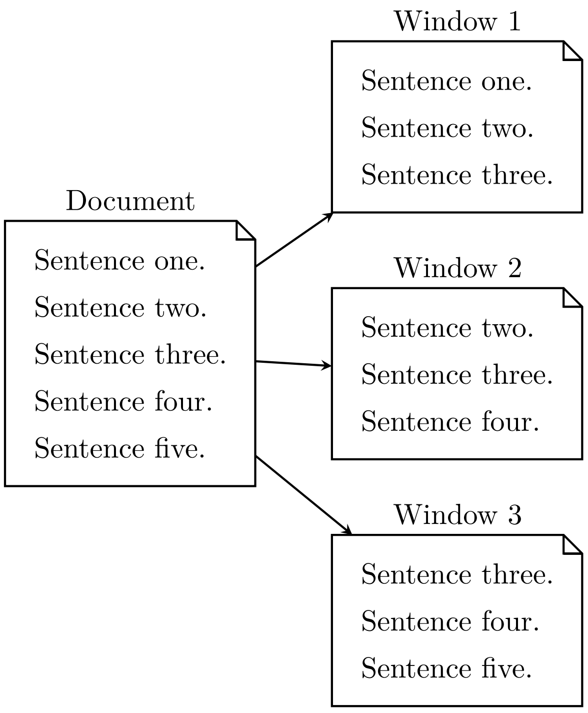
With the list of pages to download obtained, they have to be downloaded and parsed into plain text for the use with the rest of the system. While Wikipedia offers their articles in the *wikitext* markup format, they often make use of various templates which are then converted to HTML for displayal.[6] This makes it hard to get the full content out, even with the use of text conversion tools such as *Pandoc*.[7] Instead, the page is fetched in a stripped down version by adding an `action=render` parameter to the HTTP request.[8] This version contains only the main content without any sidebars and similar elements which might make the job of parsing even harder. The stripped down page is then parsed by an HTML parser into a structure that's easier to work with. From the obtained structure all unwanted elements such as images, and all the unwanted sections such as tables of contents, references, external links, and similar, are removed. Just as in section 3.1, equations and other mathematical notation are replaced with `<\m>` tokens, and finally the actual text is extracted.[9]

3.3 Paper Representation & Preprocessing

Each paper is first tokenized into sentences, and each of these sentences are further individually tokenized into words and a lower case transformation is applied. The resulting structure of documents is then searched for sentences of length shorter than a specified sentence length threshold, and such instances are concatenated to their neighbours until each sentence meets the length requirement. Finally,

[3]https://en.wikipedia.org/wiki/Wikipedia:Portal
[4]https://en.wikipedia.org/wiki/Wikipedia:User_pages
[5]https://en.wikipedia.org/wiki/Wikipedia:Namespace
[6]https://www.mediawiki.org/wiki/Wikitext
[7]http://pandoc.org/
[8]https://www.mediawiki.org/wiki/Manual:Parameters_to_index.php
[9]https://en.wikipedia.org/wiki/Wikipedia:Rendering_math

Figure 3.1: Illustration of the document windowing for a window size of 3.



moving windows of w sentences, with each window being shift by one with respect to the previous, are generated and joined into strings. An illustration of the windowing process can be seen in Figure 3.1.

The number of these windows for a given document d can be computed with the following equation:

$$|windows_w(d)| = |sentences(d)| - w + 1 \tag{3.1}$$

The $X_{[i,j]}$ notation used in these expressions, as well as throughout the rest of the thesis, represents *array slicing*, i.e., taking a subarray, or a *slice*, of an array X contained between indices i and j.[10] This operation can be expressed more formally as $X_{[i,j]} = [X_i, X_{i+1}, \ldots, X_j]$.

In order to build a set of training examples, we need to find all relevant combinations of suspicious document windows and Wikipedia article windows. For each source Wikipedia fragment and its plagiarised counterpart, we need to find all windows in the Wikipedia article and the generated paper, which contain at least a single sentence of their respective fragments. We then need to match all such Wikipedia fragment-window pairs, (F_w, Y), with all their relevant document

[10]This operation is equivalent to Python's slicing operation, i.e., `X[i:j+1]` (note the exclusive end index).

fragment-window pairs, (F_d, X), and compute the plagiarism score, *plag* (eq. 3.2) for each such match.

The value of this plagiarism score is a weighted sum of (a) the portion of the source Wikipedia fragment included in the Wikipedia window, $plag_y$ (eq. 3.4), and (b) the portion of that portion included in the paper window, $plag_x$ (eq. 3.3). Additional care is taken not to unnecessarily penalize window sizes smaller than the plagiarised fragment's sentence count by taking into account the length of the closest window of the Wikipedia fragment of size w, instead of the length of the whole fragment. This effectively means that a window containing the whole plagiarised fragment, due to the sentence count of such fragment being lower than the window size, would have the same *plag* score as a window containing only a portion of a much larger plagiarised fragment, as long as the window contained no other text other than that portion.

$$plag = w_x \cdot plag_x + w_y \cdot plag_y \tag{3.2}$$

$$plag_x = \begin{cases} \sum X_P \cap Y_P / \sum Y_P & \text{if } Y_L \neq \emptyset \vee Y_R \neq \emptyset, \\ 1 & \text{otherwise.} \end{cases} \tag{3.3}$$

$$plag_y = \begin{cases} \min\left\{1,\ \sum Y_P / \sum P_{[0, \min\{w, |P|\}]}\right\} & \text{if } Y_L \neq \emptyset \wedge Y_R = \emptyset, \\ \min\left\{1,\ \sum Y_P / \sum P_{[\max\{0, i_y + |Y_P| - w\}, i_y + |Y_P|]}\right\} & \text{if } Y_L = \emptyset \wedge Y_R \neq \emptyset, \\ 1 & \text{otherwise.} \end{cases} \tag{3.4}$$

Figure 3.2 shows how the *plag* score changes as we move our window along the relevant regions of the Wikipedia article and the suspicious paper, with three different weighing schemes for comparison. In the same plot, positions in the Wikipedia article and in the paper represent window indices, e.g., point $(0, 3)$ signifies the *plag* score between the first windows of the suspicious text and the fourth window of the Wikipedia article. The exact texts used for the computation of the said plot can be found in the table 3.2.

One important question remains – how do we interpret these scores? There are two main approaches that can be taken here, (a) we can either consider these values as signifiers of the extent to which a text is plagiarised with respect to some source and build a regression-based model which would make the same kind of predictions on the unseen texts, or (b) we can set up a binary classification model which applies a threshold function to each of these scores, taking them from the real domain to binary categories, PLAGIARISM and NON-PLAGIARISM. We have opted for the second approach, setting the threshold at an arbitrary value of 0.5, as this generally assigns NON-PLAGIARISM category to all examples which contain only a small fraction of the plagiarised fragment, e.g., a window size of five which only includes a single sentence of a five sentence plagiarised fragment.

The dataset consists of 38 plagiarised Wikipedia fragments, with a median sentence count of 3, and 27 non-plagiarised Wikipedia fragments. After windowing and matching with Wikipedia fragments, we end up with a couple of thousand of paper window–Wikipedia window examples, ranging from 864 for a windows size of one, to 3,887 examples for a window size of six.

3.4 Wikipedia Search Engine

Comparing a paper against the whole Wikipedia corpus would be incredibly inefficient, thus a search engine would go a long way in reducing the search space. A simple, configurable, latent semantic analysis–based search engine was built for this purpose and trained on the whole Wikipedia corpus. The tf–idf measure and the latent semantic analysis will be properly introduced later in sections 4.3.3 and 4.3.4, respectively.

The engine's text corpus consists of windowed text fragments instead of the whole documents. To accomplish this, each article undergoes the same transformations as the papers in the section 3.3, resulting in a sequence of windows of size w with a stride of one. The actual search will be performed on these generated windows.

To perform a meaningful search with as little noise as possible, we first need to extract the fragment's keywords. To do so, the tf–idf vector of the given fragment is obtained through the engine's corpus-fitted tf–idf transformer. Then, the tf–idf vector's indices are sorted by their tf–idf value in descending order. Finally, the desired amount of indices are taken from the top and mapped to their respective tokens, the exact number of which can be configured in the query itself. These tokens are the keywords to be used for the actual search.

The search itself, once we have the keywords, boils down to joining the keywords into a single string and computing the cosine similarity between the corpus-fitted LSA transform of the said string and each Wikipedia window's LSA vector. Similarly to the keyword extraction, the cosines' indices are then sorted and the

Figure 3.2: Example of *plag* scores for the texts in the table 3.2.

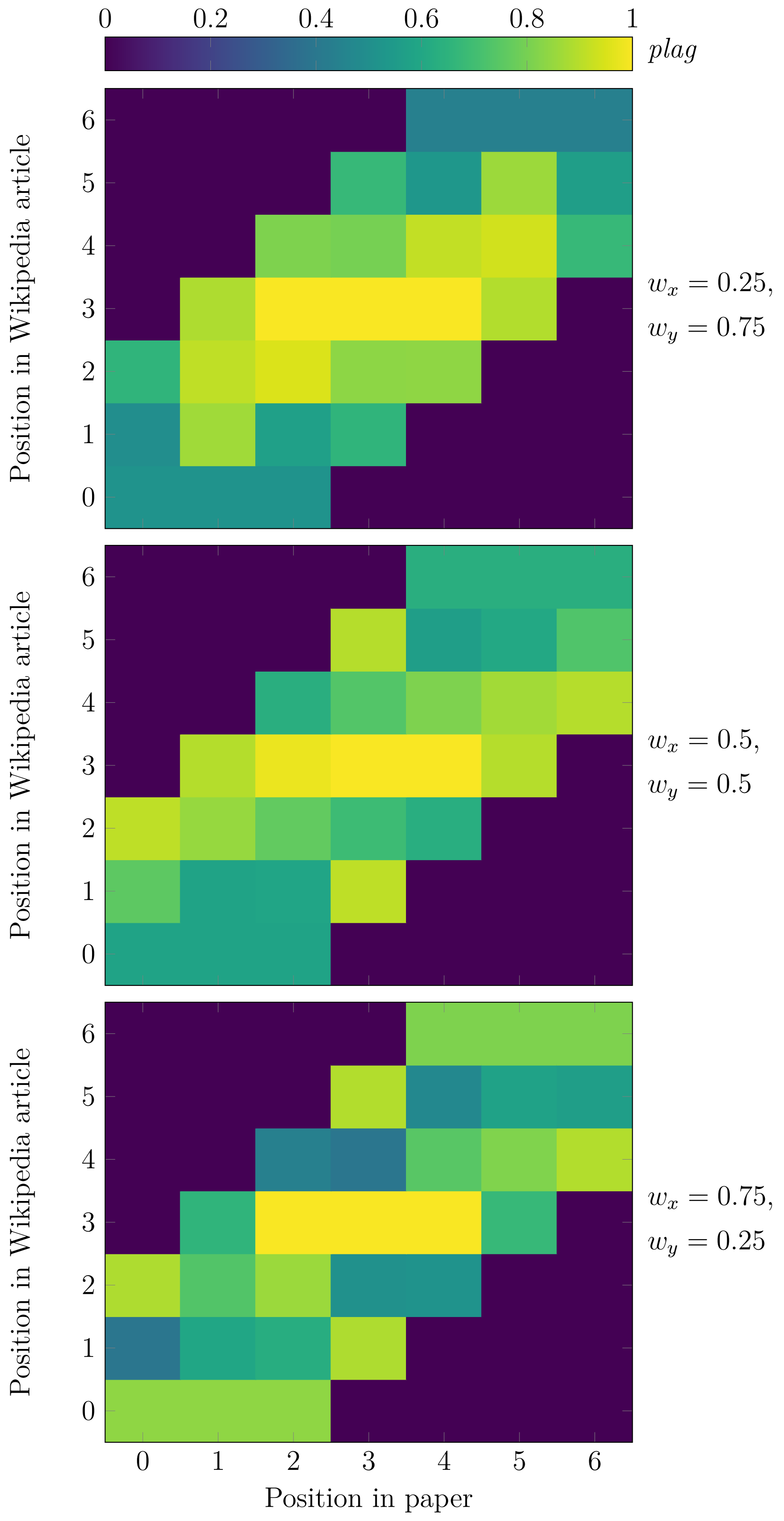


Table 3.2: Example of the sentences included in the windows around a plagiarised fragment for the window size of 3. The highlighted sentences in the first and second text represent the plagiarised fragment and its source, respectively.

English (NMT):

The main goal of the learner is to generalize one's own experience. Generalization in this context is the ability of a machine that learns to do exactly on new, unseen examples/tasks after passing the learning data set. Load training in the neural network can be modeled as a nonlinear global optimization problem. The target function can be formatted by estimating the readiness or error of a given load vector as follows: First, the loads in the network are set according to the vector load. Then the network is evaluated against the exercise sequence. Usually, the squared sum difference between prediction and target values specified in the exercise sequence is used to display the error of the current load vector. Optional global optimization techniques can then be used to minimize this targeted function. The problem of recognizing named entities is basically reduced to the problem of detection of the name, which in itself is reduced to the problem of segmentation, and to the classification itself of the detected names. Undetectable machine learning is the task of machine learning to perform a function that will describe hidden structures from "unmarked" data (classification or categorization is not included in the observations).

Wikipedia:

The CRBP algorithm can minimize the global error; this fact results in an improved stability of the algorithm, providing a unifying view on gradient calculation techniques for recurrent networks with local feedback. An interesting approach to the computation of gradient information in RNNs with arbitrary architectures was proposed by Wan and Beaufays, is based on signal-flow graphs diagrammatic derivation to obtain the BPTT batch algorithm while, based on Lee theorem for networks sensitivity calculations, its fast online version was proposed by Campolucci, Uncini and Piazza. Global optimization methods Training the weights in a neural network can be modeled as a non-linear global optimization problem. A target function can be formed to evaluate the fitness or error of a particular weight vector as follows: First, the weights in the network are set according to the weight vector. Next, the network is evaluated against the training sequence. Typically, the sum-squared-difference between the predictions and the target values specified in the training sequence is used to represent the error of the current weight vector. Arbitrary global optimization techniques may then be used to minimize this target function. The most common global optimization method for training RNNs is genetic algorithms, especially in unstructured networks. Initially, the genetic algorithm is encoded with the neural network weights in a predefined manner where one gene in the chromosome represents one weight link, henceforth; the whole network is represented as a single chromosome.

requested number of them are taken from the top and mapped to their respective Wikipedia windows.

4 Model

Here, we'll be taking a look at the semantic and syntactic features used, as well as address the task of classification. Also, we'll cover the available translation services and see how they compare against each other subjectively. The features used for classification were very much inspired by the works of Madnani et al. (2012), and that of Ji and Eisenstein (2013).

4.1 Translation Models

Before processing a suspicious document in Croatian, it has to be translated to English in order to seamlessly work with the previously built English Wikipedia corpus. To accomplish this, two state-of-the-art translation services, Google Cloud Platform's *Translate* service and the Microsoft Azure's *Translator*, were used.[1][2] Since Google offers two different models through its platform, we'll be taking a look at both of them, bringing the total number of translation models up to three.

These three models consist of: Google's *Phrase-Based Machine Translation* (PBMT) and *Neural Machine Translation* (GNMT) translation models, and the Microsoft *Translator*'s *"general"* translation model (henceforth referred to as MST). Google's, now outdated, PBMT model is, just like the Microsoft's *Translator*, a statistical, phrase-based, machine translation model. On the other hand, GNMT is a modern translation model based on recurrent neural networks. Unlike the phrase-based models, it attempts to translate the input text as a whole, without breaking it up into phrases or words, which often leads to the translations feeling much less artificial when compared with the statistical models (Wu et al., 2016). Generally, the neural network–based models have proven to be much more natural and closer to actual human translations, than the statistical models. As

[1]`https://cloud.google.com/translate/`

[2]`https://www.microsoft.com/en-us/translator/home.aspx`

Table 4.1: Complex sentence translation comparison – example one.

Source	*Problem učenja se svodi na generalizaciju prijašnjih iskustava, odnosno, modeliranje algoritma i podataka kako bi omogućili stroju da na osnovu ulaznog skupa podataka za treniranje donosi što točnije zaključke o još neviđenim podacima.*
PBMT	*The problem of learning is reduced to the generalization of previous experience, or modeling algorithms and data to enable the machine on the basis of the input data set for training yields the most accurate conclusions about the still unseen data.*
GNMT	*The learning problem is reduced to the generalization of previous experiences, ie, modeling of algorithms and data to enable the machine to draw on as accurate as possible conclusions from unexamined data based on the input data set for training.*
MST	*The problem boils down to learning generalization of previous experiences, that is, the modeling of the algorithm and data to enable the machine to on the basis of input data set for training brings as accurately as possible the conclusions of another unprecedented data.*

will be evident from the given examples and the results themselves, the neural network–based model seems to be much closer to what an actual human translation would look like when compared to statistical models. It should be noted that Microsoft too offers a solution based on neural networks, however at the time of the writing it did not support Croatian language.

Differences between these models and the methods they employ result in stark differences in the translation quality. Table 4.1 showcases these differences on an example in which all three services seem to output something of value. On the other hand, table 4.2 shows an example which proves problematic for all tested models, even resulting in an incomplete translation by Microsoft *Translator*.

4.2 Lexical Overlap Features

These features make use of basic lexical relationships between text fragments. This includes simple n-gram features, such as intersection measures, as well as more advanced metrics such as BLEU and NIST which make use of exhaustiveness and information content. This includes counts of syntactic structures, simple n-

Table 4.2: Complex sentence translation comparison – example two.

Source	*Konvolucijski sloj je glavna gradivna jedinica CNN-a. Parametri sloja se sastoje od para filtera (ili jezgri) koji mogu učiti te imaju maleno receptivno polje, ali se proširuju kroz cijelu dubinu volumena inputa.*
PBMT	*Convolutions layer is the main structural unit of CNN. Condition parameters is comprised of a pair of filters (or cores) that can learn and have small receptive field, or extend through the entire depth of the input volume.*
GNMT	*The conveying layer is CNN's main building block. The layer parameters consist of a pair of filters (or cores) that can be taught and have a small receptive field, but extend across the full depth of input volume.*
MST	*Konvolucijski layer is the main building, a unit of CNN. Parameters of the layers consist of pairs of filters (or cores) that can learn, and have a small receptive field, but extend through the entire depth of the volume of inputs.*

gram features, as well as more advanced metrics such as BLEU and NIST which make use of exhaustiveness and information content.

4.2.1 Counting Features

Word, sentence, and/or character counting features can, despite their simplicity, offer good support to other features. For this reason, they can be frequently found in many NLP task's models. In this work, only one such feature was implemented, a word count ratio of two fragments.

Another potentially useful counting feature would be a metric describing the difference in the complexity of two compared texts. Coleman and Liau (1975) propose just such a metric, the aptly named Coleman-Liau index. The index is a rather simple linear function of the ratios of the character and sentence counts to the word count. It attempts to score the text in question on a U. S. grade level scale, with 1 corresponding to the first grade of elementary school, 12 being the last grade of high school, and anything greater belonging to the university or college education.[3] For our own scoring, the two compared documents' Coleman-Liau indices were computed and their absolute difference was taken.

[3]https://www.ais.edu.hk/age-grade-guide

$$CLI(X) = 5.88 \cdot \frac{|characters(X)|}{|words(X)|} - 29.6 \cdot \frac{|sentences(X)|}{|words(X)|} - 15.8 \tag{4.1}$$

It should be noted that this is far from being the only such metric. A wide range of them were developed over the years, ranging from straightforward measures such as Coleman-Liau, to more complex ones taking syllable counts and other language constructs into account. This exact metric was chosen for being simple while still retaining useful information.

Another similar metric, the automated readability index (Senter, 1967) was also tried, but showed very little correlation with the results in the preliminary stages, and was subsequently left out entirely.

4.2.2 Common N-Gram Features

The following two measures, precision and recall, are usually used in the result analysis of prediction tasks. Here, however, we'll be using them as operations over sets, quantifying the relationship between some two sets and their intersection. Both measures' codomains cover the real interval of $[0, 1]$, where a higher score indicates better results, or in our case, similarity. These sets will represent two text fragments, or more precisely, the two text windows, being compared. Let X denote the suspicious text and Y the potential source text, for the purposes of this and all subsequent examples.

In terms of classification results, precision (eq. 4.2) is the measure of how many of our results are correctly classified, or in other words, how confident we are in our predictions. On the other hand, recall (eq. 4.3) is the measure of how many examples that should have been classifed, have actually been classified. In our specific context, precision is simply the quotient of words from the suspicious document included in the source document, while recall is the quotient of words from the source document included in the suspicious document. These metrics are often at odds with one another as increasing one usually results in the fall of the other.

$$\pi = \frac{TP}{TP + FP} = \frac{|X \cap Y|}{|X|} \tag{4.2}$$

$$\rho = \frac{TP}{TP + FN} = \frac{|X \cap Y|}{|Y|} \tag{4.3}$$

Table 4.3: Example of basic n-gram features with $n = 1$.

Sentence	π	ρ	F_1	F_{mean}
"I cannot accept such a proposal." *"I don't accept such proposals."*	0.67	0.8	0.73	0.78

Due to their dual nature, precision and recall are often used together to score the system. However, single value metrics are usually preferred over multiple value ones. Thus, metrics that combine and balance these two measures into a single value have been developed. One of these is the harmonic mean of precision and recall, the F_1 score, shown below:

$$F_1 = 2 \cdot \frac{\pi\,\rho}{\pi + \rho} \tag{4.4}$$

Lavie et al. (2004) showed that in the task of machine translation evaluation, recall corresponds much more to the human judgement of such translations than precision does. To exploit this, they devised an adjusted harmonic mean metric, F_{mean} (eq. 4.5), which weighs recall much more heavily than precision. This is in contrast to the F_1 score which accounts for both precision and recall equally. This approach showed substantially better results than precision-based metrics, such as BLEU and NIST. The aforementioned F_{mean} metric is just a special case of the more general F_β measure (eq. 4.6) with $\beta = 3$. The value of β was empirically determined by Lavie and his team, but as they say, the exact value plays little part and what's really important is the the additional weight placed onto recall.

$$F_{mean} = \frac{10\,\pi\,\rho}{9\,\pi + \rho} = F_{\beta=3} \tag{4.5}$$

$$F_\beta = (1 + \beta^2) \cdot \frac{\pi\,\rho}{\pi + \beta^2\,\rho} \tag{4.6}$$

4.2.3 DICE Family

The *DICE* coefficient (Dice, 1945), also known as the Sørensen–Dice coefficient, is a general measure of the similarity of two sets. Being so general, it has found its way to many different areas, including natural language processing, where it's used to compute the similarity between two sentences represented as sets of bigrams. The original *DICE* metric is defined as follows:

Table 4.4: Extended bigrams motivational example.

Sentence	*DICE*	*XDICE*
"I cannot accept such a proposal." *"I don't accept such proposals."*	0.22	0.375

$$DICE(X,Y) = \frac{2 \cdot |X \cap Y|}{|X| + |Y|} \tag{4.7}$$

If we substitute in the set definitions of precision (eq. 4.2) and recall (eq. 4.3) into the equation for F_1 (eq. 4.4) and simplify it, we'll get this exact equation. In fact, *DICE* and F_1, in the set-based context at least, are one and the same. Despite this, it's usefulness lies in the fact that it led to the development of two very powerful metrics which build upon the original.

Brew et al. (1996) extend the original algorithm by introducing extended bigrams, hence the name *XDICE* (extended *DICE*). Extended bigrams consist of regular bigrams with the addition of trigrams with their second element removed, also known as *1-skip-2-grams* (Guthrie et al., 2006). Equations 4.8 and 4.9 illustrate this more precisely.

$$extend(X) = bigrams(X) \cup \textit{1-skip-2-grams}(X) \tag{4.8}$$
$$\textit{1-skip-2-grams}(X) = \{(w_1, w_3) \mid (w_1, \_, w_3) \in trigrams(X)\} \tag{4.9}$$

The *XDICE* metric is then simply defined as:

$$XDICE(X,Y) = DICE(extend(X), extend(Y)) \tag{4.10}$$

The example in the table 4.4 illustrates the advantage of utilizing extended bigrams through two common patterns which the usual bigrams cannot capture at all:

- word insertion: *"such proposal"* to *"such a proposal"*, and
- trigram modification: *"I cannot accept"* to *"I don't accept"*.

Applying the original metric gives us, assuming stemming is applied beforehand, $DICE(X,Y) = 2/9 = 0.22$, since we only have a single bigram match, despite the sentences being rather similar.[4] The improved metric results in $XDICE(X,Y) =$

[4]Stemming reduces words down to their stem by removing any morphological affixes, e.g.,

6/9 = 0.375. While not a huge increase over the original, the observed effects get more pronounced as the compared sentences get longer.

Brew et al. further extend *XDICE* with *XXDICE*, a variation of the algorithm that introduces positional awareness in the form of weights based on matching extended n-grams' positions, instead of weighing every match equally, as was the case previously. For each extended n-gram, the algorithm takes into account the lesser of the two sentences' occurrence counts. Since the original algorithm doesn't explicitly account for repeating n-grams, a simple greedy approach was taken, taking into account the first non-exhasted position from the other sentence. See algorithm 4.1 for this particular implementation.

Algorithm 4.1 XXDICE Algorithm

```
 1: function XXDICE($X, Y$)
 2:     $\tilde{X} \leftarrow extend(X)$
 3:     $\tilde{Y} \leftarrow extend(Y)$
 4:     $exhausted \leftarrow$ array of size $|\tilde{Y}|$ filled with $\perp$ values
 5:     $weights \leftarrow 0$
 6:     for $i \in [0, |\tilde{X}|\rangle$ do
 7:         $j \leftarrow$ first index of $\tilde{X}_i$ in $\tilde{Y}$, such that $exhausted_j = \perp$, or else $-1$
 8:         if $j \geq 0$ then
 9:             $exhausted_j \leftarrow \top$
10:             $weights \leftarrow weights + \frac{2}{1+(i-j)^2}$
11:     return $weights/(|\tilde{X}| + |\tilde{Y}|)$
```

4.2.4 BLEU

Papineni et al. (2002) from IBM introduced the machine translation evaluation measure BLEU (short for *bilingual evaluation understudy*), which is based on the n-gram co-occurrences and a modification of the precision metric. The original BLEU algorithm compares a number of candidate translations with a number of reference translations. For the sake of simplicity, the approach presented here is based on a 1-to-1 comparison, as that's the way the metric was utilized in this work.

"derive", *"derivation"*, and similar all share the stem *"deriv"*. Stemming is the process of removing morphological affixes from words, leaving only the word stem. () Stemming is the process of reducing inflected or derived words to their word stem

Table 4.5: Modified precision motivational example.

Sentence	π	p_n
"The the the the the the the."	1.0	0.286
"The cat is on the mat."		

$$p_n = \frac{\sum_{w \in X} \min\{\text{count}(w, X), \text{count}(w, Y)\}}{|X|} \tag{4.11}$$

The count(x, X) function used in the equation 4.11, as well as in many other features in the paper, returns the number of occurrences of some value x in the collection X.[5] It can be formally defined as count$(x, X) = |\{x' \mid x' \in X,\ x' = x\}|$.

The precision measure here is modified to not take into account just any true positive, but instead to cap that count by the number of occurrences in the reference sentence (eq. 4.11). A simplified version of the example from Papineni's work shown in the table 4.5 best showcases the improvement over the original metric. This *exhaustion* effect can also be seen in *XXDICE* (see algorithm 4.1) as once the indices of a matching word in either sentence are exhausted, the weight cannot be computed and is therefore zero. Furthermore, the algorithm introduces the notion of a brevity penalty to counteract the high precision scores of short translations (eq. 4.13).

$$BLEU = BP \cdot \exp\left(\frac{1}{N} \cdot \sum_{n=1}^{N} \log p_n\right) \tag{4.12}$$

$$BP = \begin{cases} 1 & \text{if } |X| > |Y|, \\ \exp(1 - \frac{|Y|}{|X|}) & \text{otherwise.} \end{cases} \tag{4.13}$$

4.2.5 NIST

NIST's Doddington (2002) attempts to improve upon the IBM's BLEU metric by weighting the n-grams based on their information content.[6] This information content is based on the fact that the number of occurrences of an n-gram of size n are bound to be less or equal to the number of occurrences of its sub-n-gram of size $n-1$. The higher the ratio of the two occurrence counts, the more information

[5]Collection here signifies any iterable structure such as an array, list, set, etc.

[6]United States' National Institute of Standards and Technology.

can be considered to be held by such an n-gram. Just as the BLEU metric, the original purpose of the algorithm was machine translation evaluation. Unlike the BLEU measure, and for that fact most of the measures laid out here, it is not normalized to $[0, 1]$, but is rather unbounded, with a higher score indicating higher similarity.

$$\mathit{Info}(w_1 \dots w_n) = \mathrm{ld}\left(\frac{\mathrm{count}(w_1 \dots w_{n-1}, Y)}{\mathrm{count}(w_1 \dots w_n, Y)}\right) \tag{4.14}$$

The NIST score's brevity penalty also sees improvement over the original BLEU one, as it is much smoother and less sensitive to small variations in length. The β factor is taken to make the brevity penalty equal to 0.5 when the length of the hypothesis is ⅔ that of the reference. Applying simple mathematics, this gives us $\beta = \log 0.5 / \log^2 1.5$.

$$\mathit{NIST} = \mathit{BP} \cdot \sum_{n=1}^{N}\left(\frac{1}{|\mathit{n\text{-}grams}(X)|}\sum_{\mathit{n\text{-}gram} \in X} \mathit{Info}(\mathit{n\text{-}gram})\right) \tag{4.15}$$

$$\mathit{BP} = \exp\left(\beta \log^2 \min\left\{\frac{|X|}{|Y|}, 1\right\}\right) \tag{4.16}$$

4.3 Semantic Features

Semantic features utilize semantic information such as synonymy, paraphrases, and corpus-wide frequencies to provide the system with deeper "understanding" of the provided data.

4.3.1 TER-Plus

TER-Plus (Snover et al., 2008), also known as TERp, is a machine translation metric based on edit distances, i.e., the number of edits required to transform a hypothesis string to a reference string. In its original machine translation context, the hypothesis string is a translation whose similarity is tested, while the reference string, or a multitude of them, is the reference translation to test against. In our plagiarism detection context, we test the hypothetical plagiariased text fragment against its source. As with all other machine translation metrics which support multiple reference translations, we'll only be working with one such reference translation.

TER-Plus builds upon the same author's original TER metric (Snover et al., 2006), where TER stands for the *translation edit rate.* In its simplest formulation, TER can be expressed as:

$$TER(h,r) = \frac{\text{edit}(h,r)}{|r|} \tag{4.17}$$

The edits in question, along with their particular weights, i.e., costs they account for, include:

– word matches with a weight of zero;
– word insertions,
– word deletions,
– word substitutions, and
– shifts of words sequences, i.e., location changes; all with a weight of one.

TER-Plus further extends this by adding stem and synonym matches, as well as paraphrase substitutions, all of which have lower weights than their word edit counterparts. Phrasal substitution here differs from other edits in that it's specified by a vector of weights, $\langle w_1, w_2, w_3, w_4 \rangle$, which are applied as presented in equation 4.18. Here, $\Pr(p_1, p_2)$ represents the probability of p_1 and p_2 being paraphrases, while $\text{edit}(p_1, p_2)$ is equal to the number of edits required for the alignment of said phrases without the substitution being performed. The paraphrases and their respective probabilities are contained in a phrase table consiting of 14,184,361 such entries. Also, the original weights are no longer fixed, but are instead optimized against human judgements such as fluency or adequacy, in order to maximize the metric's correlation with such judgements. Naturally, this optimization applies to the new transformations as well, since they were never tied to any particular weights in the first place.

$$\begin{aligned}\text{cost}(p_1, p_2) = w_1 &\\ &+ w_2 \,\text{edit}(p_1, p_2)\, \log(\Pr(p_1, p_2))\\ &+ w_3 \,\text{edit}(p_1, p_2) \Pr(p_1, p_2)\\ &+ w_4 \,\text{edit}(p_1, p_2)\end{aligned} \tag{4.18}$$

4.3.2 METEOR

METEOR (Denkowski and Lavie, 2011) is another machine translation evaluation metric we'll be utilizing. The method assigns similarity scores by first aligning the hypothesis and the reference, and then computing similarity at sentence level. It aligns the fragments by not only considering exact word matches, but also taking into account stem, synonym, and paraphrase matches, not unlike the previously described TER-Plus metric.

After obtaining the matches, a number of optimizations are performed, such as maximizing the number of covered words, and minimizing the number of contiguous matches, or *chunks*. Then, weighted precision and recall, dependent on meta-parameter δ, are computed and parameterized harmonic mean F_α is calculated. Finally, a fragmentation penalty is applied as a function of the ratio of the average number of matched words and the number of chunks (eq. 4.19), and the final score is computed as presented in equation 4.20.

$$Pen = \gamma \cdot \left(\frac{ch}{m}\right) \tag{4.19}$$

$$METEOR = (1 - Pen) \cdot F_\alpha \tag{4.20}$$

As with TER-Plus, the meta-parameters, α through δ and the weights w_i, are optimized against human judgements.

4.3.3 Term Frequency–Inverse Document Frequency

Term frequency–inverse document frequency, or shortly tf–idf, tries to measure the significance of a word, or term, in relation to the document it's contained in. This metric is, due to its simplicity, very frequently used in many natural language processing tasks, especially in search engines, such as the one we've built previously. As can be assumed from the name, the metric is comprised of two components, the term frequency, and the inverse document frequency. Term frequency (eq. 4.21) measures the word's frequency in its document, while the inverse document frequency signifies the term's importance through the whole corpus of documents. Note that in the context of this work, document refers to a single fragment.

Note that the exact *flavor* of tf–idf presented here is based on NLTK's (Bird et al., 2009) implementation. The core feature only includes the raw counts, while the logarithmic weighing and division-by-zero prevention are specific to this particular implementation.

$$\mathrm{tf}(t,d) = \log(\mathrm{count}(t,d)) + 1 \tag{4.21}$$

$$\mathrm{idf}(t,D) = \log\left(\frac{|D|+1}{|\{d \mid d \in D,\, t \in d\}|+1}\right) + 1 \tag{4.22}$$

$$\mathrm{tfidf}(t,d,D) = \mathrm{tf}(t,d) \cdot \mathrm{idf}(t,D) \tag{4.23}$$

The tf–idf metric is frequently accompanied by cosine similarity, the idea of which is to estimate the similarity of two texts by the angle between them in the high dimensional space of tf–idf features (see equation 4.24). Thus, instead of taking the whole tf–idf vectors as a feature, which can often have tens of thousands of dimensions, we can simply summarize the resulting two tf–idf vectors into this one number.

$$\cos(A,B) = \frac{A \cdot B}{\|A\| \cdot \|B\|} \tag{4.24}$$

4.3.4 Latent Semantic Analysis

Latent semantic analysis (Landauer, 2006), also known as latent semantic indexing when used in certain problems, involves reducing the dimensionality of a term-document matrix such as tf–idf, through truncated singular value decomposition (see eq. 4.25), and thus moving to a low-dimensional *semantic* space. In the referenced equation, X_k represents our approximation of the original matrix X in the semantic space. It's a product of the orthogonal matrices U_k and V_k^T, and the diagonal matrix Σ_k, all of which were obtained by selecting k largest singular values and their vectors from the original U and V obtained through SVD. This not only greatly reduces the dimension of the tf–idf matrix, which is often unreasonably large, while also offering valuable semantic information that we would not otherwise have found with just the tf–idf by itself. In our case, the tf–idf matrix was used as the term-document matrix, although, as was previously noted in 4.3.3, a more proper term would be the term-fragment matrix.

$$X \approx X_k = U_k \, \Sigma_k \, V_k^\mathsf{T} \tag{4.25}$$

As with tf–idf, its possible to take the cosine similarity of two LSA vectors to obtain their similarity in a single number. However, unlike tf–idf, we can afford to include the whole vectors as features due to its low dimensionality. For the two compared fragments, the suspicious text and the potential source text, we take their resulting LSA vectors, $LSA(h)$ and $LSA(R)$, and their absolute difference, $|LSA(h) - LSA(r)|$ as a feature, as well as their cosine similarity. This approach is very similar to the sample vector approach of Ji and Eisenstein (2013), with the addition of cosine similarity.

4.4 Classification

Each *plag* score goes through a threshold function which turns its real value, a measure of how much something is plagiarised, into a binary category value signifying whether it's plagiarism or not. These binary plagiarism values are then used as the target against which we train and test the model through the classifier. With these categorical values, we can finally perform classification through a support vector machine.

Support vector machines (Cortes and Vapnik, 1995) are a class of machine learning classification methods which attempt to learn to classify examples by identifying the hyperplanes separating the said classes in the feature space. The method represents the high-dimensional cross product through a kernel function, $K : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$, which plays a great part in correctly separating the data. For this particular task, the radial basis function kernel (Vert et al., 2004) was utilized.

The hyper-parameters of the SVM, C and RBF kernel's γ, as well as the kernel itself, were selected by performing a cross-validated grid search with exponentially increasing values, e.g., $C, \gamma \in \{2 \times 10^{-6}, 2 \times 10^{-4}, \ldots, 2 \times 10^{4}\}$. The parameters were optimized against the weighted macro-averaged F_1 metric, described later on in section 5. In the end, the combination of parameters $C = 200$ and $\gamma = 0.0002$ proved to be provide the best results. SVM's linear kernel was included in the search alongside the RBF kernel, however it generally showed poorer results.

5 Results

To recap, after preprocessing the dataset we end up with a several thousand of examples, depending on the window size, ranging from 864 for the window size of one, to 3,887 for the window size of six, which was the largest window size tested. After the parameters were optimized, as described in section 4.4, three-fold cross-validated scoring was performed, computing the following micro-averaged metrics:

- precision (see statistical variant of equation 4.2);
- recall (see statistical variant of equation 4.2);
- accuracy, which simply represents the fraction of correct predictions, defined as $accuracy = (TP + TN)/(TP + TN + FP + FN)$; and
- the F_1 score (see eq. 4.4);

as well as the following macro-averaged metrics:

- the unweighted F_1 score; and
- the weighted F_1 score, which tries to account for imbalances in the dataset by weighing each category's individual scores according to the fraction of the total population said category makes up.

Micro-averaged metrics simply compute the metric over the whole population, while the macro-averaged ones do so for each label separately, and then average the individual results.

5.1 Remarks

The plots in Figure 5.1 show how the six tested performance metrics change with window size and the weighing scheme used. The horizontal axis shows the change in window size, the vertical displays the performance metric results, while the different lines on the plots portray the weighing schemes as described in the

Figure 5.1: Result plots for the MST translation model.

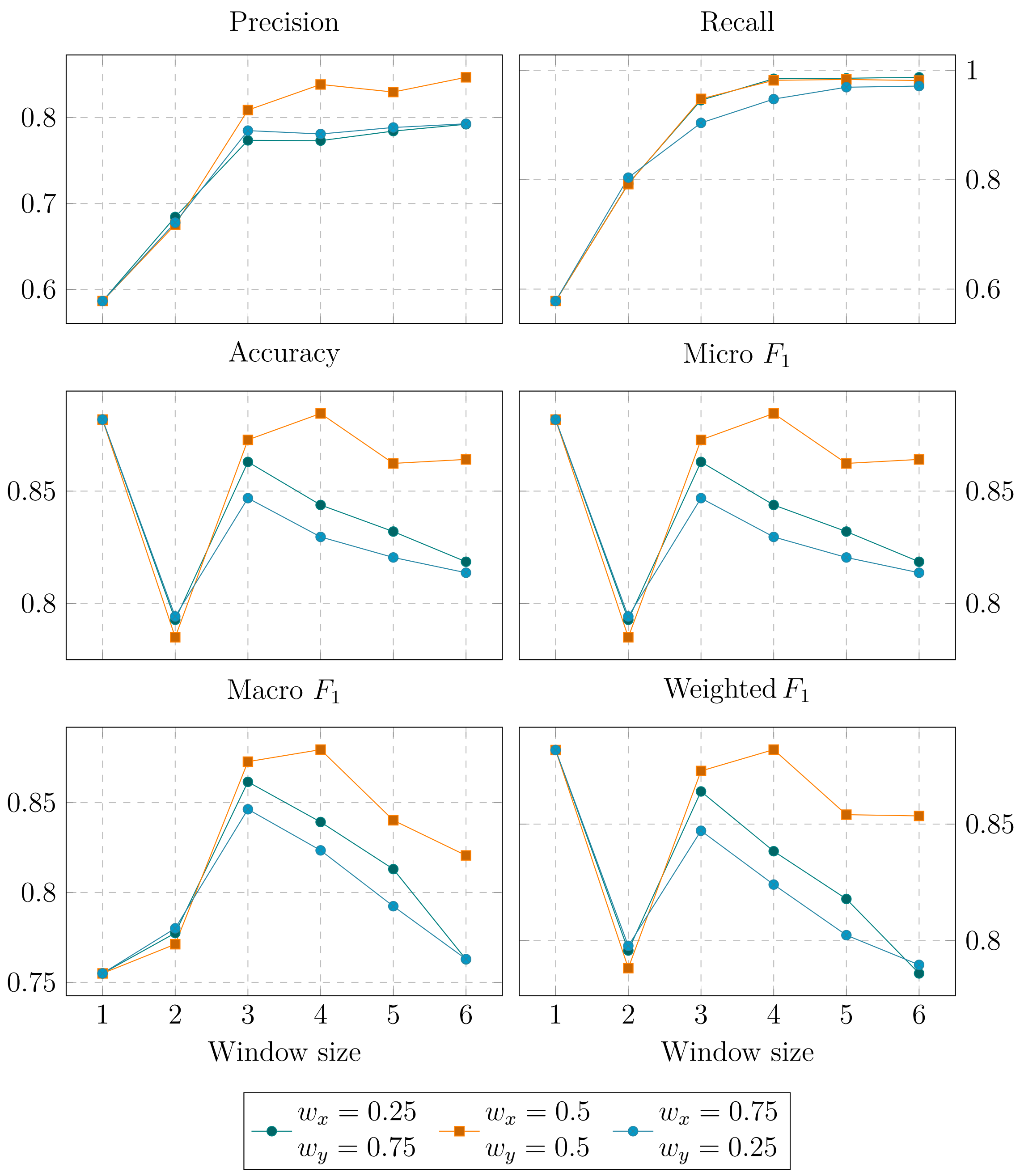


Figure 5.2: Comparison of window sizes' results.

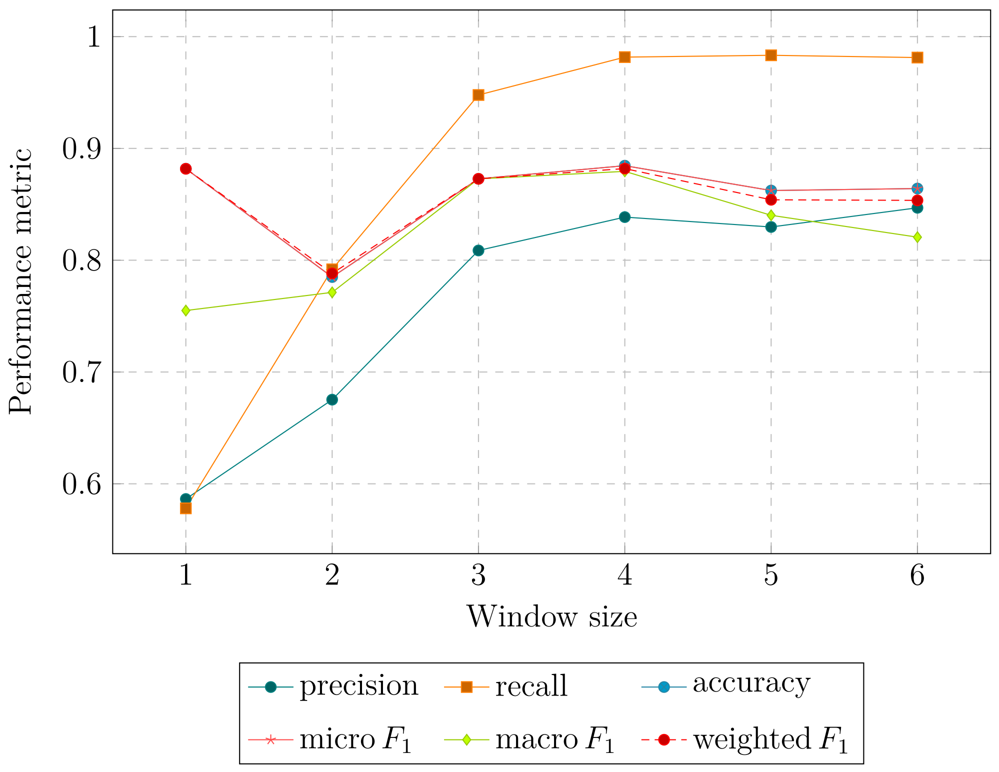


Figure 5.3: Translation model results comparison with window size of six.

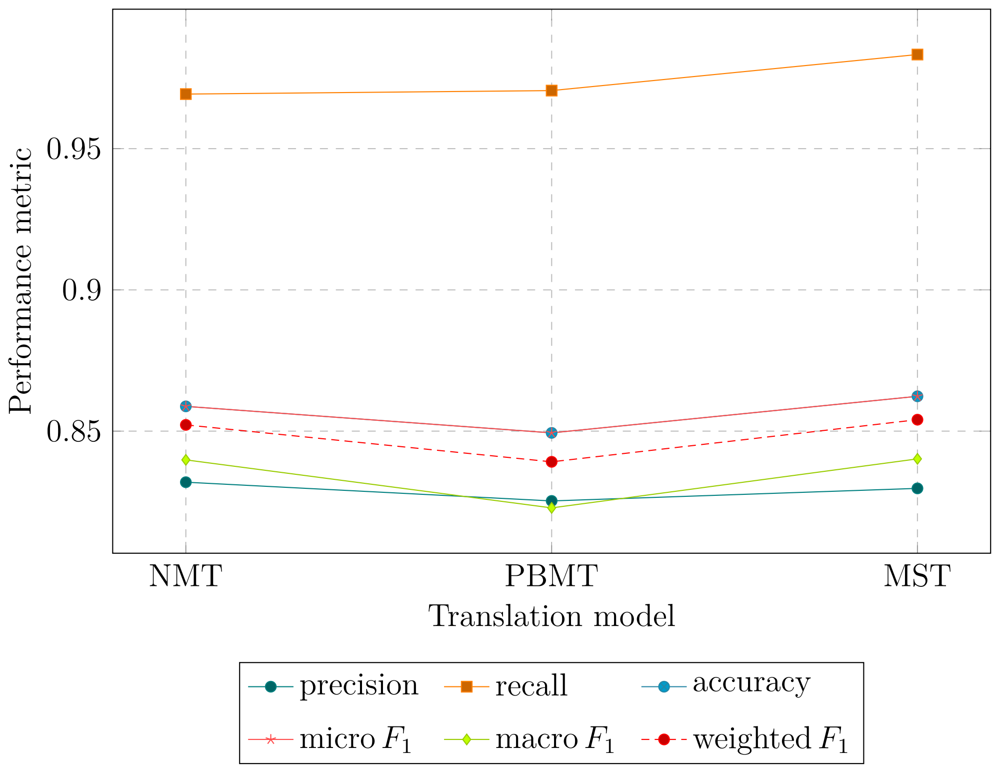


Table 5.1: Evaluation results for the weighing scheme $w_x = 0.5, w_y = 0.5$ with the MST model being used.

Window size	Precision	Recall	Accuracy	Micro F_1	Macro F_1	Weighted F_1
1	0.586	0.578	0.882	0.882	0.755	0.882
2	0.675	0.792	0.785	0.785	0.771	0.788
3	0.809	0.948	0.873	0.873	0.873	0.873
4	0.839	0.982	**0.885**	**0.885**	**0.879**	**0.882**
5	0.829	**0.983**	0.862	0.862	0.840	0.854
6	**0.847**	0.981	0.864	0.864	0.821	0.854

plot's legend. As can be seen from the plots in Figure 5.1, the *plag* score weighting scheme which places equal weight on both of its components consistently achieves the best results. The rather high results are most likely a consequence of the small size of the dataset. Applying the same model on a larger dataset would most likely result in lower results.

We can see in the Figure 5.2 how recall consistently increases with the window size. However, other metrics don't seem to be affected much after window size of three, which is incidentally the median Wikipedia fragment size in our dataset. Interestingly enough, Figure 5.3 shows that Microsoft's Translator, despite subjectively its translations not being nearly as natural as Google NMT's, seems to provide close or at times even better results than Google's NMT. Table 5.1 shows results for the weighting scheme $w_x = 0.5, w_y = 0.5$ accompanied with the MST translation model, with the best results in each metric highlighted.

These results, however, don't tell the whole story. Due to the windowing performed, each sentence is covered by a number of these windows, and is therefore included in a number of plagiarism predictions. In order to make sense of this data, we could try to coerce nearby windows into contiguous chunks of text by taking look at the positions of their source matches in their respective source documents. If we have two nearby windows in the suspicious document, and we predict that each of the windows plagiarises a source window from Wikipedia, and these two source windows are also very close, we could combine them into a single prediction.

6 Conclusion

The work presented here tackles the problem of cross-lingual plagiarism detection through use of advanced commercial translation services in an attempt to reduce the problem down to mono-lingual plagiarism detection. A subset of Wikipedia covering topics in machine learning was used as the dataset and from which synthetic example papers were built. A local search engine based on latent semantic analysis was built for the purpose of reducing the search space of the said dataset. Finally, binary classification was performed on a windowed representation of the synthetic after a number of syntactic and semantic features were extracted from it.

The model shows good results, however due to the rather small size of the dataset, the exact figures should be taken with a grain of salt. Evaluation of a larger dataset would paint a clearer picture of the model's performance.

As for future work, there is a lot that could be improved upon. The single biggest issue with the model presented here is the lack of data from which to generate papers. A dataset could be generated by translating parts of Wikipedia articles through translation services, and the resulting model could be tested against the manually translated data. This would in turn give us information on both the quality of the translation service and how appropriate it is for cross-lingual paper generation, and more importantly, the behaviour of our model when exposed to a larger corpus. Another path one could explore would be automatic generation of fragments by syntactically and semantically modifying the existing examples.

Bibliography

Steven Bird, Ewan Klein, and Edward Loper. *Natural Language Processing with Python*. O'Reilly Media, Inc., 1st publication, 2009. ISBN 0596516495, 9780596516499.

Chris Brew, David McKelvie, and Buccleuch Place. Word-pair extraction for lexicography, 1996.

Meri; Coleman and T. L. Liau. *A computer readability formula designed for machine scoring*, volume 60. Journal of Applied Psychology, 1975.

Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Mach. Learn.*, 20 (3):273–297, September 1995. ISSN 0885-6125. doi: 10.1023/A:1022627411411. URL http://dx.doi.org/10.1023/A:1022627411411.

Michael Denkowski and Alon Lavie. Meteor 1.3: Automatic metric for reliable optimization and evaluation of machine translation systems. In *Proceedings of the sixth workshop on statistical machine translation*, stranice 85–91. Association for Computational Linguistics, 2011.

Lee R. Dice. Measures of the amount of ecologic association between species. *Ecology*, 26(3):297–302, 1945. ISSN 1939-9170. doi: 10.2307/1932409. URL http://dx.doi.org/10.2307/1932409.

George Doddington. Automatic evaluation of machine translation quality using n-gram co-occurrence statistics. In *Proceedings of the Second International Conference on Human Language Technology Research*, HLT '02, stranice 138–145, San Francisco, CA, USA, 2002. Morgan Kaufmann Publishers Inc. URL http://dl.acm.org/citation.cfm?id=1289189.1289273.

Marc Franco-Salvador, Parth Gupta, and Paolo Rosso. Cross-language plagiarism detection using a multilingual semantic network. In *Proceedings of the 35th*

European Conference on Advances in Information Retrieval, ECIR'13, stranice 710–713, Berlin, Heidelberg, 2013. Springer-Verlag. ISBN 978-3-642-36972-8. doi: 10.1007/978-3-642-36973-5_66. URL http://dx.doi.org/10.1007/978-3-642-36973-5_66.

D. Guthrie, B. Allison, W. Liu, L. Guthrie, and Y. Wilks. A Closer Look at Skip-Gram Modelling. In *Proceedings of the Fifth International Conference on Language Resources and Evaluation (LREC-2006)*, Genoa, Italy, 2006.

Samer Hassan and Rada Mihalcea. Cross-lingual semantic relatedness using encyclopedic knowledge. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 3 - Volume 3*, EMNLP '09, stranice 1192–1201, Stroudsburg, PA, USA, 2009. Association for Computational Linguistics. ISBN 978-1-932432-63-3. URL http://dl.acm.org/citation.cfm?id=1699648.1699665.

Yangfeng Ji and Jacob Eisenstein. Discriminative improvements to distributional sentence similarity. In *EMNLP*, 2013.

Thomas K Landauer. *Latent semantic analysis.* Wiley Online Library, 2006.

Alon Lavie, Kenji Sagae, and Shyamsundar Jayaraman. The significance of recall in automatic metrics for mt evaluation. In *In Proceedings of the 6th Conference of the Association for Machine Translation in the Americas (AMTA-2004)*, 2004.

Nitin Madnani, Joel Tetreault, and Martin Chodorow. Re-examining machine translation metrics for paraphrase identification. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, NAACL HLT '12, stranice 182–190, Stroudsburg, PA, USA, 2012. Association for Computational Linguistics. ISBN 978-1-937284-20-6. URL http://dl.acm.org/citation.cfm?id=2382029.2382055.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: A method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ACL '02, stranice 311–318, Stroudsburg, PA, USA, 2002. Association for Computational Linguistics. doi: 10.3115/1073083.1073135. URL http://dx.doi.org/10.3115/1073083.1073135.

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

E. A. Senter, R. J.; Smith. *Automated Readibility Index.* Wright-Patterson Air Force Base, 1967.

M. Snover, N. Madnani, Bonnie J Dorr, and R. Schwartz. Terp system description. *MetricsMATR workshop at AMTA*, 2008/// 2008.

Matthew Snover, Bonnie Dorr, Richard Schwartz, Linnea Micciulla, and John Makhoul. A study of translation edit rate with targeted human annotation. In *Proceedings of association for machine translation in the Americas*, volume 200, 2006.

Jean-Philippe Vert, Koji Tsuda, and Bernhard Schölkopf. A primer on kernel methods. *Kernel Methods in Computational Biology*, stranice 35–70, 2004.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Lukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. Google's neural machine translation system: Bridging the gap between human and machine translation. *CoRR*, abs/1609.08144, 2016. URL `http://arxiv.org/abs/1609.08144`.

Appendix A
Code Snippets

In this chapter we'll go through some details of the implementation itself, the libraries used, and the problems, hopefully accompanied with solutions, encountered along the way.

A.1 Tokenization

The code presented in the listing A.1 enhances the NLTK's default pre-trained English sentence tokenizer to properly handle *"i.e."* and *"e.g."* abbreviations in the text. The default behaviour incorrectly assumes that the aforementioned abbreviations imply the end of a sentence when followed by whitespace. By accessing the tokenizer's private member variables, we can extend its internal list of abbreviations to correctly handle these cases.

Listing A.1 Enhancing NLTK sentence tokenizer's abbreviation handling.

```python
import nltk

tokenizer = nltk.data.load('tokenizers/punkt/english.pickle')
tokenizer._params.abbrev_types.update(['i.e', 'e.g'])

def sent_tokenize(text):
    return tokenizer.tokenize(text)
```

A.2 Wikipedia category pages

Obtaining the list of all pages contained under some Wikipedia category comes down to a simple depth-first search through the category tree available in the public Wikipedia API.[1]

Listing A.2 Obtaining pages contained under a Wikipedia category.

```python
from json import loads
from requests import get

def get_pages(category):
    ignored = [u'Draft', u'Portal', u'Textures', u'User'] # and many more
    visited = set()
    ps = set()
    cs = set([category])
    while cs:
        visited |= cs
        ps |= set(p for c in cs for p in apiCall(c, 'page'))
        cs = set(c_sub for c in cs for c_sub in apiCall(c, "subcat")) - visited
    return ps

def apiCall(category, type):
    url = BASE_URL.format(category=category, type=type)
    return (p['title'] for p in loads(get(url).text)['query']['categorymembers'])
```

A.3 Wikipedia Search Engine Details

The listing A.3 here accompany the search procedure description in the section 3.4. The transformations used in the listing, namely the `CountVectorizer`, the `TfidfTransformer`, and the `TruncatedSVD`, all come from the Python's *scikit-learn* (Pedregosa et al., 2011) machine learning library.[2][3][4] Pipelines mentioned

[1] https://en.wikipedia.org/w/api.php?action=query&format=json&cmlimit=500&cmprop=title&list=categorymembers&cmtitle=Category:{category}&cmtype={type}

[2] http://scikit-learn.org/0.18/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html

[3] http://scikit-learn.org/0.18/modules/generated/sklearn.feature_extraction.text.TfidfTransformer.html

[4] http://scikit-learn.org/0.18/modules/generated/sklearn.decomposition.TruncatedSVD.html

in the code snippets' comments refer to *scikit-learns* `Pipelines`, a method of chaining sequential transformations without the unnecessary boilerplate and the bookkeeping of the intermediate results.[5]

Listing A.3 Keyword extraction and Wikipedia corpus search.

```
from numpy import argsort
from itertools import izip
from operator import itemgetter

def search(fragment, n_keywords, n_results):
    # `tfidf`: pipeline of `CountVectorizer` with stemming and `TfidfTransformer`
    # `tfidf_terms`: `tfidf` feature names
    tfidf_vec = tfidf.transform([fragment])
    tfidf_rank = sorted(
        ((tfidf_terms[j], x)
         for j, x in izip(tfidf_vec.indices, tfidf_vec.data)),
        key=itemgetter(1)
    )
    keywords = next(izip(*tfidf_rank[:n_keywords]))

    # `index`: array of (Wikipedia file, its index)
    # `lsa`: pipeline of `tfidf` and `TruncatedSVD`
    cos = cosine_similarity(lsa.transform([' '.join(keywords)]), trained)[0]
    # Take `n_results` entries in descending order
    cos_ixs = argsort(cos)[:-(n_results + 1):-1]
    return [(i, index[i], cos[i]) for i in cos_ixs[:n_results]]
```

A.4 METEOR and TER-Plus implementation

Due to the complexity of these two measures and the large datasets that accompany them, no native implementation was attempted. Instead, the Java libraries, released by none other than the authors themselves, were used. To accomplish this, Python's *jnius* library was used to communicate with a Java Virtual Machine through the JNI – *Java Native Interface*.[6][7]

[5] http://scikit-learn.org/0.18/modules/generated/sklearn.pipeline.Pipeline.html
[6] http://docs.oracle.com/javase/7/docs/technotes/guides/jni/spec/jniTOC.html
[7] https://pyjnius.readthedocs.io/en/latest/

The listing A.4 shows the initial configuration of the JVM class-paths and the memory settings recommended by the library authors.

Listing A.4 Python-JVM binding setup.

```
import jnius_config

def jvm_setup():
    if not jnius_config.vm_running:
        jnius_config.add_options('-Xms1G')
        jnius_config.add_options('-Xmx3G')
        jnius_config.add_classpath(METEOR_JAR_PATH)
        jnius_config.add_classpath(TERP_JAR_PATH)
```

The METEOR library was very easy to interface to due to their clean architecture and the direct access to the scoring functions.[8]

Listing A.5 TER-Plus integration.

```
def to_trans(sentences):
    return '\n'.join(s + ' ([sys][doc][' + str(i + 1) + '])'
                     for i, s in enumerate(sentences))

def score(self, hyp, ref):
    with open(HYP_PATH, 'w') as hyp_f: hyp_f.write(to_trans(hyp))
    with open(REF_PATH, 'w') as ref_f: ref_f.write(to_trans(ref))

    TERPara.getOpts(['terp', '-h', HYP_PATH, '-r', REF_PATH])
    output = TERPlus().run()
    # `to_py_iter` transforms a Java collection into a Python iterable
    results = [output.getResult(ter_id) for ter_id in to_py_iter(output.getIds())]

    return sum(r.numEdits for r in results) / sum(r.numWords for r in results)
```

TER-Plus, on the other hand, proved to be much more difficult to cooperate with.[9] The library doesn't offer any direct access to the scoring methods and instead the arguments have to be provided in a command-line-interface-like manner and the compared texts have to be provided via files in either the XML,

[8]http://www.cs.cmu.edu/~alavie/METEOR/.

[9]https://github.com/snover/terp.

SGML, or their own TRANS format. To counter this, temporary files for the reference and candidate texts were created at the start of the scoring and each of the compared fragments were written to the files during their individual scoring time. The otherwise unavoidable excessive output and its parsing were avoided by directly computing the final result of the metric from the accessible intermediary results. The exact procedure can be seen in the listing A.5.

Still, despite all the hurdles, most of which are unavoidable anyway, the presented approach is most likely an order of magnitude faster than manually invoking the JVM through a shell library.

Cross-Lingual Plagiarism Detection from Wikipedia

Abstract

Plagiarism detection is a natural language processing task with the purpose of finding plagiarised sentences, paragraphs, or text fragments in a suspicious document, and retrieving their sources. Cross-lingual plagiarism detection further expands upon this by considering source works in languages other than the original work's language. In this particular case we consider Croatian papers with the detection of plagiarised English sources. As Wikipedia is one of the most common plagiarism sources, it was chosen as the corpus against which to perform the detection, in particular, a subset of its pertaining to machine learning. The work explores a translational model which utilizes a number of syntactic and semantic features in combination with the latent semantic analysis.

Keywords: cross-lingual plagiarism detection, latent semantic analysis, natural language processing, machine learning, Wikipedia, Croatian language.

Međujezično otkrivanje plagijata s Wikipedije

Sažetak

Detekcija plagijata je zadatak iz područja analize prirodnih jezika koji se bavi pronalaženjem plagijariziranih rečenica, paragrafa te komada teksta u sumnjivom dokumentu i dohvaćanjem njihovih izvora. U ovom slučaju uzimamo u obzir radove na hrvatskom s detekcijom plagijariziranih izvora na engleskom. Pošto je Wikipedija jedan od najčešćih izvora plagijarizma, odabrana je kao tijelo teksta u odnosu na koje izvršavamo detekciju, s ograničenjem na podskup Wikipedije koji se bavi temama iz područja strojnog učenja. Ovaj rad istražuje prijevodni model koji koristi sintaktičke i semantičke značajke u kombinaciji s latentnom semantičkom analizom.

Ključne riječi: međujezična detekcija plagijata, latentna semantička analiza, procesiranje prirodnih jezika, strojno učenje, Wikipedija, hrvatski jezik.